To Buy or Not to Buy:

Predicting E-Commerce Shoppers’ Purchase Intentions

DS4400: Professor John Rachlin

Spring 2023

Luke Abbatessa, Jocelyn Ju, and Yuting Zheng

**Abstract**

This project was intended to determine whether an e-commerce shopper would ultimately make a purchase by taking into consideration a variety of factors including bounce rate, closeness to a special day, and visitor type. To accomplish this, we found machine learning classification algorithms that would perform the best for the data types and classification class values in the dataset. Using Sci-Kit Learn and self-developed machine learning algorithms for the perceptron model and decision tree algorithm with optimal feature selections, hyperparameter tuning, and cross-validation methods, as well as the neural networks algorithm, e-commerce consumers are classified into buying or not buying based on their behaviors measured using the attribute values. Identifying major attribute values for consumer behaviors like duration of page visits, exit rates, bounce rates, and page values, businesses in the e-commerce industry can determine website features, advertisements, and user interface interactions that would attract consumers and positively influence their purchasing decisions. The best model found was our self-developed Decision Tree algorithm with all of the features.

**Background**

In a business, it is always essential to be able to predict the number of consumers that are going to buy your products based on consumer behavior in order to accurately forecast demand for the next season. This concept not only applies to in-store scenarios, but also to the e-commerce world. Businesses that are involved in e-commerce also need to have a way to predict the number of purchases for a certain product in order to more accurately plan their supply chain operations like how many products they should order from their manufacturers. Over-forecasted demands can lead to an increase in holding costs and under forecasting demand can lead to a loss in profit because when products are not in stock, consumers will look for alternatives. Understanding which customers are most likely to make a purchase allows the business to better tailor their marketing approaches to those customers. If a customer is already planning to make a purchase, it does not make sense to send a coupon, because then you will not make as much money as you would otherwise. However, if a customer will not make a purchase, sending a coupon or otherwise engaging them in the company may encourage them to do so.

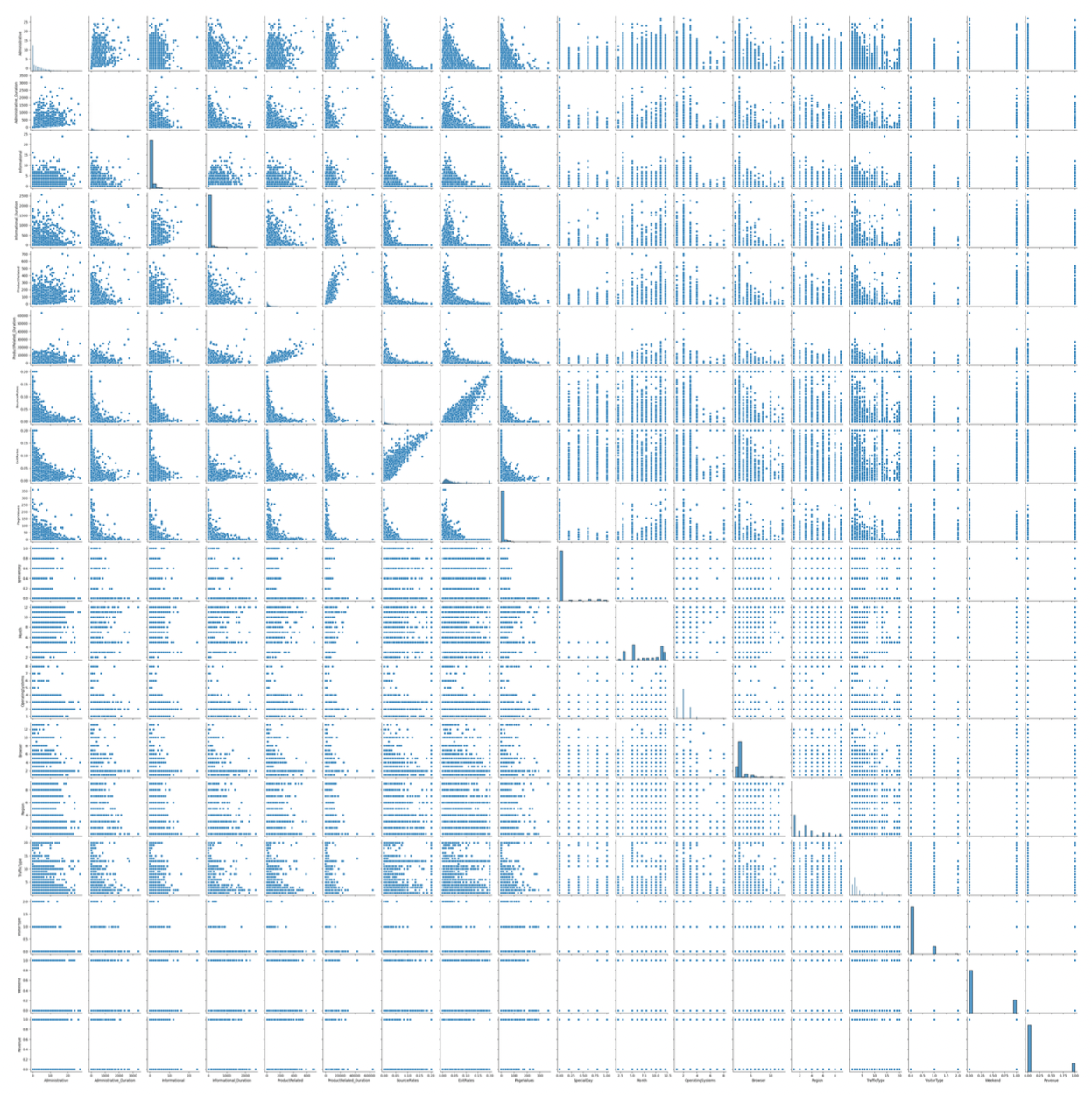
Currently, machine learning is present in the e-commerce industry, primarily through personalization with ads, informing sellers to best create relevant marketing campaigns, managing supply and demand, detecting fraud, and adjusting pricing with the market. This enables businesses (big and small) to be able to predict fluctuations and adjust accordingly. Additionally, machine learning often directs increased traffic to the business through targeted ads and marketing, such as offering return customers a 30% coupon. As machine learning and artificial intelligence become increasingly mainstream, businesses are quick to take advantage.

Prior to this project, another person, GitHub username zeglam, performed a similar analysis on this data. In their project, zeglam performed classification using Naive Bayes, KNN, SVM, Logistic Regression, Random Forest, Gradient Boost, and AdaBoost, finding the best performance with Gradient Boost, accuracy of 0.905, F1-score of 0.689, Precision value of 0.761, and Recall of 0.630. For our approach to this dataset, we opted to compare self-developed models to SciKit-Learn models, as well as incorporating neural networks in an effort to address the data from a different angle.

**Data Analysis**

The data source contains 10 numerical and 8 categorical attribute values. The 18 attributes are Administrative, Administrative\_Duration, Informational, Informational\_Duration, ProductRelated, ProductRelated\_Duration, BounceRates, ExitRates, PageValues, SpecialDay, Month, OperatingSystems, Browser, Region, TrafficType, VisitorType, Weekend, and Revenue. The main attributes we focused on were ProductRelated, ProductRelated\_Duration, ExitRates, BounceRates, PageValues, SpecialDay, Month, Region, TrafficType, VisitorType, Weekend, and Revenue. Reason being, we felt that these attributes would help us analyze customer patterns on different websites and product types to determine which website features and product types are most attractive.

In terms of data cleaning, after the dataset was read in as a Pandas DataFrame, we converted object-type variables to strings, we dropped duplicate rows, and we made sure to shuffle the rows of the data frame to eliminate any possible biases. In addition, all features with datatype “string” and “bool” were converted to an integer. This includes Month, for which months January through December were assigned numbers 1 through 12, respectively. VisitorType contained strings as well, with types “Returning\_Visitor” set to 0, “New\_Visitor” as 1, and “Other” as 2. Additionally, boolean value columns “Weekend” and “Revenue” were set to 0 for “False” and 1 for “True”.



