Report

April 15, 2020

1 Default of Credit Card Clients - Predictive Models

1.1 Introduction

In this report, we will use CatBoost, XGBoost, LightGBM, and deep neural network (DNN) to predict default of credit card clients.

1.2 Dataset

This dataset is downloaded from Kaggle.

Data recource: UCI machine learning repository

This dataset contains information on default payments, demographic factors, credit data, history of payment, and bill statements of credit card clients in Taiwan from April 2005 to September 2005.

1.3 Content

There are 25 variables:

ID: ID of each client

LIMIT_BAL: Amount of given credit in NT dollars (includes individual and family/supplementary credit

SEX: Gender (1=male, 2=female)

EDUCATION: (1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown)

MARRIAGE: Marital status (1=married, 2=single, 3=others)

AGE: Age in years

PAY_0: Repayment status in September, 2005 (-1=pay duly, 1=payment delay for one month, 2=payment delay for two months, … 8=payment delay for eight months, 9=payment delay for nine months and above)

PAY_2: Repayment status in August, 2005 (scale same as above)

PAY_3: Repayment status in July, 2005 (scale same as above)

PAY_4: Repayment status in June, 2005 (scale same as above)

PAY_5: Repayment status in May, 2005 (scale same as above)

PAY_6: Repayment status in April, 2005 (scale same as above)

BILL_AMT1: Amount of bill statement in September, 2005 (NT dollar)

BILL_AMT2: Amount of bill statement in August, 2005 (NT dollar)

BILL_AMT3: Amount of bill statement in July, 2005 (NT dollar)

BILL_AMT4: Amount of bill statement in June, 2005 (NT dollar)

BILL_AMT5: Amount of bill statement in May, 2005 (NT dollar)

BILL_AMT6: Amount of bill statement in April, 2005 (NT dollar)

PAY_AMT1: Amount of previous payment in September, 2005 (NT dollar)

PAY_AMT2: Amount of previous payment in August, 2005 (NT dollar)

PAY_AMT3: Amount of previous payment in July, 2005 (NT dollar)

PAY_AMT4: Amount of previous payment in June, 2005 (NT dollar)

PAY_AMT5: Amount of previous payment in May, 2005 (NT dollar)

PAY_AMT6: Amount of previous payment in April, 2005 (NT dollar)

default.payment.next.month: Default payment (1=yes, 0=no)

2 Load packages

2.1 Load packages

[287]:
```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import gc #Garbage Collector interface
from datetime import datetime
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from catboost import CatBoostClassifier
import lightgbm as lgb
import xgboost as xgb
```

2.2 Set parameters

Here we set few parameters for the analysis and models.

[288]:
```
# RFC_METRIC = 'gini'  #metric used for RandomForrestClassifier
# NUM_ESTIMATORS = 100 #number of estimators used for RandomForrestClassifier
# NO_JOBS = 4 #number of parallel jobs used for RandomForrestClassifier

#VALIDATION
TEST_SIZE = 0.20 # using train_test_split

#CROSS-VALIDATION
NUMBER_KFOLDS = 5 #number of KFolds for cross-validation

RANDOM_STATE = 2020

MAX_ROUNDS = 1000 #lgb iterations
EARLY_STOP = 50 #lgb early stop
OPT_ROUNDS = 1000  #To be adjusted based on best validation rounds
VERBOSE_EVAL = 50 #Print out metric result
```

3 Read the data

[289]:
```
data_df = pd.read_csv("/Users/Stylewsxcde991/Desktop/Default of Credit Card␣
↪Clients/UCI_Credit_Card.csv")
```

4 Check the data

[290]:
```
print("Default Credit Card Clients data -  rows:",data_df.shape[0]," columns:",␣
↪data_df.shape[1])
```

```
Default Credit Card Clients data -  rows: 30000  columns: 25
```

4.1 Glimpse the data

We start by looking to the data features (first 5 rows).

[291]:
```
data_df.head()
```

[291]:
```
   ID  LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  PAY_0  PAY_2  PAY_3  PAY_4  \
0   1    20000.0    2          2         1   24      2      2     -1     -1
1   2   120000.0    2          2         2   26     -1      2      0      0
2   3    90000.0    2          2         2   34      0      0      0      0
```

```
3   4    50000.0     2            2          1   37       0       0       0       0
4   5    50000.0     1            2          1   57      -1       0      -1       0

   …  BILL_AMT4  BILL_AMT5  BILL_AMT6  PAY_AMT1  PAY_AMT2  PAY_AMT3  \
0  …        0.0        0.0        0.0       0.0     689.0       0.0
1  …     3272.0     3455.0     3261.0       0.0    1000.0    1000.0
2  …    14331.0    14948.0    15549.0    1518.0    1500.0    1000.0
3  …    28314.0    28959.0    29547.0    2000.0    2019.0    1200.0
4  …    20940.0    19146.0    19131.0    2000.0   36681.0   10000.0

   PAY_AMT4  PAY_AMT5  PAY_AMT6  default.payment.next.month
0       0.0       0.0       0.0                           1
1    1000.0       0.0    2000.0                           1
2    1000.0    1000.0    5000.0                           0
3    1100.0    1069.0    1000.0                           0
4    9000.0     689.0     679.0                           0

[5 rows x 25 columns]
```

Let's look into more details to the data.

[292]:
```python
data_df.describe()
```

[292]:
```
                 ID       LIMIT_BAL           SEX     EDUCATION      MARRIAGE  \
count  30000.000000    30000.000000  30000.000000  30000.000000  30000.000000
mean   15000.500000   167484.322667      1.603733      1.853133      1.551867
std     8660.398374   129747.661567      0.489129      0.790349      0.521970
min        1.000000    10000.000000      1.000000      0.000000      0.000000
25%     7500.750000    50000.000000      1.000000      1.000000      1.000000
50%    15000.500000   140000.000000      2.000000      2.000000      2.000000
75%    22500.250000   240000.000000      2.000000      2.000000      2.000000
max    30000.000000  1000000.000000      2.000000      6.000000      3.000000

                AGE         PAY_0         PAY_2         PAY_3         PAY_4  \
count  30000.000000  30000.000000  30000.000000  30000.000000  30000.000000
mean      35.485500     -0.016700     -0.133767     -0.166200     -0.220667
std        9.217904      1.123802      1.197186      1.196868      1.169139
min       21.000000     -2.000000     -2.000000     -2.000000     -2.000000
25%       28.000000     -1.000000     -1.000000     -1.000000     -1.000000
50%       34.000000      0.000000      0.000000      0.000000      0.000000
75%       41.000000      0.000000      0.000000      0.000000      0.000000
max       79.000000      8.000000      8.000000      8.000000      8.000000

       …     BILL_AMT4      BILL_AMT5      BILL_AMT6       PAY_AMT1  \
count  …  30000.000000   30000.000000   30000.000000   30000.000000
mean   …  43262.948967   40311.400967   38871.760400    5663.580500
std    …  64332.856134   60797.155770   59554.107537   16563.280354
```

```
min     … -170000.000000  -81334.000000 -339603.000000       0.000000
25%     …    2326.750000    1763.000000    1256.000000    1000.000000
50%     …   19052.000000   18104.500000   17071.000000    2100.000000
75%     …   54506.000000   50190.500000   49198.250000    5006.000000
max     …  891586.000000  927171.000000  961664.000000  873552.000000

           PAY_AMT2      PAY_AMT3       PAY_AMT4       PAY_AMT5  \
count  3.000000e+04   30000.00000   30000.000000   30000.000000
mean   5.921163e+03    5225.68150    4826.076867    4799.387633
std    2.304087e+04   17606.96147   15666.159744   15278.305679
min    0.000000e+00       0.00000       0.000000       0.000000
25%    8.330000e+02     390.00000     296.000000     252.500000
50%    2.009000e+03    1800.00000    1500.000000    1500.000000
75%    5.000000e+03    4505.00000    4013.250000    4031.500000
max    1.684259e+06  896040.00000  621000.000000  426529.000000

            PAY_AMT6  default.payment.next.month
count   30000.000000                30000.000000
mean     5215.502567                    0.221200
std     17777.465775                    0.415062
min         0.000000                    0.000000
25%       117.750000                    0.000000
50%      1500.000000                    0.000000
75%      4000.000000                    0.000000
max    528666.000000                    1.000000

[8 rows x 25 columns]
```

There are 30,000 distinct credit card clients.

As the value 0 for default payment means 'not default' and value 1 means 'default', the mean of 0.221 means that there are 22.1% of credit card contracts that will default next month (will verify this in the next sections of this analysis).

4.2 Checking missing data

Let's check if there is any missing data.

[293]:
```
data_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 25 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   ID                          30000 non-null  int64
 1   LIMIT_BAL                   30000 non-null  float64
 2   SEX                         30000 non-null  int64
```

```
 3   EDUCATION                   30000 non-null  int64
 4   MARRIAGE                    30000 non-null  int64
 5   AGE                         30000 non-null  int64
 6   PAY_0                       30000 non-null  int64
 7   PAY_2                       30000 non-null  int64
 8   PAY_3                       30000 non-null  int64
 9   PAY_4                       30000 non-null  int64
 10  PAY_5                       30000 non-null  int64
 11  PAY_6                       30000 non-null  int64
 12  BILL_AMT1                   30000 non-null  float64
 13  BILL_AMT2                   30000 non-null  float64
 14  BILL_AMT3                   30000 non-null  float64
 15  BILL_AMT4                   30000 non-null  float64
 16  BILL_AMT5                   30000 non-null  float64
 17  BILL_AMT6                   30000 non-null  float64
 18  PAY_AMT1                    30000 non-null  float64
 19  PAY_AMT2                    30000 non-null  float64
 20  PAY_AMT3                    30000 non-null  float64
 21  PAY_AMT4                    30000 non-null  float64
 22  PAY_AMT5                    30000 non-null  float64
 23  PAY_AMT6                    30000 non-null  float64
 24  default.payment.next.month  30000 non-null  int64
dtypes: float64(13), int64(12)
memory usage: 5.7 MB
```

By the above information table, we find that all columns have 30000 observations. That is to say, there is no missing data problem.

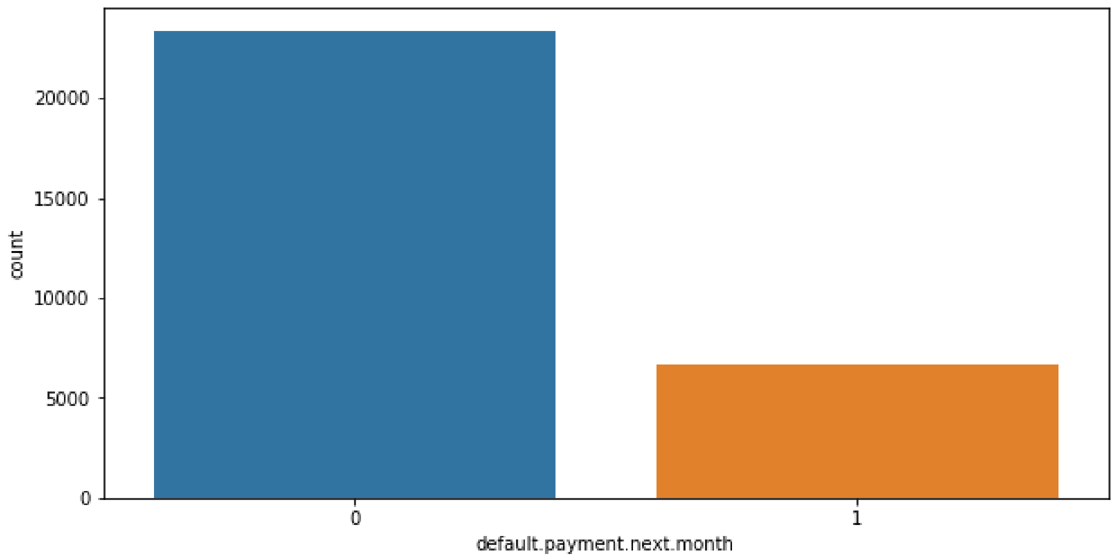
4.3 Data unbalance

Let's check data unbalance with respect with target value, i.e. default.payment.next.month.

[294]:
```
# The proportion of each target class

Not_Default,Default = data_df["default.payment.next.month"].value_counts()
print(f'Not_Default {Not_Default}')
print(f'Default {Default}')
print(f'Default proportion {round((100*Default/(Default+Not_Default)),2)}%')
plt.figure(figsize=(10,5))
sns.countplot(data_df['default.payment.next.month'])
```

```
Not_Default 23364
Default 6636
Default proportion 22.12%
```

[294]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x13664ed90>
```



According to above information, a number of 6,636 out of 30,000 (or 22%) of clients will default next month.

The data has a unbalance with respect of the target value (default.payment.next.month).

This might be a problem for our DNN models.

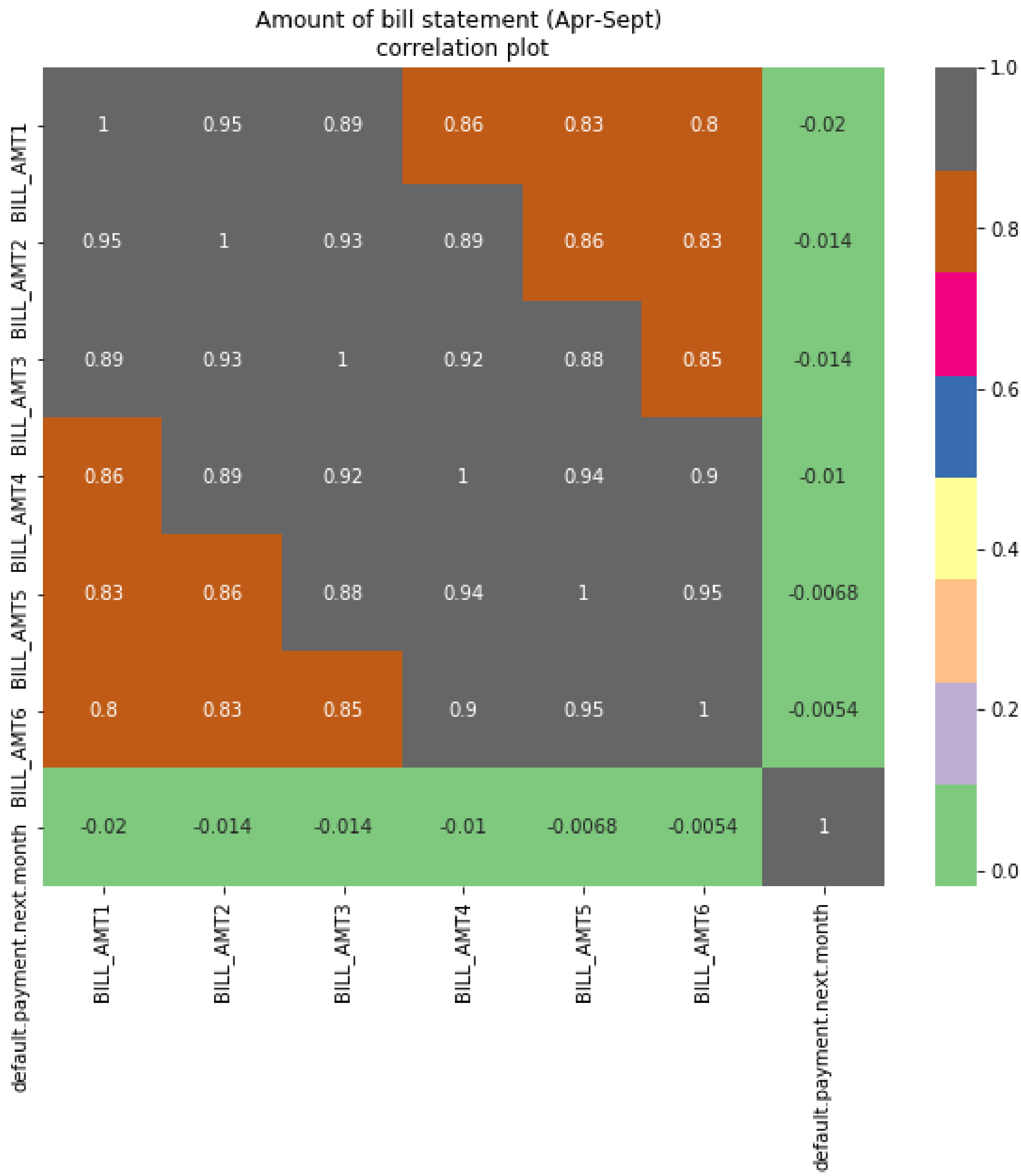
5 Explosive Data exploration

5.1 Features correlation

For the numeric values, let's represent the features correlation.

Let's check the correlation of Amount of bill statement in April - September 2005.

[513]:
```
var =␣
↪['BILL_AMT1','BILL_AMT2','BILL_AMT3','BILL_AMT4','BILL_AMT5','BILL_AMT6','default.
↪payment.next.month']

