Machine Learning

(240-674)

Wathunyu Phetpaya

6710120039, Computer Engineering, Master Degree

Prince of Songkla University

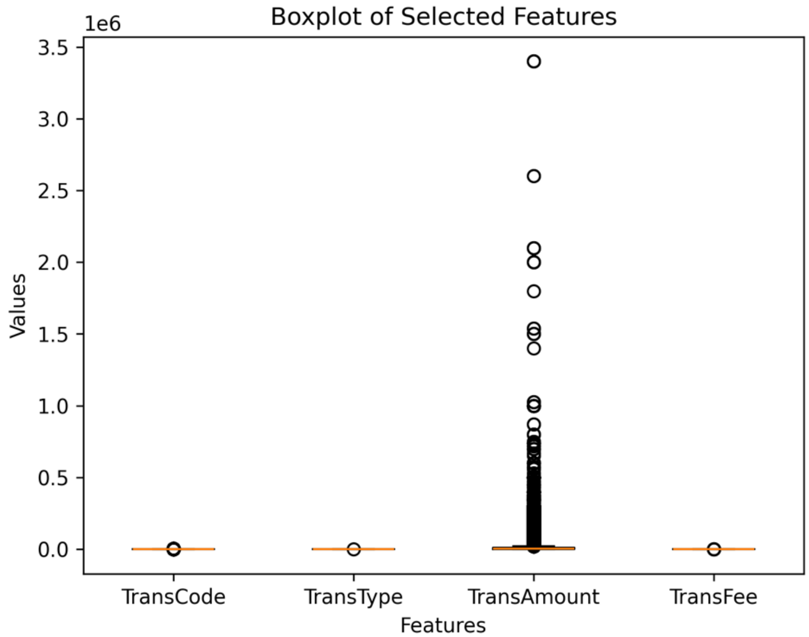
Dataset

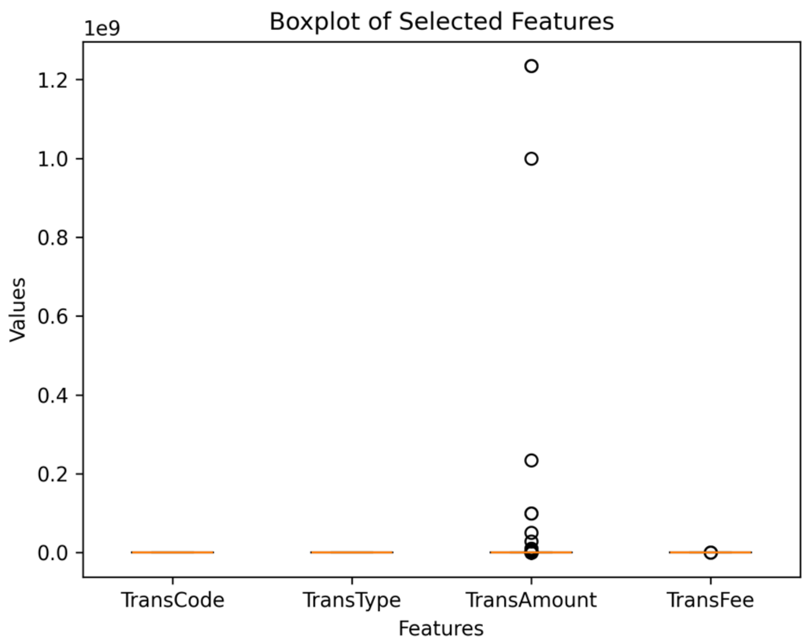
- Prince of Songkla University Cooperative Credit And Saving, Limited

