

# **Machine Learning**

**(240-674)**

Wathunyu Phetpaya

6710120039, Computer Engineering, Master Degree

Prince of Songkla University

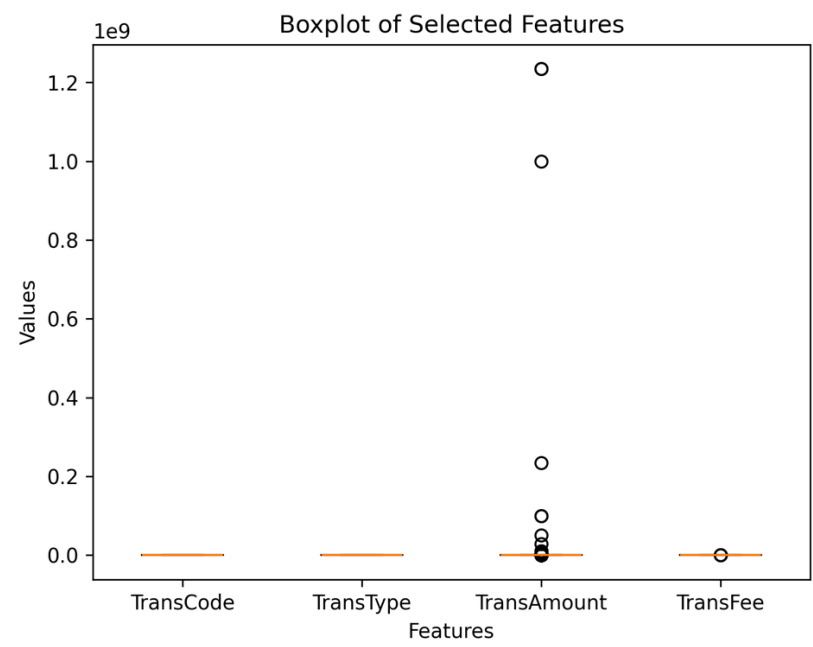
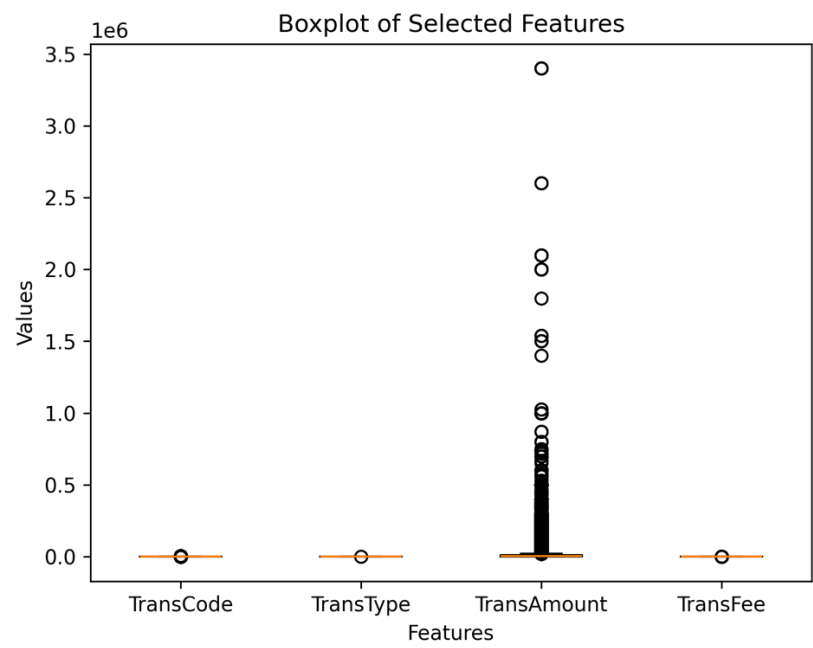
## **Dataset**