*Fig 1: Pair Plots depicting the correlation between the attributes of the data*

Plotting the correlation between attributes, as shown in Figure 1, we found there was minimal correlation between any of the explanatory variables and Revenue, our target feature. Due to this, we implemented additional methods to determine which features, if any, we would be able to discard without influencing the benchmarks of the developed models. One thing we did find from the pairplot above was that multiple explanatory variables were confounded, meaning that correlations existed with each other, which caused a false implication of causation. The unpredictability of these confounding variables can alter the reliability of our models.

**Methods**

#### **Test-Train Split**

Using Sci-Kit Learn’s train\_test\_split technique, the data was split into training and testing sets with the shuffled parameter as true. The train\_test\_split technique is meant to split data structures into random training and test subsets. It is important in the context of machine learning algorithms because it is used to treat the bias-variance trade-off. To elaborate, a model that has high bias is not flexible towards other datasets; on the other hand, a model that has high variance is likely to make poorer predictions. With train\_test\_split, the user is more likely to produce a model that has lower bias and lower variance. In the context of this project, the training data was 70% of the original dataset and the testing set was 30% of the original dataset.

#### **Feature Selection**

To determine optimal features for which to employ our models, we used tree-based feature selection and a pipeline method. Tree-based feature selection can be used to compute impurity-based feature importances, which in turn can be used to discard irrelevant features. Similarly, a pipeline method is used to evaluate feature importances and select the most relevant features; its purpose is to assemble several steps that can be cross-validated together while setting different parameters. Once these were completed, we compared the feature lists that the feature selection methods returned.

**Machine Learning Models: Employment & Rationale**

For employment, the perceptron model, the decision tree algorithm, and the neural network algorithm were utilized. Using the three different classification algorithms, we wanted to compare the five metrics scores between our own model and SciKit-Learn's model to recognize differences between results and what each of the results meant. We chose the perceptron model because we felt that it better allowed for multiple features to be compared when predicting the class. Also, from previous experience with implementing machine learning algorithms from scratch, the perceptron model took the least time to produce output, thus we felt the most confident in tuning the hyperparameters of this model. We chose the decision tree algorithm because the original dataset had a significant number of features (17), thus we felt that it could be interesting to gauge which feature(s) would produce the most uniformity upon branch splits of the tree (e.g. which branch splits would produce the leaves with the most class similarity). Lastly, we chose the neural network algorithm because we wanted to supplement our classification-based analysis of our data.

**Hyperparameter Tuning Methods**

In order to tune our machine learning algorithms to produce the best metrics while not overfitting our data, we used GridSearchCV for the decision tree algorithms. SciKit-Learn’s GridSearchCV algorithm tries all the exhaustive combinations of parameter values supplied by the user and chooses the best value for each parameter supplied. For example, if the user provides a list of values to try for three hyperparameters, then GridSearchCV will try all possible combinations; this could translate to 2o combinations of hyperparameters or more. A 10-fold cross-validation was used in the GridSearchCV algorithm with 70% of our training data split into training and validation sets; this was meant to prevent overfitting, as we are using the other 30% of our data as the testing set after we hyper tuned our parameters for the decision tree algorithm.

As for the perceptron algorithm, the only tuning that was done was feature selection because the other features for the perceptron algorithm were epochs (number of iterations) and alpha (the learning rate). The number of epochs should not be too small, because when there are not enough iterations, the set of feature values will not receive enough training to produce accurate results for the corresponding labels. If the number of iterations increases, it reaches a certain threshold, where the accuracy scores will have minor variations between each increase in iteration. Therefore, for our model, we set the number of epochs to be 1000, since we felt there would only be minor changes in the metric scores with each additional iteration. As for the alpha value, the alpha value represents the learning rate; you do not want too fast of a learning curve (might overshoot the optimal classification result) or too slow of a learning curve (time expensive). Thus, we set the learning rate at 0.0001; this was the same parameter value from lecture. We felt this to be the ideal learning rate, as tuning the learning rate was not critical since it would not be as influential.

**Cross-Validation Methods**

Cross-validation is important because it prevents the user from using the test set to iteratively tune the model at hand, which may lead to overfitting. If the test set were overtuned, it would lead to a higher chance of overfitting, which would lead to poorer predictions. Fortunately, cross-validation produces a training set and a validation set over a set number of folds; the validation sets can be tuned and optimized using performance feedback (thus acting as a pseudo test set), and they can be used to report an objective measure of predictive accuracy for evaluation purposes.We applied the 10-fold cross validation method in our hyperparameter tuning for the self developed decision tree algorithm and Sci-Kit Learn’s decision tree classifier to determine the best hyperparameters to be used for the model.

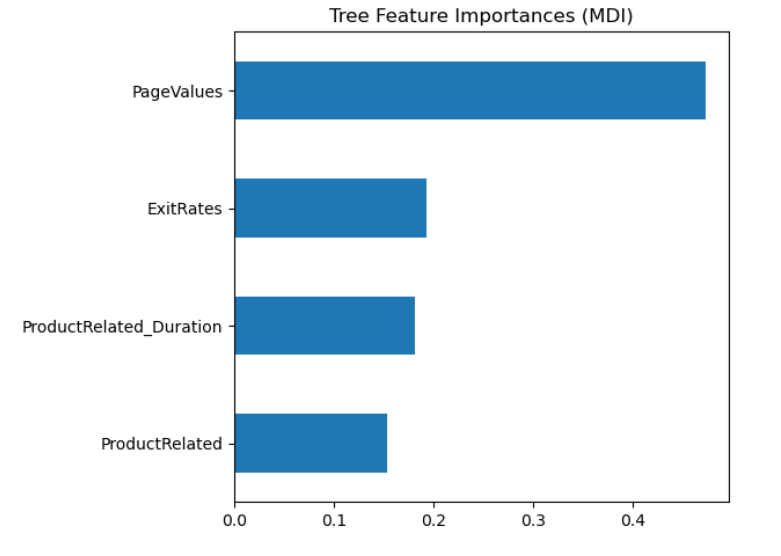
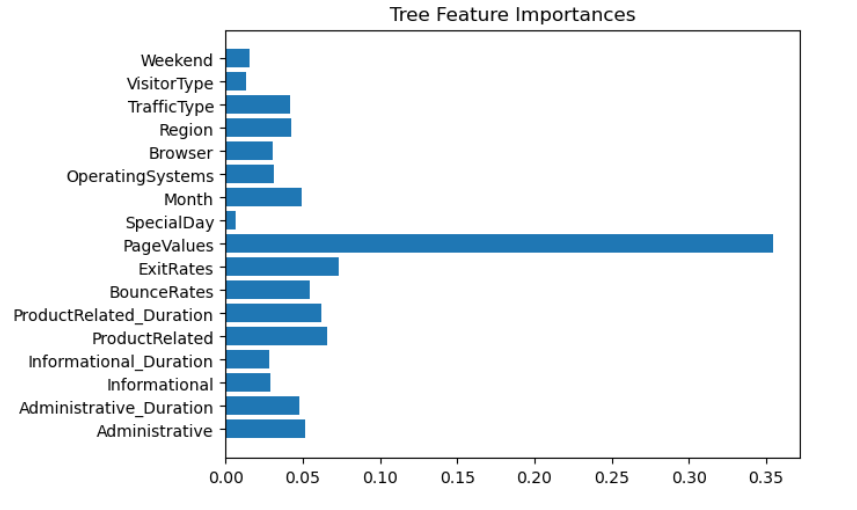
**Analysis**

This project’s broader application is to the business world. As such, ensuring that the maximum revenue is achieved is likely the greater goal. Because of this, false negatives, meaning that someone made a purchase when they were not anticipated to, are a better outcome than false positives, where an expected purchase was not made. Incorrect predictions of revenue occurring may cost the company much in terms of marketing, potential coupons, and other expectations. Thus, we placed a greater emphasis on specificity than sensitivity. Additionally, while accuracy is important, the data was skewed with approximately five times more False “Revenue” values than True.