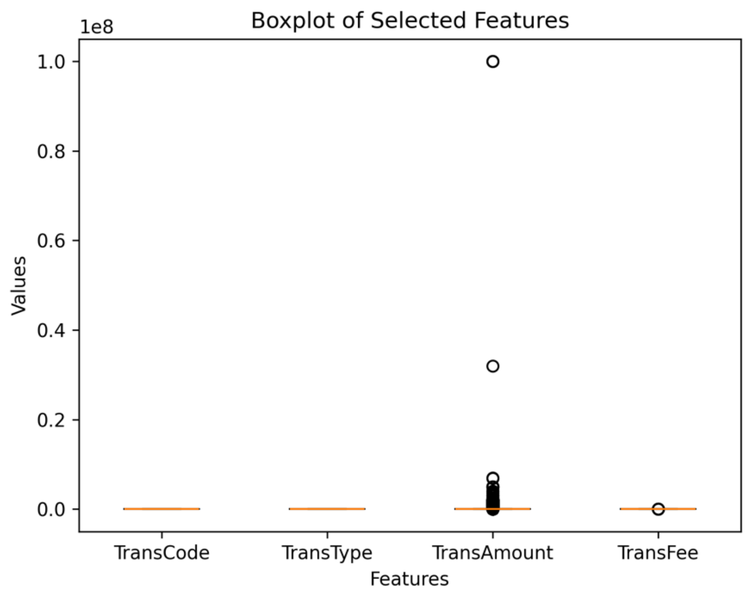
Environment

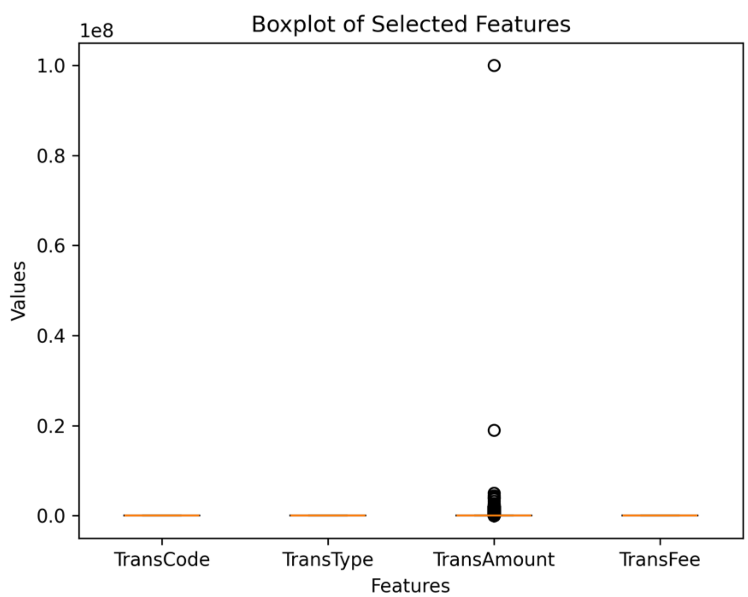
- Python Version: 3.12.0
- Library: numpy, pandas, scikit-learn, matplotlib, time, wraps
- pip install -r requirements.txt

Data Distribution






Boxplot of Selected Features
1e8
1.0
0.8
0.6
0.4
0.2
0.0
Values
TransCode
TransType
TransAmount
TransFee
Features


Boxplot of Selected Features
1e8
1.0
0.8
0.6
0.4
0.2
0.0
Values
TransCode
TransType
TransAmount
TransFee
Features

Data Transformation

```python
# Combine all the data into one DataFrame
combined_data = pd.concat([data2564, data2565, data2566, data2567])

# Convert Tran_Date to datetime and extract the month and year
combined_data['Tran_Date'] = pd.to_datetime(combined_data['Tran_Date'], format='%Y%m%d')
combined_data['Birth_date'] = pd.to_datetime(combined_data['Birth_date'], format='%Y%m%d')

# Calculate age at the time of the transaction
combined_data['Age'] = (combined_data['Tran_Date'] - combined_data['Birth_date']).dt.days // 365

# Define age ranges, including 51-60 and 61+
bins = [0, 18, 30, 40, 50, 60, 100]
labels = ['0-18', '19-30', '31-40', '41-50', '51-60', '61+']
combined_data['AgeRange'] = pd.cut(combined_data['Age'], bins=bins, labels=labels, right=False)

# Extract year and month for grouping
combined_data['YearMonth'] = combined_data['Tran_Date'].dt.to_period('M')

# Count TransCode occurrences by month and age range, setting observed=False for compatibility
transcode_counts_by_age = combined_data.groupby(['YearMonth', 'AgeRange', 'TransCode'], observed=False).size().unstack(fill_value=0)

# Reindex to ensure all age ranges are included, filling with 0 where necessary
all_age_ranges = pd.Categorical(labels, categories=labels, ordered=True)

# Fill missing values with 0 to ensure complete data for all age ranges
transcode_counts_by_age = transcode_counts_by_age.fillna(0)
```

Data Training

1. Random Forest

```python
# Create and train the Random Forest model
model_rf = RandomForestRegressor(n_estimators=100, random_state=42)
model_rf.fit(X_train, y_train)

# Prepare data for prediction in 2024 OCT
future_year_months = pd.date_range(start='2024-10-01', end='2024-10-31', freq='ME').to_period('M')
age_ranges = labels

# Create DataFrame for future predictions
future_data = []
for month in future_year_months:
    for age_range in age_ranges:
        future_data.append({'YearMonth': month, 'AgeRange': age_range})
future_data = pd.DataFrame(future_data)

# Convert categorical variables to dummy/indicator variables
future_X = pd.get_dummies(future_data[['AgeRange']], drop_first=True)
# Align with training data columns (excluding TransCode columns which are targets)
future_X = future_X.reindex(columns=X_train.columns[X_train.columns.str.startswith('AgeRange_')], fill_value=0)

# Add TransCode columns to future_X, initializing with 0 and ensuring string column names
for transcode in transcode_counts_by_age.columns:
    future_X[str(transcode)] = 0  # Convert transcode to string for column name

# Make predictions for 2024 OCT
future_predictions = model_rf.predict(future_X)

# Create DataFrame for predictions
predictions_df = pd.DataFrame(future_predictions, columns=target_columns)
predictions_df = pd.concat([future_data, predictions_df], axis=1)

# Reshape to match transcode_counts_by_age format
predictions_by_age = predictions_df.groupby(['YearMonth', 'AgeRange'])[target_columns].sum().unstack(fill_value=0)

# Display the predictions
return predictions_by_age, model_rf
```

