Less for More: Retail Returns Classification

with Logistic Regression, Support Vector Machine, RandomForest, XGBoost and Deep Neural Network

Group 3
Cheyenne | Deviyani | Min | Peter
Ryan | Kelly | Xavier

PRESENTATION FLOW

01
Recap
Problem Statement
Motivation
Literature Review

02
Data
Dataset
Data Preprocessing

03
Models
Random Forest, XGBoost, SVM, DNN, Logistic Regression

04
Results
Individual Model Results
Models Comparison

05
Prospects & Challenges
Limitation & Challenges
Future Work

06
Conclusion

01 RECAP

PROBLEM STATEMENT

"For an average company, the researchers estimate that a five percent improvement in the rate of returns has the potential to deliver real improvements" - ERC Retail Loss

Profit Losses

>$600B of losses, projected to increase year on year

Management of returns logistically intensive

Product Wastage

Returned products cause up to 5B tons of waste / year

Cannibalize profit margins

Customer Satisfaction

Customers desire seamless retail experience

Need to optimize return process and customer satisfaction

PROJECT MOTIVATION

Future-proofing for increased returns as businesses move online

Improving cost efficiency

Enhancing customer experiences for customer retention

- Craft better returns policy
- Targeted campaigns
- Identify pain points

>>>>>>>>>>

LITERATURE REVIEW 1 - DATA PREPROCESSING TECHNIQUES

Using Chi-Squared Contingency test to find highly correlated variables

Discover variables with high correlation within specified level of significance

scipy library chi2_contingency function

Chi-Square Test of Independence
(chi_2_contingency**):**

- Used for **categorical variables**
- Tests whether two categorical variables are independent of each other.
- The output includes a chi-square statistic and a p-value

(Melanie, 2024)

Encoding high dimension categorical attributes

Target encoding to replace each category with the mean (or some other aggregation) of the target variable for that category

(Udilâ, et al., 2023)

LITERATURE REVIEW 2- MODEL TUNING

GridSearchCV

- Exhaustively searches through a manually specified subset of the hyperparameter space
- Defined by a grid of hyperparameters, where every combination is evaluated
- Can be **computationally expensive**, especially with a large number of hyperparameters

RandomSearchCV

- Searches the space of hyperparameters randomly
- Number of parameter settings that are tried is a fixed budget set by the user
- **More efficient than GridSearchCV**
  (Bergstra & Bengio, 2012)

Implementing GridSearchCV and RandomSearchCV in Python (Kdnuggets, 2022)

- Hyperparameter Tuning Using Grid Search and Random Search in Python
- Search for Global optima
- Using sklearn.model_selection library
- GridSearchCV and RandomSearchCV

02 DATA

›››››››››››

DATASET

```
(100000, 14)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 14 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   order_item_id  100000 non-null  int64
 1   order_date     100000 non-null  object
 2   delivery_date  90682 non-null   object
 3   item_id        100000 non-null  int64
 4   item_size      100000 non-null  object
 5   item_color     100000 non-null  object
 6   brand_id       100000 non-null  int64
 7   item_price     100000 non-null  float64
 8   user_id        100000 non-null  int64
 9   user_title     100000 non-null  object
 10  user_dob       91275 non-null   object
 11  user_state     100000 non-null  object
 12  user_reg_date  100000 non-null  object
 13  return         100000 non-null  int64
```

- Open sourced Kaggle dataset (Kaggle BADS1920, 2024)
- Real orders by customers of a clothing store describing orders and customers
- **14 columns * 100,000 rows**

DATA PRE-PROCESSING 1

Size = 40 → Drop 1 row
Size = unsized → untouched

E.g., Gender, Age, Age groups, Delivery days, Average monthly purchases per user per order, most/least returned sizes

Missing Values & Outliers handling » Drop Values » Standardization » Features Extraction » Train-Test Split

Missing values in only delivery date & user date of birth.

Delivery days < 0 → create attribute no_delivery to represent

Standardize to string sizes.
Integers+ → Next biggest size.
Integers → Map to string sizes based on predefined limits.

80/20 Train test split

»»»»»»»»»»

DATA PRE-PROCESSING 2

Chi-squared test with p value= 0.01.
Order item id, if order during special occasion, least returned size not correlated with returns

User state, Item size

Transformation and Standardization »» Correlation Analysis »» Feature Selection »» One Hot Encoding »» Target Encoding

Log transformation for skewed variables

Standardize log transformed variables

Drop insignificant and engineered variables

Replacing high dimension features (user return rate, brand return rate) with average target value of all data points belonging to the category

>>>>>>>>>>

DATA PRE-PROCESSING OUTCOME

Original Dataset

```
(100000, 14)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 14 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   order_item_id   100000 non-null  int64
 1   order_date      100000 non-null  object
 2   delivery_date   90682 non-null   object
 3   item_id         100000 non-null  int64
 4   item_size       100000 non-null  object
 5   item_color      100000 non-null  object
 6   brand_id        100000 non-null  int64
 7   item_price      100000 non-null  float64
 8   user_id         100000 non-null  int64
 9   user_title      100000 non-null  object
 10  user_dob        91275 non-null   object
 11  user_state      100000 non-null  object
 12  user_reg_date   100000 non-null  object
 13  return          100000 non-null  int64
```

