credit-risk-01(Beta)

June 9, 2023

[ ]:
```
import numpy as np
import pandas as pd

from sklearn.preprocessing import LabelEncoder, MinMaxScaler

from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import GridSearchCVP
```

[ ]:
```
credit_c = pd.read_csv("dataset/credit_customers.csv")
credit = credit_c.copy() # create a copy of original dataset
credit.head()
```

[ ]:

	checking_status	duration	credit_history	\
0	<0	6.0	critical/other existing credit	
1	0<=X<200	48.0	existing paid	
2	no checking	12.0	critical/other existing credit	
3	<0	42.0	existing paid	
4	<0	24.0	delayed previously	

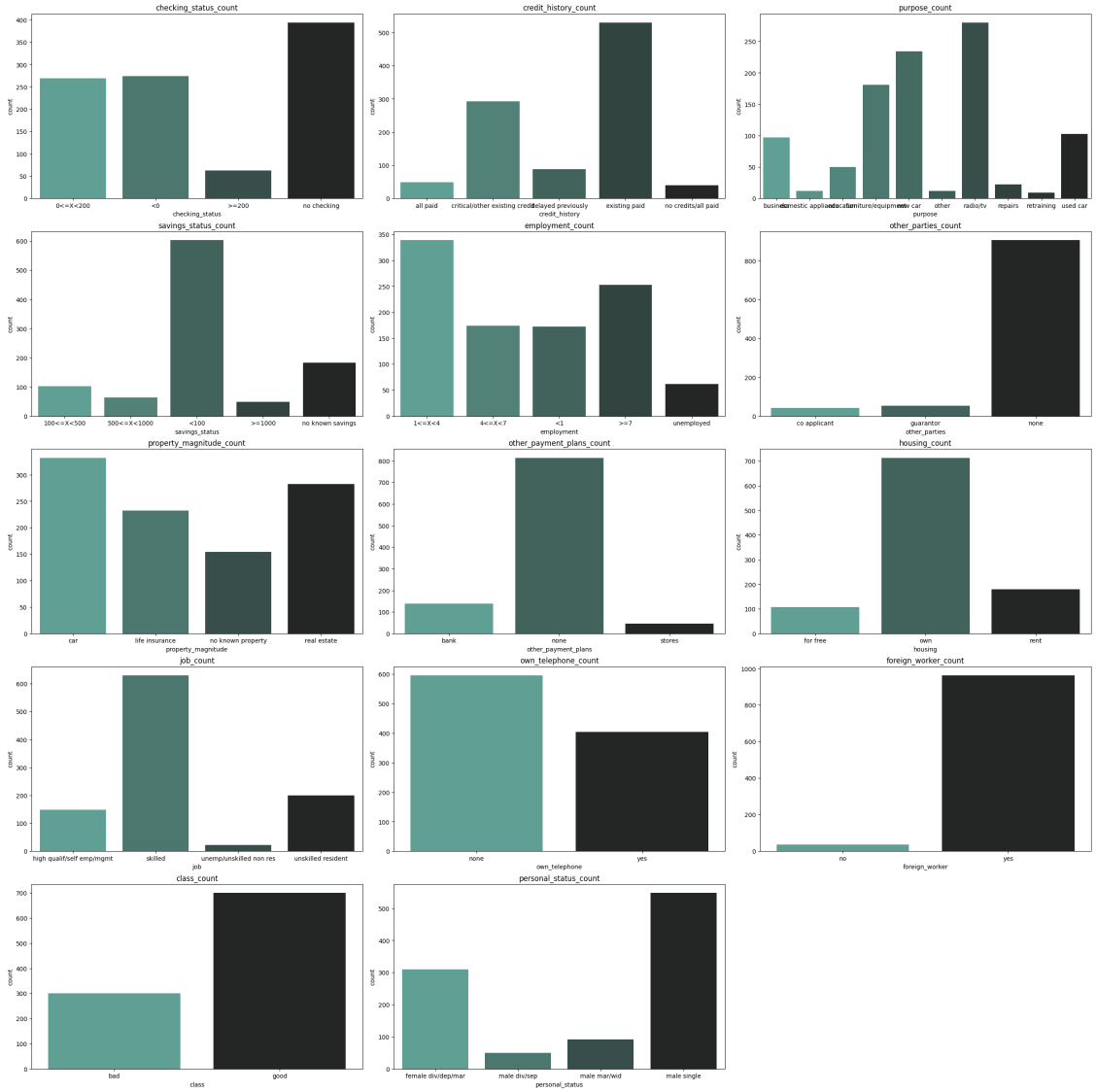
	purpose	credit_amount	savings_status	employment	\
0	radio/tv	1169.0	no known savings	>=7	
1	radio/tv	5951.0	<100	1<=X<4	
2	education	2096.0	<100	4<=X<7	
3	furniture/equipment	7882.0	<100	4<=X<7	
4	new car	4870.0	<100	1<=X<4	

	installment_commitment	personal_status	other_parties	…	\
0	4.0	male single	none	…	
1	2.0	female div/dep/mar	none	…	
2	2.0	male single	none	…	
3	2.0	male single	guarantor	…	
4	3.0	male single	none	…	

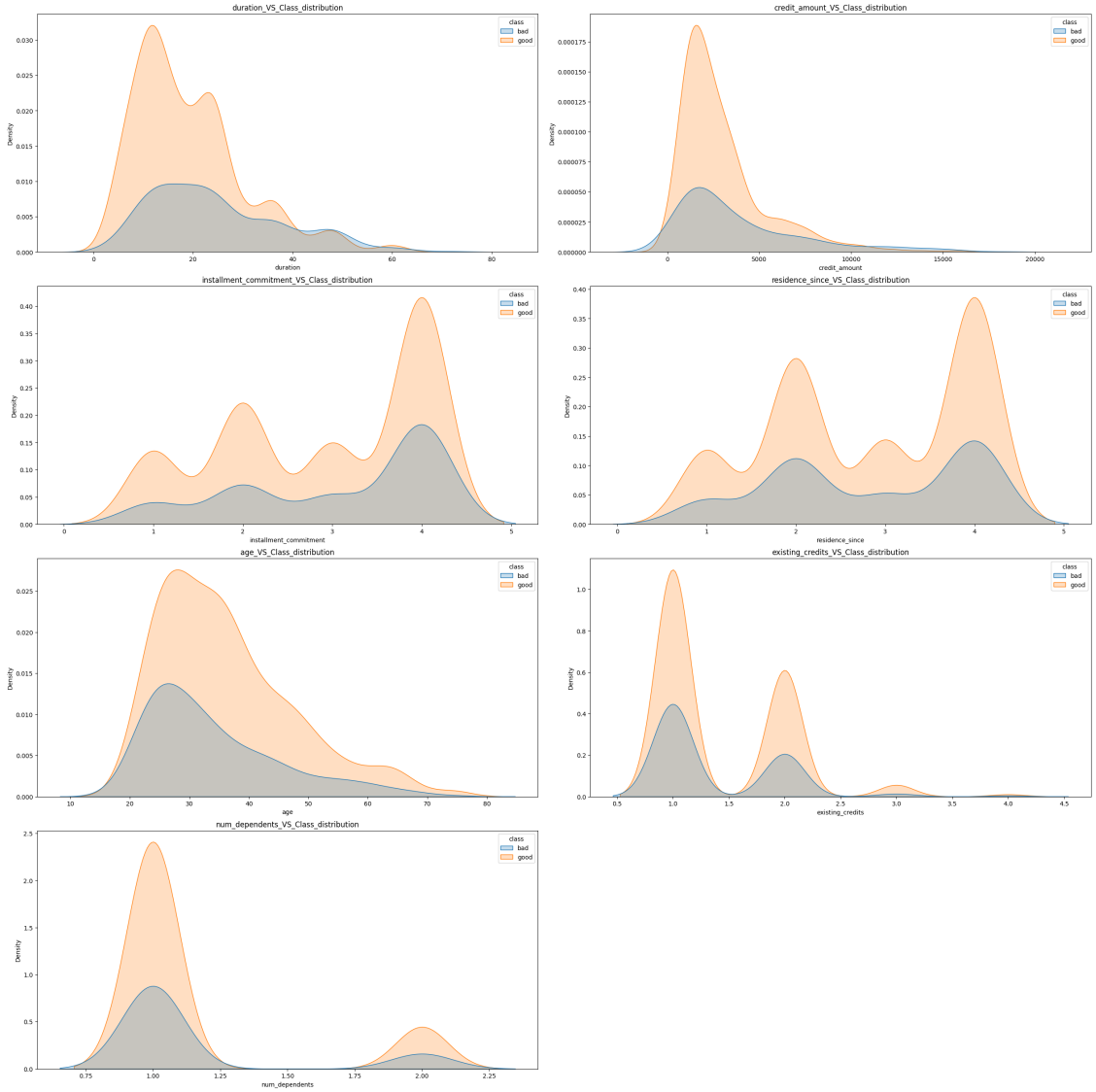
	property_magnitude	age	other_payment_plans	housing	existing_credits	\
0	real estate	67.0	none	own	2.0	
1	real estate	22.0	none	own	1.0	
2	real estate	49.0	none	own	1.0	

```
3      life insurance  45.0                 none  for free             1.0
4   no known property  53.0                 none  for free             2.0

                   job num_dependents  own_telephone foreign_worker class
0              skilled            1.0            yes            yes  good
1              skilled            1.0           none            yes   bad
2  unskilled resident            2.0           none            yes  good
3              skilled            2.0           none            yes  good
4              skilled            2.0           none            yes   bad

[5 rows x 21 columns]
```

[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

lst=['checking_status', 'credit_history', 'purpose',␣
 ↪'savings_status','employment', 'other_parties',␣
 ↪'property_magnitude','other_payment_plans', 'housing', 'job',␣
 ↪'own_telephone',
     'foreign_worker', 'class', 'personal_status']
plt.figure(figsize=(25,25),layout='constrained')
for i in range(len(array)):
    plt.subplot(5,3,i+1)
    sns.countplot(data=credit, x=array[i], palette='dark:#5A9_r',)
    plt.title(lst[i]+'_count',)
plt.show()
```



[ ]:
```
lst=['duration', 'credit_amount', 'installment_commitment','residence_since',␣
 ↪'age', 'existing_credits', 'num_dependents']
plt.figure(figsize=(25,25),layout='constrained')
for i in range(len(lst)):
    plt.subplot(4,2,i+1)
    sns.kdeplot(data=credit,x=lst[i],hue='class',fill=True)
    plt.title(lst[i]+'_VS_Class_distribution')
plt.show()
```



1 Data preparation

[ ]:
```
# Checking Null Values
credit.isnull().sum()
```

[ ]:
```
checking_status          0
duration                 0
credit_history           0
purpose                  0
credit_amount            0
savings_status           0
employment               0
```