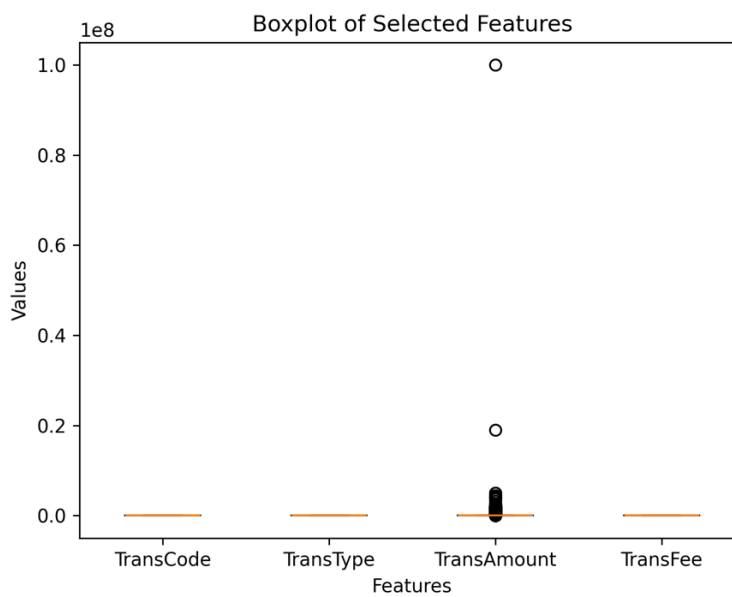
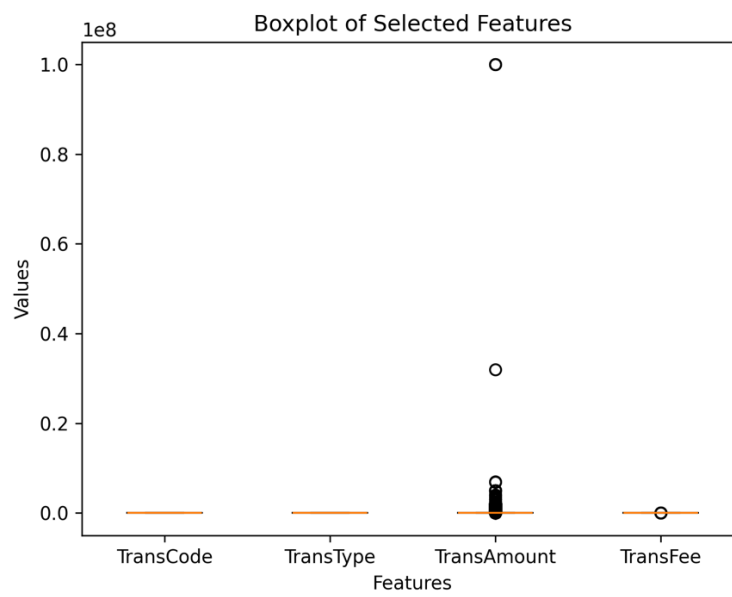
- ☐ Prince of Songkla University Cooperative Credit And Saving, Limited

## **Environment**

- ☐ Python Version: 3.12.0
- ☐ Library: numpy, pandas, scikit-learn, matplotlib, time, wraps
- ☐ `pip install -r requirements.txt`

Data Distribution





## Data Transformation

```
69 # Combine all the data into one DataFrame
70 combined_data = pd.concat([data2564, data2565, data2566, data2567])
71
72 # Convert Tran_Date to datetime and extract the month and year
73 combined_data['Tran_Date'] = pd.to_datetime(combined_data['Tran_Date'], format='%Y%m%d')
74 combined_data['Birth_date'] = pd.to_datetime(combined_data['Birth_date'], format='%Y%m%d')
75
76 # Calculate age at the time of the transaction
77 combined_data['Age'] = (combined_data['Tran_Date'] - combined_data['Birth_date']).dt.days // 365
78
79 # Define age ranges, including 51-60 and 61+
80 bins = [0, 18, 30, 40, 50, 60, 100]
81 labels = ['0-18', '19-30', '31-40', '41-50', '51-60', '61+']
82 combined_data['AgeRange'] = pd.cut(combined_data['Age'], bins=bins, labels=labels, right=False)
83
84 # Extract year and month for grouping
85 combined_data['YearMonth'] = combined_data['Tran_Date'].dt.to_period('M')
86
87 # Count TransCode occurrences by month and age range, setting observed=False for compatibility
88 transcode_counts_by_age = combined_data.groupby(['YearMonth', 'AgeRange', 'TransCode'], observed=False).size().unstack(fill_value=0)
89
90 # Reindex to ensure all age ranges are included, filling with 0 where necessary
91 all_age_ranges = pd.Categorical(labels, categories=labels, ordered=True)
92
93 # Fill missing values with 0 to ensure complete data for all age ranges
94 transcode_counts_by_age = transcode_counts_by_age.fillna(0)
```