**Train-Test Split**

Using Sci-Kit Learn’s train\_test\_split, we split the original dataset into four sectors: X\_train, X\_test, y\_train, and y\_test. X\_train and y\_train represent data frames consisting of the features and target of the training set, respectively, and they are fit to whichever model is being implemented to produce a “best fit model.” We allotted them 70% of the data to ensure the model was fed enough data for tuning/manipulating purposes. On the other hand, X\_test and y\_test represent data frames consisting of the features and target of the test set, respectively, and they are used to measure just how good the model is. We allotted them 30% of the data because the test set is solely meant to ensure the model is performing accurately. It should never be the case that the test set is used to hypertune the model, because this risks the possibility of overfitting. Additionally, for the training set, it is further split using cross fold validation into another training set as well as a validation set. All machine learning models are trained using the training set, tuned with a validation set, and then in the end tested with the testing set, making sure that our metric values provided more accurate results since the algorithm was not trained using the actual testing data.

**Feature Selection**

*Fig 2: Pipeline Feature Importance (MDI) Fig 3: Tree Feature Importance*

*Horizontal bar charts of feature importances determined by Mean Decrease in Impurity (MDI) Pipeline (Fig 2) and Tree-Based Feature Selection (Fig 3)*

The tree-based feature selection found ProductRelated, ProductRelated\_Duration, ExitRates, and PageValues to be the most valuable features. Based on the tree-based feature selection method, the feature importance value for ProductRelated was approximately 0.066, the feature importance value for ProductRelated\_Duration was approximately 0.062, the feature selection value for ExitRates was approximately 0.074, and the feature selection value for PageValues was approximately 0.354. Using the pipeline method, the selection criteria was the Mean Decrease in Impurity (MDI); the feature importance values based on MDI for ProductRelated, ProductRelated\_Duration, ExitRates, and PageValues were approximately 0.153, 0.181, 0.193, and 0.472, respectively. Ultimately, we found that ProductRelated, ProductRelated\_Duration, ExitRates, and PageValues were the best features; since the tree-based feature selection method and the pipeline method were in agreement with regards to the best features, there was no optimal feature selection technique.

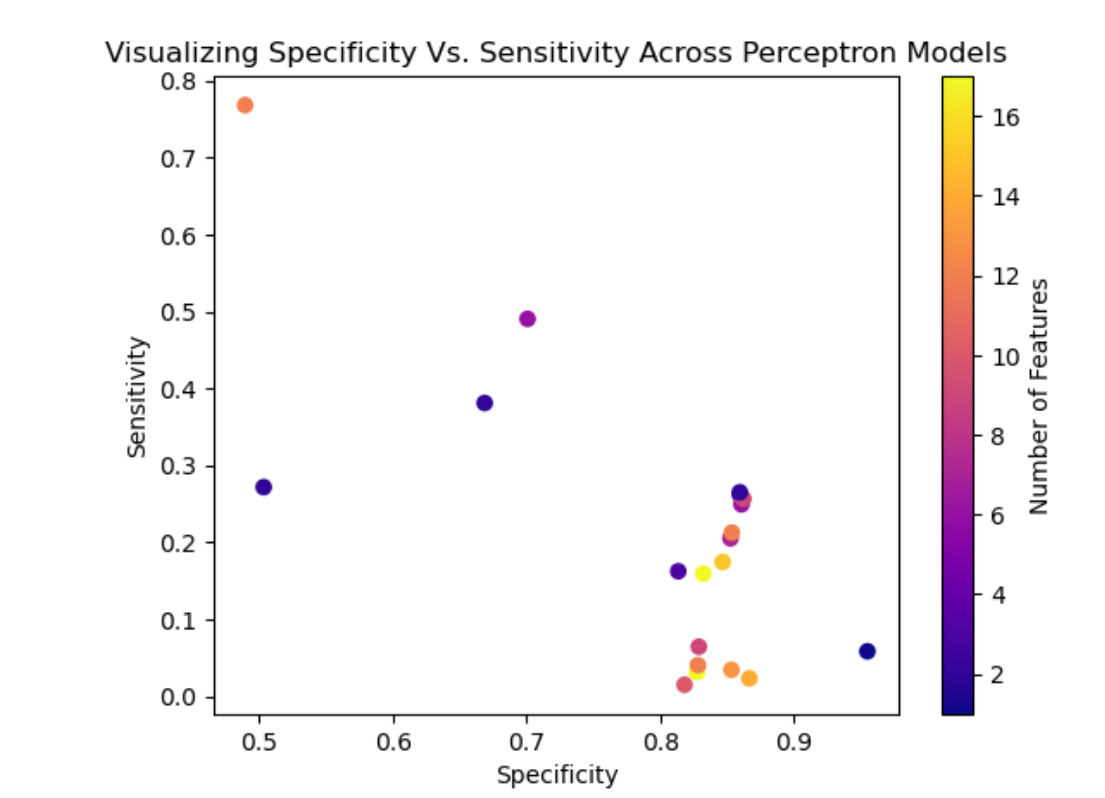
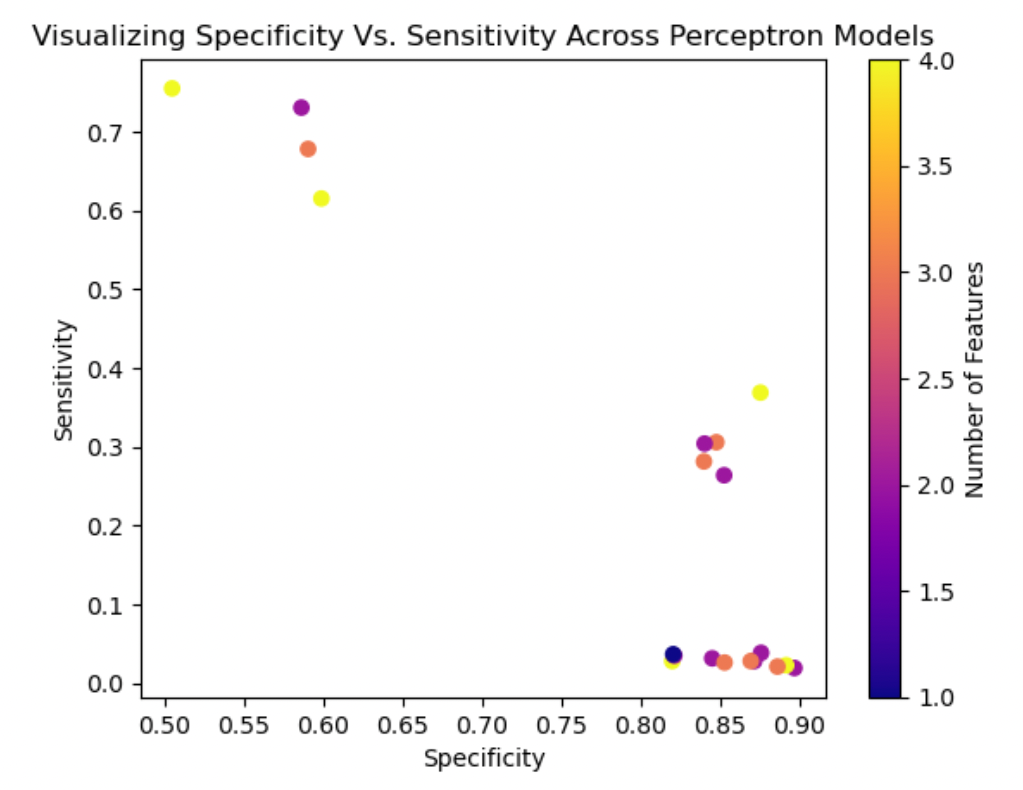
**Hyperparameter Tuning Methods**