```
installment_commitment          0
personal_status                 0
other_parties                   0
residence_since                 0
property_magnitude              0
age                             0
other_payment_plans             0
housing                         0
existing_credits                0
job                             0
num_dependents                  0
own_telephone                   0
foreign_worker                  0
class                           0
checking_status_encoder         0
credit_history_encoder          0
purpose_encoder                 0
savings_status_encoder          0
employment_encoder              0
personal_status_encoder         0
other_parties_encoder           0
property_magnitude_encoder      0
other_payment_plans_encoder     0
housing_encoder                 0
job_encoder                     0
own_telephone_encoder           0
foreign_worker_encoder          0
class_encoder                   0
dtype: int64
```

[ ]:
```python
# split and remove personal status -> gender & marital_status
new = credit['personal_status'].str.split(" ", n = 1, expand = True)

credit['gender'] = new[0]
credit['marital_status'] = new[1]

credit.drop('personal_status', axis=1, inplace=True)
```

[ ]:
```python
# Make objects categorical

obj = credit.select_dtypes(include='object')

for i in list(obj.columns):
    credit[i]=credit[i].astype('category')

credit_for_dnn = credit.copy() # copy the dataset to use for dnn
```

```
print(f"'Checking status' feature: \n{credit['checking_status'].head()}")
```

```
'Checking status' feature:
0             <0
1       0<=X<200
2    no checking
3             <0
4             <0
Name: checking_status, dtype: category
Categories (4, object): ['0<=X<200', '<0', '>=200', 'no checking']
```

[ ]:
```
# Encode features

cat = credit.select_dtypes(include='category')

for i in list(cat.columns):
    credit[i+'_encoder']=LabelEncoder().fit_transform(credit[i])
    # TODO: Change LabelEncoder to OrdinalEncoder?

num = credit.select_dtypes(include=[np.number])

scl = pd.DataFrame(MinMaxScaler().fit_transform(num.to_numpy()),columns=num.
 ↪columns)
```

[ ]:
```
scl.head()
```

[ ]:
```
   duration  credit_amount  installment_commitment  residence_since       age  \
0  0.029412       0.050567                1.000000         1.000000  0.857143
1  0.647059       0.313690                0.333333         0.333333  0.053571
2  0.117647       0.101574                0.333333         0.666667  0.535714
3  0.558824       0.419941                0.333333         1.000000  0.464286
4  0.294118       0.254209                0.666667         1.000000  0.607143

   existing_credits  num_dependents  checking_status_encoder  \
0          0.333333             0.0                 0.333333
1          0.000000             0.0                 0.000000
2          0.000000             1.0                 1.000000
3          0.000000             1.0                 0.333333
4          0.333333             1.0                 0.333333

   credit_history_encoder  purpose_encoder  …  employment_encoder  \
0                    0.25         0.666667  …                0.75
1                    0.75         0.666667  …                0.00
2                    0.25         0.222222  …                0.25
3                    0.75         0.333333  …                0.25
4                    0.50         0.444444  …                0.00

   personal_status_encoder  other_parties_encoder  property_magnitude_encoder  \
```

```
0                      1.0                    1.0                    1.000000
1                      0.0                    1.0                    1.000000
2                      1.0                    1.0                    1.000000
3                      1.0                    0.5                    0.333333
4                      1.0                    1.0                    0.666667

   other_payment_plans_encoder  housing_encoder  job_encoder  \
0                          0.5              0.5     0.333333
1                          0.5              0.5     0.333333
2                          0.5              0.5     1.000000
3                          0.5              0.0     0.333333
4                          0.5              0.0     0.333333

   own_telephone_encoder  foreign_worker_encoder  class_encoder
0                    1.0                     1.0            1.0
1                    0.0                     1.0            0.0
2                    0.0                     1.0            1.0
3                    0.0                     1.0            1.0
4                    0.0                     1.0            0.0

[5 rows x 21 columns]
```

[ ]:
```
# Split dataset

X=scl.drop('class_encoder',axis=1)
Y=scl['class_encoder']


X_train = X[0:700]
Y_train = Y[0:700]

X_test = X[700:900]
Y_test = Y[700:900]

X_val = X[900:999]
Y_val = Y[900:999]

# check for class count
from collections import Counter
print('Orig shape {}'.format((Counter(Y_train))))
```

```
Orig shape Counter({1.0: 493, 0.0: 207})
```

2 Random Forest

[ ]:
```
rf = RandomForestClassifier(random_state=42, n_jobs=-1)

rf.fit(X_train, Y_train)

print('train score: ', rf.score(X_train, Y_train))
print('val score: ', rf.score(X_val, Y_val))
```

```
train score:  1.0
val score:  0.797979797979798
```

[ ]:
```
# try with cross-validation

rf = RandomForestClassifier(random_state=42, n_jobs=-1)

params = {
    'max_depth': [2,3,4,5,10,15,20,30],
    'min_samples_leaf': [2,3,4,5,6,7,8,9,10,11],
    'n_estimators': [10,20,30,40,53,60,70,80],
}

grid_search = GridSearchCV(estimator=rf,
                           param_grid=params,
                           cv=10,
                           n_jobs=-1, verbose=1, scoring="accuracy")
```

[ ]:
```
grid_search.fit(X_train, Y_train)
```

[ ]:
```
rf_best = grid_search.best_estimator_
rf_best
```

[ ]:
```
RandomForestClassifier(max_depth=20, min_samples_leaf=6, n_estimators=30,
                       n_jobs=-1, random_state=42)
```

[ ]:
```
rf_best = RandomForestClassifier(max_depth=20, min_samples_leaf=6,␣
 ↪n_estimators=30,
                       n_jobs=-1, random_state=42)
```

[ ]:
```
rf_best.fit(X_train, Y_train)
print('train score: ', rf_best.score(X_train, Y_train))
print('val score: ', rf_best.score(X_val, Y_val))
```

```
train score:  0.8728571428571429
val score:  0.7676767676767676
```

2.1 Oversampling

[ ]:
```
from imblearn.over_sampling import RandomOverSampler

over_sampler = RandomOverSampler(random_state=42)

X_train_ores, y_train_ores = over_sampler.fit_resample(X_train, Y_train)

X_train_ores.shape, y_train_ores.shape

print('Orig shape {}'.format((Counter(Y_train))))
print('New shape {}'.format((Counter(y_train_ores))))
```

```
Orig shape Counter({1.0: 493, 0.0: 207})
New shape Counter({1.0: 493, 0.0: 493})
```

[ ]:
```
grid_search.fit(X_train_ores, y_train_ores)
```

```
Fitting 10 folds for each of 640 candidates, totalling 6400 fits
```

[ ]:
```
GridSearchCV(cv=10,
             estimator=RandomForestClassifier(n_jobs=-1, random_state=42),
             n_jobs=-1,
             param_grid={'max_depth': [2, 3, 4, 5, 10, 15, 20, 30],
                         'min_samples_leaf': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
                         'n_estimators': [10, 20, 30, 40, 53, 60, 70, 80]},
             scoring='accuracy', verbose=1)
```

[ ]:
```
rf_os_best = grid_search.best_estimator_
rf_os_best
```

[ ]:
```
RandomForestClassifier(max_depth=15, min_samples_leaf=2, n_estimators=53,
                       n_jobs=-1, random_state=42)
```

[ ]:
```
rf_os_best = RandomForestClassifier(max_depth=15, min_samples_leaf=2,␣
 ↪n_estimators=53,
                       n_jobs=-1, random_state=42)
```

[ ]:
```
rf_os_best.fit(X_train_ores, y_train_ores)
print('train score: ', rf_os_best.score(X_train_ores, y_train_ores))
print('val score: ', rf_os_best.score(X_val, Y_val))
```

```
train score:  0.9918864097363083
val score:  0.7373737373737373
```

2.2 Undersampling

[ ]:
```
from imblearn.under_sampling import RandomUnderSampler

under_sampler = RandomUnderSampler(random_state=42)

X_train_ures, y_train_ures = under_sampler.fit_resample(X_train, Y_train)

X_train_ures.shape, y_train_ures.shape

print('Orig shape {}'.format((Counter(Y_train))))
print('New shape {}'.format((Counter(y_train_ures))))
```

```
Orig shape Counter({1.0: 493, 0.0: 207})
New shape Counter({0.0: 207, 1.0: 207})
```

[ ]:
```
grid_search.fit(X_train_ures, y_train_ures)
```

```
Fitting 10 folds for each of 640 candidates, totalling 6400 fits
```

[ ]:
```
GridSearchCV(cv=10,
             estimator=RandomForestClassifier(n_jobs=-1, random_state=42),
             n_jobs=-1,
             param_grid={'max_depth': [2, 3, 4, 5, 10, 15, 20, 30],
                         'min_samples_leaf': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
                         'n_estimators': [10, 20, 30, 40, 53, 60, 70, 80]},
             scoring='accuracy', verbose=1)
```

[ ]:
```
rf_us_best = grid_search.best_estimator_
rf_us_best
```

[ ]:
```
RandomForestClassifier(max_depth=20, min_samples_leaf=2, n_estimators=80,
                       n_jobs=-1, random_state=42)
```

[ ]:
```
rf_us_best = RandomForestClassifier(max_depth=20, min_samples_leaf=2,␣
 ↪n_estimators=80,
                       n_jobs=-1, random_state=42)
```

[ ]:
```
rf_us_best.fit(X_train_ures, y_train_ures)
print('train score: ', rf_us_best.score(X_train_ures, y_train_ures))
print('val score: ', rf_us_best.score(X_val, Y_val))
```

```
train score:  0.9951690821256038
val score:  0.6767676767676768
```

2.3 Stratified Shuffle Split

[ ]:
```
# Split dataset into train and rest

from sklearn.model_selection import StratifiedShuffleSplit

splitter = StratifiedShuffleSplit(n_splits=10, test_size=0.3, random_state=42)

for train, rest in splitter.split(X,Y):
    X_train_SS = X.iloc[train]
    y_train_SS = Y.iloc[train]
    X_rest_SS = X.iloc[rest]
    y_rest_SS = Y.iloc[rest]

print(y_train_SS.value_counts())
print(y_rest_SS.value_counts())
print(X_train_SS.shape, X_rest_SS.shape)
```

```
1.0    490
0.0    210
Name: class_encoder, dtype: int64
1.0    210
0.0     90
Name: class_encoder, dtype: int64
(700, 21) (300, 21)
```

[ ]:
```
# Split rest into validation and test

splitter = StratifiedShuffleSplit(n_splits=10, test_size=0.5, random_state=42)

for val, test in splitter.split(X_rest_SS, y_rest_SS):
    X_val_SS = X_rest_SS.iloc[val]
    y_val_SS = y_rest_SS.iloc[val]
    X_test_SS = X_rest_SS.iloc[test]
    y_test_SS = y_rest_SS.iloc[test]

print(y_val_SS.value_counts())
print(y_test_SS.value_counts())
print(X_val_SS.shape, X_test_SS.shape)
```

```
1.0    105
0.0     45
Name: class_encoder, dtype: int64
1.0    105
0.0     45
Name: class_encoder, dtype: int64
(150, 21) (150, 21)
```

[ ]:
```
grid_search.fit(X_train_SS, y_train_SS)
```

```
Fitting 10 folds for each of 640 candidates, totalling 6400 fits
```

[ ]:
```
GridSearchCV(cv=10,
             estimator=RandomForestClassifier(n_jobs=-1, random_state=42),
             n_jobs=-1,
             param_grid={'max_depth': [2, 3, 4, 5, 10, 15, 20, 30],
                         'min_samples_leaf': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
                         'n_estimators': [10, 20, 30, 40, 53, 60, 70, 80]},
             scoring='accuracy', verbose=1)
```

[ ]:
```
rf_ss_best = grid_search.best_estimator_
rf_ss_best
```

[ ]:
```
RandomForestClassifier(max_depth=20, min_samples_leaf=3, n_estimators=40,
                       n_jobs=-1, random_state=42)
```

[ ]:
```
rf_ss_best = RandomForestClassifier(max_depth=20, min_samples_leaf=3,␣
 ↪n_estimators=40,
                       n_jobs=-1, random_state=42)
```

[ ]:
```
rf_ss_best.fit(X_train_SS, y_train_SS)
print('train score: ', rf_ss_best.score(X_train_SS, y_train_SS))
print('val score: ', rf_ss_best.score(X_val_SS, y_val_SS))
```