## Data Training

### 1. Random Forest

```
9 # Create and train the Random Forest model
10 model_rf = RandomForestRegressor(n_estimators=100, random_state=42)
11 model_rf.fit(X_train, y_train)
12
13 # Prepare data for prediction in 2024 OCT
14 future_year_months = pd.date_range(start='2024-10-01', end='2024-10-31', freq='ME').to_period('M')
15 age_ranges = labels
16
17 # Create DataFrame for future predictions
18 future_data = []
19 for month in future_year_months:
20     for age_range in age_ranges:
21         future_data.append({'YearMonth': month, 'AgeRange': age_range})
22 future_data = pd.DataFrame(future_data)
23
24 # Convert categorical variables to dummy/indicator variables
25 future_X = pd.get_dummies(future_data[['AgeRange']], drop_first=True)
26 # Align with training data columns (excluding TransCode columns which are targets)
27 future_X = future_X.reindex(columns=X_train.columns[X_train.columns.str.startswith('AgeRange_')], fill_value=0)
28
29 # Add TransCode columns to future_X, initializing with 0 and ensuring string column names
30 for transcode in transcode_counts_by_age.columns:
31     future_X[str(transcode)] = 0 # Convert transcode to string for column name
32
33 # Make predictions for 2024 OCT
34 future_predictions = model_rf.predict(future_X)
35
36 # Create DataFrame for predictions
37 predictions_df = pd.DataFrame(future_predictions, columns=target_columns)
38 predictions_df = pd.concat([future_data, predictions_df], axis=1)
39
40 # Reshape to match transcode_counts_by_age format
41 predictions_by_age = predictions_df.groupby(['YearMonth', 'AgeRange'])[target_columns].sum().unstack(fill_value=0)
42
43 # Display the predictions
44 return predictions_by_age, model_rf
```

## 2. Support Vector Machine

```
10 def svm_regression(transcode_counts_by_age, labels, target_columns, model_svm, X_train, X_test, y_train, y_test):
11     # Prepare the data for the model
12     data = transcode_counts_by_age.reset_index()
13     data['YearMonth'] = data['YearMonth'].astype(str)
14
15     # Convert categorical variables to dummy/indicator variables
16     X = pd.get_dummies(data[['AgeRange']], drop_first=True) # Only AgeRange for now
17     X = pd.concat([X, data[transcode_counts_by_age.columns]], axis=1) # Add TransCode columns
18
19     # Create the target variable (e.g., predicting counts for a specific TransCode)
20     target_columns = transcode_counts_by_age.columns # All TransCodes as targets
21     y = X[target_columns]
22
23     # Ensure column names are strings
24     X.columns = X.columns.astype(str) # Convert all column names to strings
25     y.columns = y.columns.astype(str) # Convert all column names to strings
26
27     # Split the data into training and testing sets
28     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
29
30     # Create and train the SVM model (using SVR for regression)
31     model_svm = MultiOutputRegressor(SVR(kernel='rbf')) # You can try different kernels like 'linear', 'poly'
32     model_svm.fit(X_train, y_train)
33
34     # Prepare data for prediction in 2024 OCT
35     future_year_months = pd.date_range(start='2024-10-01', end='2024-10-31', freq='ME').to_period('M')
36     age_ranges = labels
37
38     # Create DataFrame for future predictions
39     future_data = []
40     for month in future_year_months:
41         for age_range in age_ranges:
42             future_data.append({'YearMonth': month, 'AgeRange': age_range})
43     future_data = pd.DataFrame(future_data)
44
45     # Convert categorical variables to dummy/indicator variables
46     future_X = pd.get_dummies(future_data[['AgeRange']], drop_first=True)
47     # Get the dummy column names from training data
48     age_range_dummy_cols = X_train.columns[X_train.columns.str.startswith('AgeRange_')]
49     # Align with training data columns (excluding TransCode columns which are targets)
50     future_X = future_X.reindex(columns=age_range_dummy_cols, fill_value=0)
51
52     # Add TransCode columns to future_X, initializing with 0 and ensuring string column names
53     for transcode in transcode_counts_by_age.columns:
54         future_X[str(transcode)] = 0 # Convert transcode to string for column name
55
56     # Make predictions for 2024 OCT
57     future_predictions = model_svm.predict(future_X)
58
59     # Create DataFrame for predictions
60     predictions_df = pd.DataFrame(future_predictions, columns=target_columns)
61     predictions_df = pd.concat([future_data, predictions_df], axis=1)
62
63     # Reshape to match transcode_counts_by_age format
64     predictions_by_age = predictions_df.groupby(['YearMonth', 'AgeRange'])[target_columns].sum().unstack(fill_value=0)
65
66     # Display the predictions
67     return predictions_by_age, model_svm
```