plt.figure(figsize=(10,8))
sns.heatmap(data_df[var].corr(),cmap='Accent',annot=True)
plt.title('Amount of bill statement (Apr-Sept) \ncorrelation plot')
plt.show()
```

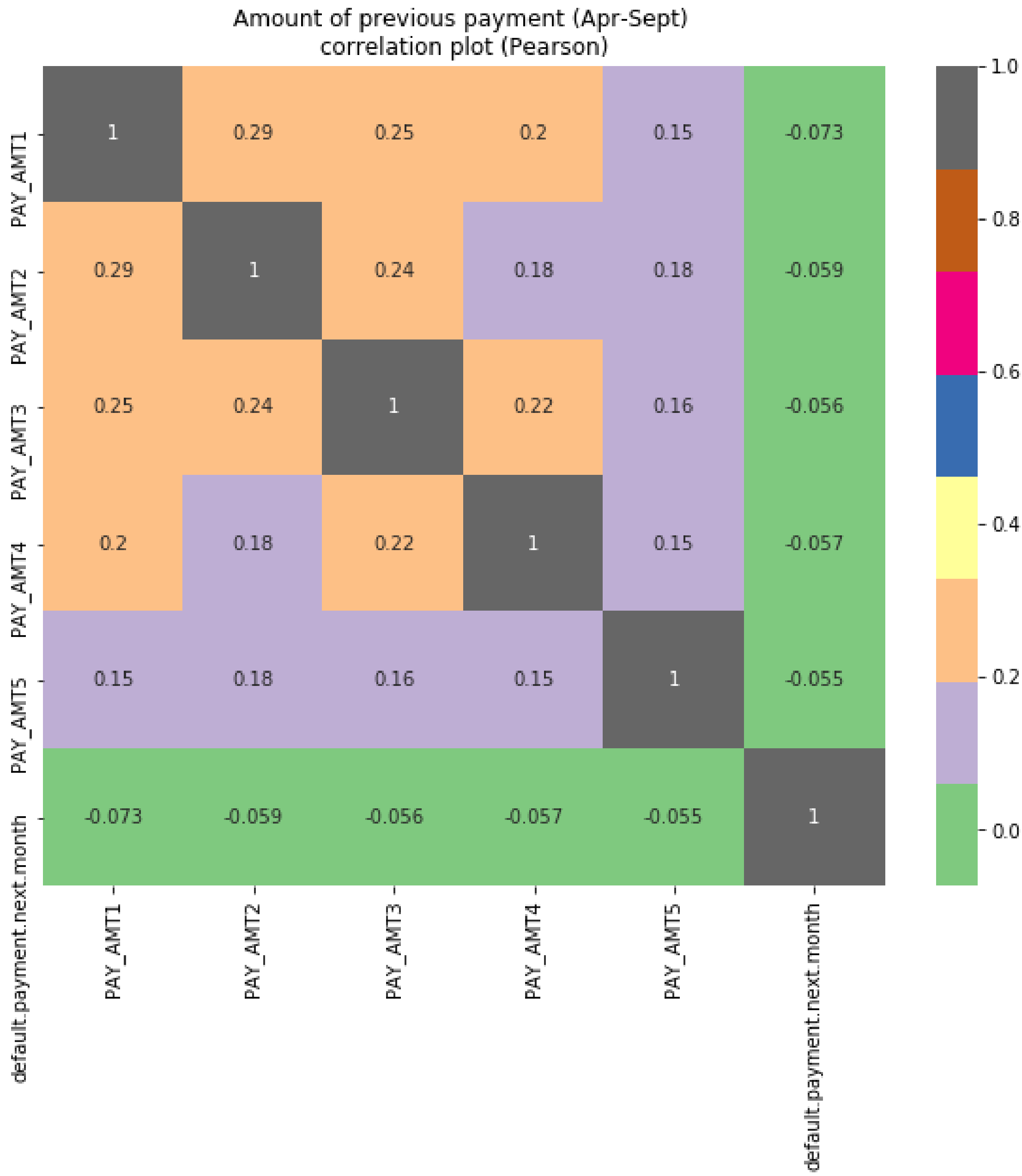


Correlation is decreasing with distance between months. Lowest correlations are between Sept-April.

Let's check the correlation of Amount of previous payment in April - September 2005.

[512]:
```
var = ['PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5','default.
↪payment.next.month']

plt.figure(figsize=(10,8))
```

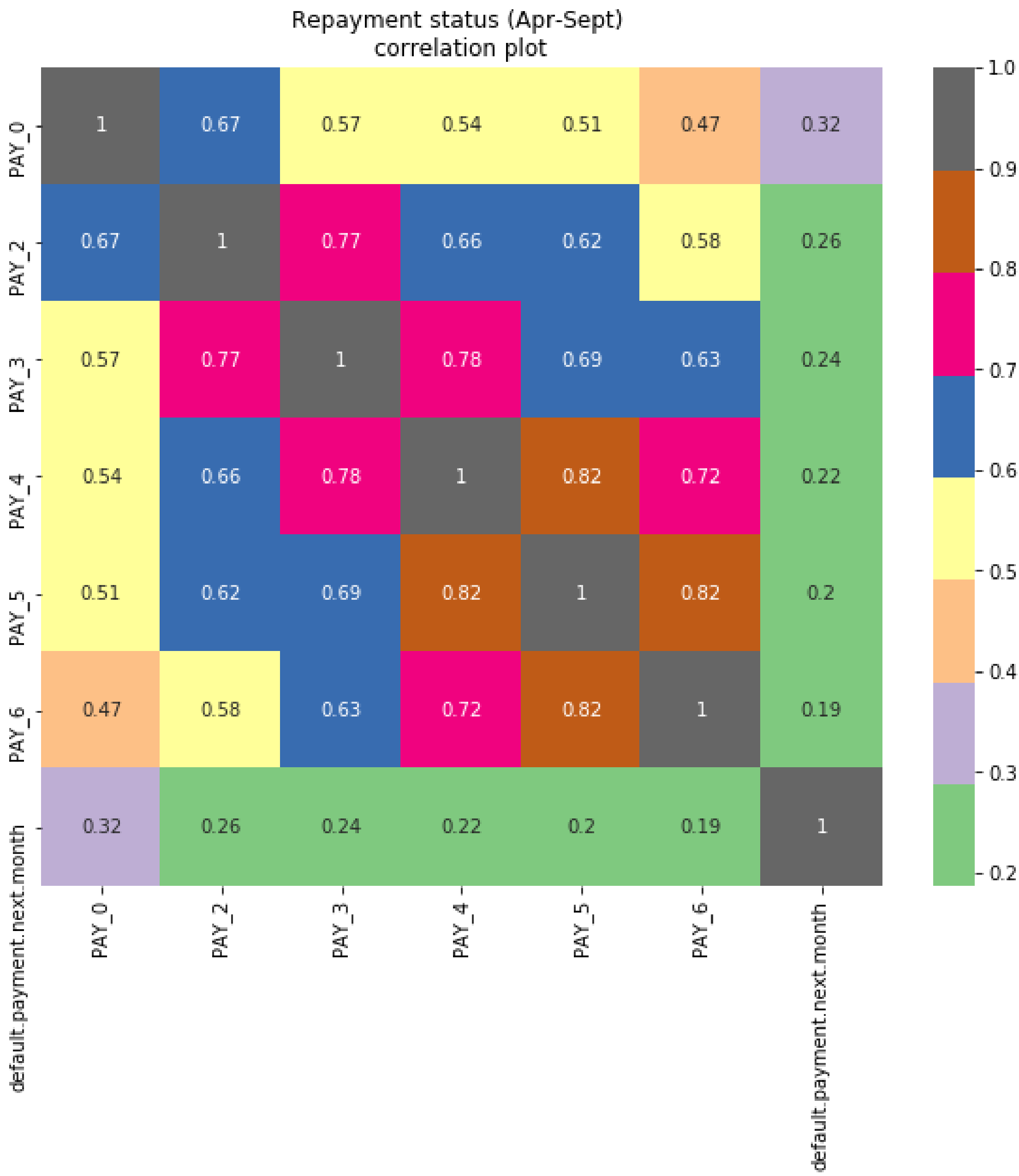
```
sns.heatmap(data_df[var].corr(),cmap='Accent',annot=True)
plt.title('Amount of previous payment (Apr-Sept) \ncorrelation plot (Pearson)')
plt.show()
```



There are no correlations between amounts of previous payments for April-Sept 2005.

Let's check the correlation between Repayment status in April - September 2005.

[514]:
```python
var = ['PAY_0','PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6','default.payment.
↪next.month']

plt.figure(figsize=(10,8))
sns.heatmap(data_df[var].corr(),cmap='Accent',annot=True)
plt.title('Repayment status (Apr-Sept) \ncorrelation plot')
plt.show()
```



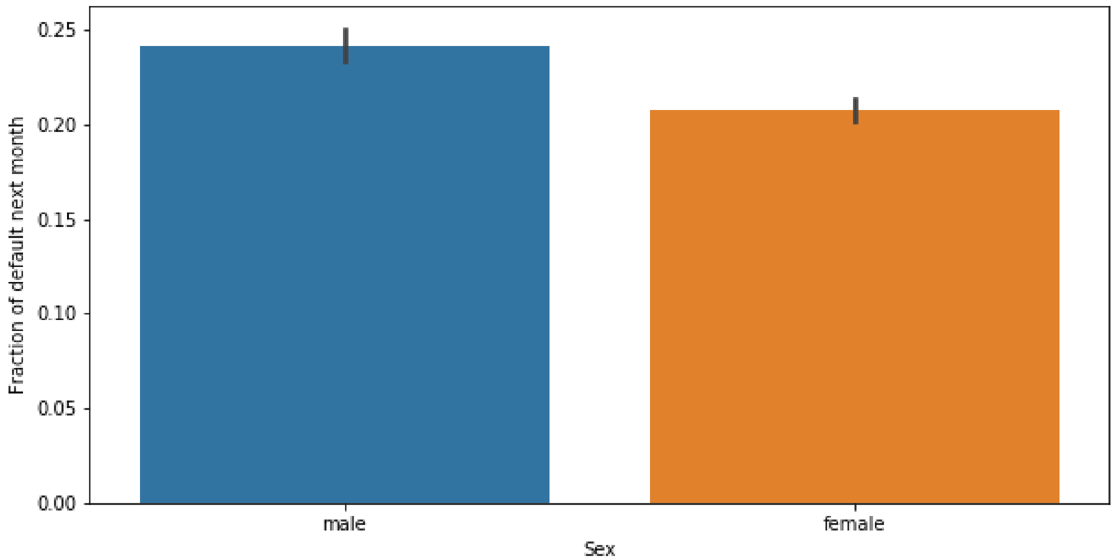
Correlation is decreasing with distance between months. Lowest correlations are between Sept-

April.

5.2 Sex

[298]:
```
plt.figure(figsize=(10,5))
ax = sns.barplot(x='SEX',y='default.payment.next.month',data=data_df)
ax.set(ylabel='Fraction of default next month')
ax.set(xlabel='Sex')
ax.set_xticklabels(['male','female'])
```

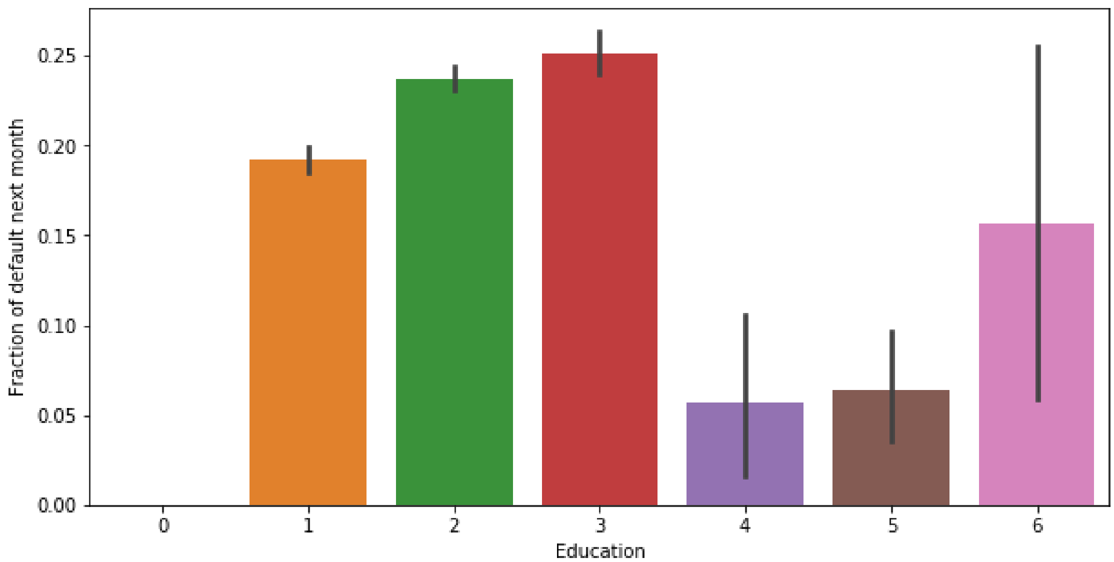
[298]:
```
[Text(0, 0, 'male'), Text(0, 0, 'female')]
```



5.3 Education

[299]:
```
plt.figure(figsize=(10,5))
ax = sns.barplot(x='EDUCATION',y='default.payment.next.month',data=data_df)
ax.set(ylabel='Fraction of default next month')
ax.set(xlabel='Education')
```

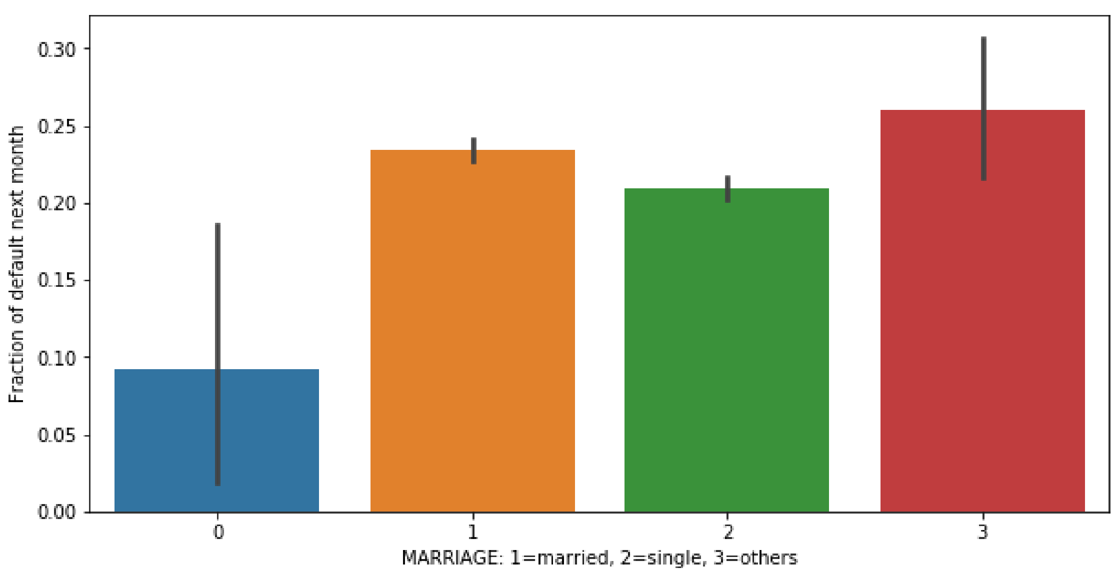
[299]:
```
[Text(0.5, 0, 'Education')]
```



5.4 Marriage

[300]:
```
plt.figure(figsize=(10,5))
ax = sns.barplot(x='MARRIAGE',y='default.payment.next.month',data=data_df)
ax.set(ylabel='Fraction of default next month')
ax.set(xlabel='MARRIAGE: 1=married, 2=single, 3=others')
```

[300]:
```
[Text(0.5, 0, 'MARRIAGE: 1=married, 2=single, 3=others')]
```



5.5 Training set, validation set, and testing set.

Let's define training set, validation set, and testing set.

[301]:
```
# Split data_df into training set and testing set
train_df, test_df = train_test_split(data_df, test_size=TEST_SIZE,␣
↪random_state=RANDOM_STATE, shuffle=True )

# Split training set into partial training set and validation set.
par_train_df, val_df = train_test_split(train_df,test_size=TEST_SIZE,␣
↪random_state=RANDOM_STATE, shuffle=True )
```

5.6 Define predictors and target values

Let's define the predictor features and the target features. Categorical features, if any, are also defined. In our case, there are no categorical feature.

[302]:
```
target = 'default.payment.next.month'
predictors = ['LIMIT_BAL', 'AGE', 'PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5',
       'PAY_6', 'BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4',
       'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3',
       'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6',
       'EDUCATION', 'SEX',
       'MARRIAGE']

# Define categorical features for catboost and LightGBM
categorical_features =['EDUCATION', 'SEX','MARRIAGE',
                       'PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5','PAY_6',]
```

[303]:
```
# The whole training set
train_X = train_df[predictors]
train_Y = train_df[target]

# Partial training set
par_train_X = par_train_df[predictors]
par_train_Y = par_train_df[target]

# Validation set
val_X = val_df[predictors]
val_Y = val_df[target]

# Test set
test_X = test_df[predictors]
test_Y = test_df[target]
```

5.7 One-hot encoder

For xgboost and DNN model, we have use one-hot encoding to encode some categorical variables.

Let's look at columns of our dataset.

[304]:
```
data_df.columns
```

[304]:
```
Index(['ID', 'LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_0',
       'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1', 'BILL_AMT2',
       'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1',
       'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6',
       'default.payment.next.month'],
      dtype='object')
```

There are some categorical features in our data: SEX, EDUCATION, MARRIAGE, AGE, PAY_0, PAY_2, PAY_3, PAY_4, PAY_5, PAY_6

Here are their definitions:

SEX: Gender (1=male, 2=female)

EDUCATION: (1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown)

MARRIAGE: Marital status (1=married, 2=single, 3=others)

PAY_0: Repayment status in September, 2005 (-1=pay duly, 1=payment delay for one month, 2=payment delay for two months, … 8=payment delay for eight months, 9=payment delay for nine months and above)

PAY_2: Repayment status in August, 2005 (scale same as above)

PAY_3: Repayment status in July, 2005 (scale same as above)

PAY_4: Repayment status in June, 2005 (scale same as above)

PAY_5: Repayment status in May, 2005 (scale same as above)

PAY_6: Repayment status in April, 2005 (scale same as above)

For PAY_0, PAY_2, PAY_3, PAY_4, PAY_5, PAY_6, these categorical features are ordinal. So label encoding is suitable for these features.

For SEX, EDUCATION, MARRIAGE, I will use one-hot encoding.

[305]:
```
OH_features = ['EDUCATION', 'SEX', 'MARRIAGE']

OH_train_X = pd.get_dummies(train_X, columns = OH_features)
OH_par_train_X = pd.get_dummies(par_train_X, columns = OH_features)
OH_val_X = pd.get_dummies(val_X, columns = OH_features)
OH_test_X = pd.get_dummies(test_X, columns = OH_features)
```

[306]:
```
OH_train_X.columns
```

[306]:

```
Index(['LIMIT_BAL', 'AGE', 'PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5',
       'PAY_6', 'BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4',
       'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3',
       'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6', 'EDUCATION_0', 'EDUCATION_1',
       'EDUCATION_2', 'EDUCATION_3', 'EDUCATION_4', 'EDUCATION_5',
       'EDUCATION_6', 'SEX_1', 'SEX_2', 'MARRIAGE_0', 'MARRIAGE_1',
       'MARRIAGE_2', 'MARRIAGE_3'],
      dtype='object')
```

[307]:

```
OH_predictors = ['LIMIT_BAL', 'AGE', 'PAY_0', 'PAY_2', 'PAY_3', 'PAY_4',␣
 ↪'PAY_5',
       'PAY_6', 'BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4',
       'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3',
       'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6', 'EDUCATION_0', 'EDUCATION_1',
       'EDUCATION_2', 'EDUCATION_3', 'EDUCATION_4', 'EDUCATION_5',
       'EDUCATION_6', 'SEX_1', 'SEX_2', 'MARRIAGE_0', 'MARRIAGE_1',
       'MARRIAGE_2', 'MARRIAGE_3']
```

6 Predictive models

6.1 Choosing a measure of success: metric

I will use some common metric in this report: accuracy, recall, precision, F1 score.

I use recall, precision, F1 score because our target class is some kind of imbalance.

[308]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score #The F-Measure is a popular metric for␣
 ↪imbalanced classification.
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
```

7 CatBoostClassifier

CatBoostClassifier is a gradient boosting for decision trees algorithm with support for handling categorical data

7.1 Prepare the model

Let's set the parameters for the model and initialize the model.

