temp

October 12, 2024

[19]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import os
# base_path = "/kaggle/input/playground-series-s4e10"
df_train = pd.read_csv(os.path.join('train.csv'))
df_test = pd.read_csv(os.path.join('test.csv'))
df_sample = pd.read_csv(os.path.join('sample_submission.csv'))
TARGET = 'loan_status'
```

Preprocessing

[20]:
```
for column in df_train.columns:
    nan_count = df_train[column].isna().sum()
    print(f"Column '{column}' has {nan_count} NaN values.")
```

```
Column 'id' has 0 NaN values.
Column 'person_age' has 0 NaN values.
Column 'person_income' has 0 NaN values.
Column 'person_home_ownership' has 0 NaN values.
Column 'person_emp_length' has 0 NaN values.
Column 'loan_intent' has 0 NaN values.
Column 'loan_grade' has 0 NaN values.
Column 'loan_amnt' has 0 NaN values.
```

```
Column 'loan_int_rate' has 0 NaN values.
Column 'loan_percent_income' has 0 NaN values.
Column 'cb_person_default_on_file' has 0 NaN values.
Column 'cb_person_cred_hist_length' has 0 NaN values.
Column 'loan_status' has 0 NaN values.
```

[21]:
```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()

for col in df_train.select_dtypes(include=['object']).columns:
    df_train[col] = label_encoder.fit_transform(df_train[col])

for col in df_test.select_dtypes(include=['object']).columns:
    df_test[col] = label_encoder.fit_transform(df_test[col])

print(df_train.head())
```

```
   id  person_age  person_income  person_home_ownership  person_emp_length  \
0   0          37          35000                      3                0.0
1   1          22          56000                      2                6.0
2   2          29          28800                      2                8.0
3   3          30          70000                      3               14.0
4   4          22          60000                      3                2.0

   loan_intent  loan_grade  loan_amnt  loan_int_rate  loan_percent_income  \
0            1           1       6000          11.49                 0.17
1            3           2       4000          13.35                 0.07
2            4           0       6000           8.90                 0.21
3            5           1      12000          11.11                 0.17
4            3           0       6000           6.92                 0.10

   cb_person_default_on_file  cb_person_cred_hist_length  loan_status
0                          0                          14            0
1                          0                           2            0
2                          0                          10            0
3                          0                           5            0
4                          0                           3            0
```

[22]:
```
print(df_train.describe())
```

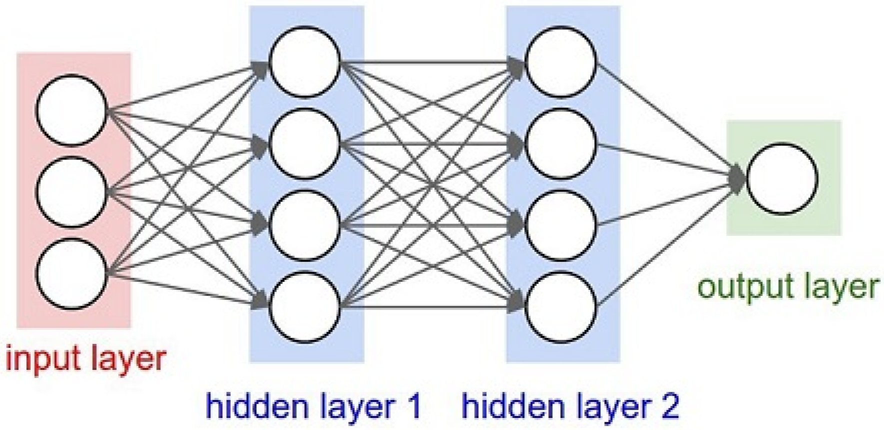
```
                 id    person_age  person_income  person_home_ownership  \
count  58645.000000  58645.000000   5.864500e+04           58645.000000
mean   29322.000000     27.550857   6.404617e+04               1.673578
std    16929.497605      6.033216   3.793111e+04               1.452534
min        0.000000     20.000000   4.200000e+03               0.000000
25%    14661.000000     23.000000   4.200000e+04               0.000000
50%    29322.000000     26.000000   5.800000e+04               3.000000
75%    43983.000000     30.000000   7.560000e+04               3.000000
```

