03-predicting-credit-card-approvals Final

March 22, 2025

Commercial banks receive *a lot* of applications for credit cards. Many of them get rejected for many reasons, like high loan balances, low income levels, or too many inquiries on an individual's credit report, for example. Manually analyzing these applications is mundane, error-prone, and time-consuming (and time is money!). Luckily, this task can be automated with the power of machine learning and pretty much every commercial bank does so nowadays. In this workbook, you will build an automatic credit card approval predictor using machine learning techniques, just like real banks do.

0.0.1 The Data

The data is a small subset of the Credit Card Approval dataset from the UCI Machine Learning Repository showing the credit card applications a bank receives. This dataset has been loaded as a `pandas` DataFrame called `cc_apps`. The last column in the dataset is the target value.

[52]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from tensorflow.keras import regularizers
import xgboost as xgb
```

0.1 Data Loading and Exploration

This section involves loading the credit card approval dataset and performing initial exploration. The data is loaded from a CSV file into a pandas DataFrame called `cc_apps`. The `head()` function is used to display the first few rows of the dataset, giving an initial view of its structure and content.

[53]:
```
# Load dataset
cc_apps = pd.read_csv("/kaggle/input/
 ↪predicting-credit-card-approvals-my-dataset/cc_approvals.data", header=None)

cc_apps.head()
```

[53]:
```
   0      1      2  3  4  5  6     7  8  9  10 11   12 13
0  b  30.83  0.000  u  g  w  v  1.25  t  t   1  g    0  +
1  a  58.67  4.460  u  g  q  h  3.04  t  t   6  g  560  +
2  a  24.50  0.500  u  g  q  h  1.50  t  f   0  g  824  +
3  b  27.83  1.540  u  g  w  v  3.75  t  t   5  g    3  +
4  b  20.17  5.625  u  g  w  v  1.71  t  f   0  s    0  +
```

0.2 Data Preprocessing

0.2.1 Handling Missing Values

The dataset likely contains missing values that need to be addressed. This may involve imputation techniques or removal of incomplete records.

[54]:
```
# Replace the '?'s with NaN in dataset
cc_apps_nans_replaced = cc_apps.replace("?", np.NaN)

# Create a copy of the NaN replacement DataFrame
cc_apps_imputed = cc_apps_nans_replaced.copy()

# Iterate over each column of cc_apps_nans_replaced and impute the most␣
 ↪frequent value for object data types and the mean for numeric data types
for col in cc_apps_imputed.columns:
    # Check if the column is of object type
    if cc_apps_imputed[col].dtypes == "object":
        # Impute with the most frequent value
        cc_apps_imputed[col] = cc_apps_imputed[col].fillna(
            cc_apps_imputed[col].value_counts().index[0]
        )
    else:
        cc_apps_imputed[col] = cc_apps_imputed[col].fillna(cc_apps_imputed[col].
 ↪mean())

cc_apps_imputed.head()
```

[54]:
```
   0      1      2  3  4  5  6     7  8  9  10 11   12 13
0  b  30.83  0.000  u  g  w  v  1.25  t  t   1  g    0  +
1  a  58.67  4.460  u  g  q  h  3.04  t  t   6  g  560  +
2  a  24.50  0.500  u  g  q  h  1.50  t  f   0  g  824  +
3  b  27.83  1.540  u  g  w  v  3.75  t  t   5  g    3  +
4  b  20.17  5.625  u  g  w  v  1.71  t  f   0  s    0  +
```

0.2.2 Encoding Categorical Variables

Many features in the dataset are categorical and need to be converted to numerical format for machine learning algorithms. This could involve techniques like one-hot encoding or label encoding.

[55]:
```
# Dummify the categorical features
cc_apps_encoded = pd.get_dummies(cc_apps_imputed, drop_first=True)
cc_apps_encoded.head()
```

[55]:
```
       2     7  10   12    0_b  1_15.17  1_15.75  1_15.83  1_15.92  1_16.00  \
0  0.000  1.25   1    0   True    False    False    False    False    False
1  4.460  3.04   6  560  False    False    False    False    False    False
2  0.500  1.50   0  824  False    False    False    False    False    False
3  1.540  3.75   5    3   True    False    False    False    False    False
4  5.625  1.71   0    0   True    False    False    False    False    False

   …    6_j    6_n    6_o    6_v    6_z   8_t    9_t   11_p   11_s   13_-
0  …  False  False  False   True  False  True   True  False  False  False
1  …  False  False  False  False  False  True   True  False  False  False
2  …  False  False  False  False  False  True  False  False  False  False
3  …  False  False  False   True  False  True   True  False  False  False
4  …  False  False  False   True  False  True  False  False   True  False

[5 rows x 383 columns]
```

0.2.3 Splitting the Dataset

The data is split into training and testing sets to allow for model evaluation on unseen data.

```
[56]: # Extract the last column as your target variable
      X = cc_apps_encoded.iloc[:, :-1].values
      y = cc_apps_encoded.iloc[:, [-1]].values

      # Split into train and test sets
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.135,␣
       ↪random_state=42)

      X_train.shape
```

```
[56]: (596, 382)
```

0.2.4 Feature Scaling

Numerical features may need to be scaled to a common range to ensure all features contribute equally to the model. This often involves using techniques like StandardScaler or MinMaxScaler.

```
[57]: # Instantiate StandardScaler and use it to rescale X_train and X_test
      scaler = StandardScaler()
      rescaledX_train = scaler.fit_transform(X_train)
      rescaledX_test = scaler.transform(X_test)
```

0.3 Model Selection and Training

0.3.1 Model Choice

An XGBoost classifier is chosen for this task, likely due to its effectiveness in handling tabular data and its ability to capture complex relationships between features.

```
[86]: # Instantiate the XGBoost classifier
      model = xgb.XGBClassifier(
          n_estimators=100,  # Number of boosting rounds
          learning_rate=0.1,  # Learning rate
          max_depth=3,  # Maximum depth of the trees
          random_state=42
      )
```

0.3.2 Model Training

The XGBoost model is trained on the preprocessed training data.

```
[87]: # Training the SVM model
      model.fit(rescaledX_train, y_train)
```

```
[87]: XGBClassifier(base_score=None, booster=None, callbacks=None,
                    colsample_bylevel=None, colsample_bynode=None,
                    colsample_bytree=None, device=None, early_stopping_rounds=None,
                    enable_categorical=False, eval_metric=None, feature_types=None,
                    gamma=None, grow_policy=None, importance_type=None,
                    interaction_constraints=None, learning_rate=0.1, max_bin=None,
                    max_cat_threshold=None, max_cat_to_onehot=None,
                    max_delta_step=None, max_depth=3, max_leaves=None,
                    min_child_weight=None, missing=nan, monotone_constraints=None,
                    multi_strategy=None, n_estimators=100, n_jobs=None,
                    num_parallel_tree=None, random_state=42, …)
```

[88]:
```
# Predict on the test set
y_pred = model.predict(rescaledX_test)

# Convert probabilities to binary class labels
y_pred_binary = (y_pred > 0.5).astype(int)
```

0.4 Model Evaluation

The trained model's performance is evaluated, using metrics such as accuracy on the test set.

[89]:
```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred_binary)
print(f"Logistic Regression Classifier Accuracy: {accuracy:.4f}")
```

```
Logistic Regression Classifier Accuracy: 0.8404
```

0.5 Results Interpretation

The final model's performance is analyzed, and insights are drawn about the most important features for credit card approval prediction.

[ ]:
```

```

[ ]:
```

```