[262]:
```
clf = CatBoostClassifier(iterations=500,
                             learning_rate=0.02,
                             depth=12,
                             eval_metric='F1',
                             random_seed = RANDOM_STATE,
                             bagging_temperature = 0.2,
                             od_type='Iter',
                             metric_period = VERBOSE_EVAL, #
                             od_wait=100)
```

7.2 Fit CatBoost

Note that we use the **"categorical_feature"** in our fitting process.

[263]:
```
clf.fit(par_train_X, par_train_Y.
 ↪values,cat_features=categorical_features,eval_set=(val_X,val_Y),verbose=True)
```

```
Warning: Overfitting detector is active, thus evaluation metric is calculated on
every iteration. 'metric_period' is ignored for evaluation metric.

0:      learn: 0.4308878        test: 0.4436137 best: 0.4436137 (0)     total:
63.1ms   remaining: 31.5s
50:     learn: 0.5020447        test: 0.4472362 best: 0.4487744 (48)    total:
23.9s    remaining: 3m 30s
100:    learn: 0.5554865        test: 0.4542626 best: 0.4542626 (100)   total:
56.8s    remaining: 3m 44s
150:    learn: 0.5907186        test: 0.4462399 best: 0.4548287 (110)   total:
1m 22s   remaining: 3m 9s
200:    learn: 0.6084542        test: 0.4491055 best: 0.4548287 (110)   total:
1m 49s   remaining: 2m 45s
Stopped by overfitting detector  (100 iterations wait)

bestTest = 0.4548286604
bestIteration = 110

Shrink model to first 111 iterations.
```

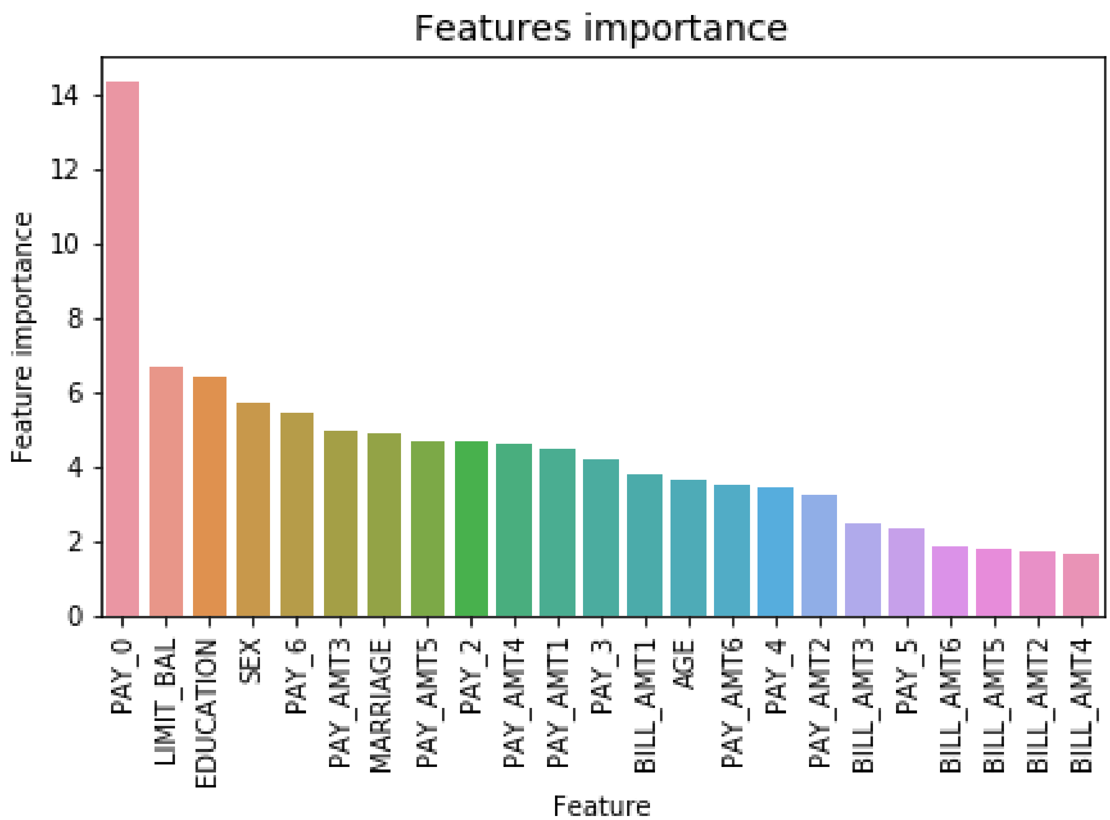
[263]: <catboost.core.CatBoostClassifier at 0x13f550650>

7.3 Features importance

Let's see the features importance.

[ ]:
```
tmp = pd.DataFrame({'Feature': predictors, 'Feature importance': clf.
 ↪feature_importances_})
tmp = tmp.sort_values(by='Feature importance',ascending=False)
```

```
plt.figure(figsize = (7,4))
plt.title('Features importance',fontsize=14)
s = sns.barplot(x='Feature',y='Feature importance',data=tmp)
s.set_xticklabels(s.get_xticklabels(),rotation=90)
plt.show()
```



7.4 Predict the target values

Let's now predict the **target** values for the **val_df** data, using predict function.

[233]:
```
# Use the whole training data to trian our model
clf = CatBoostClassifier(iterations=500,
                             learning_rate=0.02,
                             depth=12,
                             eval_metric='F1',
                             random_seed = RANDOM_STATE,
                             bagging_temperature = 0.2,
                             od_type='Iter',
                             metric_period = VERBOSE_EVAL, #
                             od_wait=100)
```

```
clf.fit(train_X, train_Y.values,eval_set=(val_X,val_Y),verbose=True)

# predict
preds_Y = clf.predict(test_X)
```

```
Warning: Overfitting detector is active, thus evaluation metric is calculated on
every iteration. 'metric_period' is ignored for evaluation metric.
0:      learn: 0.4706761        test: 0.4577861 best: 0.4577861 (0)     total:
421ms    remaining: 3m 30s
50:     learn: 0.5065732        test: 0.4962406 best: 0.4962406 (50)    total:
19s      remaining: 2m 47s
100:    learn: 0.5423010        test: 0.5321674 best: 0.5321674 (100)   total:
37.3s    remaining: 2m 27s
150:    learn: 0.5755021        test: 0.5673671 best: 0.5682520 (146)   total:
55.8s    remaining: 2m 9s
200:    learn: 0.5968457        test: 0.5888138 best: 0.5888138 (200)   total:
1m 15s   remaining: 1m 51s
250:    learn: 0.6179817        test: 0.6107711 best: 0.6107711 (250)   total:
1m 34s   remaining: 1m 34s
300:    learn: 0.6360871        test: 0.6252285 best: 0.6252285 (300)   total:
1m 54s   remaining: 1m 15s
350:    learn: 0.6526367        test: 0.6363636 best: 0.6363636 (350)   total:
2m 13s   remaining: 56.8s
400:    learn: 0.6730608        test: 0.6598802 best: 0.6598802 (399)   total:
2m 32s   remaining: 37.8s
450:    learn: 0.6936595        test: 0.6769231 best: 0.6769231 (450)   total:
2m 52s   remaining: 18.8s
499:    learn: 0.7113826        test: 0.6996508 best: 0.7012231 (495)   total:
3m 12s   remaining: 0us

bestTest = 0.7012230635
bestIteration = 495

Shrink model to first 496 iterations.
```

7.5 Confusion matrix, accuracy, precision, recall, F1 score

[234]:
```
accuracy = accuracy_score(test_Y, preds_Y)
precision = precision_score(test_Y, preds_Y)
recall = recall_score(test_Y, preds_Y)
f1 = f1_score(test_Y, preds_Y)
c_matrix = confusion_matrix(test_Y, preds_Y)
print('accuracy: '+str(accuracy)+'\n')
print('precision: '+str(precision)+'\n')
print('recall: '+str(recall)+'\n')
```

```
print('F1 score: '+str(f1)+'\n')
print('Confusion matrix: ')
print(c_matrix)
```

```
accuracy: 0.8221666666666667

precision: 0.7217261904761905

recall: 0.3553113553113553

F1 score: 0.47619047619047616

Confusion matrix:
[[4448  187]
 [ 880  485]]
```

8 XGBoostClassifier

XGBoostClassifier is a gradient boosting for decision trees algorithm with support for handling categorical data

8.1 Prepare and fit the model

[270]:
```
clf = xgb.XGBClassifier(learning_rate=0.02)
clf.fit(par_train_X,␣
↪par_train_Y,eval_set=[(par_train_X,par_train_Y),(val_X,val_Y)],early_stopping_rounds=10,␣
↪verbose=True)
```

```
[0]	validation_0-error:0.16969	validation_1-error:0.18396
Multiple eval metrics have been passed: 'validation_1-error' will be used for
early stopping.

Will train until validation_1-error hasn't improved in 10 rounds.
[1]	validation_0-error:0.16974	validation_1-error:0.18396
[2]	validation_0-error:0.16964	validation_1-error:0.18396
[3]	validation_0-error:0.16984	validation_1-error:0.18396
[4]	validation_0-error:0.16990	validation_1-error:0.18396
[5]	validation_0-error:0.16990	validation_1-error:0.18396
[6]	validation_0-error:0.16906	validation_1-error:0.18354
[7]	validation_0-error:0.16953	validation_1-error:0.18354
[8]	validation_0-error:0.16891	validation_1-error:0.18333
[9]	validation_0-error:0.16891	validation_1-error:0.18333
[10]	validation_0-error:0.16885	validation_1-error:0.18333
[11]	validation_0-error:0.16875	validation_1-error:0.18313
[12]	validation_0-error:0.16870	validation_1-error:0.18313
```

```
[13]	validation_0-error:0.16875	validation_1-error:0.18271
[14]	validation_0-error:0.16859	validation_1-error:0.18354
[15]	validation_0-error:0.16818	validation_1-error:0.18438
[16]	validation_0-error:0.16854	validation_1-error:0.18458
[17]	validation_0-error:0.16844	validation_1-error:0.18479
[18]	validation_0-error:0.16849	validation_1-error:0.18354
[19]	validation_0-error:0.16823	validation_1-error:0.18375
[20]	validation_0-error:0.16771	validation_1-error:0.18354
[21]	validation_0-error:0.16807	validation_1-error:0.18354
[22]	validation_0-error:0.16781	validation_1-error:0.18333
[23]	validation_0-error:0.16771	validation_1-error:0.18354
Stopping. Best iteration:
[13]	validation_0-error:0.16875	validation_1-error:0.18271
```

[270]:
```
XGBClassifier(base_score=0.5, booster=None, colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints=None,
              learning_rate=0.02, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=0, num_parallel_tree=1,
              objective='binary:logistic', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method=None,
              validate_parameters=False, verbosity=None)
```

8.2 Features importance

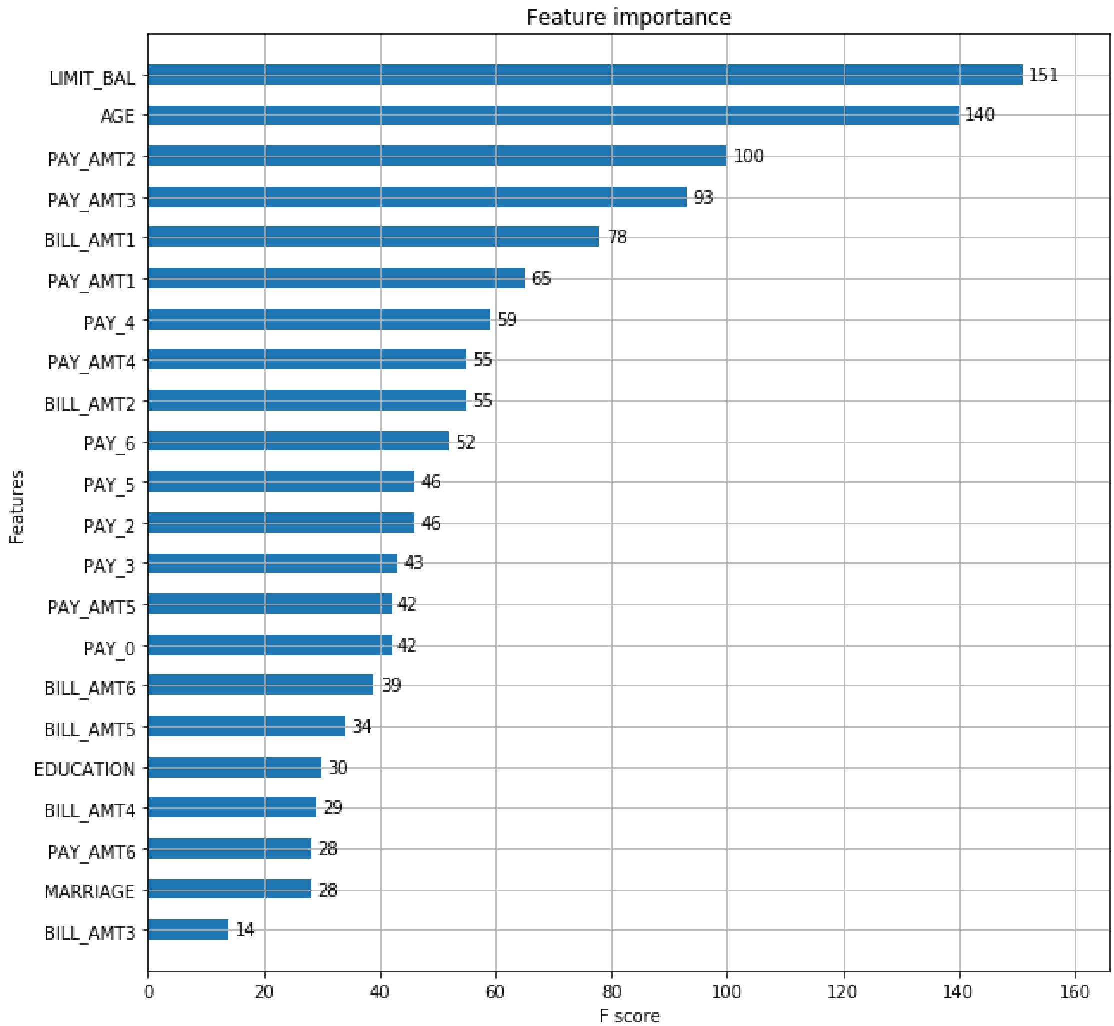
Let's see the features importance.

[271]:
```
clf.feature_importances_
```

[271]:
```
array([0.00948596, 0.00432746, 0.7013907 , 0.11981683, 0.01353244,
       0.01157393, 0.02505312, 0.01070786, 0.01198352, 0.00545711,
       0.00460776, 0.00524404, 0.00365825, 0.005003  , 0.00591743,
       0.00847558, 0.02381752, 0.00598348, 0.00435658, 0.00531497,
       0.00706675, 0.        , 0.00722563], dtype=float32)
```

[274]:
```
fig,ax = plt.subplots(figsize=(10,10))
xgb.plot_importance(clf,height=0.5,ax=ax)
```

[274]: <matplotlib.axes._subplots.AxesSubplot at 0x137398450>



8.3 Predict the target values

[269]:
```
# Use the whole data set to train our model
clf.fit(OH_train_X, train_Y)

# predict the results
preds_Y=clf.predict(OH_test_X)
```

8.4 Confusion matrix, accuracy, precision, recall, F1 score

[251]:
```
preds_Y=(preds_Y>0.5).astype(int)

accuracy = accuracy_score(test_Y, preds_Y)
precision = precision_score(test_Y, preds_Y)
recall = recall_score(test_Y, preds_Y)
f1 = f1_score(test_Y, preds_Y)
c_matrix = confusion_matrix(test_Y, preds_Y)
print('accuracy: '+str(accuracy)+'\n')
print('precision: '+str(precision)+'\n')
print('recall: '+str(recall)+'\n')
print('F1 score: '+str(f1)+'\n')
print('Confusion matrix: ')
print(c_matrix)
```

```
accuracy: 0.8188333333333333

precision: 0.7044117647058824

recall: 0.3509157509157509

F1 score: 0.4684596577017115

Confusion matrix:
[[4434  201]
 [ 886  479]]
```

9 LightBGM Classifier

Let's continue with LightGBM classifier.

9.1 Prepare and fit the model

Note that we use the "categorical_feature" parameter.