```
train score:  0.9142857142857143
val score:  0.7733333333333333
```

2.4 Oversampling with SMOTEN

[ ]:
```
from imblearn.over_sampling import SMOTEN

over_sampler_SMOTEN = SMOTEN(sampling_strategy=1, random_state=42,␣
 ↪k_neighbors=3, n_jobs=-1)

X_train_ores_smoten, y_train_ores_smoten = over_sampler_SMOTEN.fit_resample(X,␣
 ↪Y)

from collections import Counter
print('Orig shape {}'.format((Counter(Y_train))))
print('New shape {}'.format((Counter(y_train_ores_smoten))))
```

```
Orig shape Counter({1.0: 493, 0.0: 207})
New shape Counter({1.0: 700, 0.0: 700})
```

```
/home/filip/py-env/aai-2023-1/lib/python3.9/site-
packages/imblearn/over_sampling/_smote/base.py:858: FutureWarning: The parameter
`n_jobs` has been deprecated in 0.10 and will be removed in 0.12. You can pass
```

```
an nearest neighbors estimator where `n_jobs` is already set instead.
  warnings.warn(
```

[ ]:
```
X_train_ores_smoten.shape
```

[ ]:
```
(1400, 20)
```

[ ]:
```
grid_search.fit(X_train_ores_smoten, y_train_ores_smoten)
```

```
Fitting 10 folds for each of 640 candidates, totalling 6400 fits
```

[ ]:
```
GridSearchCV(cv=10,
             estimator=RandomForestClassifier(n_jobs=-1, random_state=42),
             n_jobs=-1,
             param_grid={'max_depth': [2, 3, 4, 5, 10, 15, 20, 30],
                         'min_samples_leaf': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
                         'n_estimators': [10, 20, 30, 40, 53, 60, 70, 80]},
             scoring='accuracy', verbose=1)
```

[ ]:
```
rf_os_smoten_best = grid_search.best_estimator_
rf_os_smoten_best
```

[ ]:
```
RandomForestClassifier(max_depth=15, min_samples_leaf=2, n_estimators=80,
                       n_jobs=-1, random_state=42)
```

[ ]:
```
rf_os_smoten_best = RandomForestClassifier(max_depth=15, min_samples_leaf=2,␣
 ↪n_estimators=80,
                       n_jobs=-1, random_state=42)
```

[ ]:
```
rf_os_smoten_best.fit(X_train_ores_smoten, y_train_ores_smoten)
print('train score: ', rf_os_smoten_best.score(X_train_ores_smoten,␣
 ↪y_train_ores_smoten))
print('val score: ', rf_os_smoten_best.score(X_val, Y_val))
```

```
train score:  0.9828571428571429
val score:  0.9797979797979798
```

3 Evaluation

[ ]:
```
from sklearn.metrics import␣
 ↪confusion_matrix,classification_report,accuracy_score,f1_score

def evaluate(model, X_train, Y_train, X_val, Y_val):
    y_train_pred = model.predict(X_train)
    y_val_pred = model.predict(X_val)

    conf_train = confusion_matrix(Y_train, y_train_pred)
    acc_train = accuracy_score(Y_train, y_train_pred)
```

```
    f1_train = f1_score(Y_train, y_train_pred)

    conf_val = confusion_matrix(Y_val, y_val_pred)
    acc_val = accuracy_score(Y_val, y_val_pred)
    f1_val = f1_score(Y_val, y_val_pred)

    clf = classification_report(Y_val, y_val_pred)

    train_score = model.score(X_train, Y_train)
    val_score = model.score(X_val, Y_val)

    print(f'***********  {model}  ***********')
    print('-- Training --')
    print('\n')
    print('Accuracy : ', acc_train)
    print('F1 Score : ', f1_train)
    print(10*'=====')
    print('Confusion Matrix :\n',conf_train)
    print(10*'=====')
    print('Train Score : ', train_score)
    print('\n')
    print('-- Validation --')
    print('\n')
    print('Accuracy : ', acc_val)
    print('F1 Score : ', f1_val)
    print(10*'=====')
    print('Confusion Matrix :\n',conf_val)
    print(10*'=====')
    print('Validation Score : ', val_score)
    print(10*'=====')
    print('Classification Report :\n',clf)
    print(10*'=====')
```

[ ]:
```
evaluate(rf_os_smoten_best, X_train_ores_smoten, y_train_ores_smoten, X_val,␣
↪Y_val)
```

```
***********  RandomForestClassifier(max_depth=15, min_samples_leaf=2,
n_estimators=80,
                       n_jobs=-1, random_state=42)  ***********
-- Training --


Accuracy :  0.9828571428571429
F1 Score :  0.9829787234042553
==================================================
Confusion Matrix :
 [[683  17]
 [  7 693]]
```

```
================================================
Train Score :  0.9828571428571429


-- Validation --


Accuracy :  0.9595959595959596
F1 Score :  0.9705882352941176
================================================
Confusion Matrix :
 [[29  3]
 [ 1 66]]
================================================
Validation Score :  0.9595959595959596
================================================
Classification Report :
               precision    recall  f1-score   support

         0.0       0.97      0.91      0.94        32
         1.0       0.96      0.99      0.97        67

    accuracy                           0.96        99
   macro avg       0.96      0.95      0.95        99
weighted avg       0.96      0.96      0.96        99

================================================
```

[ ]:
```
# TODO: Test with stratified + SMOTEN
```

4 Logistic regression

[ ]:
```
from sklearn.linear_model import LogisticRegression
```

[ ]:
```
def quick_fit_score(model, X_train, y_train, X_val, y_val):

    model = model.fit(X_train, y_train)

    acc_train = model.score(X_train, y_train)
    acc_val = model.score(X_val, y_val)

    print('Model : ', model)
    print('train score : ', acc_train)
    print('val score : ', acc_val)
```

[ ]:
```
lr = LogisticRegression()
quick_fit_score(lr, X_train, Y_train, X_val, Y_val)
```

```
Model :  LogisticRegression()
train score :  0.7414285714285714
val score :  0.6666666666666666
```

[ ]:
```
# Try with Stratified
lr_SS = LogisticRegression()
quick_fit_score(lr_SS, X_train_SS, y_train_SS, X_val_SS, y_val_SS)
```

```
Model :  LogisticRegression()
train score :  0.7414285714285714
val score :  0.7333333333333333
```

[ ]:
```
# Try with SMOTEN
lr_smoten = LogisticRegression()
quick_fit_score(lr_smoten, X_train_ores_smoten, y_train_ores_smoten, X_val,␣
 ↪Y_val)
```

```
Model :  LogisticRegression()
train score :  0.7457142857142857
val score :  0.6666666666666666
```

[ ]:
```
# Try cross validation
lr_cv = LogisticRegression()
parameters= {"C":np.logspace(2,3,7), "penalty":["l1","l2"], "solver":
 ↪["liblinear", "saga"]}
lr_cv = GridSearchCV(lr, parameters, cv=10, scoring='accuracy')

lr_cv.fit(X_train_SS, y_train_SS)

lr_cv_best = lr_cv.best_estimator_
```

[ ]:
```
lr_cv_best
```

[ ]:
```
LogisticRegression(C=100.0, penalty='l1', solver='liblinear')
```

[ ]:
```
quick_fit_score(lr_cv_best, X_train_SS, y_train_SS, X_val_SS, y_val_SS)
```

```
Model :  LogisticRegression(C=100.0, penalty='l1', solver='liblinear')
train score :  0.7471428571428571
val score :  0.7266666666666667
```

[ ]:
```
evaluate(lr_SS, X_train_SS, y_train_SS, X_val_SS, y_val_SS)
```

```
***********  LogisticRegression()  ***********
-- Training --


Accuracy :  0.7414285714285714
F1 Score :  0.8316279069767442
================================================
```

```
Confusion Matrix :
 [[ 72 138]
 [ 43 447]]
================================================
Train Score :  0.7414285714285714


-- Validation --


Accuracy :  0.7333333333333333
F1 Score :  0.8230088495575221
================================================
Confusion Matrix :
 [[17 28]
 [12 93]]
================================================
Validation Score :  0.7333333333333333
================================================
Classification Report :
               precision    recall  f1-score   support

         0.0       0.59      0.38      0.46        45
         1.0       0.77      0.89      0.82       105

    accuracy                           0.73       150
   macro avg       0.68      0.63      0.64       150
weighted avg       0.71      0.73      0.71       150

================================================
```

5 Deep Neural Network

[ ]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import models
from tensorflow.keras import layers
from tensorflow.keras import optimizers
from tensorflow.keras import losses
from tensorflow.keras import metrics
```

```
2023-05-12 14:54:30.339620: I tensorflow/core/platform/cpu_feature_guard.cc:193]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  AVX2 AVX512F AVX512_VNNI FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
```

```
2023-05-12 14:54:30.926336: I tensorflow/core/util/port.cc:104] oneDNN custom
operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn them
off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2023-05-12 14:54:30.965216: W
tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could
not load dynamic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot
open shared object file: No such file or directory
2023-05-12 14:54:30.965271: I
tensorflow/compiler/xla/stream_executor/cuda/cudart_stub.cc:29] Ignore above
cudart dlerror if you do not have a GPU set up on your machine.
2023-05-12 14:54:32.111365: W
tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could
not load dynamic library 'libnvinfer.so.7'; dlerror: libnvinfer.so.7: cannot
open shared object file: No such file or directory
2023-05-12 14:54:32.111606: W
tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could
not load dynamic library 'libnvinfer_plugin.so.7'; dlerror:
libnvinfer_plugin.so.7: cannot open shared object file: No such file or
directory
2023-05-12 14:54:32.111613: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Cannot
dlopen some TensorRT libraries. If you would like to use Nvidia GPU with
TensorRT, please make sure the missing libraries mentioned above are installed
properly.
```

[ ]:
```
# dataset with categories and number values
credit_for_dnn.head()
```

[ ]:

	checking_status	duration	credit_history	\
0	<0	6.0	critical/other existing credit	
1	0<=X<200	48.0	existing paid	
2	no checking	12.0	critical/other existing credit	
3	<0	42.0	existing paid	
4	<0	24.0	delayed previously	

	purpose	credit_amount	savings_status	employment	\
0	radio/tv	1169.0	no known savings	>=7	
1	radio/tv	5951.0	<100	1<=X<4	
2	education	2096.0	<100	4<=X<7	
3	furniture/equipment	7882.0	<100	4<=X<7	
4	new car	4870.0	<100	1<=X<4	

	installment_commitment	other_parties	residence_since	…	\
0	4.0	none	4.0	…	
1	2.0	none	2.0	…	
2	2.0	none	3.0	…	

```
3                    2.0     guarantor              4.0  …
4                    3.0          none              4.0  …

  other_payment_plans   housing existing_credits                job  \
0                none       own              2.0            skilled
1                none       own              1.0            skilled
2                none       own              1.0  unskilled resident
3                none  for free              1.0            skilled
4                none  for free              2.0            skilled

   num_dependents own_telephone  foreign_worker class  gender marital_status
0             1.0           yes             yes  good    male         single
1             1.0          none             yes   bad  female    div/dep/mar
2             2.0          none             yes  good    male         single
3             2.0          none             yes  good    male         single
4             2.0          none             yes   bad    male         single

[5 rows x 22 columns]
```