GridSearchCV is important for hyperparameter tuning because based on the parameters that the user feeds it with regards to a specific estimator, it will return the optimal values for each parameter, in turn leading to the optimal metrics produced for the model. GridSearchCV also includes it own cross validation parameter that you can set true to not only hyper tune the parameters but also allows to make sure that the hyperparameters are tuned across multiple iterations of data set, allowing for the elimination of unexpected outliers in the data set, affecting the accuracy of the model. The SciKit-Learn version of GridSearchCV was used on the SciKit-Learn decision tree classifier. The parameters set for the GridSearchCV are checking the different criterions (the different methods to use for establishing the splitting of the branches), the different maximum depths (the global maximum numbers of branches), and the different minimum numbers of instances (how many instances of unique classes remain at a certain node after a split). The criterions used were gini, entropy, and log\_loss. The max\_depth parameter had a range of values from 1 t0 20, as did the min\_samples\_split parameter. The best parameters found were criterion: entropy, max\_depth: 4, and min\_samples\_split: 2. The resulting accuracy score was 0.8965. For the self-developed decision tree algorithm, we implemented our own version of GridSearchCV where we plugged in a range of values to check for each parameter; it should be noted that, in addition to the same parameters used in GridSearchCV, we also included the different target impurities (the threshold scores resulting from a branch split indicative of uniformity of classes) as a parameter. The final results were entropy as the best criterion with a best\_depth value of 6, a best min\_instance value of 2, and a best target\_impurtiy value of 0.0. This produced an accuracy score of 0.8850.

For the perceptron model, the only hyperparameter tuning involved selecting the features that we passed into the perceptron model for both our self-developed perceptron algorithm and SciKit-Learn’s version of the perceptron model. We found the best hyperparameters to be the features ProductRelated, ProductRelated\_Duration, ExitRates, and PageValues. The score of our self-developed perceptron model was 0.7262 and after hyperparameter tuning and cross validation check, the score decreased to 0.4918. The scores for the SciKit-Learn model before feature selection tuning was 0.8726 and after tuning, it increased to 0.8788.

**Perceptron Model**

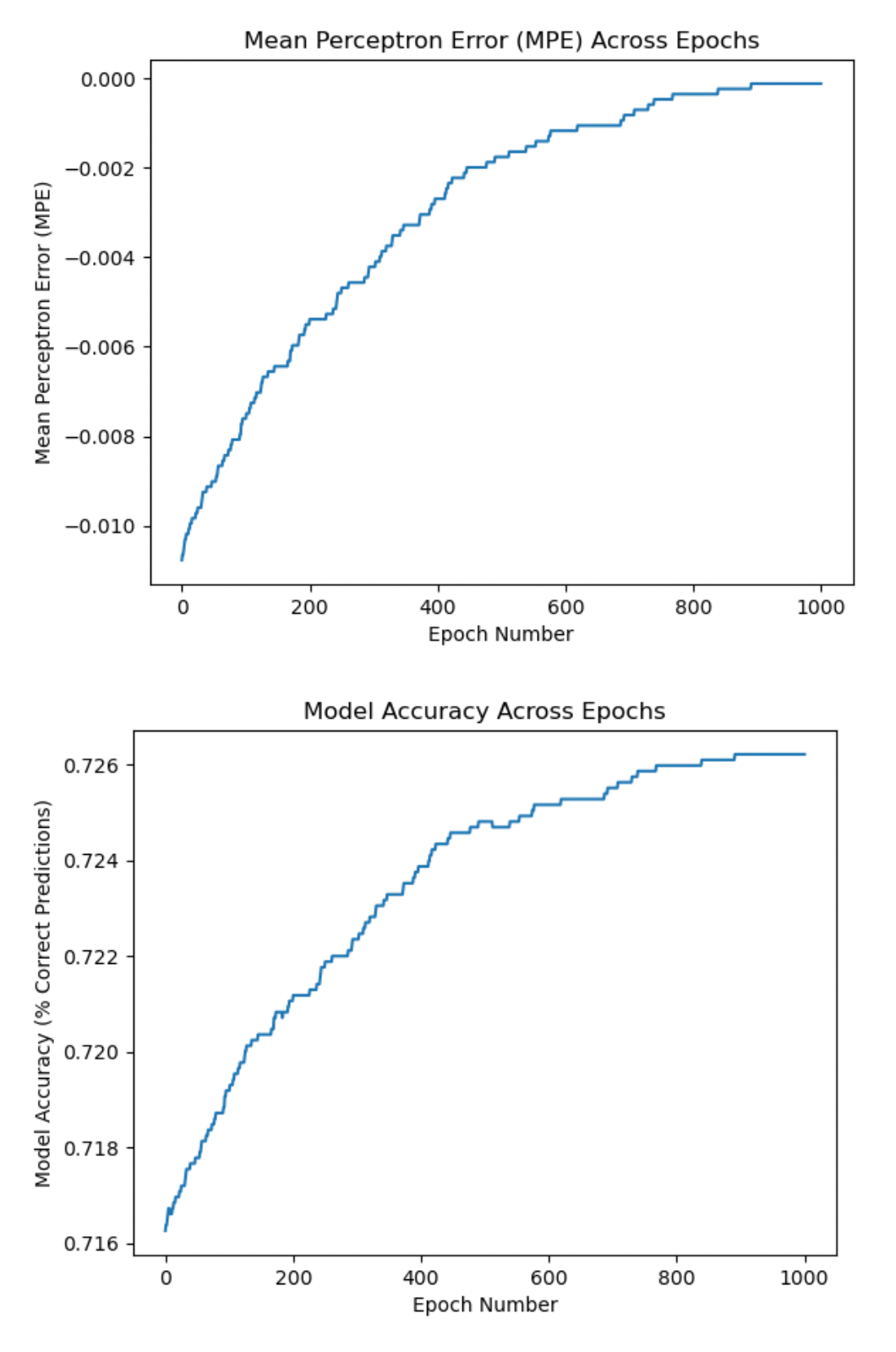
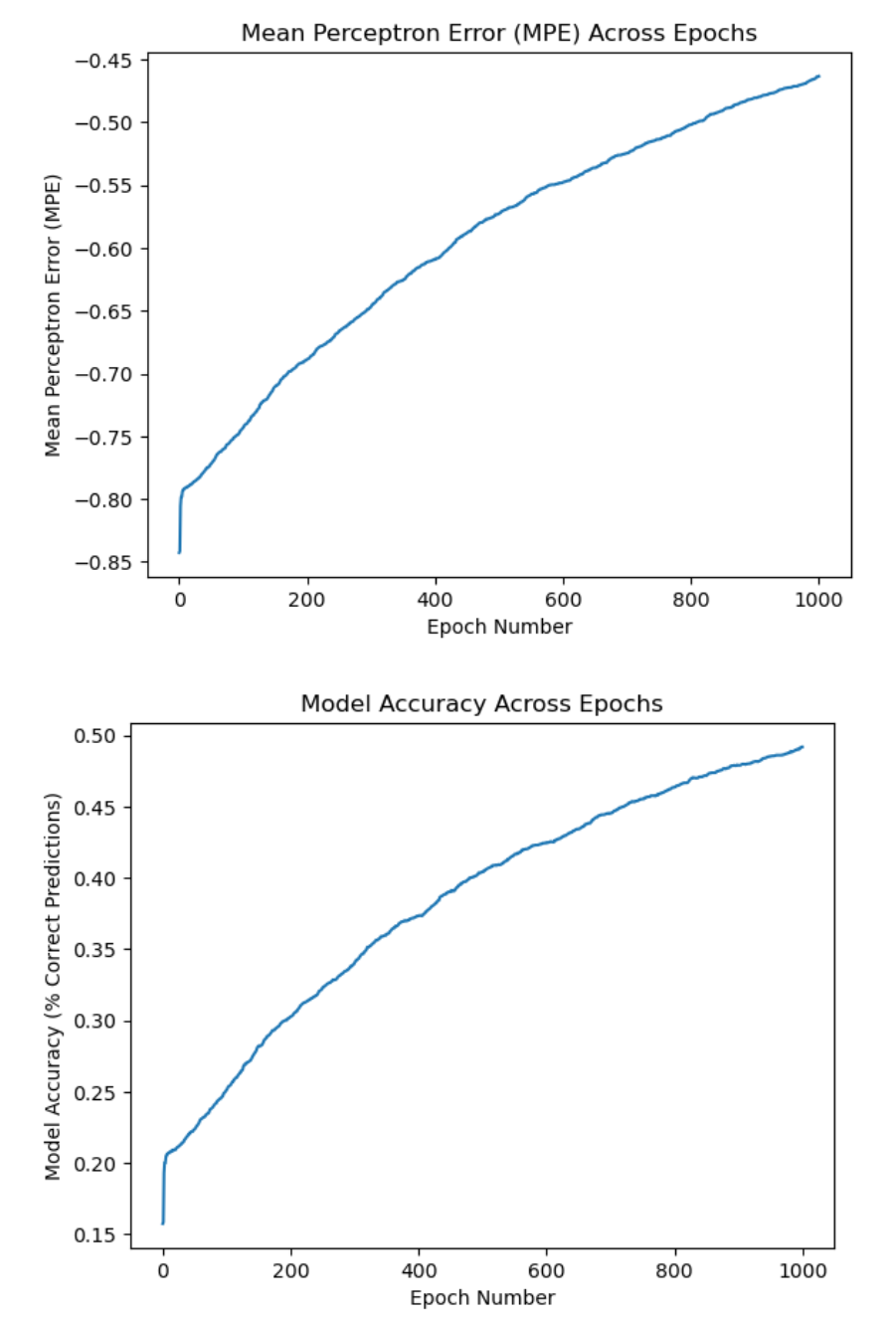
Two perceptron models were used for our models for the classification analysis and for comparison purposes: our own perceptron model and SciKit-Learn’s perceptron model. For our self-developed perceptron algorithm, we first ran it against all features in the data set with an alpha value of 0.0001 and epochs of 1000. Looking at the results of the model in terms of the five metrics score, the accuracy was 0.7262, the sensitivity was 0.125, the specificity was 0.8377, the precision was 0.1249, and the F1-score was 0.1250. After tuning the perceptron model via feature selection methods, the perceptron algorithm was ran again with only the selected features of Product\_Related, ProductRelated\_Duration, ExitRates, and PageValues, with alpha the same at 0.0001 and the number of epochs constant at 1000. Some scores became worse, while some actually improved. The accuracy went from 0.7262 to 0.4918. This decrease in accuracy was surprising, but it can be explained because when you decrease the number of features, the algorithm might not have enough features to learn and improve itself when making predictions, thus decreasing the number of correct predictions out of the total number of predictions made. The sensitivity score improved from 0.125 to 0.8566. This is a huge improvement in the sensitivity score, which means that with the selected features passed into the algorithm the true positive rate improves the model at predicting customers who are actually buying and not mislabeling them as not buying, which is desired. The specificity rate, however, decreased from 0.8377 to 0.4243; this decrease in specificity rate is undesired but is not very influential on our objective, since mislabeling a customer as not buying is not increasing additional costs to the company and is actually adding extra revenue to the business. The precision score increased from 0.1249 to 0.2160, which is good since our objective is to help businesses identify customers that are not buying and then target their marketing strategies towards them to help them expand their customer base. The F1-score increased from 0.1250 to 0.3449, but this score is the least of our concerns for this project since our data includes biases (there are an overwhelming amount of “False” instances in the Revenue column), so this score is not a great indicator of accuracy of the model for this data set.