[276]:
```
clf = lgb.LGBMClassifier()
clf.fit(par_train_X,␣
↪par_train_Y,eval_set=[(par_train_X,par_train_Y),(val_X,val_Y)],
        early_stopping_rounds=10,␣
↪verbose=True,categorical_feature=categorical_features)
```

```
/opt/anaconda3/lib/python3.7/site-packages/lightgbm/basic.py:1295: UserWarning:
categorical_feature in Dataset is overridden.
New categorical_feature is ['EDUCATION', 'MARRIAGE', 'PAY_0', 'PAY_2', 'PAY_3',
```

```
'PAY_4', 'PAY_5', 'PAY_6', 'SEX']
  'New categorical_feature is {}'.format(sorted(list(categorical_feature))))
```

```
[1]     training's binary_logloss: 0.50461      valid_1's binary_logloss:
0.510877
Training until validation scores don't improve for 10 rounds
[2]     training's binary_logloss: 0.489156     valid_1's binary_logloss:
0.496524
[3]     training's binary_logloss: 0.476756     valid_1's binary_logloss:
0.485603
[4]     training's binary_logloss: 0.466744     valid_1's binary_logloss:
0.476619
[5]     training's binary_logloss: 0.458496     valid_1's binary_logloss:
0.469308
[6]     training's binary_logloss: 0.451498     valid_1's binary_logloss:
0.463626
[7]     training's binary_logloss: 0.445647     valid_1's binary_logloss:
0.458814
[8]     training's binary_logloss: 0.440528     valid_1's binary_logloss:
0.454839
[9]     training's binary_logloss: 0.436143     valid_1's binary_logloss:
0.451608
[10]    training's binary_logloss: 0.432177     valid_1's binary_logloss:
0.449173
[11]    training's binary_logloss: 0.428814     valid_1's binary_logloss:
0.447153
[12]    training's binary_logloss: 0.425591     valid_1's binary_logloss:
0.444943
[13]    training's binary_logloss: 0.422904     valid_1's binary_logloss:
0.443788
[14]    training's binary_logloss: 0.420429     valid_1's binary_logloss:
0.442844
[15]    training's binary_logloss: 0.417866     valid_1's binary_logloss:
0.441797
[16]    training's binary_logloss: 0.415732     valid_1's binary_logloss:
0.440961
[17]    training's binary_logloss: 0.413664     valid_1's binary_logloss:
0.440025
[18]    training's binary_logloss: 0.411787     valid_1's binary_logloss:
0.439451
[19]    training's binary_logloss: 0.409948     valid_1's binary_logloss:
0.438978
[20]    training's binary_logloss: 0.408402     valid_1's binary_logloss:
0.438813
[21]    training's binary_logloss: 0.406766     valid_1's binary_logloss:
0.438309
[22]    training's binary_logloss: 0.405281     valid_1's binary_logloss:
0.438013
```

```
[23]	training's binary_logloss: 0.40386	valid_1's binary_logloss:
0.438005
[24]	training's binary_logloss: 0.40251	valid_1's binary_logloss:
0.437931
[25]	training's binary_logloss: 0.401319	valid_1's binary_logloss:
0.437557
[26]	training's binary_logloss: 0.400068	valid_1's binary_logloss:
0.437566
[27]	training's binary_logloss: 0.398946	valid_1's binary_logloss:
0.437893
[28]	training's binary_logloss: 0.397783	valid_1's binary_logloss:
0.437794
[29]	training's binary_logloss: 0.396551	valid_1's binary_logloss:
0.437648
[30]	training's binary_logloss: 0.395443	valid_1's binary_logloss:
0.437877
[31]	training's binary_logloss: 0.394245	valid_1's binary_logloss:
0.438218
[32]	training's binary_logloss: 0.393162	valid_1's binary_logloss:
0.438194
[33]	training's binary_logloss: 0.392254	valid_1's binary_logloss:
0.438137
[34]	training's binary_logloss: 0.391339	valid_1's binary_logloss:
0.438082
[35]	training's binary_logloss: 0.390402	valid_1's binary_logloss:
0.437898
Early stopping, best iteration is:
[25]	training's binary_logloss: 0.401319	valid_1's binary_logloss:
0.437557
```

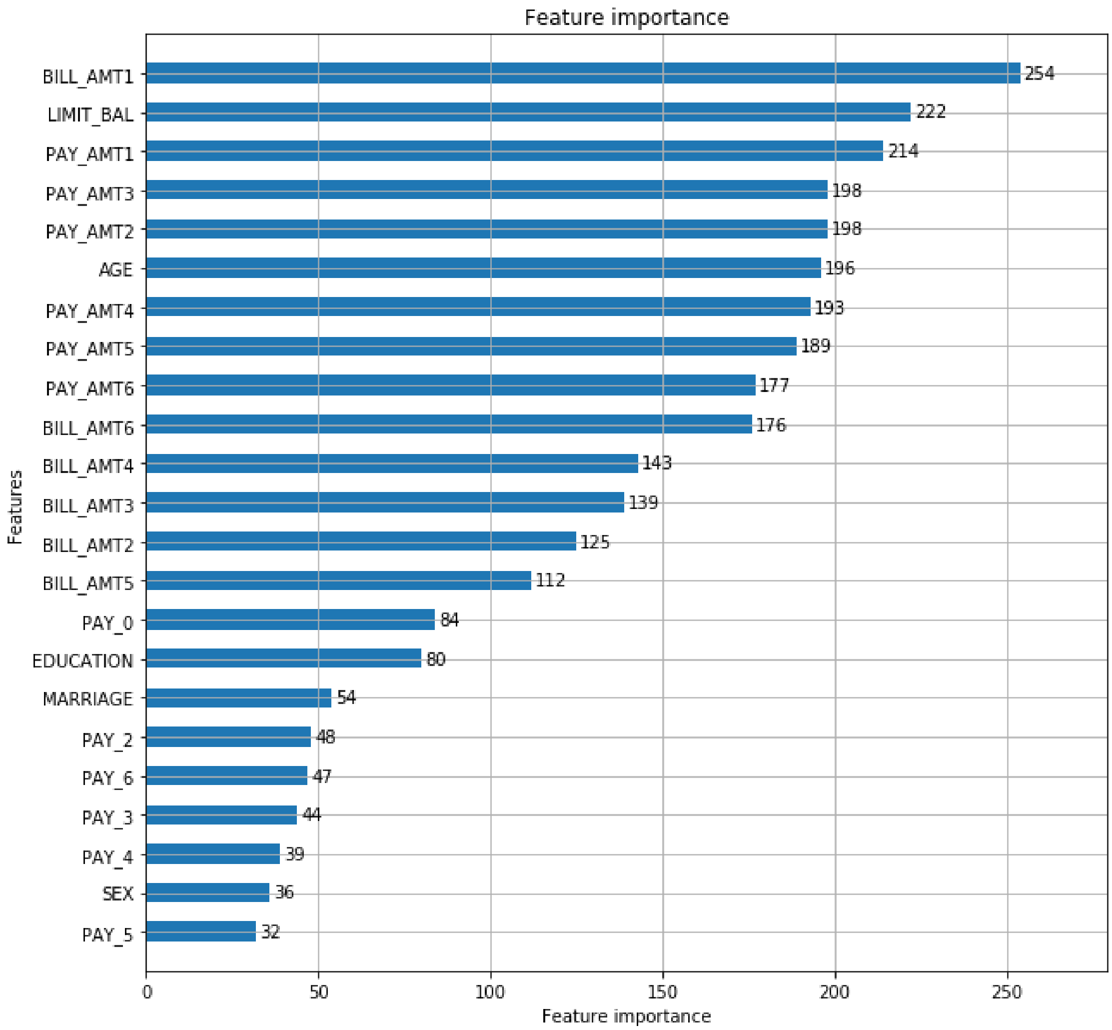
[276]:
```
LGBMClassifier(boosting_type='gbdt', class_weight=None, colsample_bytree=1.0,
               importance_type='split', learning_rate=0.1, max_depth=-1,
               min_child_samples=20, min_child_weight=0.001, min_split_gain=0.0,
               n_estimators=100, n_jobs=-1, num_leaves=31, objective=None,
               random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=True,
               subsample=1.0, subsample_for_bin=200000, subsample_freq=0)
```

9.2 Features importance

Let's see the features importance.

[281]:
```python
fig,ax = plt.subplots(figsize=(10,10))
lgb.plot_importance(clf,height=0.5,ax=ax)
```

[281]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x141706310>
```



9.3 Predict the target values

[279]:
```
# Use the whole data set to train our model
clf = lgb.LGBMClassifier()
clf.fit(train_X, train_Y)

# predict the results
preds_Y=clf.predict(test_X)
```

9.4 Confusion matrix, accuracy, precision, recall, F1 score

[280]:
```
preds_Y=(preds_Y>0.5).astype(int)

accuracy = accuracy_score(test_Y, preds_Y)
precision = precision_score(test_Y, preds_Y)
recall = recall_score(test_Y, preds_Y)
f1 = f1_score(test_Y, preds_Y)
c_matrix = confusion_matrix(test_Y, preds_Y)
print('accuracy: '+str(accuracy)+'\n')
print('precision: '+str(precision)+'\n')
print('recall: '+str(recall)+'\n')
print('F1 score: '+str(f1)+'\n')
print('Confusion matrix: ')
print(c_matrix)
```

```
accuracy: 0.825

precision: 0.7292576419213974

recall: 0.367032967032967

F1 score: 0.48830409356725146

Confusion matrix:
[[4449  186]
 [ 864  501]]
```

10 DNN model

One thing to note is that, we use the whole training data and the "validation_split" parameter when we fit our model. We use "validation_split" to automatically create our validation set.

[384]:
```
from keras import models
from keras import layers
```

10.1 Data scaling: Standardization

I use data scaling in order to improve the stability and performance of DNN models.

Reference: https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/

Remark: in general, in tree-based models the scale of the features does not matter.

Reference: https://datascience.stackexchange.com/questions/22036/how-does-lightgbm-deal-with-value-scale

Reference: https://datascience.stackexchange.com/questions/16225/would-you-recommend-feature-normalization-when-using-boosting-trees

Now, let's scale our training set and test set.

Reference: https://datascience.stackexchange.com/questions/39932/feature-scaling-both-training-and-test-data

[385]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler().fit(OH_train_X) # Use the whole train set to fit our␣
 ↪standardscaler.

OH_train_X = sc.transform(OH_train_X)
OH_test_X = sc.transform(OH_test_X)

# Transform dataframe to array, so we can use our DNN models.
OH_train_X = np.asarray(OH_train_X)
OH_test_X = np.asarray(OH_test_X)
train_Y = np.asarray(train_Y)
```

10.2 Calculated class weight

[386]:
```
data_df["default.payment.next.month"].value_counts()
```

[386]:
```
0    23364
1     6636
Name: default.payment.next.month, dtype: int64
```

[387]:
```
# Calculate class weight
NotDefault = 23364
Default = 6636
total_count = 23364 + 6636

weight_no_default = (1/NotDefault)*(total_count)/2.0
weight_default = (1/Default)*(total_count)/2.0

class_weights = {0:weight_no_default, 1:weight_default}
```

10.3 Model1

Goal: get great results in training set.