2. Support Vector Machine

```python
def svm_regression(transcode_counts_by_age, labels, target_columns, model_svm, X_train, X_test, y_train, y_test):
    # Prepare the data for the model
    data = transcode_counts_by_age.reset_index()
    data['YearMonth'] = data['YearMonth'].astype(str)

    # Convert categorical variables to dummy/indicator variables
    X = pd.get_dummies(data[['AgeRange']], drop_first=True)  # Only AgeRange for now
    X = pd.concat([X, data[transcode_counts_by_age.columns]], axis=1) # Add TransCode columns

    # Create the target variable (e.g., predicting counts for a specific TransCode)
    target_columns = transcode_counts_by_age.columns  # All TransCodes as targets
    y = X[target_columns]

    # Ensure column names are strings
    X.columns = X.columns.astype(str)  # Convert all column names to strings
    y.columns = y.columns.astype(str)  # Convert all column names to strings

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Create and train the SVM model (using SVR for regression)
    model_svm = MultiOutputRegressor(SVR(kernel='rbf'))  # You can try different kernels like 'linear', 'poly'
    model_svm.fit(X_train, y_train)

    # Prepare data for prediction in 2024 OCT
    future_year_months = pd.date_range(start='2024-10-01', end='2024-10-31', freq='ME').to_period('M')
    age_ranges = labels

    # Create DataFrame for future predictions
    future_data = []
    for month in future_year_months:
        for age_range in age_ranges:
            future_data.append({'YearMonth': month, 'AgeRange': age_range})
    future_data = pd.DataFrame(future_data)

    # Convert categorical variables to dummy/indicator variables
    future_X = pd.get_dummies(future_data[['AgeRange']], drop_first=True)
    # Get the dummy column names from training data
    age_range_dummy_cols = X_train.columns[X_train.columns.str.startswith('AgeRange_')]
    # Align with training data columns (excluding TransCode columns which are targets)
    future_X = future_X.reindex(columns=age_range_dummy_cols, fill_value=0)

    # Add TransCode columns to future_X, initializing with 0 and ensuring string column names
    for transcode in transcode_counts_by_age.columns:
        future_X[str(transcode)] = 0  # Convert transcode to string for column name

    # Make predictions for 2024 OCT
    future_predictions = model_svm.predict(future_X)

    # Create DataFrame for predictions
    predictions_df = pd.DataFrame(future_predictions, columns=target_columns)
    predictions_df = pd.concat([future_data, predictions_df], axis=1)

    # Reshape to match transcode_counts_by_age format
    predictions_by_age = predictions_df.groupby(['YearMonth', 'AgeRange'])[target_columns].sum().unstack(fill_value=0)

    # Display the predictions
    return predictions_by_age, model_svm
```

3. Neural Network

```python
@time_measure
def neural_net(transcode_counts_by_age, labels, target_columns, model_nn, X_train, X_test, y_train, y_test, epochs=100):

    # Get the columns before scaling
    X_train_columns = X_train.columns

    # Scale the data
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Create the neural network model
    model_nn = tf.keras.models.Sequential([
        tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
        tf.keras.layers.Dense(32, activation='relu'),
        tf.keras.layers.Dense(len(target_columns))  # Output layer with number of TransCodes
    ])

    # Compile the model
    model_nn.compile(optimizer='adam', loss='mse')  # Mean squared error for regression

    # Train the model
    model_nn.fit(X_train, y_train, epochs=epochs, batch_size=32, validation_data=(X_test, y_test))

    # Prepare data for prediction in 2024 OCT
    future_year_months = pd.date_range(start='2024-10-01', end='2024-10-31', freq='ME').to_period('M')
    age_ranges = labels

    # Create DataFrame for future predictions
    future_data = []
    for month in future_year_months:
        for age_range in age_ranges:
            future_data.append({'YearMonth': month, 'AgeRange': age_range})
    future_data = pd.DataFrame(future_data)

    # Convert categorical variables to dummy/indicator variables
    future_X = pd.get_dummies(future_data[['AgeRange']], drop_first=True)

    # Use X_train_columns instead of X_train.columns
    future_X = future_X.reindex(columns=X_train_columns[X_train_columns.str.startswith('AgeRange_')], fill_value=0)

    # Add TransCode columns to future_X, initializing with 0 and ensuring string column names
    for transcode in transcode_counts_by_age.columns:
        future_X[str(transcode)] = 0  # Convert transcode to string for column name

    # Scale the future data
    future_X_scaled = scaler.transform(future_X) # Scale the future_X DataFrame using the same scaler

    # Make predictions for 2024 using the scaled future data
    future_predictions = model_nn.predict(future_X_scaled)

    # Create DataFrame for predictions
    predictions_df = pd.DataFrame(future_predictions, columns=target_columns)
    predictions_df = pd.concat([future_data, predictions_df], axis=1)

    # Reshape to match transcode_counts_by_age format
    predictions_by_age = predictions_df.groupby(['YearMonth', 'AgeRange'])[target_columns].sum().unstack(fill_value=0)
```

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 image and natural language processing to time series forecasting, neural networks have proven their effectiveness in extracting valuable insights and making accurate predictions.