```
max    58644.000000    123.000000   1.900000e+06               3.000000

       person_emp_length   loan_intent    loan_grade     loan_amnt  \
count       58645.000000  58645.000000  58645.000000  58645.000000
mean            4.701015      2.519430      1.066638   9217.556518
std             3.959784      1.722896      1.046181   5563.807384
min             0.000000      0.000000      0.000000    500.000000
25%             2.000000      1.000000      0.000000   5000.000000
50%             4.000000      3.000000      1.000000   8000.000000
75%             7.000000      4.000000      2.000000  12000.000000
max           123.000000      5.000000      6.000000  35000.000000

       loan_int_rate  loan_percent_income  cb_person_default_on_file  \
count   58645.000000         58645.000000               58645.000000
mean       10.677874             0.159238                   0.148384
std         3.034697             0.091692                   0.355484
min         5.420000             0.000000                   0.000000
25%         7.880000             0.090000                   0.000000
50%        10.750000             0.140000                   0.000000
75%        12.990000             0.210000                   0.000000
max        23.220000             0.830000                   1.000000

       cb_person_cred_hist_length   loan_status
count                58645.000000  58645.000000
mean                     5.813556      0.142382
std                      4.029196      0.349445
min                      2.000000      0.000000
25%                      3.000000      0.000000
50%                      4.000000      0.000000
75%                      8.000000      0.000000
max                     30.000000      1.000000
```

[40]:

```
X_train = df_train.drop(['id','loan_status'], axis=1).values
# print(X_train[0,:])
y_train = df_train['loan_status'].values