* *

*Fig 5: Perceptron- All Features Fig 6: Perceptron- Selected Features*

*Scatterplot of the Self-Developed Perceptron Models’ Specificity vs. Sensitivity, color coded to represent number of features with all features available (Fig 5) and with the top four features available (Fig 6)*

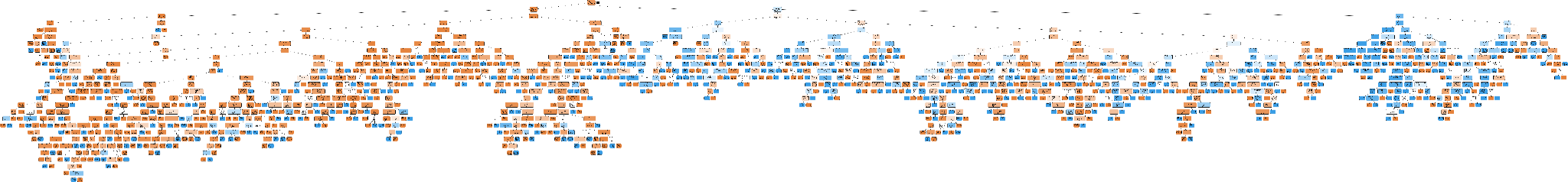
With regards to SciKit-Learn’s model, we implemented two different instances of the model: one with all of the features of the dataset, and one with the features selected by both the tree-based feature selection method and the pipeline method. For the first instance, our results included an accuracy of roughly 0.873, a sensitivity of roughly 0.236, a specificity of roughly 0.991, a precision of roughly 0.825, and a f1-score of roughly 0.367. For the second instance, our results included an accuracy of roughly 0.879, a sensitivity of roughly 0.309, a specificity of roughly 0.984, a precision of roughly 0.783, and a f1-score of roughly 0.444. On the positive side, with certain features discarded from the model, our accuracy, sensitivity, and f1-score increased; on the negative side, our specificity and precision decreased. The negatives outweigh the positives because from the perspective of a company trying to sell its products, false positives are more costly, meaning precision should be maximized, but in this case it is not.

* *

*Fig 4: Mean Perceptron Error and Model Accuracy across Epochs line plots for Self-Developed Perceptron Algorithm with all features (left) and Self-Developed Perceptron Algorithm with only the selected features (right)*