[ ]:
```
X = credit_for_dnn.drop('class', axis=1)
y = credit_for_dnn['class']
```

[ ]:
```
X.shape
```

[ ]:
```
(1000, 21)
```

5.1 Data preparation

[ ]:
```
splitter = StratifiedShuffleSplit(n_splits=10, test_size=0.3, random_state=42)

for train, rest in splitter.split(X,y):
    X_train_SS = X.iloc[train]
    y_train_SS = y.iloc[train]
    X_rest_SS = X.iloc[rest]
    y_rest_SS = y.iloc[rest]

print(y_train_SS.value_counts())
print(y_rest_SS.value_counts())
print(X_train_SS.shape, X_rest_SS.shape)
```

```
good    490
bad     210
Name: class, dtype: int64
good    210
bad      90
Name: class, dtype: int64
(700, 21) (300, 21)
```

[ ]:
```
splitter = StratifiedShuffleSplit(n_splits=10, test_size=0.5, random_state=42)

for val, test in splitter.split(X_rest_SS, y_rest_SS):
    X_val_SS = X_rest_SS.iloc[val]
    y_val_SS = y_rest_SS.iloc[val]
    X_test_SS = X_rest_SS.iloc[test]
    y_test_SS = y_rest_SS.iloc[test]

print(y_val_SS.value_counts())
print(y_test_SS.value_counts())
print(X_val_SS.shape, X_test_SS.shape)
```

```
good    105
bad      45
Name: class, dtype: int64
good    105
bad      45
Name: class, dtype: int64
(150, 21) (150, 21)
```

[ ]:
```
def list_inputs(X):
  inputs = {}
  for name, column in X.items():
    dtype = column.dtype
    if dtype == 'category':
      dtype = tf.string
    else:
      dtype = tf.float64
    inputs[name] = tf.keras.Input(shape=(1,), name=name, dtype=dtype)

  return inputs

train_inputs = list_inputs(X_train_SS)
val_inputs = list_inputs(X_val_SS)
test_inputs = list_inputs(X_test_SS)

train_inputs
```

[ ]:
```
{'checking_status': <KerasTensor: shape=(None, 1) dtype=string (created by layer
'checking_status')>,
 'duration': <KerasTensor: shape=(None, 1) dtype=float64 (created by layer
'duration')>,
 'credit_history': <KerasTensor: shape=(None, 1) dtype=string (created by layer
'credit_history')>,
 'purpose': <KerasTensor: shape=(None, 1) dtype=string (created by layer
'purpose')>,
 'credit_amount': <KerasTensor: shape=(None, 1) dtype=float64 (created by layer
'credit_amount')>,
```

```
 'savings_status': <KerasTensor: shape=(None, 1) dtype=string (created by layer
'savings_status')>,
 'employment': <KerasTensor: shape=(None, 1) dtype=string (created by layer
'employment')>,
 'installment_commitment': <KerasTensor: shape=(None, 1) dtype=float64 (created
by layer 'installment_commitment')>,
 'other_parties': <KerasTensor: shape=(None, 1) dtype=string (created by layer
'other_parties')>,
 'residence_since': <KerasTensor: shape=(None, 1) dtype=float64 (created by
layer 'residence_since')>,
 'property_magnitude': <KerasTensor: shape=(None, 1) dtype=string (created by
layer 'property_magnitude')>,
 'age': <KerasTensor: shape=(None, 1) dtype=float64 (created by layer 'age')>,
 'other_payment_plans': <KerasTensor: shape=(None, 1) dtype=string (created by
layer 'other_payment_plans')>,
 'housing': <KerasTensor: shape=(None, 1) dtype=string (created by layer
'housing')>,
 'existing_credits': <KerasTensor: shape=(None, 1) dtype=float64 (created by
layer 'existing_credits')>,
 'job': <KerasTensor: shape=(None, 1) dtype=string (created by layer 'job')>,
 'num_dependents': <KerasTensor: shape=(None, 1) dtype=float64 (created by layer
'num_dependents')>,
 'own_telephone': <KerasTensor: shape=(None, 1) dtype=string (created by layer
'own_telephone')>,
 'foreign_worker': <KerasTensor: shape=(None, 1) dtype=string (created by layer
'foreign_worker')>,
 'gender': <KerasTensor: shape=(None, 1) dtype=string (created by layer
'gender')>,
 'marital_status': <KerasTensor: shape=(None, 1) dtype=string (created by layer
'marital_status')>}
```

[ ]:
```python
def preprocess_inputs(inputs, X):
  preprocessed_inputs = []

  for name, input in inputs.items():
    if input.dtype == tf.float64:
      continue

    lookup = layers.StringLookup(vocabulary=np.unique(X[name]))
    one_hot = layers.CategoryEncoding(num_tokens=lookup.vocabulary_size())

    x = lookup(input)
    x = one_hot(x)
    preprocessed_inputs.append(x)

  return preprocessed_inputs
```

```
preprocessed_inputs_train = preprocess_inputs(train_inputs, X_train_SS)
preprocessed_inputs_val = preprocess_inputs(val_inputs, X_val_SS)
preprocessed_inputs_test = preprocess_inputs(test_inputs, X_test_SS)

preprocessed_inputs_train
```

```
2023-05-12 15:06:00.898691: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:967] could not
open file to read NUMA node: /sys/bus/pci/devices/0000:01:00.0/numa_node
Your kernel may have been built without NUMA support.
2023-05-12 15:06:00.899055: W
tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could
not load dynamic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot
open shared object file: No such file or directory
2023-05-12 15:06:00.899148: W
tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could
not load dynamic library 'libcublas.so.11'; dlerror: libcublas.so.11: cannot
open shared object file: No such file or directory
2023-05-12 15:06:00.899196: W
tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could
not load dynamic library 'libcublasLt.so.11'; dlerror: libcublasLt.so.11: cannot
open shared object file: No such file or directory
2023-05-12 15:06:00.899245: W
tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could
not load dynamic library 'libcufft.so.10'; dlerror: libcufft.so.10: cannot open
shared object file: No such file or directory
2023-05-12 15:06:00.899358: W
tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could
not load dynamic library 'libcurand.so.10'; dlerror: libcurand.so.10: cannot
open shared object file: No such file or directory
2023-05-12 15:06:00.899408: W
tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could
not load dynamic library 'libcusolver.so.11'; dlerror: libcusolver.so.11: cannot
open shared object file: No such file or directory
2023-05-12 15:06:00.899455: W
tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could
not load dynamic library 'libcusparse.so.11'; dlerror: libcusparse.so.11: cannot
open shared object file: No such file or directory
2023-05-12 15:06:00.899487: W
tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64] Could
not load dynamic library 'libcudnn.so.8'; dlerror: libcudnn.so.8: cannot open
shared object file: No such file or directory
2023-05-12 15:06:00.899519: W
tensorflow/core/common_runtime/gpu/gpu_device.cc:1934] Cannot dlopen some GPU
libraries. Please make sure the missing libraries mentioned above are installed
properly if you would like to use GPU. Follow the guide at
https://www.tensorflow.org/install/gpu for how to download and setup the
required libraries for your platform.
```

```
Skipping registering GPU devices…
2023-05-12 15:06:00.902477: I tensorflow/core/platform/cpu_feature_guard.cc:193]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  AVX2 AVX512F AVX512_VNNI FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
```

[ ]:
```
[<KerasTensor: shape=(None, 5) dtype=float32 (created by layer
'category_encoding')>,
 <KerasTensor: shape=(None, 6) dtype=float32 (created by layer
'category_encoding_1')>,
 <KerasTensor: shape=(None, 11) dtype=float32 (created by layer
'category_encoding_2')>,
 <KerasTensor: shape=(None, 6) dtype=float32 (created by layer
'category_encoding_3')>,
 <KerasTensor: shape=(None, 6) dtype=float32 (created by layer
'category_encoding_4')>,
 <KerasTensor: shape=(None, 4) dtype=float32 (created by layer
'category_encoding_5')>,
 <KerasTensor: shape=(None, 5) dtype=float32 (created by layer
'category_encoding_6')>,
 <KerasTensor: shape=(None, 4) dtype=float32 (created by layer
'category_encoding_7')>,
 <KerasTensor: shape=(None, 4) dtype=float32 (created by layer
'category_encoding_8')>,
 <KerasTensor: shape=(None, 5) dtype=float32 (created by layer
'category_encoding_9')>,
 <KerasTensor: shape=(None, 3) dtype=float32 (created by layer
'category_encoding_10')>,
 <KerasTensor: shape=(None, 3) dtype=float32 (created by layer
'category_encoding_11')>,
 <KerasTensor: shape=(None, 3) dtype=float32 (created by layer
'category_encoding_12')>,
 <KerasTensor: shape=(None, 5) dtype=float32 (created by layer
'category_encoding_13')>]
```

[ ]:
```
def normalize_num_inputs(inputs, X):

    numeric_inputs = {name:input for name,input in inputs.items()
                      if input.dtype==tf.float64}

    x = layers.Concatenate()(list(numeric_inputs.values()))
    norm = layers.Normalization()
    norm.adapt(np.array(X[numeric_inputs.keys()]))
    all_num_inputs = norm(x)
    return all_num_inputs
```

```
all_num_inputs_train = normalize_num_inputs(train_inputs, X_train_SS)
all_num_inputs_val = normalize_num_inputs(val_inputs, X_val_SS)
all_num_inputs_test = normalize_num_inputs(test_inputs, X_test_SS)

preprocessed_inputs_train.append(all_num_inputs_train)
preprocessed_inputs_val.append(all_num_inputs_val)
preprocessed_inputs_test.append(all_num_inputs_test)
```

[ ]:
```
preprocessed_inputs_cat_train = layers.Concatenate()(preprocessed_inputs_train)
preprocessed_inputs_cat_val = layers.Concatenate()(preprocessed_inputs_val)
preprocessed_inputs_cat_test = layers.Concatenate()(preprocessed_inputs_test)

credit_preprocessing_train = tf.keras.Model(train_inputs,␣
 ↪preprocessed_inputs_cat_train)
credit_preprocessing_val = tf.keras.Model(val_inputs,␣
 ↪preprocessed_inputs_cat_val)
credit_preprocessing_test = tf.keras.Model(test_inputs,␣
 ↪preprocessed_inputs_cat_test)
```

[ ]:
```
credit_features_dict_train = {name: np.array(value)
                         for name, value in X_train_SS.items()}
credit_features_dict_val = {name: np.array(value)
                         for name, value in X_val_SS.items()}
credit_features_dict_test = {name: np.array(value)
                         for name, value in X_test_SS.items()}
```

[ ]:
```
# Sample example with string one-hots and numeric values concatenated
features_dict = {name:values[0:1] for name, values in␣
 ↪credit_features_dict_train.items()}
credit_preprocessing_train(features_dict)
```