X_test = df_test.values
```

```
[3.700e+01 3.500e+04 3.000e+00 0.000e+00 1.000e+00 1.000e+00 6.000e+03
 1.149e+01 1.700e-01 0.000e+00 1.400e+01]
```



[41]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
# from sklearn.metrics import roc_auc_score
# from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_train.drop(['id','loan_status'], axis=1))
# Create the model
model = Sequential()
model.add(Dense(4, input_dim=X_scaled.shape[1], activation='relu'))  # First␣
↪hidden layer
model.add(Dense(4, activation='relu'))  # Second hidden layer
model.add(Dense(1, activation='sigmoid'))  # Output layer

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='sgd')

# Fit the model
model.fit(X_scaled, y_train, epochs=100, batch_size=10, verbose=1)
```

```
Epoch 1/100
```

```
c:\Users\Big_Guppy\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
5865/5865           18s 3ms/step -
loss: 0.3834
Epoch 2/100
5865/5865           15s 3ms/step -
loss: 0.2877
Epoch 3/100
```

```
5865/5865              12s 2ms/step -
loss: 0.2568
Epoch 4/100
5865/5865              19s 2ms/step -
loss: 0.2385
Epoch 5/100
5865/5865              24s 2ms/step -
loss: 0.2310
Epoch 6/100
5865/5865              20s 2ms/step -
loss: 0.2287
Epoch 7/100
5865/5865              22s 3ms/step -
loss: 0.2331
Epoch 8/100
5865/5865              28s 4ms/step -
loss: 0.2254
Epoch 9/100
5865/5865              32s 2ms/step -
loss: 0.2257
Epoch 10/100
5865/5865              25s 3ms/step -
loss: 0.2250
Epoch 11/100
5865/5865              20s 3ms/step -
loss: 0.2251
Epoch 12/100
5865/5865              17s 3ms/step -
loss: 0.2177
Epoch 13/100
5865/5865              15s 3ms/step -
loss: 0.2138
Epoch 14/100
5865/5865              14s 2ms/step -
loss: 0.2164
Epoch 15/100
5865/5865              27s 3ms/step -
loss: 0.2189
Epoch 16/100
5865/5865              17s 3ms/step -
loss: 0.2163
Epoch 17/100
5865/5865              22s 3ms/step -
loss: 0.2170
Epoch 18/100
5865/5865              18s 3ms/step -
loss: 0.2098
Epoch 19/100
```

```
5865/5865           19s 3ms/step -
loss: 0.2149
Epoch 20/100
5865/5865           17s 3ms/step -
loss: 0.2143
Epoch 21/100
5865/5865           17s 3ms/step -
loss: 0.2110
Epoch 22/100
5865/5865           18s 3ms/step -
loss: 0.2086
Epoch 23/100
5865/5865           17s 2ms/step -
loss: 0.2117
Epoch 24/100
5865/5865           13s 2ms/step -
loss: 0.2108
Epoch 25/100
5865/5865           22s 2ms/step -
loss: 0.2073
Epoch 26/100
5865/5865           16s 3ms/step -
loss: 0.2088
Epoch 27/100
5865/5865           12s 2ms/step -
loss: 0.2133
Epoch 28/100
5865/5865           12s 2ms/step -
loss: 0.2111
Epoch 29/100
5865/5865           13s 2ms/step -
loss: 0.2097
Epoch 30/100
5865/5865           15s 3ms/step -
loss: 0.2120
Epoch 31/100
5865/5865           12s 2ms/step -
loss: 0.2110
Epoch 32/100
5865/5865           29s 3ms/step -
loss: 0.2086
Epoch 33/100
5865/5865           28s 5ms/step -
loss: 0.2124
Epoch 34/100
5865/5865           44s 5ms/step -
loss: 0.2064
Epoch 35/100
```

```
5865/5865          32s 5ms/step -
loss: 0.2102
Epoch 36/100
5865/5865          36s 4ms/step -
loss: 0.2127
Epoch 37/100
5865/5865          44s 5ms/step -
loss: 0.2111
Epoch 38/100
5865/5865          46s 6ms/step -
loss: 0.2094
Epoch 39/100
5865/5865          16s 3ms/step -
loss: 0.2092
Epoch 40/100
5865/5865          20s 3ms/step -
loss: 0.2133
Epoch 41/100
5865/5865          18s 3ms/step -
loss: 0.2066
Epoch 42/100
5865/5865          17s 3ms/step -
loss: 0.2134
Epoch 43/100
5865/5865          16s 3ms/step -
loss: 0.2119
Epoch 44/100
5865/5865          25s 3ms/step -
loss: 0.2134
Epoch 45/100
5865/5865          24s 4ms/step -
loss: 0.2067
Epoch 46/100
5865/5865          17s 3ms/step -
loss: 0.2081
Epoch 47/100
5865/5865          21s 4ms/step -
loss: 0.2103
Epoch 48/100
5865/5865          22s 4ms/step -
loss: 0.2084
Epoch 49/100
5865/5865          38s 3ms/step -
loss: 0.2094
Epoch 50/100
5865/5865          30s 5ms/step -
loss: 0.2078
Epoch 51/100
```

```
5865/5865            27s 5ms/step -