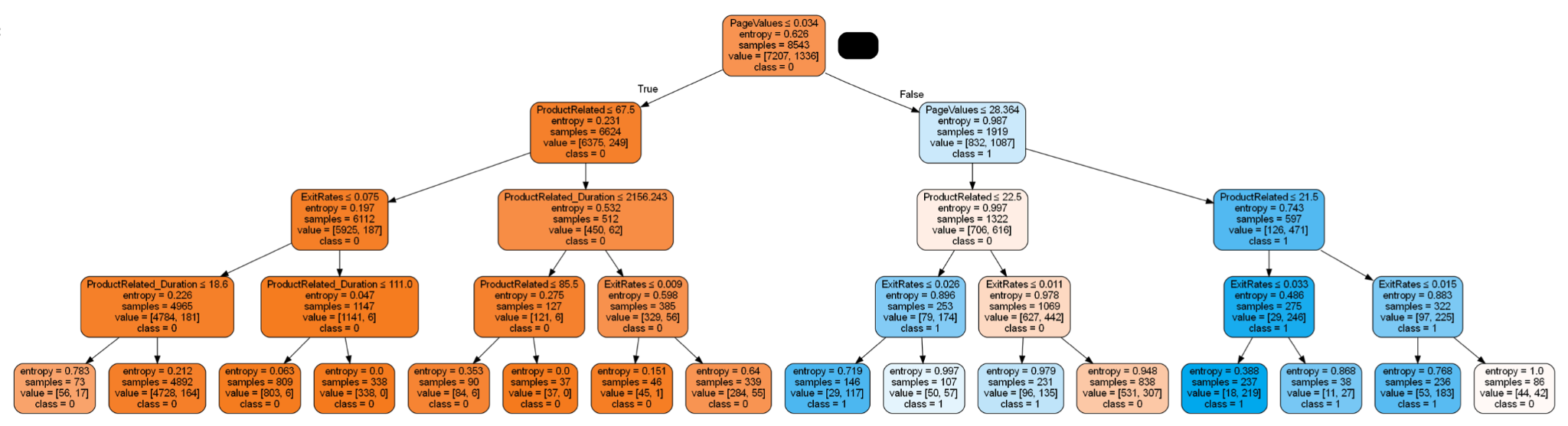
**Decision Tree Algorithm**

To further our analysis of the data, we implemented two decision tree algorithms: a self-developed model (SDM) and the SciKit-Learn model (SKL). Our SDM was first run with all of the features with initial parameters of a max\_depth of 4, a target impurity of 0.0 and a minimum number of unique class instances left in the branches of 2. This resulted in an accuracy of 0.8888, sensitivity of 0.4603, specificity of 0.9682, precision of 0.7287, and f1-score of 0.5642. After reducing the features to the top four and hyperparameter tuning to find the ideal max\_depth, min\_instance, and impurity values, we implemented our algorithm again. This change reduced the accuracy to 0.8850, specificity to 0.3986, and f1-score to 0.5200. However, the feature reduction did increase specificity score to 0.9751 and precision to 0.7475. This may have been due to the limited number of features restricting the splits that may have increased our metrics. Though the specificity and precision increased from all of the features to the selected features, the better model was the SDM that included all of the features, as the weighted scores were overall greater than the SDM with selected features.



*Fig 7: SciKit-Learn Decision Tree- All Features*

We then implemented the SKL with all of the features (Fig 7). After, we ran the SKL with selected features and the optimal hyperparameters, using entropy as our criterion, a max\_depth of 4, and a min\_samples\_split of 2.

**

*Fig 8: SciKit-Learn Decision Tree- Selected Features*

From SKL all features to SKL specific features (Fig 8), the accuracy increased from 0.8553 to 0.8837, sensitivity decreased from 0.5612 to 0.4965, specificity increased from 0.9097 to 0.9553, precision increased from 0.535 to 0.6730, and f1-score increased from 0.5478 to 0.5714. Thus, the SKL with specific features and optimal hyperparameters was the more effective model overall, with the increases outweighing the decline in sensitivity.

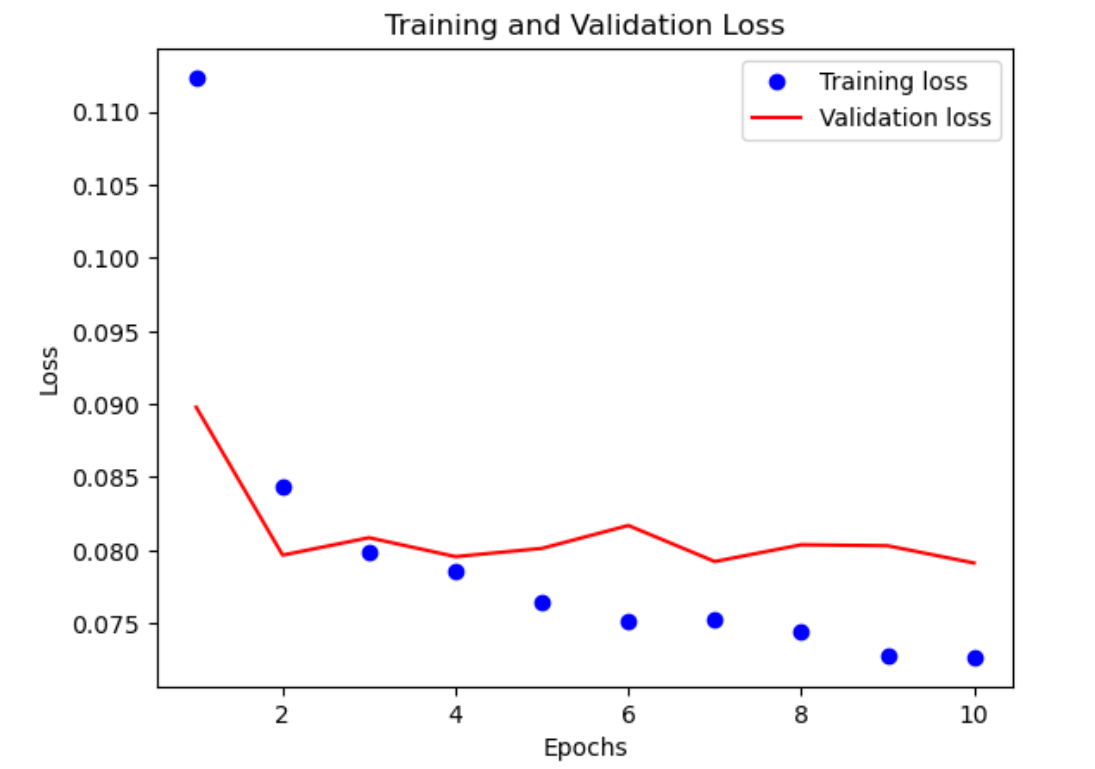
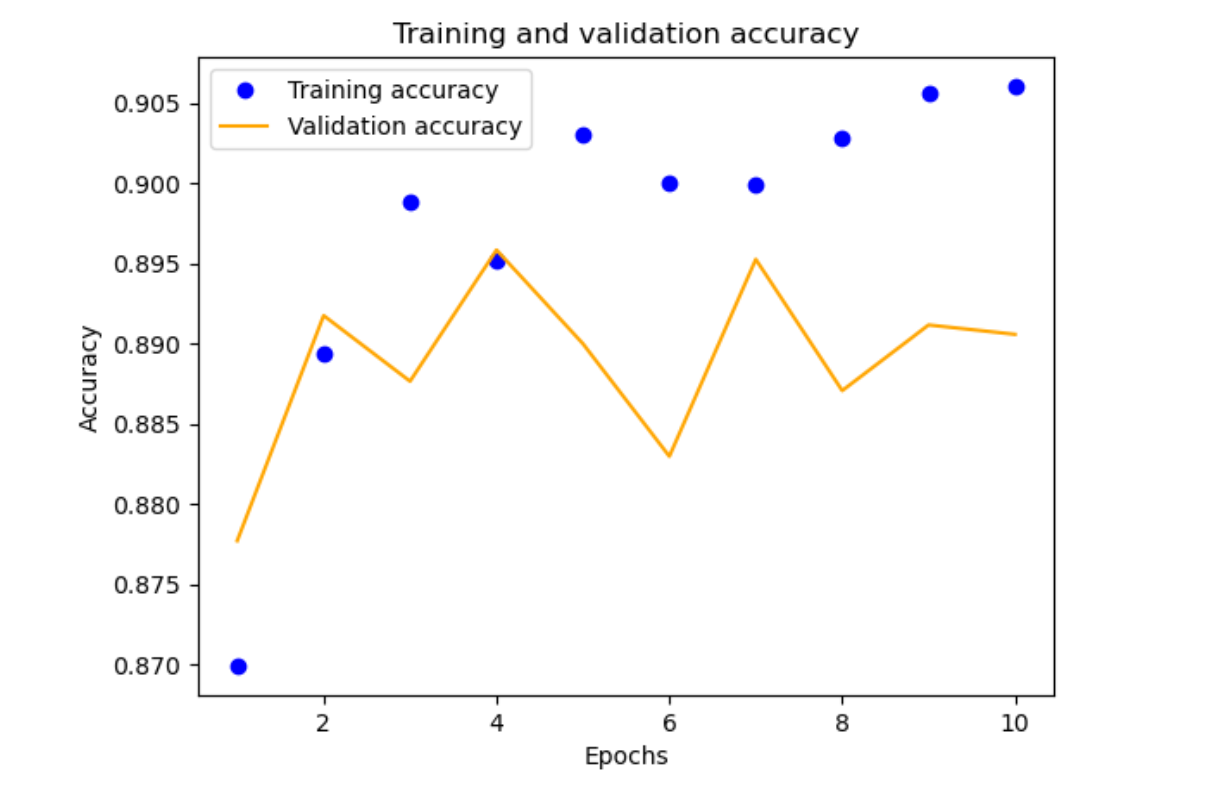
**Neural Networks**

In terms of neural networks, we implemented the tensorflow library for creating a neural network for classification. Initially, the data is split into a training set and a test set. The distribution of data in the two different sets are shown in Figure 9, with approximately 2/3 of our data as the training set and around 1/3 of the data as the test set. The distribution of true values between the test set and the training set is not ideal as if the model predicts everything as false, it can also still reach a high level of accuracy since a large portion of the test set contains false values.