[478]:
```
# Training the DNN model 1
model = models.Sequential()
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dense(100, activation='relu'))
```

```python
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

history = model.fit(OH_train_X,
                    train_Y,
                    epochs=200,
                    batch_size=50,
                    validation_split=0.2,
                    class_weight = class_weights) # Note that the
 ↪"validation_split" parameter
```

```
Train on 19200 samples, validate on 4800 samples
Epoch 1/200
19200/19200 [==============================] - 2s 116us/step - loss: 0.5984 -
accuracy: 0.7478 - val_loss: 0.5945 - val_accuracy: 0.7667
Epoch 2/200
19200/19200 [==============================] - 2s 83us/step - loss: 0.5762 -
accuracy: 0.7586 - val_loss: 0.5817 - val_accuracy: 0.7475
Epoch 3/200
19200/19200 [==============================] - 2s 83us/step - loss: 0.5665 -
accuracy: 0.7609 - val_loss: 0.5852 - val_accuracy: 0.7004
Epoch 4/200
19200/19200 [==============================] - 2s 84us/step - loss: 0.5609 -
accuracy: 0.7487 - val_loss: 0.5837 - val_accuracy: 0.7460
Epoch 5/200
19200/19200 [==============================] - 2s 92us/step - loss: 0.5545 -
accuracy: 0.7565 - val_loss: 0.5924 - val_accuracy: 0.6615
Epoch 6/200
19200/19200 [==============================] - 2s 87us/step - loss: 0.5540 -
accuracy: 0.7560 - val_loss: 0.5899 - val_accuracy: 0.7340
Epoch 7/200
19200/19200 [==============================] - 2s 85us/step - loss: 0.5468 -
accuracy: 0.7606 - val_loss: 0.5792 - val_accuracy: 0.7575
Epoch 8/200
19200/19200 [==============================] - 2s 84us/step - loss: 0.5432 -
accuracy: 0.7601 - val_loss: 0.5834 - val_accuracy: 0.7342
Epoch 9/200
```

```
19200/19200 [==============================] - 2s 85us/step - loss: 0.5357 -
accuracy: 0.7644 - val_loss: 0.5898 - val_accuracy: 0.7617
Epoch 10/200
19200/19200 [==============================] - 2s 83us/step - loss: 0.5335 -
accuracy: 0.7625 - val_loss: 0.5908 - val_accuracy: 0.7163
Epoch 11/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.5303 -
accuracy: 0.7688 - val_loss: 0.6045 - val_accuracy: 0.7577
Epoch 12/200
19200/19200 [==============================] - 2s 84us/step - loss: 0.5265 -
accuracy: 0.7640 - val_loss: 0.5939 - val_accuracy: 0.7525
Epoch 13/200
19200/19200 [==============================] - 2s 95us/step - loss: 0.5238 -
accuracy: 0.7620 - val_loss: 0.6014 - val_accuracy: 0.7527
Epoch 14/200
19200/19200 [==============================] - 2s 90us/step - loss: 0.5198 -
accuracy: 0.7590 - val_loss: 0.6004 - val_accuracy: 0.7100
Epoch 15/200
19200/19200 [==============================] - 2s 85us/step - loss: 0.5142 -
accuracy: 0.7658 - val_loss: 0.6432 - val_accuracy: 0.7458
Epoch 16/200
19200/19200 [==============================] - 2s 83us/step - loss: 0.5096 -
accuracy: 0.7699 - val_loss: 0.6065 - val_accuracy: 0.6871
Epoch 17/200
19200/19200 [==============================] - 2s 88us/step - loss: 0.5053 -
accuracy: 0.7633 - val_loss: 0.6602 - val_accuracy: 0.7398
Epoch 18/200
19200/19200 [==============================] - 2s 83us/step - loss: 0.5017 -
accuracy: 0.7734 - val_loss: 0.6138 - val_accuracy: 0.7158
Epoch 19/200
19200/19200 [==============================] - 2s 84us/step - loss: 0.4957 -
accuracy: 0.7673 - val_loss: 0.6379 - val_accuracy: 0.7394
Epoch 20/200
19200/19200 [==============================] - 2s 85us/step - loss: 0.4899 -
accuracy: 0.7683 - val_loss: 0.6406 - val_accuracy: 0.7596
Epoch 21/200
19200/19200 [==============================] - 2s 83us/step - loss: 0.4903 -
accuracy: 0.7698 - val_loss: 0.6890 - val_accuracy: 0.7265
Epoch 22/200
19200/19200 [==============================] - 2s 85us/step - loss: 0.4822 -
accuracy: 0.7718 - val_loss: 0.6826 - val_accuracy: 0.7319
Epoch 23/200
19200/19200 [==============================] - 2s 84us/step - loss: 0.4778 -
accuracy: 0.7702 - val_loss: 0.7220 - val_accuracy: 0.7473
Epoch 24/200
19200/19200 [==============================] - 2s 85us/step - loss: 0.4703 -
accuracy: 0.7783 - val_loss: 0.6718 - val_accuracy: 0.7292
Epoch 25/200
```

```
19200/19200 [==============================] - 2s 84us/step - loss: 0.4683 -
accuracy: 0.7772 - val_loss: 0.6532 - val_accuracy: 0.6881
Epoch 26/200
19200/19200 [==============================] - 2s 83us/step - loss: 0.4642 -
accuracy: 0.7845 - val_loss: 0.6973 - val_accuracy: 0.7231
Epoch 27/200
19200/19200 [==============================] - 2s 85us/step - loss: 0.4518 -
accuracy: 0.7840 - val_loss: 0.7409 - val_accuracy: 0.7342
Epoch 28/200
19200/19200 [==============================] - 2s 88us/step - loss: 0.4500 -
accuracy: 0.7882 - val_loss: 0.7495 - val_accuracy: 0.7444
Epoch 29/200
19200/19200 [==============================] - 2s 85us/step - loss: 0.4450 -
accuracy: 0.7860 - val_loss: 0.7596 - val_accuracy: 0.7219
Epoch 30/200
19200/19200 [==============================] - 2s 100us/step - loss: 0.4390 -
accuracy: 0.7936 - val_loss: 0.8167 - val_accuracy: 0.7496
Epoch 31/200
19200/19200 [==============================] - 2s 85us/step - loss: 0.4357 -
accuracy: 0.7989 - val_loss: 0.8346 - val_accuracy: 0.6910
Epoch 32/200
19200/19200 [==============================] - 2s 85us/step - loss: 0.4269 -
accuracy: 0.8012 - val_loss: 0.8707 - val_accuracy: 0.7133
Epoch 33/200
19200/19200 [==============================] - 2s 88us/step - loss: 0.4216 -
accuracy: 0.8001 - val_loss: 0.8843 - val_accuracy: 0.7021
Epoch 34/200
19200/19200 [==============================] - 2s 85us/step - loss: 0.4204 -
accuracy: 0.8040 - val_loss: 0.7661 - val_accuracy: 0.7013
Epoch 35/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.4197 -
accuracy: 0.8026 - val_loss: 0.9247 - val_accuracy: 0.6908
Epoch 36/200
19200/19200 [==============================] - 2s 93us/step - loss: 0.4075 -
accuracy: 0.8079 - val_loss: 1.1026 - val_accuracy: 0.7156
Epoch 37/200
19200/19200 [==============================] - 2s 89us/step - loss: 0.4030 -
accuracy: 0.8158 - val_loss: 1.0807 - val_accuracy: 0.7415
Epoch 38/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.4012 -
accuracy: 0.8134 - val_loss: 0.9418 - val_accuracy: 0.7113
Epoch 39/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.3943 -
accuracy: 0.8140 - val_loss: 0.9328 - val_accuracy: 0.7058
Epoch 40/200
19200/19200 [==============================] - 2s 95us/step - loss: 0.3856 -
accuracy: 0.8193 - val_loss: 0.9530 - val_accuracy: 0.7221
Epoch 41/200
```

```
19200/19200 [==============================] - 2s 86us/step - loss: 0.3808 -
accuracy: 0.8214 - val_loss: 1.3613 - val_accuracy: 0.7006
Epoch 42/200
19200/19200 [==============================] - 2s 92us/step - loss: 0.3846 -
accuracy: 0.8229 - val_loss: 0.8204 - val_accuracy: 0.7273
Epoch 43/200
19200/19200 [==============================] - 2s 84us/step - loss: 0.3807 -
accuracy: 0.8241 - val_loss: 1.0636 - val_accuracy: 0.7100
Epoch 44/200
19200/19200 [==============================] - 2s 84us/step - loss: 0.3707 -
accuracy: 0.8274 - val_loss: 1.0362 - val_accuracy: 0.7054
Epoch 45/200
19200/19200 [==============================] - 2s 88us/step - loss: 0.3685 -
accuracy: 0.8284 - val_loss: 1.1036 - val_accuracy: 0.7113
Epoch 46/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.3659 -
accuracy: 0.8270 - val_loss: 1.2545 - val_accuracy: 0.7196
Epoch 47/200
19200/19200 [==============================] - 2s 88us/step - loss: 0.3633 -
accuracy: 0.8278 - val_loss: 1.2005 - val_accuracy: 0.6981
Epoch 48/200
19200/19200 [==============================] - 2s 84us/step - loss: 0.3498 -
accuracy: 0.8372 - val_loss: 1.0658 - val_accuracy: 0.7304
Epoch 49/200
19200/19200 [==============================] - 2s 85us/step - loss: 0.3507 -
accuracy: 0.8338 - val_loss: 1.2746 - val_accuracy: 0.7006
Epoch 50/200
19200/19200 [==============================] - 2s 84us/step - loss: 0.3516 -
accuracy: 0.8320 - val_loss: 1.1694 - val_accuracy: 0.7346
Epoch 51/200
19200/19200 [==============================] - 2s 84us/step - loss: 0.3501 -
accuracy: 0.8386 - val_loss: 1.2412 - val_accuracy: 0.7175
Epoch 52/200
19200/19200 [==============================] - 2s 85us/step - loss: 0.3428 -
accuracy: 0.8412 - val_loss: 1.3504 - val_accuracy: 0.7177
Epoch 53/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.3437 -
accuracy: 0.8378 - val_loss: 1.3033 - val_accuracy: 0.7302
Epoch 54/200
19200/19200 [==============================] - 2s 88us/step - loss: 0.3323 -
accuracy: 0.8394 - val_loss: 1.1280 - val_accuracy: 0.7171
Epoch 55/200
19200/19200 [==============================] - 2s 87us/step - loss: 0.3333 -
accuracy: 0.8371 - val_loss: 1.4098 - val_accuracy: 0.7065
Epoch 56/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.3262 -
accuracy: 0.8483 - val_loss: 1.4978 - val_accuracy: 0.7075
Epoch 57/200
```

```
19200/19200 [==============================] - 2s 85us/step - loss: 0.3317 -
accuracy: 0.8399 - val_loss: 1.6230 - val_accuracy: 0.7048
Epoch 58/200
19200/19200 [==============================] - 2s 85us/step - loss: 0.3225 -
accuracy: 0.8519 - val_loss: 1.4823 - val_accuracy: 0.7167
Epoch 59/200
19200/19200 [==============================] - 2s 87us/step - loss: 0.3245 -
accuracy: 0.8478 - val_loss: 1.2624 - val_accuracy: 0.6996
Epoch 60/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.3126 -
accuracy: 0.8473 - val_loss: 1.5198 - val_accuracy: 0.7146
Epoch 61/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.3173 -
accuracy: 0.8487 - val_loss: 1.3201 - val_accuracy: 0.7121
Epoch 62/200
19200/19200 [==============================] - 2s 85us/step - loss: 0.3058 -
accuracy: 0.8535 - val_loss: 1.4064 - val_accuracy: 0.7319
Epoch 63/200
19200/19200 [==============================] - 2s 85us/step - loss: 0.3013 -
accuracy: 0.8592 - val_loss: 1.5091 - val_accuracy: 0.6998
Epoch 64/200
19200/19200 [==============================] - 2s 85us/step - loss: 0.2997 -
accuracy: 0.8544 - val_loss: 1.4488 - val_accuracy: 0.6862
Epoch 65/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.3041 -
accuracy: 0.8548 - val_loss: 1.2310 - val_accuracy: 0.7252
Epoch 66/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.2986 -
accuracy: 0.8606 - val_loss: 1.2946 - val_accuracy: 0.6942
Epoch 67/200
19200/19200 [==============================] - 2s 94us/step - loss: 0.2929 -
accuracy: 0.8561 - val_loss: 1.6763 - val_accuracy: 0.7025
Epoch 68/200
19200/19200 [==============================] - 2s 85us/step - loss: 0.2965 -
accuracy: 0.8558 - val_loss: 1.2676 - val_accuracy: 0.6917
Epoch 69/200
19200/19200 [==============================] - 2s 85us/step - loss: 0.2945 -
accuracy: 0.8597 - val_loss: 1.4050 - val_accuracy: 0.7206
Epoch 70/200
19200/19200 [==============================] - 2s 85us/step - loss: 0.2904 -
accuracy: 0.8628 - val_loss: 1.4332 - val_accuracy: 0.7100
Epoch 71/200
19200/19200 [==============================] - 2s 90us/step - loss: 0.2768 -
accuracy: 0.8647 - val_loss: 2.0846 - val_accuracy: 0.7115
Epoch 72/200
19200/19200 [==============================] - 2s 94us/step - loss: 0.2847 -
accuracy: 0.8578 - val_loss: 1.7852 - val_accuracy: 0.7171
Epoch 73/200
```

```
19200/19200 [==============================] - 2s 90us/step - loss: 0.2776 -
accuracy: 0.8606 - val_loss: 1.9285 - val_accuracy: 0.7183
Epoch 74/200
19200/19200 [==============================] - 2s 88us/step - loss: 0.2886 -
accuracy: 0.8559 - val_loss: 2.2079 - val_accuracy: 0.7027
Epoch 75/200
19200/19200 [==============================] - 2s 85us/step - loss: 0.2868 -
accuracy: 0.8609 - val_loss: 1.5773 - val_accuracy: 0.7023
Epoch 76/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.2775 -
accuracy: 0.8646 - val_loss: 1.7162 - val_accuracy: 0.7175
Epoch 77/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.2650 -
accuracy: 0.8725 - val_loss: 1.7492 - val_accuracy: 0.7138
Epoch 78/200
19200/19200 [==============================] - 2s 95us/step - loss: 0.2758 -
accuracy: 0.8619 - val_loss: 1.6342 - val_accuracy: 0.7202
Epoch 79/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.2732 -
accuracy: 0.8684 - val_loss: 1.4660 - val_accuracy: 0.7000
Epoch 80/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.2668 -
accuracy: 0.8693 - val_loss: 2.7126 - val_accuracy: 0.6994
Epoch 81/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.2608 -
accuracy: 0.8727 - val_loss: 2.5044 - val_accuracy: 0.6985
Epoch 82/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.2649 -
accuracy: 0.8689 - val_loss: 2.0555 - val_accuracy: 0.7121
Epoch 83/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.2612 -
accuracy: 0.8701 - val_loss: 2.1730 - val_accuracy: 0.6856
Epoch 84/200
19200/19200 [==============================] - 2s 91us/step - loss: 0.2644 -
accuracy: 0.8657 - val_loss: 1.7628 - val_accuracy: 0.7140
Epoch 85/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.2580 -
accuracy: 0.8737 - val_loss: 2.4163 - val_accuracy: 0.7025
Epoch 86/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.2559 -
accuracy: 0.8769 - val_loss: 1.5558 - val_accuracy: 0.6969
Epoch 87/200
19200/19200 [==============================] - 2s 89us/step - loss: 0.2511 -
accuracy: 0.8807 - val_loss: 2.4349 - val_accuracy: 0.7019
Epoch 88/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.2473 -
accuracy: 0.8781 - val_loss: 1.6509 - val_accuracy: 0.6746
Epoch 89/200
```

```
19200/19200 [==============================] - 2s 86us/step - loss: 0.2570 -
accuracy: 0.8755 - val_loss: 1.9480 - val_accuracy: 0.6985
Epoch 90/200
19200/19200 [==============================] - 2s 87us/step - loss: 0.2531 -
accuracy: 0.8763 - val_loss: 2.2757 - val_accuracy: 0.6829
Epoch 91/200
19200/19200 [==============================] - 2s 87us/step - loss: 0.2626 -
accuracy: 0.8714 - val_loss: 1.7131 - val_accuracy: 0.7273
Epoch 92/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.2450 -
accuracy: 0.8810 - val_loss: 2.2226 - val_accuracy: 0.6960
Epoch 93/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.2432 -
accuracy: 0.8791 - val_loss: 2.3551 - val_accuracy: 0.6963
Epoch 94/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.2400 -
accuracy: 0.8828 - val_loss: 2.4381 - val_accuracy: 0.6988
Epoch 95/200
19200/19200 [==============================] - 2s 85us/step - loss: 0.2452 -
accuracy: 0.8832 - val_loss: 1.7638 - val_accuracy: 0.7129
Epoch 96/200
19200/19200 [==============================] - 2s 85us/step - loss: 0.2321 -
accuracy: 0.8832 - val_loss: 2.6836 - val_accuracy: 0.7379
Epoch 97/200
19200/19200 [==============================] - 2s 87us/step - loss: 0.2398 -
accuracy: 0.8819 - val_loss: 1.8252 - val_accuracy: 0.7023
Epoch 98/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.2332 -
accuracy: 0.8806 - val_loss: 2.7424 - val_accuracy: 0.7069
Epoch 99/200
19200/19200 [==============================] - 2s 87us/step - loss: 0.2319 -
accuracy: 0.8866 - val_loss: 2.7233 - val_accuracy: 0.7142
Epoch 100/200
19200/19200 [==============================] - 2s 89us/step - loss: 0.2416 -
accuracy: 0.8816 - val_loss: 2.6193 - val_accuracy: 0.7240
Epoch 101/200
19200/19200 [==============================] - 2s 96us/step - loss: 0.2401 -
accuracy: 0.8835 - val_loss: 2.1551 - val_accuracy: 0.6923
Epoch 102/200
19200/19200 [==============================] - 2s 87us/step - loss: 0.2266 -
accuracy: 0.8871 - val_loss: 2.4055 - val_accuracy: 0.7171
Epoch 103/200
19200/19200 [==============================] - 2s 85us/step - loss: 0.2270 -
accuracy: 0.8860 - val_loss: 2.0561 - val_accuracy: 0.6969
Epoch 104/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.2292 -
accuracy: 0.8831 - val_loss: 2.0879 - val_accuracy: 0.7069
Epoch 105/200
```

```
19200/19200 [==============================] - 2s 86us/step - loss: 0.2264 -
accuracy: 0.8884 - val_loss: 2.0786 - val_accuracy: 0.6904
Epoch 106/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.2169 -
accuracy: 0.8932 - val_loss: 2.2924 - val_accuracy: 0.6975
Epoch 107/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.2174 -
accuracy: 0.8929 - val_loss: 2.6606 - val_accuracy: 0.7113
Epoch 108/200
19200/19200 [==============================] - 2s 92us/step - loss: 0.2245 -
accuracy: 0.8909 - val_loss: 2.3564 - val_accuracy: 0.7192
Epoch 109/200
19200/19200 [==============================] - 2s 89us/step - loss: 0.2226 -
accuracy: 0.8914 - val_loss: 1.4551 - val_accuracy: 0.6723
Epoch 110/200
19200/19200 [==============================] - 2s 86us/step - loss: 0.2198 -
accuracy: 0.8918 - val_loss: 2.5299 - val_accuracy: 0.7098
Epoch 111/200
19200/19200 [==============================] - 2s 87us/step - loss: 0.2288 -
accuracy: 0.8870 - val_loss: 2.7953 - val_accuracy: 0.7177
Epoch 112/200
19200/19200 [==============================] - 2s 90us/step - loss: 0.2100 -
accuracy: 0.8980 - val_loss: 3.1128 - val_accuracy: 0.7077
Epoch 113/200
19200/19200 [==============================] - 2s 90us/step - loss: 0.2198 -
accuracy: 0.8906 - val_loss: 1.8188 - val_accuracy: 0.6969
Epoch 114/200
19200/19200 [==============================] - 2s 94us/step - loss: 0.2092 -
accuracy: 0.8964 - val_loss: 2.6084 - val_accuracy: 0.7223
Epoch 115/200
19200/19200 [==============================] - 2s 92us/step - loss: 0.2112 -
accuracy: 0.8994 - val_loss: 2.0068 - val_accuracy: 0.7167
Epoch 116/200
19200/19200 [==============================] - 2s 87us/step - loss: 0.2094 -
accuracy: 0.8947 - val_loss: 2.7010 - val_accuracy: 0.7152
Epoch 117/200
19200/19200 [==============================] - 2s 106us/step - loss: 0.2081 -
accuracy: 0.8995 - val_loss: 2.5771 - val_accuracy: 0.7035
Epoch 118/200
19200/19200 [==============================] - 2s 116us/step - loss: 0.2120 -
accuracy: 0.8957 - val_loss: 1.7348 - val_accuracy: 0.6915
Epoch 119/200
19200/19200 [==============================] - 2s 120us/step - loss: 0.1990 -
accuracy: 0.9035 - val_loss: 2.3227 - val_accuracy: 0.7081
Epoch 120/200
19200/19200 [==============================] - 2s 110us/step - loss: 0.1943 -
accuracy: 0.9046 - val_loss: 2.7618 - val_accuracy: 0.7160
Epoch 121/200
```

```
19200/19200 [==============================] - 2s 95us/step - loss: 0.2149 -
accuracy: 0.8978 - val_loss: 2.1072 - val_accuracy: 0.7106
Epoch 122/200
19200/19200 [==============================] - 2s 92us/step - loss: 0.2050 -
accuracy: 0.9008 - val_loss: 2.2465 - val_accuracy: 0.6971
Epoch 123/200
19200/19200 [==============================] - 2s 91us/step - loss: 0.2007 -
accuracy: 0.9003 - val_loss: 2.1089 - val_accuracy: 0.7123
Epoch 124/200
19200/19200 [==============================] - 2s 114us/step - loss: 0.1952 -
accuracy: 0.9039 - val_loss: 2.3501 - val_accuracy: 0.7221
Epoch 125/200
19200/19200 [==============================] - 2s 112us/step - loss: 0.2049 -
accuracy: 0.8977 - val_loss: 2.1886 - val_accuracy: 0.7267
Epoch 126/200
19200/19200 [==============================] - 2s 85us/step - loss: 0.2035 -
accuracy: 0.9020 - val_loss: 3.0922 - val_accuracy: 0.6985
Epoch 127/200
19200/19200 [==============================] - 2s 80us/step - loss: 0.2047 -
accuracy: 0.8999 - val_loss: 2.8324 - val_accuracy: 0.7092
Epoch 128/200
19200/19200 [==============================] - 2s 97us/step - loss: 0.1943 -
accuracy: 0.9049 - val_loss: 3.5924 - val_accuracy: 0.7004
Epoch 129/200
19200/19200 [==============================] - 2s 91us/step - loss: 0.1990 -
accuracy: 0.9032 - val_loss: 3.0706 - val_accuracy: 0.7217
Epoch 130/200
19200/19200 [==============================] - 2s 124us/step - loss: 0.1972 -
accuracy: 0.9048 - val_loss: 2.1141 - val_accuracy: 0.6946
Epoch 131/200
19200/19200 [==============================] - 2s 123us/step - loss: 0.1955 -
accuracy: 0.9064 - val_loss: 2.8003 - val_accuracy: 0.7069
Epoch 132/200
19200/19200 [==============================] - 2s 106us/step - loss: 0.1918 -
accuracy: 0.9067 - val_loss: 3.3583 - val_accuracy: 0.7121
Epoch 133/200
19200/19200 [==============================] - 2s 106us/step - loss: 0.1898 -
accuracy: 0.9086 - val_loss: 3.5681 - val_accuracy: 0.7202
Epoch 134/200
19200/19200 [==============================] - 2s 105us/step - loss: 0.2054 -
accuracy: 0.9019 - val_loss: 2.5601 - val_accuracy: 0.7000
Epoch 135/200
19200/19200 [==============================] - 2s 109us/step - loss: 0.1902 -
accuracy: 0.9085 - val_loss: 4.5453 - val_accuracy: 0.7262
Epoch 136/200
19200/19200 [==============================] - 2s 116us/step - loss: 0.1928 -
accuracy: 0.9071 - val_loss: 5.7248 - val_accuracy: 0.7152
Epoch 137/200
```

```
19200/19200 [==============================] - 2s 109us/step - loss: 0.1991 -
accuracy: 0.9031 - val_loss: 3.3480 - val_accuracy: 0.6992
Epoch 138/200
19200/19200 [==============================] - 2s 110us/step - loss: 0.1902 -
accuracy: 0.9083 - val_loss: 2.3743 - val_accuracy: 0.6938
Epoch 139/200
19200/19200 [==============================] - 2s 130us/step - loss: 0.1861 -
accuracy: 0.9071 - val_loss: 4.8512 - val_accuracy: 0.6992
Epoch 140/200
19200/19200 [==============================] - 3s 131us/step - loss: 0.1898 -
accuracy: 0.9107 - val_loss: 2.4764 - val_accuracy: 0.7040
Epoch 141/200
19200/19200 [==============================] - 2s 100us/step - loss: 0.1775 -
accuracy: 0.9135 - val_loss: 3.3897 - val_accuracy: 0.7152
Epoch 142/200
19200/19200 [==============================] - 2s 91us/step - loss: 0.1804 -
accuracy: 0.9065 - val_loss: 3.9702 - val_accuracy: 0.7050
Epoch 143/200
19200/19200 [==============================] - 2s 89us/step - loss: 0.1920 -
accuracy: 0.9031 - val_loss: 2.8933 - val_accuracy: 0.7000
Epoch 144/200
19200/19200 [==============================] - 2s 102us/step - loss: 0.1919 -
accuracy: 0.9067 - val_loss: 2.5031 - val_accuracy: 0.7063
Epoch 145/200
19200/19200 [==============================] - 2s 91us/step - loss: 0.1838 -
accuracy: 0.9114 - val_loss: 3.0283 - val_accuracy: 0.6942
Epoch 146/200
19200/19200 [==============================] - 2s 92us/step - loss: 0.1839 -
accuracy: 0.9094 - val_loss: 4.0682 - val_accuracy: 0.6904
Epoch 147/200
19200/19200 [==============================] - 2s 95us/step - loss: 0.2040 -
accuracy: 0.8998 - val_loss: 2.7932 - val_accuracy: 0.7067
Epoch 148/200
19200/19200 [==============================] - 2s 93us/step - loss: 0.1885 -
accuracy: 0.9089 - val_loss: 3.2708 - val_accuracy: 0.6973
Epoch 149/200
19200/19200 [==============================] - 2s 91us/step - loss: 0.1778 -
accuracy: 0.9130 - val_loss: 4.0338 - val_accuracy: 0.7102
Epoch 150/200
19200/19200 [==============================] - 2s 92us/step - loss: 0.1771 -
accuracy: 0.9112 - val_loss: 5.2165 - val_accuracy: 0.6994
Epoch 151/200
19200/19200 [==============================] - 2s 100us/step - loss: 0.1781 -
accuracy: 0.9119 - val_loss: 3.9651 - val_accuracy: 0.7133
Epoch 152/200
19200/19200 [==============================] - 2s 121us/step - loss: 0.1817 -
accuracy: 0.9081 - val_loss: 3.8409 - val_accuracy: 0.7144
Epoch 153/200
```