- Open sourced Kaggle dataset (Kaggle BADS1920, 2024)
- Real orders by customers of a clothing store describing orders and customers
- **14 columns * 100,000 rows**

Pre-processing

Pre-processed Dataset

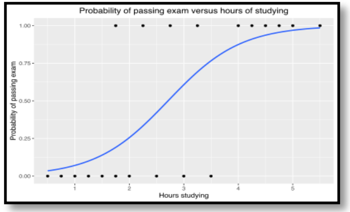
```
(99999, 40)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99999 entries, 0 to 99998
Data columns (total 40 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   return                          99999 non-null  int64
 1   no_delivery                     99999 non-null  int64
 2   is_female                       99999 non-null  int64
 3   is_male                         99999 non-null  int64
 4   is_bday                         99999 non-null  int64
 5   most_returned_item              99999 non-null  int64
 6   least_returned_item             99999 non-null  int64
 7   most_returned_color             99999 non-null  int64
 8   least_returned_color            99999 non-null  int64
 9   log_item_price                  99999 non-null  float64
 10  log_no_items                    99999 non-null  float64
 11  log_age                         99999 non-null  float64
 12  log_avg_freq_purchases          99999 non-null  float64
 13  user_state_Baden-Wuerttemberg   99999 non-null  int64
 14  user_state_Bavaria              99999 non-null  int64
 15  user_state_Berlin               99999 non-null  int64
 16  user_state_Brandenburg          99999 non-null  int64
 17  user_state_Bremen               99999 non-null  int64
 18  user_state_Hamburg              99999 non-null  int64
...
 38  user_return_rate                99999 non-null  float64
 39  brand_return_rate               99999 non-null  float64
```

- 1 row dropped
- Feature Engineered: 12 columns
- Target Encoded: 2 columns
- One-Hot Encoded: 25 columns
- Target Variable: 1 column
- **40 columns * 99,999 rows**


Dataset
Decision Tree-1
Decision Tree-2
Decision Tree-N
Result-1
Result-2
Result-N
Majority Voting / Averaging
Final Result

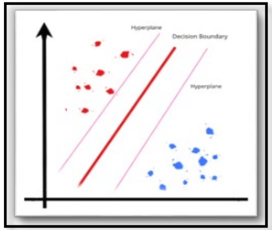
03 MODELS

MODELS



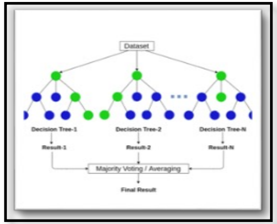
Logistic Regression

- Simple and interpretable model
- Relatively fast and good performance



SVM

- Effective for high-dimensional data
- Robust to overfitting
- Good generalization performance

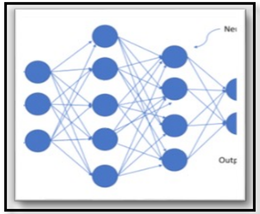


Random Forest

- Improvement on decision tree
- Robust to overfitting and noise
- Interpretable

XGBoost

- Capable of handling large datasets with high-dimensional features
- Effective in capturing complex relationships
- Adopts a "slow-learning" approach



Deep Neural Network

- Capable of learning complex patterns
- Effective for large datasets with high-dimensional features
- Can classify patterns that were not trained on

EVALUATION CRITERIA

Main Aim : Find the best performing model based on the **Test ROC – AUC Score**

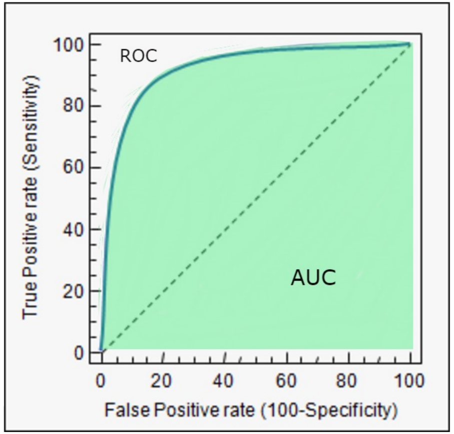


Figure 3.1: Example of ROC-AUC curve

- ROC - AUC score tells us how well a model can differentiate between positive or negative instances **across all classification thresholds**
- Range of ROC – AUC score : 0 to 1 (where 0.5 indicates random guessing and 1 indicates perfect performance)
- We will be evaluating using the **test** ROC – AUC score as we want to see how well the model can predict based on unseen data