**

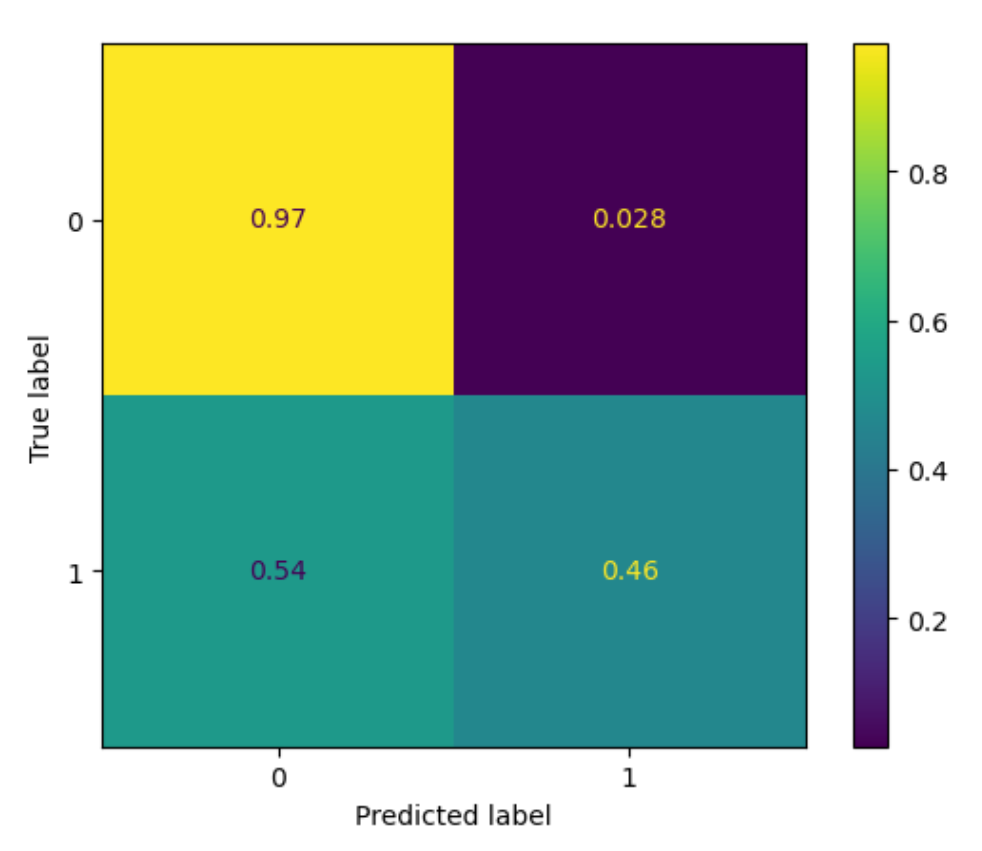
*Fig 9: Stacked bar plot of the number of samples in the testing versus training set*

After splitting the data set into training and test sets, the values in the data were normalized using the mean and standard deviation values. The model that we used for the neural network classification is “Sequential,” adding dense and dropout values with “relu” as the activation criteria. From the model summary, it was discovered that all 10,625 parameters passed into the model were trainable parameters, increasing the predictive power of the neural network classification model. The neural network model was compiled with Adam as the optimizer, loss measured in mse and returned metric values of mae and accuracy. The model is then trained with the training X and y values with 10 iterations, with a batch size of 2, a verbose of 0, and a validation split of 0.2. The model is then evaluated using mean squared error (MSE), mean absolute error (MAE), root mean squared error (RMSE), and the total mean squared absolute error (TMAE). The error values are pretty small, with a MSE value of 0.08, MAE value of 0.152, RMSE value of 0.421, and a TMAE value of 0.249. This shows that while the error for each individual iteration was small, the total error across all iterations is pretty considerable.

**

*Fig 10: Training and Validation accuracy plot for the Neural Network Fig 11: Training and Validation Loss plot for Neural Network*

Seen in Figure 10, the training and validation accuracy for the neural network, while the points fluctuated from epoch to epoch, both saw an overall increase. Similarly, in Figure 11, both the training and validation loss decreased overall, though the loss from epoch to epoch varied less so than the accuracy. The overall trend for this model was that the validation data performed worse than the training data in both cases, with lower accuracy and higher loss. This may have been due to the skewed nature of our data, with such a great quantity of “False” values. While the data likely would have benefitted from a greater number of epochs run, the neural network increased its accuracy and decreased its loss over time.

**

*Fig 12: Visualized Confusion Matrix for the Neural Network*

The confusion matrix as shown above in Figure 12 shows the number of true negative, true positive, false negative, and false positive scores of the neural networks model on the testing set after training the model on the training set. The true negative score was 0.97, the false positive score was 0.028, the false negative score was 0.54, and the true positive score was 0.46. This indicates that our neural network was exceptional at detecting negative values, but less so at detecting positives. Because our data was so greatly skewed to the “False” values, this outcome is understandable.

**Best Model Selection**

The five metrics that we produced for each of our models are accuracy, specificity, sensitivity, precision, and f1-score. The accuracy score represents the number of correct predictions of a model in comparison to the total number of predictions made. The specificity score represents the true negative rate, while the sensitivity score represents the true positive rate. The precision score measures the ability of the model to not label a negative sample as positive, with 1 being the best and 0 being the worst. The F1-score measures the model’s accuracy on a dataset, combining the precision and recall values of the model.

Looking at our dataset and the objectives of our project, we assigned weights to the five metric scores for each model and came up with a weighted total score for each model as the determinant for the best model [Appendix 1.1]. The weights were selected based on an order of importance for each of the five metric scores, based on our discretion of the score importance for our data set and the objectives of our project. Based on our objective of determining whether customers are going to buy or not buy, we ranked precision and specificity as equally important with a weight of 0.30, we ranked accuracy at a weight of 0.20, sensitivity as 0.15, and f1-score at 0.05. The precision and specificity scores are at 0.30 weight because we want our model to predict customers that are buying as buying, and customers who are not buying as not buying. This would help the business identify which segments of customers are already in their customer base, so businesses can focus on expanding their markets into customers that are not already buying from them and then identifying marketing techniques that will attract those customers not buying, to incentivize them to buy. The accuracy is the number of accurate predictions in comparison with the total number of predictions made, and since our objective is to correctly identify customers who are not being from the business, our model should focus on accurately identifying customers that are not buying. The weight of the sensitivity score is 0.15 because the sensitivity score is the true positive rate and that is not as important since falsely identifying customers as positive is not as harmful to our business, advertising to additional customers that are have already bought from the business is not as harmful as not identifying customers that are not already our customers, since businesses always want to attract more customers and expand their customer base.

The top four models that we found were the SKLearn Perceptron with all features, SKLearn Perceptron with selected features, self-developed Decision Tree with all features, and self-developed Decision Tree with selected features [Appendix 1]. These models were fairly similar in regards to accuracy and specificity metrics, which was our primary focus. After applying the weights to each of the scores, there are actually top five models that have the four highest total weighted scores [Appendix 1.1] and these models are the SKLearn Perceptron Model with selected features, the self-developed Decision Tree model with all features, the self-developed Decision Tree model with selected features, the SKLearn Decision Tree model with selected features, and the Neural Networks model. Since the self-developed Decision Tree model with all features appears in both tables [Appendix 1, 1.1], it is the best model since it has top scores across all five metrics weighted equally and also the top scores for the total weighted score after applying our weight distributions.