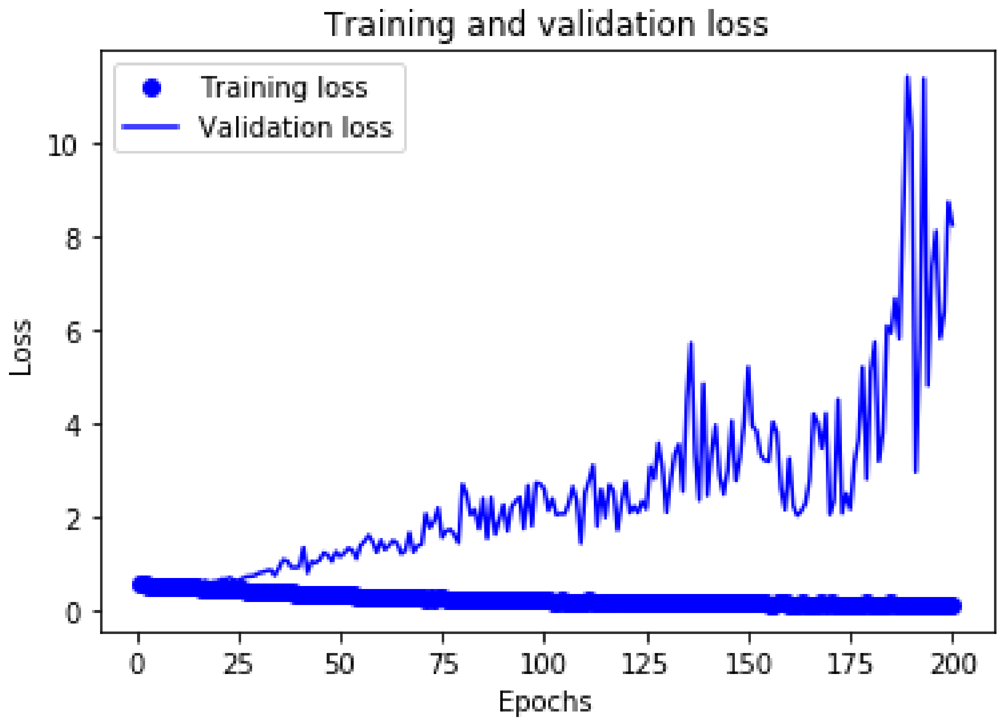
```
19200/19200 [==============================] - 2s 102us/step - loss: 0.1846 -
accuracy: 0.9125 - val_loss: 3.3400 - val_accuracy: 0.7025
Epoch 154/200
19200/19200 [==============================] - 2s 115us/step - loss: 0.1795 -
accuracy: 0.9112 - val_loss: 3.2233 - val_accuracy: 0.7094
Epoch 155/200
19200/19200 [==============================] - 2s 90us/step - loss: 0.1706 -
accuracy: 0.9148 - val_loss: 3.1951 - val_accuracy: 0.6919
Epoch 156/200
19200/19200 [==============================] - 2s 88us/step - loss: 0.1678 -
accuracy: 0.9148 - val_loss: 4.0392 - val_accuracy: 0.7233
Epoch 157/200
19200/19200 [==============================] - 2s 89us/step - loss: 0.1797 -
accuracy: 0.9102 - val_loss: 3.8115 - val_accuracy: 0.7079
Epoch 158/200
19200/19200 [==============================] - 2s 90us/step - loss: 0.1731 -
accuracy: 0.9153 - val_loss: 2.6289 - val_accuracy: 0.7113
Epoch 159/200
19200/19200 [==============================] - 2s 88us/step - loss: 0.1753 -
accuracy: 0.9159 - val_loss: 2.1713 - val_accuracy: 0.6973
Epoch 160/200
19200/19200 [==============================] - 2s 89us/step - loss: 0.1601 -
accuracy: 0.9207 - val_loss: 3.2644 - val_accuracy: 0.7058
Epoch 161/200
19200/19200 [==============================] - 2s 93us/step - loss: 0.1700 -
accuracy: 0.9176 - val_loss: 2.2559 - val_accuracy: 0.7167
Epoch 162/200
19200/19200 [==============================] - 2s 91us/step - loss: 0.1704 -
accuracy: 0.9198 - val_loss: 2.0468 - val_accuracy: 0.7006
Epoch 163/200
19200/19200 [==============================] - 2s 98us/step - loss: 0.1803 -
accuracy: 0.9134 - val_loss: 2.1443 - val_accuracy: 0.7154
Epoch 164/200
19200/19200 [==============================] - 2s 91us/step - loss: 0.1692 -
accuracy: 0.9152 - val_loss: 2.3086 - val_accuracy: 0.7025
Epoch 165/200
19200/19200 [==============================] - 2s 92us/step - loss: 0.1686 -
accuracy: 0.9167 - val_loss: 2.8064 - val_accuracy: 0.7042
Epoch 166/200
19200/19200 [==============================] - 2s 92us/step - loss: 0.1651 -
accuracy: 0.9194 - val_loss: 4.2116 - val_accuracy: 0.7171
Epoch 167/200
19200/19200 [==============================] - 2s 101us/step - loss: 0.1673 -
accuracy: 0.9185 - val_loss: 4.0143 - val_accuracy: 0.6935
Epoch 168/200
19200/19200 [==============================] - 2s 91us/step - loss: 0.1726 -
accuracy: 0.9151 - val_loss: 3.4871 - val_accuracy: 0.7088
Epoch 169/200
```

```
19200/19200 [==============================] - 2s 92us/step - loss: 0.1580 -
accuracy: 0.9226 - val_loss: 4.2231 - val_accuracy: 0.7073
Epoch 170/200
19200/19200 [==============================] - 2s 91us/step - loss: 0.1857 -
accuracy: 0.9095 - val_loss: 2.0728 - val_accuracy: 0.7158
Epoch 171/200
19200/19200 [==============================] - 2s 92us/step - loss: 0.1605 -
accuracy: 0.9243 - val_loss: 2.3611 - val_accuracy: 0.7223
Epoch 172/200
19200/19200 [==============================] - 2s 103us/step - loss: 0.1556 -
accuracy: 0.9251 - val_loss: 4.5169 - val_accuracy: 0.7056
Epoch 173/200
19200/19200 [==============================] - 2s 97us/step - loss: 0.1594 -
accuracy: 0.9226 - val_loss: 2.0940 - val_accuracy: 0.7052
Epoch 174/200
19200/19200 [==============================] - 2s 108us/step - loss: 0.1702 -
accuracy: 0.9170 - val_loss: 2.5126 - val_accuracy: 0.6944
Epoch 175/200
19200/19200 [==============================] - 2s 94us/step - loss: 0.1614 -
accuracy: 0.9198 - val_loss: 2.1845 - val_accuracy: 0.6915
Epoch 176/200
19200/19200 [==============================] - 2s 94us/step - loss: 0.1551 -
accuracy: 0.9231 - val_loss: 3.1835 - val_accuracy: 0.7133
Epoch 177/200
19200/19200 [==============================] - 2s 124us/step - loss: 0.1502 -
accuracy: 0.9254 - val_loss: 3.6752 - val_accuracy: 0.7033
Epoch 178/200
19200/19200 [==============================] - 2s 116us/step - loss: 0.1563 -
accuracy: 0.9231 - val_loss: 5.2108 - val_accuracy: 0.6983
Epoch 179/200
19200/19200 [==============================] - 2s 116us/step - loss: 0.1839 -
accuracy: 0.9068 - val_loss: 2.8279 - val_accuracy: 0.6992
Epoch 180/200
19200/19200 [==============================] - 2s 123us/step - loss: 0.1699 -
accuracy: 0.9158 - val_loss: 5.1350 - val_accuracy: 0.7088
Epoch 181/200
19200/19200 [==============================] - 2s 114us/step - loss: 0.1634 -
accuracy: 0.9201 - val_loss: 5.7503 - val_accuracy: 0.7188
Epoch 182/200
19200/19200 [==============================] - 2s 92us/step - loss: 0.1638 -
accuracy: 0.9248 - val_loss: 3.2009 - val_accuracy: 0.7038
Epoch 183/200
19200/19200 [==============================] - 2s 93us/step - loss: 0.1471 -
accuracy: 0.9265 - val_loss: 3.8039 - val_accuracy: 0.7054
Epoch 184/200
19200/19200 [==============================] - 2s 92us/step - loss: 0.1546 -
accuracy: 0.9246 - val_loss: 6.0918 - val_accuracy: 0.7167
Epoch 185/200
```

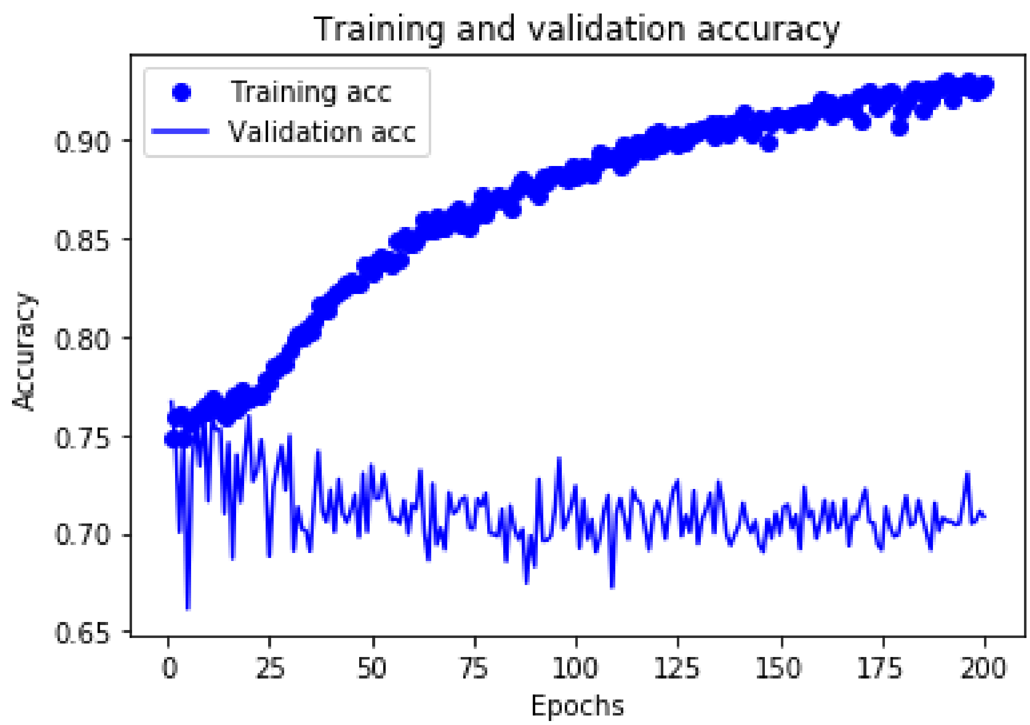
```
19200/19200 [==============================] - 2s 97us/step - loss: 0.1709 -
accuracy: 0.9156 - val_loss: 5.9413 - val_accuracy: 0.7075
Epoch 186/200
19200/19200 [==============================] - 2s 98us/step - loss: 0.1581 -
accuracy: 0.9263 - val_loss: 6.6725 - val_accuracy: 0.6998
Epoch 187/200
19200/19200 [==============================] - 2s 116us/step - loss: 0.1586 -
accuracy: 0.9216 - val_loss: 5.8205 - val_accuracy: 0.6915
Epoch 188/200
19200/19200 [==============================] - 2s 96us/step - loss: 0.1527 -
accuracy: 0.9268 - val_loss: 8.6074 - val_accuracy: 0.7152
Epoch 189/200
19200/19200 [==============================] - 2s 108us/step - loss: 0.1478 -
accuracy: 0.9269 - val_loss: 11.4070 - val_accuracy: 0.7010
Epoch 190/200
19200/19200 [==============================] - 2s 119us/step - loss: 0.1480 -
accuracy: 0.9291 - val_loss: 10.2122 - val_accuracy: 0.7077
Epoch 191/200
19200/19200 [==============================] - 2s 89us/step - loss: 0.1479 -
accuracy: 0.9302 - val_loss: 2.9767 - val_accuracy: 0.7060
Epoch 192/200
19200/19200 [==============================] - 2s 97us/step - loss: 0.1648 -
accuracy: 0.9208 - val_loss: 5.7300 - val_accuracy: 0.7058
Epoch 193/200
19200/19200 [==============================] - 2s 99us/step - loss: 0.1484 -
accuracy: 0.9283 - val_loss: 11.3659 - val_accuracy: 0.7042
Epoch 194/200
19200/19200 [==============================] - 2s 83us/step - loss: 0.1540 -
accuracy: 0.9276 - val_loss: 4.8226 - val_accuracy: 0.7048
Epoch 195/200
19200/19200 [==============================] - 2s 105us/step - loss: 0.1470 -
accuracy: 0.9274 - val_loss: 7.3405 - val_accuracy: 0.7163
Epoch 196/200
19200/19200 [==============================] - 2s 121us/step - loss: 0.1475 -
accuracy: 0.9299 - val_loss: 8.1153 - val_accuracy: 0.7304
Epoch 197/200
19200/19200 [==============================] - 2s 120us/step - loss: 0.1550 -
accuracy: 0.9258 - val_loss: 5.8277 - val_accuracy: 0.7048
Epoch 198/200
19200/19200 [==============================] - 2s 91us/step - loss: 0.1541 -
accuracy: 0.9257 - val_loss: 6.3774 - val_accuracy: 0.7063
Epoch 199/200
19200/19200 [==============================] - 2s 117us/step - loss: 0.1477 -
accuracy: 0.9264 - val_loss: 8.7343 - val_accuracy: 0.7110
Epoch 200/200
19200/19200 [==============================] - 2s 93us/step - loss: 0.1492 -
accuracy: 0.9292 - val_loss: 8.2471 - val_accuracy: 0.7083
```

10.4 Plot the results of loss and accuracy

[479]:
```
# plot the results of loss values from the training set and validtion set
import matplotlib.pyplot as plt
history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']

epochs = range(1, len(history_dict['accuracy']) + 1)

plt.plot(epochs, loss_values, 'bo', label='Training loss') # bo is for blue dot
plt.plot(epochs, val_loss_values, 'b', label='Validation loss') # b is for blue␣
 ↪line
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

#plt.clf()
acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



Training and validation loss
Training loss
Validation loss
Loss
Epochs
0
2
4
6
8
10
0
25
50
75
100
125
150
175
200



Training and validation accuracy
Training acc
Validation acc
Accuracy
Epochs
0.65
0.70
0.75
0.80
0.85
0.90
0
25
50
75
100
125
150
175
200

According to above plots, we find that we have significant overfitting problem. (The validation loss is out of control).

Therefore, let's use the 'dropout' techiniques in our next DNN model to prevent this problem.

10.5 Model2

Let's use the 'dropout' techiniques to prevent this problem.

[504]:
```
# Training the DNN model 2
# Use "dropout" to prevent overfitting
model = models.Sequential()
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dropout(0.1))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

history = model.fit(OH_train_X,
                    train_Y,
                    epochs=200,
                    batch_size=50,
                    validation_split=0.2,
                    class_weight = class_weights) # Note that the␣
↪"validation_split" parameter
```