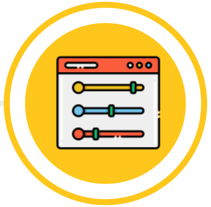
››››››››››

OUR STEPS FOR EACH MODEL

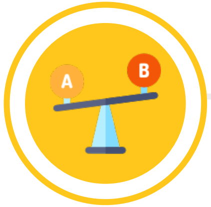
Construct a Baseline Model

- Use default/predefined parameters
- Obtain the test ROC-AUC score

Hyperparameter Tuning



- Custom grid search function to tune each model, with 5-fold cross-validation
- Find the optimal values of parameters that maximise train ROC-AUC score during cross validation
- Fit the optimal parameters into model and obtain the test ROC-AUC score



Comparison of all models

- Compare the test ROC - AUC scores of each model under the particular model category
- Model with the highest ROC – AUC score will be the best model for that particular category
- We will finally compare all the best models from different model categories to select the ultimate best model

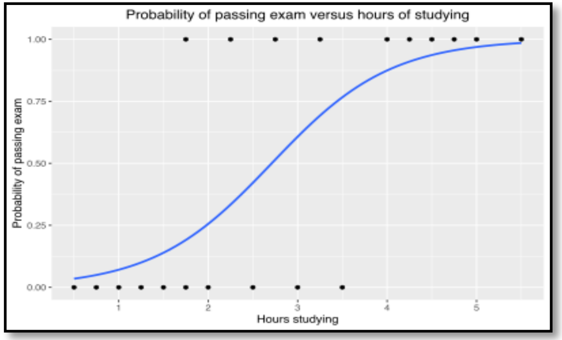
Note: It is possible for the baseline model to outperform the tuned models

>>>>>>>>>>

04 RESULTS

LOGISTIC REGRESSION

Model



Baseline Model Test ROC-AUC Score:

0.757

Library

sklearn.linear_model.LogisticRegression

Parameters

- Default Parameters were used
- As per the LogisticRegression() function within the LogisticRegression module

LOGISTIC REGRESSION

Regularization Techniques

LASSO	Ridge	Elastic Net
• Penalizes absolute value of coefficients • Shrink coefficients of less important predictors to exactly zero (model simplification and variable selection) • In case of correlated predictors, LASSO selects one predictor and drives the coefficients of others to zero	• Penalizes squared value of coefficients • Shrinks coefficients of less predictors proportionally (close to zero) • Reduces impact of multicollinearity by reducing the influence of correlated predictors	• Combines LASSO penalty and Ridge penalty • Selects one variable from group of correlated variables and assign it non-zero coefficient while shrinking coefficients of other correlated variables towards zero

Table 4.1: Summary of regularization techniques

LOGISTIC REGRESSION

Hyperparameter Tuning

- Tuned hyperparameter 'C' with range [0.001, 0.01, 0.1, 1, 10, 100, 100]
- Tuned hyperparameter 'l1_ratio' for Elastic Net with range [0.1, 0.3, 0.5, 0.7, 0.9]

Logistic Regression (Baseline)
No hyperparameter tuning
Test ROC-AUC score: 0.7566

Logistic Regression + LASSO	Logistic Regression + Ridge	Logistic Regression + Elastic Net
Hyperparameter(s) to tune and their ranges: 'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]	Hyperparameter(s) to tune and their ranges: 'C': [0.001, 0.01, 0.1, 1, 10, 100, 100]'	Hyperparameter(s) to tune and their ranges: 'C': [0.001, 0.01, 0.1, 1, 10, 100], 'l1_ratio': [0.1, 0.3, 0.5, 0.7, 0.9]
Optimal value of hyperparameter(s): {'C':100}	Optimal value of hyperparameter(s): {'C':10}	Optimal value of hyperparameter(s): {'C':10} {'l1_ratio:0.1}
CV ROC- AUC score: 0.74557	**CV ROC-AUC score: 0.74577**	**CV ROC-AUC score: 0.74557**

Table 4.2: Hyperparameters and CV/Test ROC-AUC scores across different parameters and regularization techniques

LOGISTIC REGRESSION

Interaction Effects

Interaction Terms
log_item_price * is_bday
log_age * log_avg_freq_purchases
log_no_items * log_avg_freq_purchases
log_item_price * brand_return_rate
most_returned_colour * brand_return_rate least_returned_colour * brand_return_rate

Table 4.3 Interaction Terms Explored

LOGISTIC REGRESSION

Including Interaction Terms

Logistic Regression with interaction terms (Baseline)	**Logistic Regression with interaction terms + LASSO**	**Logistic Regression with interaction terms + Ridge**	**Logistic Regression with interaction terms + Elastic Net**
No hyperparameter tuning	Hyperparameter(s) to tune and their ranges: 'C': [0.001, 0.01, 0.1, 1, 10, 100]	Hyperparameter(s) to tune and their ranges: 'C': [0.001, 0.01, 0.1, 1, 10, 100]'	Hyperparameter(s) to tune and their ranges: 'C': [0.001, 0.01, 0.1, 1, 10, 100], 'l1_ratio': [0.1, 0.3, 0.5, 0.7, 0.9]
	Optimal value of hyperparameter(s): {'C':100}	Optimal value of hyperparameter(s): {'C':10}	Optimal value of hyperparameter(s): {'C':10} {'l1_ratio:0.1}
Test ROC-AUC score: 0.7564	**CV ROC- AUC score: 0.7454**	**CV ROC-AUC score: 0.7454**	**CV ROC-AUC score: 0.7454**

Figure 4.4: CV/Test ROC-AUC scores with interaction terms

LOGISTIC REGRESSION

Since CV AUC-ROC scores were similar across the penalty methods (for with and without interaction terms), we opted for **elastic net** due to its ability to perform variable selection and regularization simultaneously.