**Conclusion**

Ultimately, we found the self-developed decision tree algorithm with all features to be the best model hypertuned with parameters entropy for criterion; 4 for maximum depth; 2 for minimum instances; and 0.0 for target impurity score. While this model is fairly reliable, it is also important to note that the dataset had a large number of “False” values for the Revenue, indicating that the samples in this data set were skewed.

Within the broader context of business, our results indicate that while there is still further tuning that can be done, machine learning has potential for guiding decisions in the future, as we are sure it does currently. A small Etsy seller might be able to use our decision tree to make ends meet. A larger corporation could determine who to send coupons to for that week, potentially allowing a small child to buy a toy for their sibling. Businesses can implement our solution to make more effective decisions for themselves and their customer base. There is also potential for this project to be used in more immoral ways, such as taking advantage of vulnerable people, and it is important to be aware of this to combat it.

If we were to complete this project again, we would implement hyperparameter tuning with the perceptron model such that we would experiment with different values for the alpha and epochs parameters, we would implement more neural networks algorithms to compare with each other, we would take a sample from the original dataset representative of more equal proportions of “True” and “False” to alleviate some of the skewness of the data, and we would determine a fair way to assign weights to the five metric scores to take into account the varying importances of the particular metrics. Additionally, we would try another method of classification– clustering the customers into specific groups based on similar behaviors.

**Author Contributions**

**Luke Abbatessa**

* Wrote sections of the final report
* Debugged errors in the decision tree algorithm
* Wrote the tree based feature selection and pipeline feature selection methods
* Created the decision tree algorithm using sk learn
* Implemented GridSearch CV for our hyperparameter tuning
* Created functions for cleaning up our data file
* Performed Exploratory Data Analysis on the data file
* Debugged errors in decision tree algorithm
* Helped implement both perceptron models with all features
* Implemented 10 cross fold validation
* Implemented train\_test\_split

**Jocelyn Ju**

* Organized and added sections to the poster
* Worked with Yuting to create the neural networks model
* Helped implement the selected feature models and perceptron models
* Wrote sections of the final report
* Contextualized models in the broader scope of the project
* Created correlation matrix for neural networks
* Created the decision tree diagram for SciKit-Learn decision tree algorithm
* Added decorations to the poster
* Formatted final paper and resources
* Created SciKit-Learn decision tree algorithm

**Yuting Zheng**

* Created sections of the poster
* Wrote the Abstract, Background, Neural Networks, Perceptron Model, Best Model Selection sections of the report
* Utilized decision tree algorithm from HW4 and corrected sections of the model to make it fit for our data set for this project
* Commented sections of the code
* Debugged errors in our self-developed machine learning algorithms
* Worked with Jocelyn to create the neural networks model
* Came up with ideas for feature selections
* Organized final model reports into tables in the report
* Created helper functions for our code and cleaned the code up by creating functions for sections of code that were repeatedly used
* Created a test data set containing only the selected features found from feature selection
* Increased efficiency for hyperparameter tuning for self-developed decision tree algorithm
* Added classification reports for final testing results of the machine learning algorithms
* Helped to create the perceptron model for selected features

**References**

“Ecommerce Machine Learning: Business Benefits + Use Cases.” *BigCommerce*,

https://www.bigcommerce.com/articles/ecommerce/machine-learning/.

Galarnyk, Michael. “Understanding Train Test Split.” *Built In*, 28 July 2022,

https://builtin.com/data-science/train-test-split.

Grant, Peter. “An Introduction to Bias-Variance Tradeoff.” *Built In*, 2 Dec. 2021,

https://builtin.com/data-science/\bias-variance-tradeoff.

Hashmi, Farukh. “How to Find Best Hyperparameters Using GridSearchCV in Python.”

*Thinking Neuron*, Thinking Neuron, https://thinkingneuron.com/how-to-find-best- hyperparameters-using-gridsearchcv-in-python/

Jain, Vipin, et al. “An Overview of Electronic Commerce (e-Commerce).” Journal of

Contemporary Issues in Business and Government, vol. 27, no. 3, 2021, https://doi.org/10.47750/cibg.2021.27.03.090.

Kumar, Manoj, et al. “Model-Based and Sequential Feature Selection.” Scikit-Learn,

https://scikit-learn.org/stable/auto\_examples/feature\_selection/plot\_select\_from\_model\_diabetes.html#sphx-glr-auto-examples-feature-selection-plot-select-from-model-diabetes-py.

Sakar, C., and Yomi Kastro. “Online Shoppers Purchasing Intention Dataset.” UC Irvine

Machine Learning Repository, UC Irvine, 30 Aug. 2018, https://archive-beta.ics.uci.edu/dataset/468/online+shoppers+purchasing+intention+dataset.

Zeglam. “Zeglam/Online-Shoppers-Intention-Prediction: Predict the Intention (Make a

Purchase or Not) of e-Commerce Website Visitors.” GitHub, https://github.com/zeglam/Online- shoppers-intention-prediction.

**Appendix**

**FINAL RESULTS**

| Model | Accuracy | Sensitivity | Specificity | Precision | F1-Score |
| --- | --- | --- | --- | --- | --- |
| Self-dev Perceptron (all features) | 0.7262 | 0.125 | 0.8377 | 0.1249 | 0.1250 |
| **SKLearn Perceptron (all features)** | **0.8726** | **0.2358** | **0.9907** | **0.8246** | **0.3667** |
| Self-dev Perceptron (selected features) | 0.4918 | 0.8566 | 0.4243 | 0.2160 | 0.3449 |
| **SKLearn Perceptron (selected features)** | **0.8788** | **0.3094** | **0.9841** | **0.7832** | **0.4436** |
| **Self-dev DTree (all features)** | **0.8888** | **0.4603** | **0.9682** | **0.7287** | **0.5642** |
| SKLearn DTree (all features) | 0.8553 | 0.5612 | 0.9097 | 0.535 | 0.5478 |
| **Self-dev DTree (selected features)** | **0.8850** | **0.3986** | **0.9751** | **0.7475** | **0.5200** |
| SKLearn DTree (selected features) | 0.8837 | 0.4965 | 0.9553 | 0.6730 | 0.5714 |
| Neural Networks | 0.8837 | 0.4965 | 0.9553 | 0.6730 | 0.5714 |

*Appendix 1: Table of summary of the five metric scores for each model*

| Model | Accuracy | Sensitivity | Specificity | Precision | F1-Score | Total Score |
| --- | --- | --- | --- | --- | --- | --- |
| Self-dev Perceptron (all features) | 0.2179 | 0.0188 | 0.1675 | 0.0375 | 0.0063 | 0.4479 |
| SKLearn Perceptron (all features) | 0.2618 | 0.0354 | 0.1981 | 0.2474 | 0.0183 | 0.7610 |
| Self-dev Perceptron (selected features) | 0.1475 | 0.1285 | 0.0849 | 0.0648 | 0.0172 | 0.4429 |
| **SKLearn Perceptron (selected features)** | **0.2636** | **0.0464** | **0.1968** | **0.2350** | **0.0222** | **0.7640** |
| **Self-dev DTree (all features)** | **0.2666** | **0.0690** | **0.1936** | **0.2186** | **0.0282** | **0.7761** |
| SKLearn DTree (all features) | 0.2566 | 0.0842 | 0.1819 | 0.1605 | 0.0274 | 0.7106 |
| **Self-dev DTree (selected features)** | **0.2655** | **0.0598** | **0.1950** | **0.2243** | **0.0260** | **0.7706** |
| **SKLearn DTree (selected features)** | **0.2651** | **0.0745** | **0.1911** | **0.2019** | **0.0286** | **0.7611** |
| **Neural Networks** | **0.2651** | **0.0745** | **0.1911** | **0.2019** | **0.0286** | **0.7611** |

*Appendix 1.1: Table of summary of the five metric scores for each model multiplied by their corresponding weights (Accuracy-0.20, Sensitivity-0.3, Specificity-0.15, Precision-0.3, F1-Score-0.05)*