loss: 0.2081
Epoch 52/100
5865/5865            46s 5ms/step -
loss: 0.2089
Epoch 53/100
5865/5865            35s 4ms/step -
loss: 0.2049
Epoch 54/100
5865/5865            46s 5ms/step -
loss: 0.2076
Epoch 55/100
5865/5865            38s 4ms/step -
loss: 0.2092
Epoch 56/100
5865/5865            25s 4ms/step -
loss: 0.2055
Epoch 57/100
5865/5865            24s 4ms/step -
loss: 0.2042
Epoch 58/100
5865/5865            46s 5ms/step -
loss: 0.2059
Epoch 59/100
5865/5865            40s 4ms/step -
loss: 0.2089
Epoch 60/100
5865/5865            37s 4ms/step -
loss: 0.2082
Epoch 61/100
5865/5865            54s 6ms/step -
loss: 0.2100
Epoch 62/100
5865/5865            31s 4ms/step -
loss: 0.2061
Epoch 63/100
5865/5865            45s 5ms/step -
loss: 0.2060
Epoch 64/100
5865/5865            24s 4ms/step -
loss: 0.2118
Epoch 65/100
5865/5865            40s 7ms/step -
loss: 0.2086
Epoch 66/100
5865/5865            52s 8ms/step -
loss: 0.2117
Epoch 67/100
```

```
5865/5865              61s 5ms/step -
loss: 0.2146
Epoch 68/100
5865/5865              23s 4ms/step -
loss: 0.2100
Epoch 69/100
5865/5865              23s 4ms/step -
loss: 0.2070
Epoch 70/100
5865/5865              40s 4ms/step -
loss: 0.2049
Epoch 71/100
5865/5865              22s 4ms/step -
loss: 0.2079
Epoch 72/100
5865/5865              23s 4ms/step -
loss: 0.2085
Epoch 73/100
5865/5865              22s 4ms/step -
loss: 0.2086
Epoch 74/100
5865/5865              40s 3ms/step -
loss: 0.2058
Epoch 75/100
5865/5865              22s 4ms/step -
loss: 0.2094
Epoch 76/100
5865/5865              22s 4ms/step -
loss: 0.2060
Epoch 77/100
5865/5865              40s 3ms/step -
loss: 0.2112
Epoch 78/100
5865/5865              20s 3ms/step -
loss: 0.2063
Epoch 79/100
5865/5865              19s 3ms/step -
loss: 0.2072
Epoch 80/100
5865/5865              21s 3ms/step -
loss: 0.2099
Epoch 81/100
5865/5865              19s 3ms/step -
loss: 0.2059
Epoch 82/100
5865/5865              19s 3ms/step -
loss: 0.2097
Epoch 83/100
```

```
5865/5865              31s 5ms/step -
loss: 0.2091
Epoch 84/100
5865/5865              38s 4ms/step -
loss: 0.2108
Epoch 85/100
5865/5865              24s 4ms/step -
loss: 0.2082
Epoch 86/100
5865/5865              27s 4ms/step -
loss: 0.2072
Epoch 87/100
5865/5865              39s 4ms/step -
loss: 0.2084
Epoch 88/100
5865/5865              18s 3ms/step -
loss: 0.2123
Epoch 89/100
5865/5865              16s 2ms/step -
loss: 0.2051
Epoch 90/100
5865/5865              18s 3ms/step -
loss: 0.2058
Epoch 91/100
5865/5865              19s 3ms/step -
loss: 0.2075
Epoch 92/100
5865/5865              20s 3ms/step -
loss: 0.2075
Epoch 93/100
5865/5865              17s 3ms/step -
loss: 0.2067
Epoch 94/100
5865/5865              17s 3ms/step -
loss: 0.2107
Epoch 95/100
5865/5865              17s 3ms/step -
loss: 0.2105
Epoch 96/100
5865/5865              22s 3ms/step -
loss: 0.2093
Epoch 97/100
5865/5865              17s 3ms/step -
loss: 0.2030
Epoch 98/100
5865/5865              17s 3ms/step -
loss: 0.2064
Epoch 99/100
```

```
5865/5865              23s 3ms/step -
loss: 0.2066
Epoch 100/100
5865/5865              20s 3ms/step -
loss: 0.2093
```

[41]: <keras.src.callbacks.history.History at 0x1500617b110>

[30]:
```python
print(df_test['id'])
```

```
0        58645
1        58646
2        58647
3        58648
4        58649
         …
39093    97738
39094    97739
39095    97740
39096    97741
39097    97742
Name: id, Length: 39098, dtype: int64
```

[42]:
```python
# Assuming 'id' is the first column in the NumPy array
X_test_ids = df_test['id']  # Get the 'id' values (first column)
X_test_scaled = scaler.transform(df_test.drop('id', axis=1))  # Scale the test␣
 ↪set features

# Make predictions
y_test_pred_probs = model.predict(X_test_scaled).flatten()

# Prepare the submission DataFrame
submission_df = pd.DataFrame({
    'id': X_test_ids,
    'loan_status': y_test_pred_probs
})

# Save to CSV
submission_df.to_csv('submission.csv', index=False)
```

```
1222/1222              4s 3ms/step
```