We further tuned the 'max_iter' for each elastic net model, with range [100,200,300].

Log_Reg + Elastic Net: Optimal 'max_iter' = 200, CV ROC_AUC Score = 0.7456
Log_Reg_InteractionTerm + Elastic Net: Optimal 'max_iter' = 100, CV ROC_AUC Score = 0.7454

Model Comparison

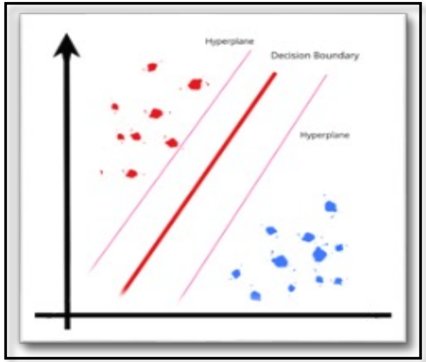
Model	LogReg (Baseline)	LogReg + Elastic Net (Tuned)	LogReg_InteractionTerm (Baseline)	LogReg_InteractionTerm + Elastic Net (Tuned)
Test ROC-AUC Score	0.7566	0.7566	0.7564	0.7564

Figure 4.5: Comparison of models Test ROC-AUC Scores

Note: LogReg (Baseline) had a higher Test ROC-AUC score of 0.7566285 than LogReg + Elastic Net(Tuned) which had a Test ROC-AUC score of 0.7566278 when considering more decimal points

SVM

Model



Baseline Model Test ROC-AUC Score:

0.686

Library

sklearn.svm.LinearSVC

Parameters

- **C :** regularization strength & trade-off between model complexity and training error
- **dual :** primal or dual formulation of linear SVM
- **max_iter :** maximum number of iterations

Range of values considered

- C: [0.1, 1, 10, 100],
- dual: [True, False],
- max_iter: [500, 1000, 2000]

SVM

Baseline Model

Parameters

- **C** = 1
- **dual** = True
- **max_iter** = 1000

Test ROC-AUC Score:

0.686

Hyperparameter tuned Model

Parameters

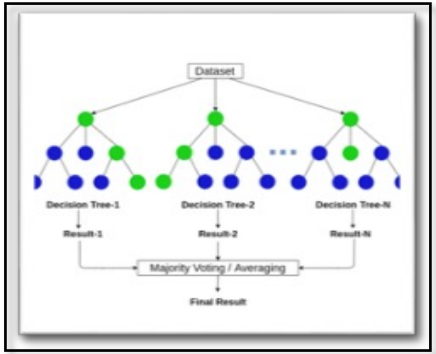
- **C** = 1.3
- **dual** = False
- **max_iter** = 500

Test ROC-AUC Score:

0.686

RANDOM FOREST

Model



Baseline Model Test ROC-AUC Score:

0.756

Library

Sklearn.ensemble.RandomForestClassifier

Parameters

- n_estimators = 100
- max_depth = None
- min_samples_split = 2
- max_features = sqrt(features)

HYPERPARAMETER TUNING FOR RANDOM FOREST

Hyperparameter Tuning

Parameter	Explanation
N_estimators : [50,100,150]	Defines the number of trees. More trees generally lead to better performance but it also increases computational complexity. Therefore, we have selected these estimators which believe to have a well spread.
Max_depth : [None, 10, 20]	A deeper tree can capture more complex relationships in the data, but it also increases the risk of overfitting therefore, we explore a range between none and 20 that allow the model to capture different level of complexity.
Min_samples_split : [2,5]	The default value is 2. This means that if any terminal node has >2 observations, we will further split it into subnodes. Default value of 2 poses the issue that a tree often keeps on splitting until the nodes are completely pure, causing it to grow in size and therefore overfits the data. Therefore, we explore a min split of 5 as well to reduce overfitting.
Max_features : ['sqrt', 'log2']	The maximum number of features allowed to be tried for each decision tree,
Criterion: [‘gini’, ‘entropy’]	The function used to measure the quality of a split in a decision tree. Gini impurity assesses misclassification likelihood based on label distribution, while entropy measures average information required for classification.