```
Train on 19200 samples, validate on 4800 samples
Epoch 1/30
19200/19200 [==============================] - 3s 165us/step - loss: 0.6858 -
accuracy: 0.5519 - val_loss: 0.6953 - val_accuracy: 0.7783
```

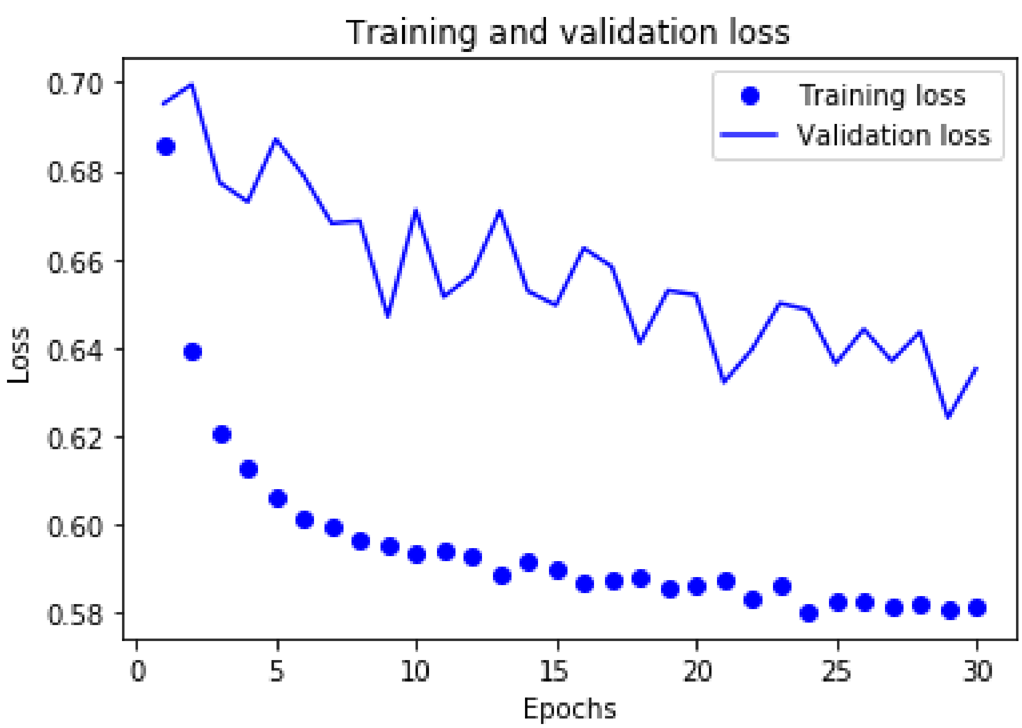
```
Epoch 2/30
19200/19200 [==============================] - 3s 155us/step - loss: 0.6393 -
accuracy: 0.7455 - val_loss: 0.6995 - val_accuracy: 0.7873
Epoch 3/30
19200/19200 [==============================] - 2s 118us/step - loss: 0.6207 -
accuracy: 0.7506 - val_loss: 0.6773 - val_accuracy: 0.8062
Epoch 4/30
19200/19200 [==============================] - 2s 118us/step - loss: 0.6127 -
accuracy: 0.7333 - val_loss: 0.6730 - val_accuracy: 0.8108
Epoch 5/30
19200/19200 [==============================] - 2s 119us/step - loss: 0.6064 -
accuracy: 0.7348 - val_loss: 0.6872 - val_accuracy: 0.8102
Epoch 6/30
19200/19200 [==============================] - 2s 113us/step - loss: 0.6016 -
accuracy: 0.7289 - val_loss: 0.6789 - val_accuracy: 0.8140
Epoch 7/30
19200/19200 [==============================] - 2s 118us/step - loss: 0.5998 -
accuracy: 0.7201 - val_loss: 0.6681 - val_accuracy: 0.8098
Epoch 8/30
19200/19200 [==============================] - 2s 116us/step - loss: 0.5966 -
accuracy: 0.7181 - val_loss: 0.6687 - val_accuracy: 0.8098
Epoch 9/30
19200/19200 [==============================] - 2s 118us/step - loss: 0.5955 -
accuracy: 0.7138 - val_loss: 0.6471 - val_accuracy: 0.8010
Epoch 10/30
19200/19200 [==============================] - 2s 116us/step - loss: 0.5934 -
accuracy: 0.7252 - val_loss: 0.6712 - val_accuracy: 0.8152
Epoch 11/30
19200/19200 [==============================] - 2s 128us/step - loss: 0.5940 -
accuracy: 0.7229 - val_loss: 0.6516 - val_accuracy: 0.8104
Epoch 12/30
19200/19200 [==============================] - 2s 123us/step - loss: 0.5932 -
accuracy: 0.7290 - val_loss: 0.6565 - val_accuracy: 0.8092
Epoch 13/30
19200/19200 [==============================] - 3s 148us/step - loss: 0.5890 -
accuracy: 0.7238 - val_loss: 0.6710 - val_accuracy: 0.8142
Epoch 14/30
19200/19200 [==============================] - 2s 124us/step - loss: 0.5917 -
accuracy: 0.7343 - val_loss: 0.6528 - val_accuracy: 0.8081
Epoch 15/30
19200/19200 [==============================] - 2s 121us/step - loss: 0.5898 -
accuracy: 0.7259 - val_loss: 0.6496 - val_accuracy: 0.8031
Epoch 16/30
19200/19200 [==============================] - 3s 134us/step - loss: 0.5868 -
accuracy: 0.7378 - val_loss: 0.6625 - val_accuracy: 0.8115
Epoch 17/30
19200/19200 [==============================] - 2s 125us/step - loss: 0.5874 -
accuracy: 0.7424 - val_loss: 0.6584 - val_accuracy: 0.8138
```

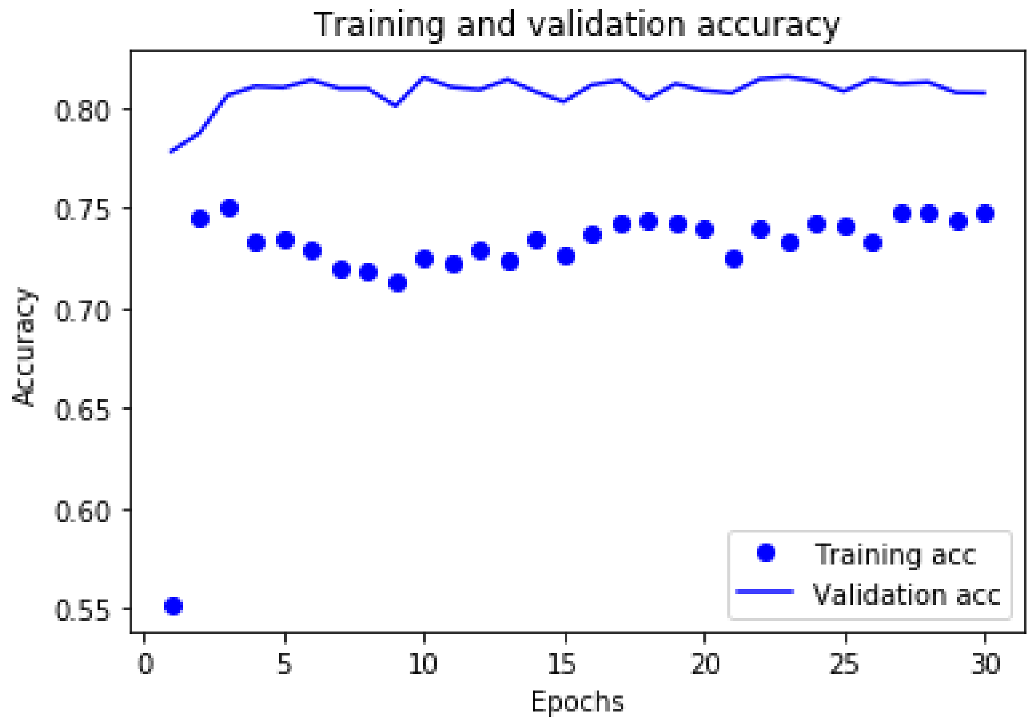
```
Epoch 18/30
19200/19200 [==============================] - 3s 132us/step - loss: 0.5881 -
accuracy: 0.7439 - val_loss: 0.6411 - val_accuracy: 0.8044
Epoch 19/30
19200/19200 [==============================] - 3s 140us/step - loss: 0.5859 -
accuracy: 0.7421 - val_loss: 0.6530 - val_accuracy: 0.8121
Epoch 20/30
19200/19200 [==============================] - 2s 124us/step - loss: 0.5860 -
accuracy: 0.7393 - val_loss: 0.6520 - val_accuracy: 0.8087
Epoch 21/30
19200/19200 [==============================] - 2s 116us/step - loss: 0.5873 -
accuracy: 0.7257 - val_loss: 0.6322 - val_accuracy: 0.8075
Epoch 22/30
19200/19200 [==============================] - 2s 123us/step - loss: 0.5831 -
accuracy: 0.7401 - val_loss: 0.6396 - val_accuracy: 0.8144
Epoch 23/30
19200/19200 [==============================] - 3s 133us/step - loss: 0.5866 -
accuracy: 0.7333 - val_loss: 0.6501 - val_accuracy: 0.8156
Epoch 24/30
19200/19200 [==============================] - 3s 143us/step - loss: 0.5803 -
accuracy: 0.7430 - val_loss: 0.6486 - val_accuracy: 0.8133
Epoch 25/30
19200/19200 [==============================] - 3s 131us/step - loss: 0.5827 -
accuracy: 0.7406 - val_loss: 0.6364 - val_accuracy: 0.8083
Epoch 26/30
19200/19200 [==============================] - 2s 108us/step - loss: 0.5828 -
accuracy: 0.7338 - val_loss: 0.6442 - val_accuracy: 0.8144
Epoch 27/30
19200/19200 [==============================] - 3s 138us/step - loss: 0.5817 -
accuracy: 0.7485 - val_loss: 0.6370 - val_accuracy: 0.8121
Epoch 28/30
19200/19200 [==============================] - 2s 123us/step - loss: 0.5820 -
accuracy: 0.7472 - val_loss: 0.6436 - val_accuracy: 0.8129
Epoch 29/30
19200/19200 [==============================] - 2s 112us/step - loss: 0.5809 -
accuracy: 0.7440 - val_loss: 0.6242 - val_accuracy: 0.8077
Epoch 30/30
19200/19200 [==============================] - 2s 114us/step - loss: 0.5815 -
accuracy: 0.7477 - val_loss: 0.6353 - val_accuracy: 0.8075
```

10.6 Plot the results of loss and accuracy

[505]:
```python
# plot the results of loss values from the training set and validtion set
import matplotlib.pyplot as plt
history_dict = history.history
loss_values = history_dict['loss']
```

```python
val_loss_values = history_dict['val_loss']

epochs = range(1, len(history_dict['accuracy']) + 1)

plt.plot(epochs, loss_values, 'bo', label='Training loss') # bo is for blue dot
plt.plot(epochs, val_loss_values, 'b', label='Validation loss') # b is for blue␣
↪line
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

#plt.clf()
acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```





According to above plots, we have overcome the overfitting problem. That's great!

10.7 Use the whole training set to train our DNN model

Now, let's use the whole training set to train our DNN model .