[ ]:
```
<tf.Tensor: shape=(1, 77), dtype=float32, numpy=
array([[ 0.        ,  0.        ,  1.        ,  0.        ,  0.        ,
         0.        ,  0.        ,  0.        ,  0.        ,  1.        ,
         0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
         0.        ,  1.        ,  0.        ,  0.        ,  0.        ,
         0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
         0.        ,  0.        ,  1.        ,  0.        ,  1.        ,
         0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
         0.        ,  0.        ,  1.        ,  0.        ,  1.        ,
         0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
         1.        ,  0.        ,  0.        ,  0.        ,  1.        ,
         0.        ,  0.        ,  0.        ,  1.        ,  0.        ,
         0.        ,  0.        ,  1.        ,  0.        ,  0.        ,
         0.        ,  1.        ,  0.        ,  0.        ,  1.        ,
         0.        ,  0.        ,  0.        ,  1.        ,  0.        ,
```

```
        -0.7461104 , -0.84751564,  0.93219185, -0.79911137, -1.0706813 ,
        -0.7181895 , -0.4082481 ]], dtype=float32)>
```

[ ]:
```
y_enc_train = LabelEncoder().fit_transform(y_train_SS)
y_enc_val = LabelEncoder().fit_transform(y_val_SS)
y_enc_test = LabelEncoder().fit_transform(y_test_SS)

y_enc_train[0:10]
```

[ ]:
```
array([0, 1, 1, 1, 1, 1, 1, 1, 1, 0])
```

[ ]:
```
# Check values of a sample

import itertools

def slices(features):
  for i in itertools.count(1): # change int to find another sample
    # For each feature take index `i`
    example = {name:values[i] for name, values in features.items()}
    yield example

for example in slices(credit_features_dict_train):
  for name, value in example.items():
    print(f"{name:40s}: {value}")
  break
```

```
checking_status                          : 0<=X<200
duration                                 : 10.0
credit_history                           : all paid
purpose                                  : radio/tv
credit_amount                            : 1048.0
savings_status                           : <100
employment                               : 1<=X<4
installment_commitment                   : 4.0
other_parties                            : none
residence_since                          : 4.0
property_magnitude                       : real estate
age                                      : 23.0
other_payment_plans                      : stores
housing                                  : own
existing_credits                         : 1.0
job                                      : unskilled resident
num_dependents                           : 1.0
own_telephone                            : none
foreign_worker                           : yes
gender                                   : male
marital_status                           : single
```

5.2 Create batches

[ ]:
```
credit_ds_train = tf.data.Dataset.
 ↪from_tensor_slices((credit_features_dict_train, y_enc_train))
credit_ds_val = tf.data.Dataset.from_tensor_slices((credit_features_dict_val,␣
 ↪y_enc_val))
credit_ds_test = tf.data.Dataset.from_tensor_slices((credit_features_dict_test,␣
 ↪y_enc_test))
```

[ ]:
```
credit_batches_train = credit_ds_train.shuffle(len(y_enc_train)).batch(32)
credit_batches_val = credit_ds_val.shuffle(len(y_enc_val)).batch(32)
credit_batches_test = credit_ds_test.shuffle(len(y_enc_test)).batch(32)
```

5.3 Define and fit the model

[ ]:
```
def dnn_model(preprocessing_head, inputs):
  body = tf.keras.Sequential([
    layers.Dense(16),
    layers.Dense(8),
    layers.Dense(1)
  ])

  preprocessed_inputs = preprocessing_head(inputs)
  result = body(preprocessed_inputs)
  model = tf.keras.Model(inputs, result)

  model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
                metrics=tf.keras.metrics.BinaryAccuracy(),
                optimizer=tf.keras.optimizers.Adam())
  return model

dnn_model_1 = dnn_model(credit_preprocessing_train, train_inputs)
```

[ ]:
```
dnn_model_1.summary()
```

```
Model: "model_19"
__________________________________________________________________________________
__________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================
==================
 age (InputLayer)               [(None, 1)]          0           []

 checking_status (InputLayer)   [(None, 1)]          0           []

 credit_amount (InputLayer)     [(None, 1)]          0           []

 credit_history (InputLayer)    [(None, 1)]          0           []
```

```
duration (InputLayer)          [(None, 1)]          0           []

employment (InputLayer)        [(None, 1)]          0           []

existing_credits (InputLayer)  [(None, 1)]          0           []

foreign_worker (InputLayer)    [(None, 1)]          0           []

gender (InputLayer)            [(None, 1)]          0           []

housing (InputLayer)           [(None, 1)]          0           []

installment_commitment (InputL  [(None, 1)]         0           []
ayer)

job (InputLayer)               [(None, 1)]          0           []

marital_status (InputLayer)    [(None, 1)]          0           []

num_dependents (InputLayer)    [(None, 1)]          0           []

other_parties (InputLayer)     [(None, 1)]          0           []

other_payment_plans (InputLaye  [(None, 1)]         0           []
r)

own_telephone (InputLayer)     [(None, 1)]          0           []

property_magnitude (InputLayer  [(None, 1)]         0           []
)

purpose (InputLayer)           [(None, 1)]          0           []

residence_since (InputLayer)   [(None, 1)]          0           []

savings_status (InputLayer)    [(None, 1)]          0           []

model (Functional)             (None, 77)           15          ['age[0][0]',
'checking_status[0][0]',
'credit_amount[0][0]',
'credit_history[0][0]',
'duration[0][0]',
'employment[0][0]',
'existing_credits[0][0]',
'foreign_worker[0][0]',
'gender[0][0]',
'housing[0][0]',
```

```
'installment_commitment[0][0]',
                                                                    'job[0][0]',
'marital_status[0][0]',
'num_dependents[0][0]',
'other_parties[0][0]',
'other_payment_plans[0][0]',
'own_telephone[0][0]',
'property_magnitude[0][0]',
'purpose[0][0]',
'residence_since[0][0]',
'savings_status[0][0]']

 sequential_16 (Sequential)     (None, 1)            1393
['model[16][0]']

=================================================================================
==================
Total params: 1,408
Trainable params: 1,393
Non-trainable params: 15
_________________________________________________________________________________
__________________
```

[ ]:
```
epochs = 100
```

[ ]:
```
history = dnn_model_1.fit(credit_batches_train, epochs=epochs, verbose=1)
```

```
Epoch 1/100
22/22 [==============================] - 2s 3ms/step - loss: 0.7256 -
binary_accuracy: 0.5486
Epoch 2/100
22/22 [==============================] - 0s 3ms/step - loss: 0.6369 -
binary_accuracy: 0.6229
Epoch 3/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5870 -
binary_accuracy: 0.6700
Epoch 4/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5523 -
binary_accuracy: 0.7029
Epoch 5/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5262 -
binary_accuracy: 0.7314
Epoch 6/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5065 -
binary_accuracy: 0.7429
Epoch 7/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4913 -
binary_accuracy: 0.7586
```

```
Epoch 8/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4796 -
binary_accuracy: 0.7629
Epoch 9/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4711 -
binary_accuracy: 0.7643
Epoch 10/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4652 -
binary_accuracy: 0.7657
Epoch 11/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4592 -
binary_accuracy: 0.7700
Epoch 12/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4556 -
binary_accuracy: 0.7743
Epoch 13/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4549 -
binary_accuracy: 0.7657
Epoch 14/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4510 -
binary_accuracy: 0.7657
Epoch 15/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4495 -
binary_accuracy: 0.7714
Epoch 16/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4487 -
binary_accuracy: 0.7700
Epoch 17/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4481 -
binary_accuracy: 0.7729
Epoch 18/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4462 -
binary_accuracy: 0.7757
Epoch 19/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4456 -
binary_accuracy: 0.7786
Epoch 20/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4439 -
binary_accuracy: 0.7729
Epoch 21/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4439 -
binary_accuracy: 0.7743
Epoch 22/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4432 -
binary_accuracy: 0.7700
Epoch 23/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4427 -
binary_accuracy: 0.7729
```

```
Epoch 24/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4415 -
binary_accuracy: 0.7729
Epoch 25/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4431 -
binary_accuracy: 0.7700
Epoch 26/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4448 -
binary_accuracy: 0.7700
Epoch 27/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4424 -
binary_accuracy: 0.7871
Epoch 28/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4408 -
binary_accuracy: 0.7786
Epoch 29/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4414 -
binary_accuracy: 0.7800
Epoch 30/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4425 -
binary_accuracy: 0.7714
Epoch 31/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4421 -
binary_accuracy: 0.7829
Epoch 32/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4398 -
binary_accuracy: 0.7800
Epoch 33/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4399 -
binary_accuracy: 0.7814
Epoch 34/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4449 -
binary_accuracy: 0.7786
Epoch 35/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4424 -
binary_accuracy: 0.7757
Epoch 36/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4423 -
binary_accuracy: 0.7743
Epoch 37/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4408 -
binary_accuracy: 0.7843
Epoch 38/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4430 -
binary_accuracy: 0.7729
Epoch 39/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4399 -
binary_accuracy: 0.7843
```

```
Epoch 40/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4402 -
binary_accuracy: 0.7743
Epoch 41/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4409 -
binary_accuracy: 0.7771
Epoch 42/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4390 -
binary_accuracy: 0.7800
Epoch 43/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4400 -
binary_accuracy: 0.7771
Epoch 44/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4417 -
binary_accuracy: 0.7814
Epoch 45/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4394 -
binary_accuracy: 0.7786
Epoch 46/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4393 -
binary_accuracy: 0.7814
Epoch 47/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4423 -
binary_accuracy: 0.7743
Epoch 48/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4412 -
binary_accuracy: 0.7829
Epoch 49/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4392 -
binary_accuracy: 0.7786
Epoch 50/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4398 -
binary_accuracy: 0.7757
Epoch 51/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4397 -
binary_accuracy: 0.7757
Epoch 52/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4404 -
binary_accuracy: 0.7786
Epoch 53/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4414 -
binary_accuracy: 0.7786
Epoch 54/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4384 -
binary_accuracy: 0.7757
Epoch 55/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4396 -
binary_accuracy: 0.7829
```

```
Epoch 56/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4388 -
binary_accuracy: 0.7800
Epoch 57/100
22/22 [==============================] - 0s 3ms/step - loss: 0.4394 -
binary_accuracy: 0.7871
Epoch 58/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4395 -
binary_accuracy: 0.7757
Epoch 59/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4395 -
binary_accuracy: 0.7814
Epoch 60/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4383 -
binary_accuracy: 0.7829
Epoch 61/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4392 -
binary_accuracy: 0.7800
Epoch 62/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4382 -
binary_accuracy: 0.7843
Epoch 63/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4397 -
binary_accuracy: 0.7871
Epoch 64/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4388 -
binary_accuracy: 0.7700
Epoch 65/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4380 -
binary_accuracy: 0.7800
Epoch 66/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4377 -
binary_accuracy: 0.7829
Epoch 67/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4385 -
binary_accuracy: 0.7700
Epoch 68/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4395 -
binary_accuracy: 0.7729
Epoch 69/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4387 -
binary_accuracy: 0.7714
Epoch 70/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4390 -
binary_accuracy: 0.7843
Epoch 71/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4396 -
binary_accuracy: 0.7771
```

```
Epoch 72/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4392 -
binary_accuracy: 0.7771
Epoch 73/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4380 -
binary_accuracy: 0.7771
Epoch 74/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4385 -
binary_accuracy: 0.7771
Epoch 75/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4382 -
binary_accuracy: 0.7843
Epoch 76/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4378 -
binary_accuracy: 0.7729
Epoch 77/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4387 -
binary_accuracy: 0.7771
Epoch 78/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4391 -
binary_accuracy: 0.7871
Epoch 79/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4375 -
binary_accuracy: 0.7829
Epoch 80/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4385 -
binary_accuracy: 0.7871
Epoch 81/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4398 -
binary_accuracy: 0.7786
Epoch 82/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4378 -
binary_accuracy: 0.7814
Epoch 83/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4382 -
binary_accuracy: 0.7800
Epoch 84/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4387 -
binary_accuracy: 0.7771
Epoch 85/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4375 -
binary_accuracy: 0.7743
Epoch 86/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4390 -
binary_accuracy: 0.7814
Epoch 87/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4391 -
binary_accuracy: 0.7729
```

```
Epoch 88/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4391 -
binary_accuracy: 0.7800
Epoch 89/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4386 -
binary_accuracy: 0.7786
Epoch 90/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4381 -
binary_accuracy: 0.7829
Epoch 91/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4380 -
binary_accuracy: 0.7771
Epoch 92/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4385 -
binary_accuracy: 0.7800
Epoch 93/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4396 -
binary_accuracy: 0.7814
Epoch 94/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4376 -
binary_accuracy: 0.7757
Epoch 95/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4376 -
binary_accuracy: 0.7771
Epoch 96/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4375 -
binary_accuracy: 0.7771
Epoch 97/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4380 -
binary_accuracy: 0.7757
Epoch 98/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4387 -
binary_accuracy: 0.7843
Epoch 99/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4390 -
binary_accuracy: 0.7800
Epoch 100/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4392 -
binary_accuracy: 0.7786
```