### 3. Neural Network

```
8 @time_measure
9 def neural_net(transcode_counts_by_age, labels, target_columns, model_nn, X_train, X_test, y_train, y_test, epochs=100):
10
11     # Get the columns before scaling
12     X_train_columns = X_train.columns
13
14     # Scale the data
15     scaler = StandardScaler()
16     X_train = scaler.fit_transform(X_train)
17     X_test = scaler.transform(X_test)
18
19     # Create the neural network model
20     model_nn = tf.keras.models.Sequential([
21         tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
22         tf.keras.layers.Dense(32, activation='relu'),
23         tf.keras.layers.Dense(len(target_columns)) # Output layer with number of TransCodes
24     ])
25
26     # Compile the model
27     model_nn.compile(optimizer='adam', loss='mse') # Mean squared error for regression
28
29     # Train the model
30     model_nn.fit(X_train, y_train, epochs=epochs, batch_size=32, validation_data=(X_test, y_test))
31
32     # Prepare data for prediction in 2024 OCT
33     future_year_months = pd.date_range(start='2024-10-01', end='2024-10-31', freq='ME').to_period('M')
34     A age_ranges = labels
35
36     # Create DataFrame for future predictions
37     future_data = []
38     for month in future_year_months:
39         for age_range in age_ranges:
40             future_data.append({'YearMonth': month, 'AgeRange': age_range})
41     future_data = pd.DataFrame(future_data)
42
43     # Convert categorical variables to dummy/indicator variables
44     future_X = pd.get_dummies(future_data[['AgeRange']], drop_first=True)
45
46     # Use X_train_columns instead of X_train.columns
47     future_X = future_X.reindex(columns=X_train_columns[X_train_columns.str.startswith('AgeRange')], fill_value=0)
48
49     # Add TransCode columns to future_X, initializing with 0 and ensuring string column names
50     for transcode in transcode_counts_by_age.columns:
51         future_X[str(transcode)] = 0 # Convert transcode to string for column name
52
53     # Scale the future data
54     future_X_scaled = scaler.transform(future_X) # Scale the future_X DataFrame using the same scaler
55
56     # Make predictions for 2024 using the scaled future data
57     future_predictions = model_nn.predict(future_X_scaled)
58
59     # Create DataFrame for predictions
60     predictions_df = pd.DataFrame(future_predictions, columns=target_columns)
61     predictions_df = pd.concat([future_data, predictions_df], axis=1)
62
63     # Reshape to match transcode_counts_by_age format
64     predictions_by_age = predictions_df.groupby(['YearMonth', 'AgeRange'])[target_columns].sum().unstack(fill_value=0)
65
```

## Result

- We're tasked with identifying the most suitable machine learning model (Random Forest, SVM, or Neural Network) for a dataset where the age values for individuals between 0-18 consistently remain zero. This is a common scenario in real-world datasets, especially when dealing with demographic data.

## Model Comparison:

### 1. Random Forest:

- Strengths:
  - Handles both classification and regression tasks well.
  - Can handle missing values and categorical data.
  - Provides feature importance scores, aiding in interpretability.
- Weaknesses:
  - May overfit on noisy data.
  - Can be computationally expensive for large datasets.
- Suitability:
  - Well-suited for the problem if the relationship between features and the target variable is non-linear.
  - Can effectively handle the zero values in the 0-18 age range.

### 2. Support Vector Machines (SVM):

- Strengths:
  - Effective for both classification and regression.
  - Can handle high-dimensional data.
  - Performs well on smaller datasets.
- Weaknesses:

- Sensitive to the choice of kernel.
  - Not as scalable as random forests for large datasets.
  - Suitability:
    - Well-suited for binary classification problems.
    - Might struggle with the large number of features that are often associated with neural networks.
3. Neural Networks:
- Strengths:
    - Can learn complex patterns in data.
    - Can handle large datasets.
    - Highly flexible due to various architecture.
  - Weaknesses:
    - Require significant computational resources and data.
    - Can be difficult to interpret.
    - Overfitting can be a problem if not properly regularized.
  - Suitability:
    - Well-suited for large datasets with complex relationships between features.
    - Can handle the zero values in the 0-18 age range, especially if combined with appropriate data preprocessing and architecture.

In conclusion, neural networks have demonstrated their exceptional capabilities in modeling and analyzing complex data. Their ability to learn complex patterns, handle non-linear relationships, and adapt to diverse data types make them a versatile tool for a wide range of applications. From excel dataset to time series forecasting, neural networks have proven their effectiveness in extracting valuable insights and making accurate predictions.