Figure 4.6: Explanations of hyperparameters chosen

Optimal Parameter:

{ 'max_depth’: 5, 'max_features’: log2,
'min_samples_split’: 2, 'n_estimators’: 26,
‘criterion’: gini }

Test ROC-AUC score

0.762

XGBOOST

Model

Library

Xgboost

Parameters

- **Number of Boosting Rounds**: The number of rounds required for the model to reach convergence
- **Objective:** The cost function that the algorithm tries to minimize
- **Booster:** The type of model being used as the base learner
- **ETA (Learning Rate)**: The contribution of each tree to the overall XGBoost model
- **Note that the final model uses regression-related parameters, default tree-related parameters are used for the baseline model*

Baseline Model Test ROC-AUC Score:

0.741

XGBOOST

XGBoost utilises grid search from the xgboost package to find the optimal hyperparameters

Number of Boosting Rounds	The number of rounds required for the model to reach convergence
Objective	The type of error that the model will try to minimize.
Booster	The base of the model, can be either tree-based or linear-based. Linear boosters are similar to that of LASSO regression.
ETA (Learning Rate)	The contribution of each boost to the overall XGBoost model

Figure 4.7: Description of hyperparameters for XGBoost

During parameter tuning, we firstly considered whether to use a linear-based or tree-based model – ultimately the linear-based model performed better. The parameters in figure 4.7 are thus the considered parameters for the final model. The parameters for the tree-based model (such as max depth and subsample) are identical to the hyperparameters in a Random Forest model.

XGBOOST

Hyperparameter Tuning

Phase 1:

Parameter
Objective : binary:logistic, reg:squarederror
Booster : gbtree, gblinear

Phase 2:

Parameter
Step 1 **Lambda :** 0, 0.1, 0.2, 0.3, 0.4, 0.5
Step 2 **Lambda :** 0, 0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05

Phase 3:

Parameter
Step 1 **ETA (Learning Rate) :** 0.01, 0.05, 0.1, 0.2, 0.3
Step 2 **ETA (Learning Rate) :** 0.0005, 0.001, 0.005, 0.01, 0.015, 0.02, 0.025

Optimal value of Hyperparameters:

{ 'n_estimators' = 109, 'lambda' = 0.00001, 'eta' = 0.015,
'booster' = 'gblinear', 'objective' = 'binary:logistic', }

ROC-AUC Score:

0.768

XGBOOST

Baseline Model

<u>Parameters</u>

- **Max Depth** = 3
- **Minimum Child Weight** = 1
- **Subsample** = 1
- **Column Sample by Tree** = 1
- **ETA (Learning Rate)** = 0.03
- **Booster** = Tree

Loss Function: Binary Logistic

Test ROC-AUC Score:

<u>0.741</u>

Hyperparameter-tuned Model

<u>Parameters</u>

- **Number of Boosting Rounds** = 163
- **Lambda =** 0
- **ETA (Learning Rate)** = 0.01
- **Booster** = Linear

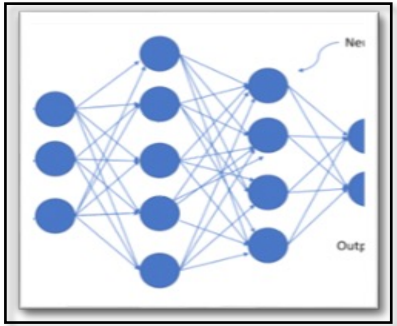
Loss Function: Regression Squared Error

Test ROC-AUC Score:

<u>0.767</u>

Deep Neural Network (DNN)

Model



Baseline Model Test ROC-AUC Score:

0.752

Library

Keras

Parameters

- **Input Layer nodes :** 40
- **# Hidden Layer :** 1
- **Hidden Layer nodes :** 27, **Activation function :** relu
- **Output Layer nodes :** 1, **Activation function :** sigmoid
- **Epochs :** 11
- **Batch size :** 32

Deep Neural Network (DNN)

Baseline Model

<u>Parameters</u>

- **Input Layer nodes** = 40
- **# Hidden Layer** = 1 (Heaton, 2017)
- **Hidden Layer nodes** = 27(Heaton, 2017), **Activation function** = relu
- **Output Layer nodes** = 1, **Activation function** = sigmoid
- **Epochs** = 11 (Epoch : An essential notion in real-time programming, 2023)
- **Batch size** = 32 (Yoshua, 2012)

Test ROC-AUC Score:

0.752

Hyperparameter-tuned Model

<u>Parameters</u>

- **Input Layer nodes** = 35
- **# Hidden Layer** = 1
- **Hidden Layer nodes** =27, **Activation function** = relu
- **Output Layer nodes** = 1, **Activation function** = sigmoid
- **Epochs** = 5
- **Batch size** = 32

Test ROC-AUC Score:

0.753

MODELS COMPARISON

Model	Optimal Hyperparameter	Test ROC-AUC (3 decimal places)
Logistic Regression without Interaction Terms (Baseline)	Default parameters by sklearn.linear_model.LogisticRegression • Penalty : 'l2' • C (inverse of regularization strength): 1.0	0.757
SVM	Strength of the regularization (c):1.3, max iteration: 500	0.686
Random Forest	• Maximum Depth: 5, Minimum Samples to Split Node: 2 • Max Features: log2, Number of trees: 26 • Criterion: Gini	0.762
XGBoost	• Number of Boosting Rounds = 134 • Lambda = 0.00001 • ETA (Learning Rate) = 0.015 • Booster = Linear • Loss Function = Binary Logistic	0.768
DNN	• Input layer nodes: 35, Epochs: 5 • Number of hidden layers: 1, Hidden layer nodes: 27 • ADAM optimizer	0.753

Figure 4.8: Comparison of hyperparameter tuned models results

BEST MODEL METRICS

Metrics	Test Score (3 decimal places)
ROC-AUC	0.768
Accuracy	0.700
Precision	0.645
Recall	0.736
F1 Score	0.687

Figure 4.9: Binary classification metrics of best model

BEST MODEL VISUALISATION

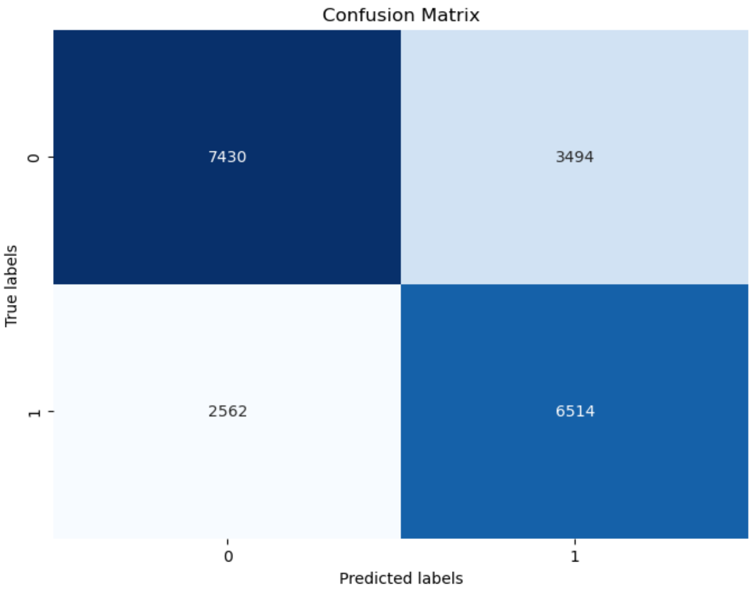


Figure 4.10: Confusion Matrix of best model

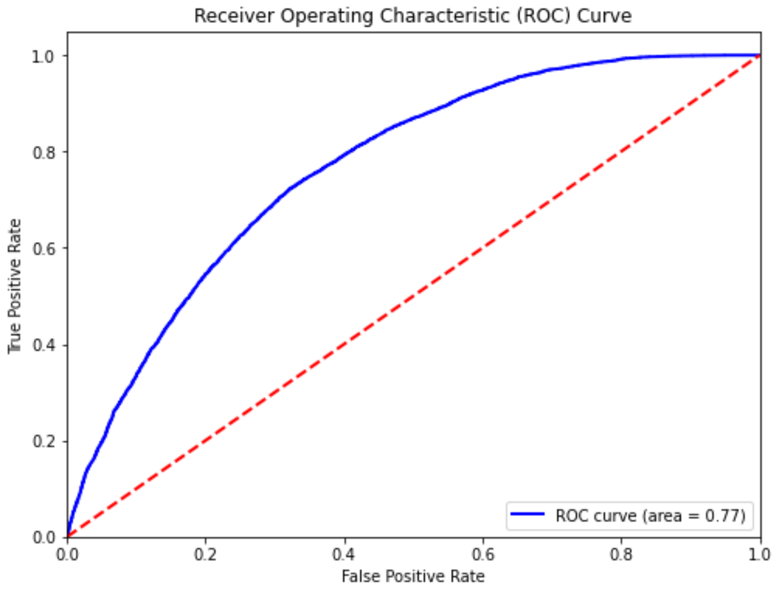


Figure 4.11: ROC Curve of best model

05

Prospects & Challenges

LIMITATIONS & CHALLENGES

High Dimensionality

High computational power → difficult to try more hyperparameters

Limited Data Records

- Only some time-centric information
- Insufficient to spot more trends and create features

1 data source

- Data limited to 1 data source of fashion retailer in Germany
- Model might not generalize well to other countries/industries

FUTURE WORK: DIMENSION REDUCTION

```
AUC: 0.7677417195434493
Accuracy: 0.6999611805690149
Precision: 0.6452017764047113
Recall: 0.7363375936535919
```

39 columns

Baseline

Baseline metrics with our best model: XGBoost

Baseline

PCA

Principle Component Analysis

Linear dimensionality reduction to reduce dimensionality of high-dimensions data by seeking uncorrelated components,

```
AUC: 0.7225629137424336
Accuracy: 0.6568363107807313
Precision: 0.6316331198536139
Recall: 0.6085279858968708
```

7 components

```
AUC: 0.7346542617699692
Accuracy: 0.6678067336180146
Precision: 0.6378013713780137
Recall: 0.6354120758043191
```