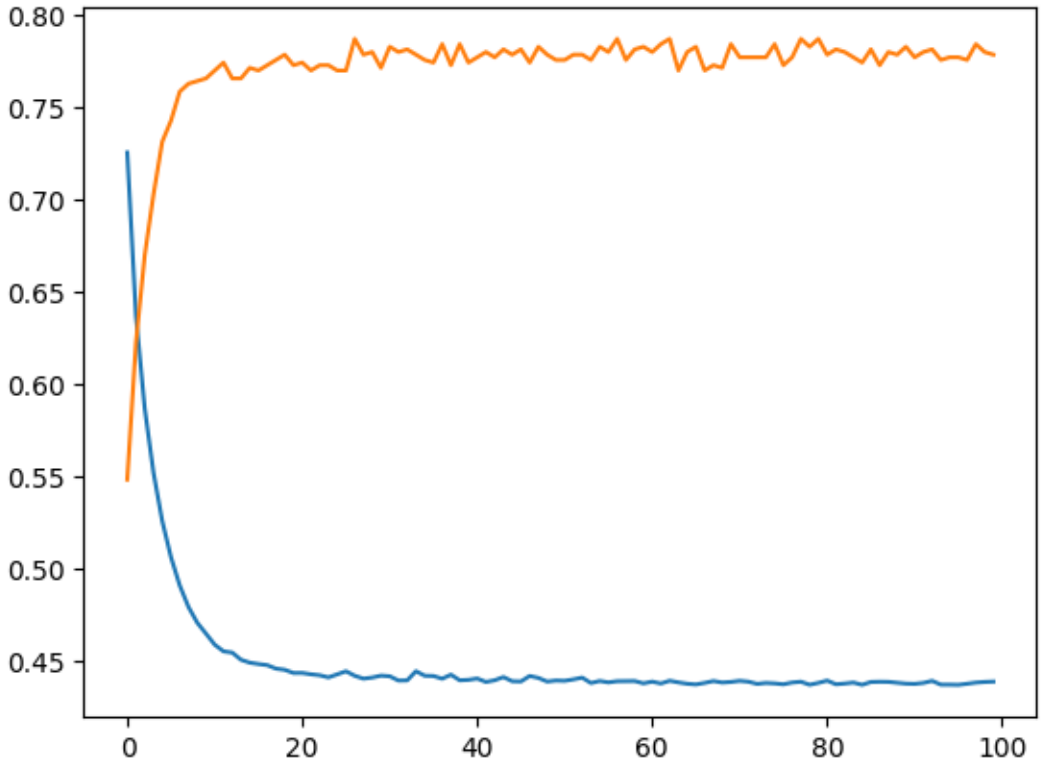
[ ]:
```
dnn_model_1.evaluate(credit_batches_train)
```

```
22/22 [==============================] - 0s 2ms/step - loss: 0.4343 -
binary_accuracy: 0.7814
```

[ ]:
```
[0.43432143330574036, 0.7814285755157471]
```

[ ]:
```
plt.plot([i for i in range(epochs)], history.history['loss'], history.
  ↪history['binary_accuracy'])
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7f5ea0369250>,
 <matplotlib.lines.Line2D at 0x7f5ea03692b0>]
```



[ ]:
```
dnn_model_1.evaluate(credit_batches_val)
```

```
5/5 [==============================] - 0s 3ms/step - loss: 0.5423 -
binary_accuracy: 0.7000
```

[ ]:
```
[0.542330801486969, 0.699999988079071]
```

[ ]:
```
# new model with different activation functions

def dnn_model(preprocessing_head, inputs):
  body = tf.keras.Sequential([
    layers.Dense(64, activation='relu'),
    layers.Dense(32, activation='relu'),
    layers.Dense(16, activation='relu'),
    layers.Dense(1, activation='sigmoid')
  ])
```

```
  preprocessed_inputs = preprocessing_head(inputs)
  result = body(preprocessed_inputs)
  model = tf.keras.Model(inputs, result)

  model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
                metrics=tf.keras.metrics.BinaryAccuracy(),
                optimizer=tf.keras.optimizers.Adam())
  return model

dnn_model_relu_sigmoid = dnn_model(credit_preprocessing_train, train_inputs)
```

[ ]:
```
history = dnn_model_relu_sigmoid.fit(credit_batches_train, epochs=epochs,␣
 ↪verbose='auto')
```

```
Epoch 1/100
/home/filip/py-env/aai-2023-1/lib/python3.9/site-packages/keras/backend.py:5676:
UserWarning: "`binary_crossentropy` received `from_logits=True`, but the
`output` argument was produced by a Sigmoid activation and thus does not
represent logits. Was this intended?
  output, from_logits = _get_logits(
22/22 [==============================] - 1s 3ms/step - loss: 0.6876 -
binary_accuracy: 0.5657
Epoch 2/100
22/22 [==============================] - 0s 3ms/step - loss: 0.6034 -
binary_accuracy: 0.7000
Epoch 3/100
22/22 [==============================] - 0s 3ms/step - loss: 0.5679 -
binary_accuracy: 0.7029
Epoch 4/100
22/22 [==============================] - 0s 3ms/step - loss: 0.5335 -
binary_accuracy: 0.7286
Epoch 5/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4997 -
binary_accuracy: 0.7500
Epoch 6/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4649 -
binary_accuracy: 0.7814
Epoch 7/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4412 -
binary_accuracy: 0.7871
Epoch 8/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4201 -
binary_accuracy: 0.8071
Epoch 9/100
22/22 [==============================] - 0s 2ms/step - loss: 0.4071 -
binary_accuracy: 0.8186
```

```
Epoch 10/100
22/22 [==============================] - 0s 2ms/step - loss: 0.3921 -
binary_accuracy: 0.8157
Epoch 11/100
22/22 [==============================] - 0s 2ms/step - loss: 0.3740 -
binary_accuracy: 0.8200
Epoch 12/100
22/22 [==============================] - 0s 2ms/step - loss: 0.3583 -
binary_accuracy: 0.8371
Epoch 13/100
22/22 [==============================] - 0s 2ms/step - loss: 0.3408 -
binary_accuracy: 0.8486
Epoch 14/100
22/22 [==============================] - 0s 2ms/step - loss: 0.3334 -
binary_accuracy: 0.8500
Epoch 15/100
22/22 [==============================] - 0s 2ms/step - loss: 0.3115 -
binary_accuracy: 0.8700
Epoch 16/100
22/22 [==============================] - 0s 2ms/step - loss: 0.3030 -
binary_accuracy: 0.8757
Epoch 17/100
22/22 [==============================] - 0s 2ms/step - loss: 0.2761 -
binary_accuracy: 0.9029
Epoch 18/100
22/22 [==============================] - 0s 2ms/step - loss: 0.2588 -
binary_accuracy: 0.9000
Epoch 19/100
22/22 [==============================] - 0s 2ms/step - loss: 0.2468 -
binary_accuracy: 0.9129
Epoch 20/100
22/22 [==============================] - 0s 2ms/step - loss: 0.2233 -
binary_accuracy: 0.9286
Epoch 21/100
22/22 [==============================] - 0s 2ms/step - loss: 0.2038 -
binary_accuracy: 0.9314
Epoch 22/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1816 -
binary_accuracy: 0.9386
Epoch 23/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1702 -
binary_accuracy: 0.9543
Epoch 24/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1522 -
binary_accuracy: 0.9529
Epoch 25/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1341 -
binary_accuracy: 0.9614
```

```
Epoch 26/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1179 -
binary_accuracy: 0.9714
Epoch 27/100
22/22 [==============================] - 0s 2ms/step - loss: 0.1056 -
binary_accuracy: 0.9786
Epoch 28/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0861 -
binary_accuracy: 0.9814
Epoch 29/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0751 -
binary_accuracy: 0.9857
Epoch 30/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0714 -
binary_accuracy: 0.9857
Epoch 31/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0523 -
binary_accuracy: 0.9957
Epoch 32/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0441 -
binary_accuracy: 0.9957
Epoch 33/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0369 -
binary_accuracy: 0.9971
Epoch 34/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0313 -
binary_accuracy: 0.9971
Epoch 35/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0264 -
binary_accuracy: 0.9971
Epoch 36/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0220 -
binary_accuracy: 0.9986
Epoch 37/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0180 -
binary_accuracy: 0.9986
Epoch 38/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0157 -
binary_accuracy: 0.9986
Epoch 39/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0143 -
binary_accuracy: 1.0000
Epoch 40/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0113 -
binary_accuracy: 1.0000
Epoch 41/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0099 -
binary_accuracy: 1.0000
```

```
Epoch 42/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0085 -
binary_accuracy: 1.0000
Epoch 43/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0074 -
binary_accuracy: 1.0000
Epoch 44/100
22/22 [==============================] - 0s 3ms/step - loss: 0.0066 -
binary_accuracy: 1.0000
Epoch 45/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0058 -
binary_accuracy: 1.0000
Epoch 46/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0053 -
binary_accuracy: 1.0000
Epoch 47/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0048 -
binary_accuracy: 1.0000
Epoch 48/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0043 -
binary_accuracy: 1.0000
Epoch 49/100
22/22 [==============================] - 0s 3ms/step - loss: 0.0040 -
binary_accuracy: 1.0000
Epoch 50/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0038 -
binary_accuracy: 1.0000
Epoch 51/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0032 -
binary_accuracy: 1.0000
Epoch 52/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0030 -
binary_accuracy: 1.0000
Epoch 53/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0027 -
binary_accuracy: 1.0000
Epoch 54/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0026 -
binary_accuracy: 1.0000
Epoch 55/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0024 -
binary_accuracy: 1.0000
Epoch 56/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0023 -
binary_accuracy: 1.0000
Epoch 57/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0021 -
binary_accuracy: 1.0000
```

```
Epoch 58/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0020 -
binary_accuracy: 1.0000
Epoch 59/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0018 -
binary_accuracy: 1.0000
Epoch 60/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0017 -
binary_accuracy: 1.0000
Epoch 61/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0016 -
binary_accuracy: 1.0000
Epoch 62/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0015 -
binary_accuracy: 1.0000
Epoch 63/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0015 -
binary_accuracy: 1.0000
Epoch 64/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0014 -
binary_accuracy: 1.0000
Epoch 65/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0013 -
binary_accuracy: 1.0000
Epoch 66/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0012 -
binary_accuracy: 1.0000
Epoch 67/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0012 -
binary_accuracy: 1.0000
Epoch 68/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0011 -
binary_accuracy: 1.0000
Epoch 69/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0011 -
binary_accuracy: 1.0000
Epoch 70/100
22/22 [==============================] - 0s 2ms/step - loss: 0.0011 -
binary_accuracy: 1.0000
Epoch 71/100
22/22 [==============================] - 0s 2ms/step - loss: 9.6865e-04 -
binary_accuracy: 1.0000
Epoch 72/100
22/22 [==============================] - 0s 2ms/step - loss: 9.2122e-04 -
binary_accuracy: 1.0000
Epoch 73/100
22/22 [==============================] - 0s 2ms/step - loss: 8.9306e-04 -
binary_accuracy: 1.0000
```

```
Epoch 74/100
22/22 [==============================] - 0s 2ms/step - loss: 8.4052e-04 -
binary_accuracy: 1.0000
Epoch 75/100
22/22 [==============================] - 0s 2ms/step - loss: 8.0872e-04 -
binary_accuracy: 1.0000
Epoch 76/100
22/22 [==============================] - 0s 2ms/step - loss: 7.8107e-04 -
binary_accuracy: 1.0000
Epoch 77/100
22/22 [==============================] - 0s 2ms/step - loss: 7.4200e-04 -
binary_accuracy: 1.0000
Epoch 78/100
22/22 [==============================] - 0s 2ms/step - loss: 7.0876e-04 -
binary_accuracy: 1.0000
Epoch 79/100
22/22 [==============================] - 0s 2ms/step - loss: 6.7317e-04 -
binary_accuracy: 1.0000
Epoch 80/100
22/22 [==============================] - 0s 2ms/step - loss: 6.4450e-04 -
binary_accuracy: 1.0000
Epoch 81/100
22/22 [==============================] - 0s 2ms/step - loss: 6.3894e-04 -
binary_accuracy: 1.0000
Epoch 82/100
22/22 [==============================] - 0s 2ms/step - loss: 5.9556e-04 -
binary_accuracy: 1.0000
Epoch 83/100
22/22 [==============================] - 0s 2ms/step - loss: 5.8088e-04 -
binary_accuracy: 1.0000
Epoch 84/100
22/22 [==============================] - 0s 2ms/step - loss: 5.5271e-04 -
binary_accuracy: 1.0000
Epoch 85/100
22/22 [==============================] - 0s 2ms/step - loss: 5.2765e-04 -
binary_accuracy: 1.0000
Epoch 86/100
22/22 [==============================] - 0s 2ms/step - loss: 5.1495e-04 -
binary_accuracy: 1.0000
Epoch 87/100
22/22 [==============================] - 0s 2ms/step - loss: 4.9692e-04 -
binary_accuracy: 1.0000
Epoch 88/100
22/22 [==============================] - 0s 2ms/step - loss: 4.8501e-04 -
binary_accuracy: 1.0000
Epoch 89/100
22/22 [==============================] - 0s 2ms/step - loss: 4.6633e-04 -
binary_accuracy: 1.0000
```

```
Epoch 90/100
22/22 [==============================] - 0s 2ms/step - loss: 4.4070e-04 -
binary_accuracy: 1.0000
Epoch 91/100
22/22 [==============================] - 0s 2ms/step - loss: 4.2607e-04 -
binary_accuracy: 1.0000
Epoch 92/100
22/22 [==============================] - 0s 2ms/step - loss: 4.0934e-04 -
binary_accuracy: 1.0000
Epoch 93/100
22/22 [==============================] - 0s 2ms/step - loss: 4.0031e-04 -
binary_accuracy: 1.0000
Epoch 94/100
22/22 [==============================] - 0s 2ms/step - loss: 3.8763e-04 -
binary_accuracy: 1.0000
Epoch 95/100
22/22 [==============================] - 0s 2ms/step - loss: 3.7587e-04 -
binary_accuracy: 1.0000
Epoch 96/100
22/22 [==============================] - 0s 2ms/step - loss: 3.6396e-04 -
binary_accuracy: 1.0000
Epoch 97/100
22/22 [==============================] - 0s 2ms/step - loss: 3.4742e-04 -
binary_accuracy: 1.0000
Epoch 98/100
22/22 [==============================] - 0s 2ms/step - loss: 3.3786e-04 -
binary_accuracy: 1.0000
Epoch 99/100
22/22 [==============================] - 0s 2ms/step - loss: 3.2923e-04 -
binary_accuracy: 1.0000
Epoch 100/100
22/22 [==============================] - 0s 2ms/step - loss: 3.1609e-04 -
binary_accuracy: 1.0000
```