[509]:
```
# Try
# Using whole training data to train our model

model = models.Sequential()
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(100, activation='relu'))
```

```python
model.add(layers.Dropout(0.5))
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dropout(0.1))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

history = model.fit(OH_train_X,
                    train_Y,
                    epochs=200,
                    batch_size=50,
                    class_weight = class_weights)
```

```
Epoch 1/200
24000/24000 [==============================] - 4s 171us/step - loss: 0.6757 -
accuracy: 0.5931
Epoch 2/200
24000/24000 [==============================] - 2s 102us/step - loss: 0.6343 -
accuracy: 0.7444
Epoch 3/200
24000/24000 [==============================] - 3s 109us/step - loss: 0.6138 -
accuracy: 0.7359
Epoch 4/200
24000/24000 [==============================] - 2s 100us/step - loss: 0.6084 -
accuracy: 0.7309
Epoch 5/200
24000/24000 [==============================] - 3s 121us/step - loss: 0.6052 -
accuracy: 0.7268
Epoch 6/200
24000/24000 [==============================] - 3s 116us/step - loss: 0.5990 -
accuracy: 0.7132
Epoch 7/200
24000/24000 [==============================] - 2s 98us/step - loss: 0.5988 -
accuracy: 0.7145
Epoch 8/200
24000/24000 [==============================] - 2s 96us/step - loss: 0.5983 -
accuracy: 0.7154
Epoch 9/200
24000/24000 [==============================] - 2s 97us/step - loss: 0.5929 -
accuracy: 0.7193
Epoch 10/200
24000/24000 [==============================] - 2s 99us/step - loss: 0.5909 -
accuracy: 0.7004
```

```
Epoch 11/200
24000/24000 [==============================] - 2s 99us/step - loss: 0.5922 -
accuracy: 0.7250
Epoch 12/200
24000/24000 [==============================] - 2s 97us/step - loss: 0.5930 -
accuracy: 0.7232
Epoch 13/200
24000/24000 [==============================] - 2s 99us/step - loss: 0.5898 -
accuracy: 0.7279
Epoch 14/200
24000/24000 [==============================] - 2s 99us/step - loss: 0.5885 -
accuracy: 0.7297
Epoch 15/200
24000/24000 [==============================] - 2s 103us/step - loss: 0.5894 -
accuracy: 0.7249
Epoch 16/200
24000/24000 [==============================] - 2s 98us/step - loss: 0.5886 -
accuracy: 0.7299
Epoch 17/200
24000/24000 [==============================] - 2s 98us/step - loss: 0.5879 -
accuracy: 0.7276
Epoch 18/200
24000/24000 [==============================] - 2s 99us/step - loss: 0.5870 -
accuracy: 0.7251
Epoch 19/200
24000/24000 [==============================] - 2s 99us/step - loss: 0.5863 -
accuracy: 0.7242
Epoch 20/200
24000/24000 [==============================] - 2s 99us/step - loss: 0.5864 -
accuracy: 0.7270
Epoch 21/200
24000/24000 [==============================] - 2s 99us/step - loss: 0.5848 -
accuracy: 0.7285
Epoch 22/200
24000/24000 [==============================] - 2s 99us/step - loss: 0.5824 -
accuracy: 0.7303
Epoch 23/200
24000/24000 [==============================] - 2s 99us/step - loss: 0.5840 -
accuracy: 0.7258
Epoch 24/200
24000/24000 [==============================] - 2s 98us/step - loss: 0.5844 -
accuracy: 0.7306
Epoch 25/200
24000/24000 [==============================] - 2s 98us/step - loss: 0.5835 -
accuracy: 0.7300
Epoch 26/200
24000/24000 [==============================] - 2s 99us/step - loss: 0.5818 -
accuracy: 0.7330
```

```
Epoch 27/200
24000/24000 [==============================] - 2s 99us/step - loss: 0.5824 -
accuracy: 0.7335
Epoch 28/200
24000/24000 [==============================] - 2s 98us/step - loss: 0.5826 -
accuracy: 0.7352
Epoch 29/200
24000/24000 [==============================] - 2s 99us/step - loss: 0.5822 -
accuracy: 0.7310
Epoch 30/200
24000/24000 [==============================] - 2s 100us/step - loss: 0.5828 -
accuracy: 0.7326
Epoch 31/200
24000/24000 [==============================] - 2s 103us/step - loss: 0.5829 -
accuracy: 0.7359
Epoch 32/200
24000/24000 [==============================] - 2s 103us/step - loss: 0.5822 -
accuracy: 0.7375
Epoch 33/200
24000/24000 [==============================] - 2s 102us/step - loss: 0.5803 -
accuracy: 0.7360
Epoch 34/200
24000/24000 [==============================] - 2s 102us/step - loss: 0.5804 -
accuracy: 0.7408
Epoch 35/200
24000/24000 [==============================] - 2s 102us/step - loss: 0.5821 -
accuracy: 0.7369
Epoch 36/200
24000/24000 [==============================] - 2s 102us/step - loss: 0.5801 -
accuracy: 0.7447
Epoch 37/200
24000/24000 [==============================] - 3s 116us/step - loss: 0.5831 -
accuracy: 0.7334
Epoch 38/200
24000/24000 [==============================] - 2s 100us/step - loss: 0.5809 -
accuracy: 0.7358
Epoch 39/200
24000/24000 [==============================] - 2s 102us/step - loss: 0.5776 -
accuracy: 0.7472
Epoch 40/200
24000/24000 [==============================] - 2s 103us/step - loss: 0.5823 -
accuracy: 0.7451
Epoch 41/200
24000/24000 [==============================] - 2s 101us/step - loss: 0.5792 -
accuracy: 0.7495
Epoch 42/200
24000/24000 [==============================] - 3s 127us/step - loss: 0.5781 -
accuracy: 0.7389
```

```
Epoch 43/200
24000/24000 [==============================] - 3s 109us/step - loss: 0.5785 -
accuracy: 0.7509
Epoch 44/200
24000/24000 [==============================] - 3s 106us/step - loss: 0.5773 -
accuracy: 0.7406
Epoch 45/200
24000/24000 [==============================] - 3s 107us/step - loss: 0.5802 -
accuracy: 0.7459
Epoch 46/200
24000/24000 [==============================] - 3s 109us/step - loss: 0.5790 -
accuracy: 0.7465
Epoch 47/200
24000/24000 [==============================] - 3s 106us/step - loss: 0.5796 -
accuracy: 0.7442
Epoch 48/200
24000/24000 [==============================] - 3s 108us/step - loss: 0.5784 -
accuracy: 0.7513
Epoch 49/200
24000/24000 [==============================] - 3s 109us/step - loss: 0.5781 -
accuracy: 0.7402
Epoch 50/200
24000/24000 [==============================] - 3s 109us/step - loss: 0.5789 -
accuracy: 0.7406
Epoch 51/200
24000/24000 [==============================] - 3s 109us/step - loss: 0.5790 -
accuracy: 0.7428
Epoch 52/200
24000/24000 [==============================] - 3s 118us/step - loss: 0.5768 -
accuracy: 0.7397
Epoch 53/200
24000/24000 [==============================] - 3s 113us/step - loss: 0.5796 -
accuracy: 0.7452
Epoch 54/200
24000/24000 [==============================] - 3s 113us/step - loss: 0.5778 -
accuracy: 0.7415
Epoch 55/200
24000/24000 [==============================] - 3s 110us/step - loss: 0.5796 -
accuracy: 0.7492
Epoch 56/200
24000/24000 [==============================] - 3s 110us/step - loss: 0.5775 -
accuracy: 0.7447
Epoch 57/200
24000/24000 [==============================] - 3s 112us/step - loss: 0.5782 -
accuracy: 0.7386
Epoch 58/200
24000/24000 [==============================] - 3s 113us/step - loss: 0.5767 -
accuracy: 0.7396
```

```
Epoch 59/200
24000/24000 [==============================] - 3s 116us/step - loss: 0.5794 -
accuracy: 0.7381
Epoch 60/200
24000/24000 [==============================] - 3s 111us/step - loss: 0.5797 -
accuracy: 0.7414
Epoch 61/200
24000/24000 [==============================] - 3s 114us/step - loss: 0.5757 -
accuracy: 0.7393
Epoch 62/200
24000/24000 [==============================] - 3s 120us/step - loss: 0.5783 -
accuracy: 0.74000s - loss: 0.5782 - accuracy: 0.74
Epoch 63/200
24000/24000 [==============================] - 3s 114us/step - loss: 0.5808 -
accuracy: 0.7388
Epoch 64/200
24000/24000 [==============================] - 3s 126us/step - loss: 0.5793 -
accuracy: 0.7415
Epoch 65/200
24000/24000 [==============================] - 3s 125us/step - loss: 0.5792 -
accuracy: 0.7397
Epoch 66/200
24000/24000 [==============================] - 3s 122us/step - loss: 0.5765 -
accuracy: 0.7386
Epoch 67/200
24000/24000 [==============================] - 3s 114us/step - loss: 0.5790 -
accuracy: 0.7408
Epoch 68/200
24000/24000 [==============================] - 3s 119us/step - loss: 0.5755 -
accuracy: 0.7427
Epoch 69/200
24000/24000 [==============================] - 3s 130us/step - loss: 0.5796 -
accuracy: 0.7496
Epoch 70/200
24000/24000 [==============================] - 3s 106us/step - loss: 0.5793 -
accuracy: 0.7463
Epoch 71/200
24000/24000 [==============================] - 3s 105us/step - loss: 0.5791 -
accuracy: 0.7408
Epoch 72/200
24000/24000 [==============================] - 3s 108us/step - loss: 0.5776 -
accuracy: 0.7374
Epoch 73/200
24000/24000 [==============================] - 3s 119us/step - loss: 0.5774 -
accuracy: 0.7450
Epoch 74/200
24000/24000 [==============================] - 3s 106us/step - loss: 0.5767 -
accuracy: 0.7416
```

```
Epoch 75/200
24000/24000 [==============================] - 3s 111us/step - loss: 0.5777 -
accuracy: 0.7426
Epoch 76/200
24000/24000 [==============================] - 3s 108us/step - loss: 0.5794 -
accuracy: 0.7503
Epoch 77/200
24000/24000 [==============================] - 3s 139us/step - loss: 0.5786 -
accuracy: 0.7369
Epoch 78/200
24000/24000 [==============================] - 3s 137us/step - loss: 0.5764 -
accuracy: 0.7421
Epoch 79/200
24000/24000 [==============================] - 3s 117us/step - loss: 0.5784 -
accuracy: 0.7361
Epoch 80/200
24000/24000 [==============================] - 3s 113us/step - loss: 0.5776 -
accuracy: 0.7413
Epoch 81/200
24000/24000 [==============================] - 3s 108us/step - loss: 0.5781 -
accuracy: 0.7431
Epoch 82/200
24000/24000 [==============================] - 3s 110us/step - loss: 0.5767 -
accuracy: 0.7446
Epoch 83/200
24000/24000 [==============================] - 3s 110us/step - loss: 0.5774 -
accuracy: 0.7479
Epoch 84/200
24000/24000 [==============================] - 3s 115us/step - loss: 0.5744 -
accuracy: 0.7474
Epoch 85/200
24000/24000 [==============================] - 3s 109us/step - loss: 0.5755 -
accuracy: 0.7420
Epoch 86/200
24000/24000 [==============================] - 3s 127us/step - loss: 0.5766 -
accuracy: 0.7420
Epoch 87/200
24000/24000 [==============================] - 3s 121us/step - loss: 0.5751 -
accuracy: 0.7426
Epoch 88/200
24000/24000 [==============================] - 3s 108us/step - loss: 0.5795 -
accuracy: 0.7391
Epoch 89/200
24000/24000 [==============================] - 3s 110us/step - loss: 0.5758 -
accuracy: 0.7436
Epoch 90/200
24000/24000 [==============================] - 3s 111us/step - loss: 0.5750 -
accuracy: 0.7449
```

```
Epoch 91/200
24000/24000 [==============================] - 3s 110us/step - loss: 0.5794 -
accuracy: 0.7431
Epoch 92/200
24000/24000 [==============================] - 3s 111us/step - loss: 0.5776 -
accuracy: 0.7495
Epoch 93/200
24000/24000 [==============================] - 3s 112us/step - loss: 0.5748 -
accuracy: 0.7411
Epoch 94/200
24000/24000 [==============================] - 3s 112us/step - loss: 0.5787 -
accuracy: 0.7475
Epoch 95/200
24000/24000 [==============================] - 3s 134us/step - loss: 0.5781 -
accuracy: 0.7409
Epoch 96/200
24000/24000 [==============================] - 3s 128us/step - loss: 0.5806 -
accuracy: 0.7408
Epoch 97/200
24000/24000 [==============================] - 3s 120us/step - loss: 0.5770 -
accuracy: 0.7537
Epoch 98/200
24000/24000 [==============================] - 3s 109us/step - loss: 0.5758 -
accuracy: 0.7492
Epoch 99/200
24000/24000 [==============================] - 3s 131us/step - loss: 0.5787 -
accuracy: 0.7407
Epoch 100/200
24000/24000 [==============================] - 3s 142us/step - loss: 0.5793 -
accuracy: 0.7499
Epoch 101/200
24000/24000 [==============================] - 3s 119us/step - loss: 0.5756 -
accuracy: 0.7490
Epoch 102/200
24000/24000 [==============================] - 3s 109us/step - loss: 0.5781 -
accuracy: 0.7457
Epoch 103/200
24000/24000 [==============================] - 3s 109us/step - loss: 0.5772 -
accuracy: 0.7482
Epoch 104/200
24000/24000 [==============================] - 3s 134us/step - loss: 0.5785 -
accuracy: 0.7455
Epoch 105/200
24000/24000 [==============================] - 3s 128us/step - loss: 0.5770 -
accuracy: 0.7444
Epoch 106/200
24000/24000 [==============================] - 3s 124us/step - loss: 0.5770 -
accuracy: 0.7442
```

```
Epoch 107/200
24000/24000 [==============================] - 3s 130us/step - loss: 0.5790 -
accuracy: 0.7401
Epoch 108/200
24000/24000 [==============================] - 3s 106us/step - loss: 0.5740 -
accuracy: 0.7421
Epoch 109/200
24000/24000 [==============================] - 3s 107us/step - loss: 0.5760 -
accuracy: 0.7430
Epoch 110/200
24000/24000 [==============================] - 3s 107us/step - loss: 0.5752 -
accuracy: 0.7358
Epoch 111/200
24000/24000 [==============================] - 3s 110us/step - loss: 0.5771 -
accuracy: 0.7374
Epoch 112/200
24000/24000 [==============================] - 3s 108us/step - loss: 0.5748 -
accuracy: 0.7416
Epoch 113/200
24000/24000 [==============================] - 3s 108us/step - loss: 0.5756 -
accuracy: 0.7433
Epoch 114/200
24000/24000 [==============================] - 3s 107us/step - loss: 0.5741 -
accuracy: 0.7382
Epoch 115/200
24000/24000 [==============================] - 3s 108us/step - loss: 0.5753 -
accuracy: 0.7379
Epoch 116/200
24000/24000 [==============================] - 3s 109us/step - loss: 0.5751 -
accuracy: 0.7401
Epoch 117/200
24000/24000 [==============================] - 3s 107us/step - loss: 0.5772 -
accuracy: 0.7480
Epoch 118/200
24000/24000 [==============================] - 3s 110us/step - loss: 0.5745 -
accuracy: 0.7437
Epoch 119/200
24000/24000 [==============================] - 3s 116us/step - loss: 0.5747 -
accuracy: 0.7440
Epoch 120/200
24000/24000 [==============================] - 3s 112us/step - loss: 0.5755 -
accuracy: 0.7461
Epoch 121/200
24000/24000 [==============================] - 3s 112us/step - loss: 0.5750 -
accuracy: 0.7490
Epoch 122/200
24000/24000 [==============================] - 3s 111us/step - loss: 0.5764 -
accuracy: 0.7490
```

```
Epoch 123/200
24000/24000 [==============================] - 3s 123us/step - loss: 0.5750 -
accuracy: 0.7430
Epoch 124/200
24000/24000 [==============================] - 3s 117us/step - loss: 0.5748 -
accuracy: 0.7438
Epoch 125/200
24000/24000 [==============================] - 3s 115us/step - loss: 0.5755 -
accuracy: 0.7449
Epoch 126/200
24000/24000 [==============================] - 3s 113us/step - loss: 0.5759 -
accuracy: 0.7456
Epoch 127/200
24000/24000 [==============================] - 3s 113us/step - loss: 0.5764 -
accuracy: 0.7500
Epoch 128/200
24000/24000 [==============================] - 3s 110us/step - loss: 0.5785 -
accuracy: 0.7440
Epoch 129/200
24000/24000 [==============================] - 3s 113us/step - loss: 0.5742 -
accuracy: 0.7496
Epoch 130/200
24000/24000 [==============================] - 3s 111us/step - loss: 0.5760 -
accuracy: 0.7385
Epoch 131/200
24000/24000 [==============================] - 3s 110us/step - loss: 0.5761 -
accuracy: 0.7481
Epoch 132/200
24000/24000 [==============================] - 3s 110us/step - loss: 0.5768 -
accuracy: 0.7448
Epoch 133/200
24000/24000 [==============================] - 3s 109us/step - loss: 0.5752 -
accuracy: 0.7505
Epoch 134/200
24000/24000 [==============================] - 3s 116us/step - loss: 0.5742 -
accuracy: 0.7389
Epoch 135/200
24000/24000 [==============================] - 3s 110us/step - loss: 0.5734 -
accuracy: 0.7399
Epoch 136/200
24000/24000 [==============================] - 3s 111us/step - loss: 0.5742 -
accuracy: 0.7416
Epoch 137/200
24000/24000 [==============================] - 3s 113us/step - loss: 0.5765 -
accuracy: 0.7400
Epoch 138/200
24000/24000 [==============================] - 3s 110us/step - loss: 0.5754 -
accuracy: 0.7440
```

```
Epoch 139/200
24000/24000 [==============================] - 3s 109us/step - loss: 0.5775 -
accuracy: 0.7426
Epoch 140/200
24000/24000 [==============================] - 3s 110us/step - loss: 0.5764 -
accuracy: 0.7451
Epoch 141/200
24000/24000 [==============================] - 3s 115us/step - loss: 0.5770 -
accuracy: 0.7484
Epoch 142/200
24000/24000 [==============================] - 3s 109us/step - loss: 0.5757 -
accuracy: 0.7412
Epoch 143/200
24000/24000 [==============================] - 3s 109us/step - loss: 0.5779 -
accuracy: 0.7412
Epoch 144/200
24000/24000 [==============================] - 3s 116us/step - loss: 0.5741 -
accuracy: 0.7480
Epoch 145/200
24000/24000 [==============================] - 3s 110us/step - loss: 0.5776 -
accuracy: 0.7464
Epoch 146/200
24000/24000 [==============================] - 3s 122us/step - loss: 0.5760 -
accuracy: 0.7530
Epoch 147/200
24000/24000 [==============================] - 3s 128us/step - loss: 0.5739 -
accuracy: 0.7492
Epoch 148/200
24000/24000 [==============================] - 3s 112us/step - loss: 0.5758 -
accuracy: 0.7484
Epoch 149/200
24000/24000 [==============================] - 3s 115us/step - loss: 0.5744 -
accuracy: 0.7492
Epoch 150/200
24000/24000 [==============================] - 3s 111us/step - loss: 0.5747 -
accuracy: 0.7472
Epoch 151/200
24000/24000 [==============================] - 3s 109us/step - loss: 0.5780 -
accuracy: 0.7458
Epoch 152/200
24000/24000 [==============================] - 3s 111us/step - loss: 0.5758 -
accuracy: 0.7491
Epoch 153/200
24000/24000 [==============================] - 3s 113us/step - loss: 0.5789 -
accuracy: 0.7384
Epoch 154/200
24000/24000 [==============================] - 3s 112us/step - loss: 0.5752 -
accuracy: 0.7462
```

```
Epoch 155/200
24000/24000 [==============================] - 3s 117us/step - loss: 0.5777 -
accuracy: 0.7448
Epoch 156/200
24000/24000 [==============================] - 3s 111us/step - loss: 0.5766 -
accuracy: 0.7406
Epoch 157/200
24000/24000 [==============================] - 3s 110us/step - loss: 0.5767 -
accuracy: 0.7377
Epoch 158/200
24000/24000 [==============================] - 3s 111us/step - loss: 0.5771 -
accuracy: 0.7347
Epoch 159/200
24000/24000 [==============================] - 3s 111us/step - loss: 0.5749 -
accuracy: 0.7428
Epoch 160/200
24000/24000 [==============================] - 3s 111us/step - loss: 0.5747 -
accuracy: 0.7495
Epoch 161/200
24000/24000 [==============================] - 3s 111us/step - loss: 0.5783 -
accuracy: 0.7455
Epoch 162/200
24000/24000 [==============================] - 3s 112us/step - loss: 0.5755 -
accuracy: 0.7386
Epoch 163/200
24000/24000 [==============================] - 3s 115us/step - loss: 0.5750 -
accuracy: 0.7499
Epoch 164/200
24000/24000 [==============================] - 3s 111us/step - loss: 0.5750 -
accuracy: 0.7393
Epoch 165/200
24000/24000 [==============================] - 3s 110us/step - loss: 0.5762 -
accuracy: 0.7423
Epoch 166/200
24000/24000 [==============================] - 3s 118us/step - loss: 0.5743 -
accuracy: 0.7496
Epoch 167/200
24000/24000 [==============================] - 3s 120us/step - loss: 0.5774 -
accuracy: 0.7474
Epoch 168/200
24000/24000 [==============================] - 3s 126us/step - loss: 0.5808 -
accuracy: 0.7355
Epoch 169/200
24000/24000 [==============================] - 3s 136us/step - loss: 0.5753 -
accuracy: 0.7321
Epoch 170/200
24000/24000 [==============================] - 3s 132us/step - loss: 0.5761 -
accuracy: 0.7374
```

```
Epoch 171/200
24000/24000 [==============================] - 3s 128us/step - loss: 0.5759 -
accuracy: 0.7426
Epoch 172/200
24000/24000 [==============================] - 3s 123us/step - loss: 0.5764 -
accuracy: 0.7388
Epoch 173/200
24000/24000 [==============================] - 3s 139us/step - loss: 0.5762 -
accuracy: 0.7511
Epoch 174/200
24000/24000 [==============================] - 3s 136us/step - loss: 0.5762 -
accuracy: 0.7505
Epoch 175/200
24000/24000 [==============================] - 3s 142us/step - loss: 0.5738 -
accuracy: 0.7529
Epoch 176/200
24000/24000 [==============================] - 3s 133us/step - loss: 0.5743 -
accuracy: 0.7505
Epoch 177/200
24000/24000 [==============================] - 3s 114us/step - loss: 0.5759 -
accuracy: 0.7444
Epoch 178/200
24000/24000 [==============================] - 3s 108us/step - loss: 0.5811 -
accuracy: 0.7487
Epoch 179/200
24000/24000 [==============================] - 3s 137us/step - loss: 0.5765 -
accuracy: 0.7474
Epoch 180/200
24000/24000 [==============================] - 3s 117us/step - loss: 0.5741 -
accuracy: 0.7479
Epoch 181/200
24000/24000 [==============================] - 3s 133us/step - loss: 0.5764 -
accuracy: 0.7424
Epoch 182/200
24000/24000 [==============================] - 3s 104us/step - loss: 0.5742 -
accuracy: 0.7508
Epoch 183/200
24000/24000 [==============================] - 3s 105us/step - loss: 0.5747 -
accuracy: 0.7484
Epoch 184/200
24000/24000 [==============================] - 3s 125us/step - loss: 0.5721 -
accuracy: 0.7455
Epoch 185/200
24000/24000 [==============================] - 3s 115us/step - loss: 0.5775 -
accuracy: 0.7426
Epoch 186/200
24000/24000 [==============================] - 3s 120us/step - loss: 0.5764 -
accuracy: 0.7453
```

```
Epoch 187/200
24000/24000 [==============================] - 3s 116us/step - loss: 0.5758 -
accuracy: 0.7446
Epoch 188/200
24000/24000 [==============================] - 3s 107us/step - loss: 0.5763 -
accuracy: 0.7384
Epoch 189/200
24000/24000 [==============================] - 3s 122us/step - loss: 0.5764 -
accuracy: 0.7433
Epoch 190/200
24000/24000 [==============================] - 3s 108us/step - loss: 0.5767 -
accuracy: 0.7430
Epoch 191/200
24000/24000 [==============================] - 3s 114us/step - loss: 0.5760 -
accuracy: 0.7450
Epoch 192/200
24000/24000 [==============================] - 3s 143us/step - loss: 0.5771 -
accuracy: 0.7435
Epoch 193/200
24000/24000 [==============================] - 3s 140us/step - loss: 0.5745 -
accuracy: 0.7368
Epoch 194/200
24000/24000 [==============================] - 3s 142us/step - loss: 0.5790 -
accuracy: 0.7392
Epoch 195/200
24000/24000 [==============================] - 3s 132us/step - loss: 0.5798 -
accuracy: 0.7425
Epoch 196/200
24000/24000 [==============================] - 3s 128us/step - loss: 0.5774 -
accuracy: 0.7392
Epoch 197/200
24000/24000 [==============================] - 3s 115us/step - loss: 0.5789 -
accuracy: 0.7372
Epoch 198/200
24000/24000 [==============================] - 3s 109us/step - loss: 0.5756 -
accuracy: 0.7396
Epoch 199/200
24000/24000 [==============================] - 3s 107us/step - loss: 0.5744 -
accuracy: 0.7473
Epoch 200/200
24000/24000 [==============================] - 3s 110us/step - loss: 0.5762 -
accuracy: 0.7387
```

10.8 predict the target values of testing set.

[516]:
```
preds_Y = model.predict(OH_test_X)
preds_Y=(preds_Y>0.5).astype(int)
```

10.9 Confusion matrix, accuracy, precision, recall, F1 score

[510]:
```
accuracy = accuracy_score(test_Y, preds_Y)
precision = precision_score(test_Y, preds_Y)
recall = recall_score(test_Y, preds_Y)
f1 = f1_score(test_Y, preds_Y)
c_matrix = confusion_matrix(test_Y, preds_Y)
print('accuracy: '+str(accuracy)+'\n')
print('precision: '+str(precision)+'\n')
print('recall: '+str(recall)+'\n')
print('F1 score: '+str(f1)+'\n')
print('Confusion matrix: ')
print(c_matrix)
```

```
accuracy: 0.8075

precision: 0.5908304498269896

recall: 0.5003663003663004

F1 score: 0.5418484728282427

Confusion matrix:
[[4162  473]
 [ 682  683]]
```

11 Conclusion

In this report, we use different machine learning models to predict default of credit cards next month:

CatBoost, XGBoost, LightGBM, Deep Neural Networks (DNN model).

To apply different models, I did various data pre-processing, such as one-hot encoding and standarization.

By using CatBoost, XGBoost, and LightGBM, we can get accuracy above 80% in our testing set.

By using DNN models, we have lots of flexibility of network architecture. In particular, by using "dropout" techniques and enough training epochs, we can get pretty great result in our testing set.

11.1 Compare different models

11.1.1 CatBoost:

accuracy: 0.8221666666666667

precision: 0.7217261904761905

recall: 0.3553113553113553

F1 score: 0.47619047619047616

11.1.2 XGBoost:

accuracy: 0.8188333333333333

precision: 0.7044117647058824

recall: 0.3509157509157509

F1 score: 0.4684596577017115

11.1.3 LightGBM:

accuracy: 0.825

precision: 0.7292576419213974

recall: 0.367032967032967

F1 score: 0.48830409356725146

11.1.4 DNN model:

accuracy: 0.8075

precision: 0.5908304498269896

recall: 0.5003663003663004

F1 score: 0.5418484728282427

According to above information, our DNN model is the best model in terms of F1 score.

One thing to note is that, because we are dealing with imbalanced dataset, F1 score will be a more reasonable metric than accuracy. Therefore, in this report, DNN model will be the best model when we want to predict default of credit cards next month.

11.2 Future works

In our future work, I will spend more time to tune the hyperparameters in our DNN models, such as numbers of hidden layers and units.

Hopefully, we can get even better results by using DNN models after tuning the hyperparameters.