7 components

Independent Component Analysis

Non-linear dimensionality reduction by finding statistically independent components

ICA

UMAP

Uniform Manifold Approximation and Projection

Non-linear dimensionality reduction by constructing low-dimensional representation of data

```
AUC: 0.5121532565879665
Accuracy: 0.5
Precision: 0.0
Recall: 0.0
```

7 components

>>>>>>>>>>

FUTURE WORK: DIMENSIONS REDUCTION

Original Dataset

```
(100000, 14)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 14 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   order_item_id  100000 non-null  int64
 1   order_date     100000 non-null  object
 2   delivery_date  90682 non-null   object
 3   item_id        100000 non-null  int64
 4   item_size      100000 non-null  object
 5   item_color     100000 non-null  object
 6   brand_id       100000 non-null  int64
 7   item_price     100000 non-null  float64
 8   user_id        100000 non-null  int64
 9   user_title     100000 non-null  object
 10  user_dob       91275 non-null   object
 11  user_state     100000 non-null  object
 12  user_reg_date  100000 non-null  object
 13  return         100000 non-null  int64
```

- Open sourced Kaggle dataset (Kaggle BADS1920, 2024)
- Real orders by customers of a clothing store describing orders and customers
- **14 columns * 100,000 rows**

Pre-processing

Preprocessed Dataset

```
(99999, 40)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99999 entries, 0 to 99998
Data columns (total 40 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   return                          99999 non-null  int64
 1   no_delivery                     99999 non-null  int64
 2   is_female                       99999 non-null  int64
 3   is_male                         99999 non-null  int64
 4   is_bday                         99999 non-null  int64
 5   most_returned_item              99999 non-null  int64
 6   least_returned_item             99999 non-null  int64
 7   most_returned_color             99999 non-null  int64
 8   least_returned_color            99999 non-null  int64
 9   log_item_price                  99999 non-null  float64
 10  log_no_items                    99999 non-null  float64
 11  log_age                         99999 non-null  float64
 12  log_avg_freq_purchases          99999 non-null  float64
 13  user_state_Baden-Wuerttemberg   99999 non-null  int64
 14  user_state_Bavaria              99999 non-null  int64
 15  user_state_Berlin               99999 non-null  int64
 16  user_state_Brandenburg          99999 non-null  int64
 17  user_state_Bremen               99999 non-null  int64
 18  user_state_Hamburg              99999 non-null  int64
...
 38  user_return_rate                99999 non-null  float64
 39  brand_return_rate               99999 non-null  float64
```

- 40 columns * 99,999 rows
- 1 row dropped
- Feature Engineered: 12 columns
- Target Encoded: 2 columns
- One-Hot Encoded: 25 columns
- Target Variable: 1 column

- Principle Component Analysis (PCA)
- **Independent Component Analysis (ICA)**
- Uniform Manifold Approximation and Projection (UMAP)

X variables after ICA

```
array([[ 0.66347922,  1.11901305,  1.08303885, ..., -1.34552265,
         0.2893769 , -0.46463207],
       [ 1.05797168,  0.22859941,  4.94143397, ..., -1.4499733 ,
         0.02935536,  0.67147536],
       [-0.30880849,  0.0519598 ,  0.13418745, ...,  0.88625827,
         0.17833542,  0.42846051],
       ...,
       [ 0.7980744 , -0.23315639,  0.82193617, ..., -1.213692  ,
        -4.6864155 ,  1.13544907],
       [ 0.7980744 , -0.23315639,  0.82193617, ..., -1.213692  ,
        -4.6864155 ,  1.13544907],
       [ 1.44312825,  0.05009556, -0.50165389, ..., -0.33460742,
         0.23379291,  0.50140081]])
```

- 7 columns * 99,999 rows

06 CONCLUSION

MODEL FINDINGS

Features that make XGBoost model a great predictive model for return prediction context

Non-linear Relationships

- Return prediction models often require capturing complex, non-linear relationships between features and the likelihood of return.
- XGBoost's ability to model complex interactions between features makes it well-suited for capturing these non-linear relationships, leading to more accurate predictions.

Model Performance

- Speed and scalability is necessary for model training and prediction in real-time or large-scale applications
- XGBoost is known for its high performance and efficiency, making it suitable for large datasets commonly encountered in return prediction problems.

Regularization

- Overfitting is a common challenge in return prediction due to the complexity of the data.
- XGBoost's built-in regularization techniques, such as controlling tree depth and leaf weights, help prevent overfitting and improve the model's generalization ability.