[ ]:
```
dnn_model_relu_sigmoid.evaluate(credit_batches_train)
```

```
22/22 [==============================] - 0s 2ms/step - loss: 3.0556e-04 -
binary_accuracy: 1.0000
```

[ ]:
```
[0.00030555881676264107, 1.0]
```

[ ]:
```
dnn_model_relu_sigmoid.evaluate(credit_batches_val)
```

```
5/5 [==============================] - 0s 2ms/step - loss: 1.6638 -
binary_accuracy: 0.7667

/home/filip/py-env/aai-2023-1/lib/python3.9/site-packages/keras/backend.py:5676:
UserWarning: "`binary_crossentropy` received `from_logits=True`, but the
`output` argument was produced by a Sigmoid activation and thus does not
```

```
represent logits. Was this intended?
  output, from_logits = _get_logits(
```

[ ]:
```
[1.6638044118881226, 0.7666666507720947]
```

[ ]:
```
# Try with dropout layers

def dnn_model(preprocessing_head, inputs):
  body = tf.keras.Sequential([
    layers.Dropout(0.5),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(32, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(16, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(1, activation='sigmoid')
  ])

  preprocessed_inputs = preprocessing_head(inputs)
  result = body(preprocessed_inputs)
  model = tf.keras.Model(inputs, result)

  model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
                metrics=tf.keras.metrics.BinaryAccuracy(),
                optimizer=tf.keras.optimizers.Adam())
  return model

dnn_model_dropouts = dnn_model(credit_preprocessing_train, train_inputs)
```