MODEL FINDINGS

- Model most important features (weights)
- Positive direction indicates most useful features

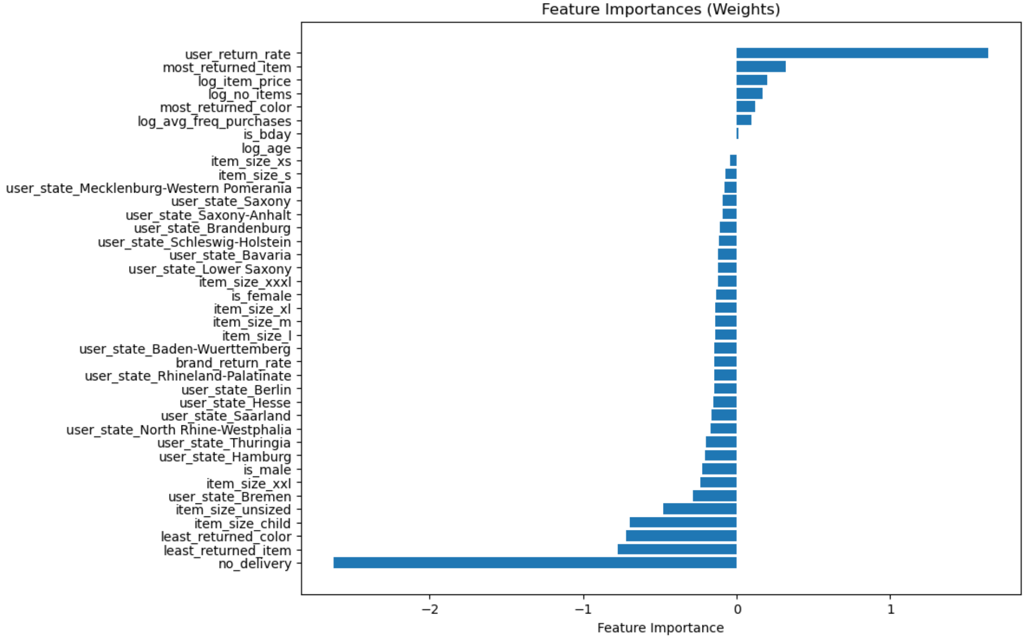


Figure 6.1: Most important features

RECOMMENDATIONS

- Policy recommendations to retailers
- Based on most important features (by weightage) found from XGBoost model

Important feature(s)	Recommendation(s)
most_returned_item	<u>Reapply lessons learned from successful factors</u> • Apply learnings to most_returned_item • Leverage customer feedback and reviews to identify specific pain points or areas for improvement associated with most returned item
log_item_price	<u>Customer Feedback on more expensive items</u> • Conduct testing and customer surveys • Gather insights on areas for improvement • Use feedback to enhance product quality and customer satisfaction
log_no_items	<u>Bundle Discounts</u> • Provide incentives for purchasing multiple items together • Encourage more deliberate purchases and increase transaction value
most_returned_colour	<u>Market Analysis</u> • Customer taste and preference for color profiles • Expected vs Actual color • Quality of clothes

Figure 6.3: Recommendations for important features

>>>>>>>>>>

RECOMMENDATIONS

- Policy recommendations to retailers
- Based on most important features (by weightage) found from XGBoost model

Important feature(s)	Recommendation(s)
log_avg_freq_purchases	<u>Purchase Analysis</u> • Examine average frequent purchases and item combinations • Identify trends in the number of items and sizes bought together • Utilize data to inform inventory and marketing strategies
is_bday	<u>Purchase Analysis</u> • Identify lead time before birthday, the purchases were made • Examine what the customers purchases returned were • Identify possible root causes for returning item on birthday

Figure 6.4: Recommendations for important features

THANKS!

Bergstra, J., & Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. Journal of Machine Learning Research, 13, 281–305. Retrieved from https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

Epoch : An essential notion in real-time programming. DataScientest. (2023, June 2). https://datascientest.com/en/epoch-an-essential-notion#:~:text=A%20larger%20number%20of%20epochs,to%20optimally%20modify%20the%20weights

Heaton, J. (2017, June 1). The number of hidden layers. Heaton Research. https://www.heatonresearch.com/2017/06/01/hidden-layers.html

Kdnuggets, Hyperparameter Tuning Using Grid Search and Random Search in Python. KDnuggets,2022

Melanie. (2024, February 11). Calculate correlation between two variables: How do you measure dependence?. DataScientest. https://datascientest.com/en/calculate-correlation-between-two-variables-how-do-you-measure-dependence

Selvaraj, N. (2022, October 5). Hyperparameter Tuning Using Grid Search and Random Search in Python. KDnuggets. https://www.kdnuggets.com/2022/10/hyperparameter-tuning-grid-search-random-search-python.html

Udilâ, A.-I., Ionescu, A., & Katsifodimos, A. (2023). Encoding Methods for Categorical Data: A Comparative Analysis for Linear Models, Decision Trees, and Support Vector Machines. https://repository.tudelft.nl/islandora/object/uuid:10b91b99-2685-4a45-b44e-48fbbf808ce2/datastream/OBJ/download#:~:text=Target%20encoding%2C%20also%20known%20as

Yoshua, B. (2012, September 16). Practical Recommendations for Gradient-Based Training of Deep Arthitectures. https://arxiv.org/pdf/1206.5533.pdf