[ ]:
```
history = dnn_model_dropouts.fit(credit_batches_train, epochs=epochs, verbose=1)
```

```
Epoch 1/100

/home/filip/py-env/aai-2023-1/lib/python3.9/site-packages/keras/backend.py:5676:
UserWarning: "`binary_crossentropy` received `from_logits=True`, but the
`output` argument was produced by a Sigmoid activation and thus does not
represent logits. Was this intended?
  output, from_logits = _get_logits(

22/22 [==============================] - 1s 3ms/step - loss: 0.7863 -
binary_accuracy: 0.5714
Epoch 2/100
22/22 [==============================] - 0s 3ms/step - loss: 0.7441 -
binary_accuracy: 0.5857
Epoch 3/100
22/22 [==============================] - 0s 3ms/step - loss: 0.6829 -
binary_accuracy: 0.6186
Epoch 4/100
```

```
22/22 [==============================] - 0s 3ms/step - loss: 0.7065 -
binary_accuracy: 0.6114
Epoch 5/100
22/22 [==============================] - 0s 3ms/step - loss: 0.6769 -
binary_accuracy: 0.6014
Epoch 6/100
22/22 [==============================] - 0s 2ms/step - loss: 0.7047 -
binary_accuracy: 0.6157
Epoch 7/100
22/22 [==============================] - 0s 2ms/step - loss: 0.6475 -
binary_accuracy: 0.6443
Epoch 8/100
22/22 [==============================] - 0s 2ms/step - loss: 0.6502 -
binary_accuracy: 0.6529
Epoch 9/100
22/22 [==============================] - 0s 3ms/step - loss: 0.6500 -
binary_accuracy: 0.6600
Epoch 10/100
22/22 [==============================] - 0s 2ms/step - loss: 0.6272 -
binary_accuracy: 0.6843
Epoch 11/100
22/22 [==============================] - 0s 2ms/step - loss: 0.6310 -
binary_accuracy: 0.6771
Epoch 12/100
22/22 [==============================] - 0s 2ms/step - loss: 0.6533 -
binary_accuracy: 0.6786
Epoch 13/100
22/22 [==============================] - 0s 2ms/step - loss: 0.6296 -
binary_accuracy: 0.6700
Epoch 14/100
22/22 [==============================] - 0s 2ms/step - loss: 0.6417 -
binary_accuracy: 0.6686
Epoch 15/100
22/22 [==============================] - 0s 3ms/step - loss: 0.6239 -
binary_accuracy: 0.6829
Epoch 16/100
22/22 [==============================] - 0s 2ms/step - loss: 0.6195 -
binary_accuracy: 0.6957
Epoch 17/100
22/22 [==============================] - 0s 2ms/step - loss: 0.6365 -
binary_accuracy: 0.6871
Epoch 18/100
22/22 [==============================] - 0s 2ms/step - loss: 0.6132 -
binary_accuracy: 0.6929
Epoch 19/100
22/22 [==============================] - 0s 2ms/step - loss: 0.6249 -
binary_accuracy: 0.6929
Epoch 20/100
```

```
22/22 [==============================] - 0s 2ms/step - loss: 0.6073 -
binary_accuracy: 0.6986
Epoch 21/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5970 -
binary_accuracy: 0.6986
Epoch 22/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5856 -
binary_accuracy: 0.7000
Epoch 23/100
22/22 [==============================] - 0s 2ms/step - loss: 0.6202 -
binary_accuracy: 0.6943
Epoch 24/100
22/22 [==============================] - 0s 2ms/step - loss: 0.6084 -
binary_accuracy: 0.6914
Epoch 25/100
22/22 [==============================] - 0s 2ms/step - loss: 0.6079 -
binary_accuracy: 0.6929
Epoch 26/100
22/22 [==============================] - 0s 2ms/step - loss: 0.6010 -
binary_accuracy: 0.6957
Epoch 27/100
22/22 [==============================] - 0s 2ms/step - loss: 0.6064 -
binary_accuracy: 0.6943
Epoch 28/100
22/22 [==============================] - 0s 2ms/step - loss: 0.6030 -
binary_accuracy: 0.7071
Epoch 29/100
22/22 [==============================] - 0s 2ms/step - loss: 0.6017 -
binary_accuracy: 0.7000
Epoch 30/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5854 -
binary_accuracy: 0.7014
Epoch 31/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5908 -
binary_accuracy: 0.7086
Epoch 32/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5859 -
binary_accuracy: 0.7029
Epoch 33/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5909 -
binary_accuracy: 0.6986
Epoch 34/100
22/22 [==============================] - 0s 2ms/step - loss: 0.6072 -
binary_accuracy: 0.7029
Epoch 35/100
22/22 [==============================] - 0s 2ms/step - loss: 0.6003 -
binary_accuracy: 0.7000
Epoch 36/100
```

```
22/22 [==============================] - 0s 2ms/step - loss: 0.5871 -
binary_accuracy: 0.7043
Epoch 37/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5930 -
binary_accuracy: 0.7014
Epoch 38/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5924 -
binary_accuracy: 0.7029
Epoch 39/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5853 -
binary_accuracy: 0.6971
Epoch 40/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5759 -
binary_accuracy: 0.7057
Epoch 41/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5856 -
binary_accuracy: 0.7043
Epoch 42/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5816 -
binary_accuracy: 0.6986
Epoch 43/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5864 -
binary_accuracy: 0.7029
Epoch 44/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5729 -
binary_accuracy: 0.7014
Epoch 45/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5770 -
binary_accuracy: 0.7000
Epoch 46/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5786 -
binary_accuracy: 0.7043
Epoch 47/100
22/22 [==============================] - 0s 2ms/step - loss: 0.6012 -
binary_accuracy: 0.7029
Epoch 48/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5811 -
binary_accuracy: 0.7000
Epoch 49/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5964 -
binary_accuracy: 0.7029
Epoch 50/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5619 -
binary_accuracy: 0.7014
Epoch 51/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5791 -
binary_accuracy: 0.7043
Epoch 52/100
```

```
22/22 [==============================] - 0s 2ms/step - loss: 0.5740 -
binary_accuracy: 0.6943
Epoch 53/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5796 -
binary_accuracy: 0.7043
Epoch 54/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5838 -
binary_accuracy: 0.7057
Epoch 55/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5729 -
binary_accuracy: 0.7014
Epoch 56/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5698 -
binary_accuracy: 0.7043
Epoch 57/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5685 -
binary_accuracy: 0.7057
Epoch 58/100
22/22 [==============================] - 0s 3ms/step - loss: 0.5711 -
binary_accuracy: 0.7029
Epoch 59/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5626 -
binary_accuracy: 0.7129
Epoch 60/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5849 -
binary_accuracy: 0.6957
Epoch 61/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5713 -
binary_accuracy: 0.6957
Epoch 62/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5666 -
binary_accuracy: 0.7029
Epoch 63/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5623 -
binary_accuracy: 0.7029
Epoch 64/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5714 -
binary_accuracy: 0.7000
Epoch 65/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5593 -
binary_accuracy: 0.7000
Epoch 66/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5495 -
binary_accuracy: 0.6986
Epoch 67/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5675 -
binary_accuracy: 0.7071
Epoch 68/100
```

```
22/22 [==============================] - 0s 2ms/step - loss: 0.5522 -
binary_accuracy: 0.7043
Epoch 69/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5511 -
binary_accuracy: 0.7057
Epoch 70/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5657 -
binary_accuracy: 0.7157
Epoch 71/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5380 -
binary_accuracy: 0.7100
Epoch 72/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5499 -
binary_accuracy: 0.7071
Epoch 73/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5700 -
binary_accuracy: 0.7086
Epoch 74/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5377 -
binary_accuracy: 0.7086
Epoch 75/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5481 -
binary_accuracy: 0.7143
Epoch 76/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5685 -
binary_accuracy: 0.6957
Epoch 77/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5637 -
binary_accuracy: 0.7186
Epoch 78/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5648 -
binary_accuracy: 0.7086
Epoch 79/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5632 -
binary_accuracy: 0.7114
Epoch 80/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5368 -
binary_accuracy: 0.7100
Epoch 81/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5636 -
binary_accuracy: 0.7129
Epoch 82/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5725 -
binary_accuracy: 0.7100
Epoch 83/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5481 -
binary_accuracy: 0.7186
Epoch 84/100
```

```
22/22 [==============================] - 0s 2ms/step - loss: 0.5596 -
binary_accuracy: 0.7143
Epoch 85/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5589 -
binary_accuracy: 0.7129
Epoch 86/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5591 -
binary_accuracy: 0.7043
Epoch 87/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5543 -
binary_accuracy: 0.7143
Epoch 88/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5634 -
binary_accuracy: 0.6986
Epoch 89/100
22/22 [==============================] - 0s 3ms/step - loss: 0.5724 -
binary_accuracy: 0.7257
Epoch 90/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5690 -
binary_accuracy: 0.7057
Epoch 91/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5618 -
binary_accuracy: 0.7029
Epoch 92/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5532 -
binary_accuracy: 0.7057
Epoch 93/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5602 -
binary_accuracy: 0.7071
Epoch 94/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5552 -
binary_accuracy: 0.7114
Epoch 95/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5854 -
binary_accuracy: 0.7071
Epoch 96/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5724 -
binary_accuracy: 0.7071
Epoch 97/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5543 -
binary_accuracy: 0.7000
Epoch 98/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5444 -
binary_accuracy: 0.7043
Epoch 99/100
22/22 [==============================] - 0s 2ms/step - loss: 0.5448 -
binary_accuracy: 0.7186
Epoch 100/100
```

```
22/22 [==============================] - 0s 2ms/step - loss: 0.5645 -
binary_accuracy: 0.7000
```

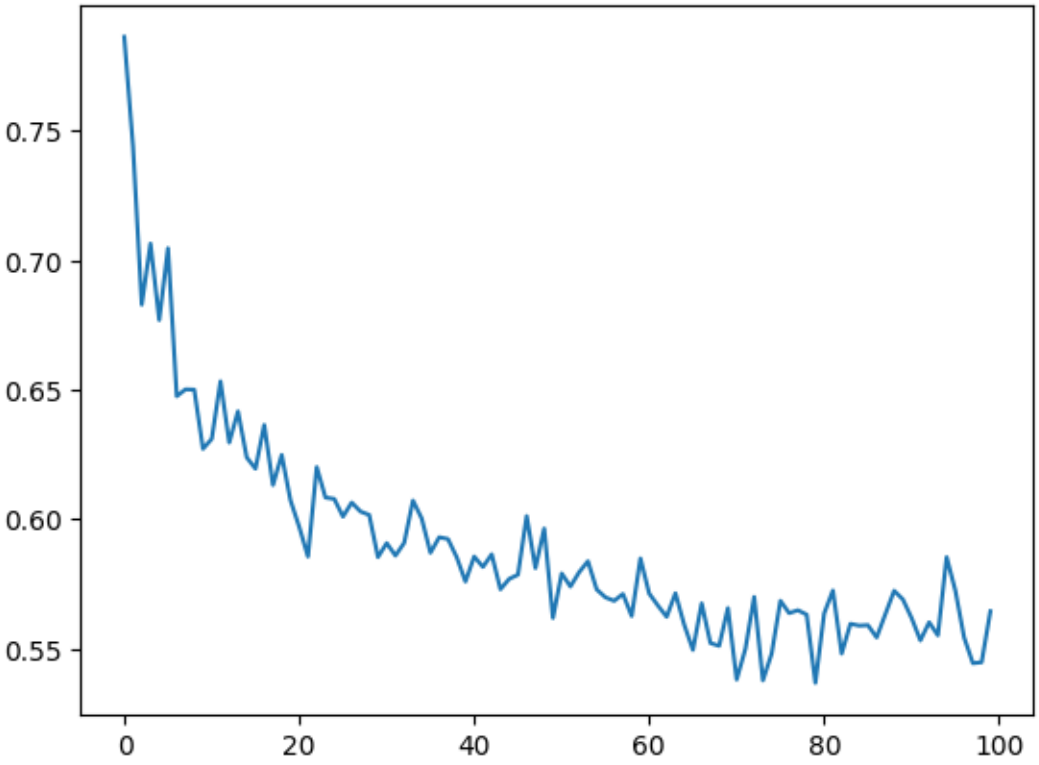
[ ]:
```
dnn_model_dropouts.evaluate(credit_batches_train)
```

```
22/22 [==============================] - 0s 2ms/step - loss: 0.5008 -
binary_accuracy: 0.7014
```

[ ]:
```
[0.500798761844635, 0.7014285922050476]
```

[ ]:
```
plt.plot([i for i in range(epochs)], history.history['loss'])
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7f5f243bac70>]
```



[ ]:
```
del dnn_model
del history
```

[ ]:
```
# Decrease dropout and learning rate

def dnn_model(preprocessing_head, inputs):
  body = tf.keras.Sequential([
    layers.Dropout(0.2),
    layers.Dense(64, activation='relu'),
```

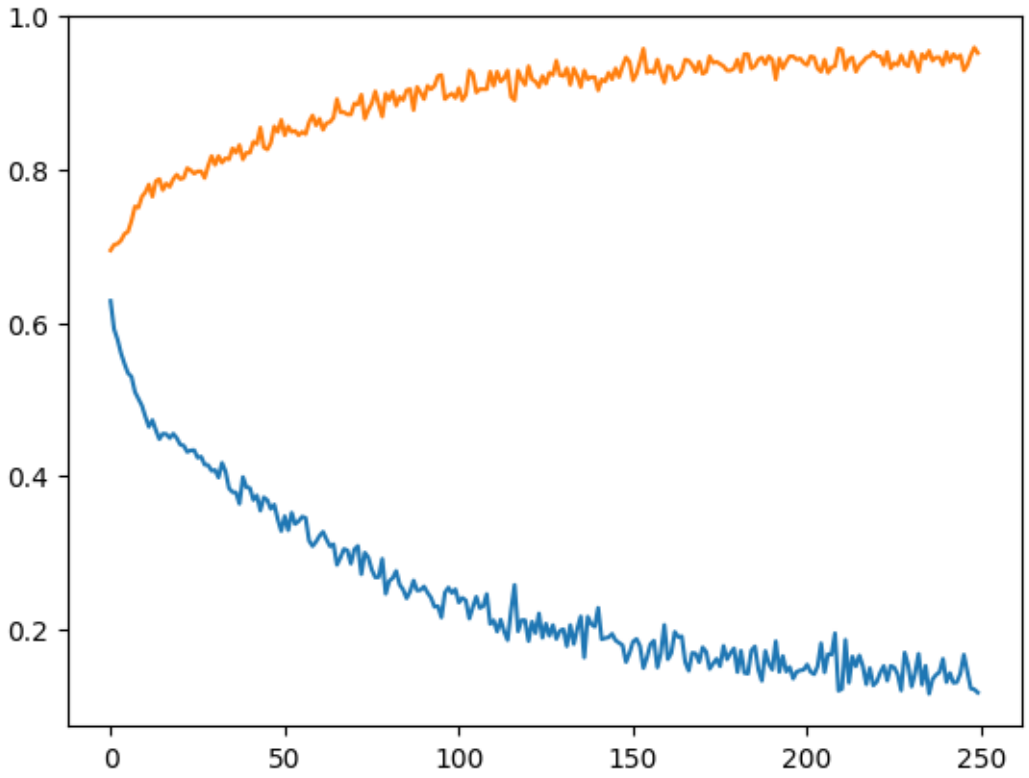
```
    layers.Dense(32, activation='relu'),
    layers.Dense(16, activation='relu'),
    layers.Dense(1, activation='sigmoid')
  ])

  preprocessed_inputs = preprocessing_head(inputs)
  result = body(preprocessed_inputs)
  model = tf.keras.Model(inputs, result)

  model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
                metrics=tf.keras.metrics.BinaryAccuracy(),
                optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005))
  return model

dnn_model_dropout_1 = dnn_model(credit_preprocessing_train, train_inputs)
```

[ ]:
```
epochs = 250
history = dnn_model_dropout_1.fit(credit_batches_train, epochs=epochs,␣
 ↪verbose=0)
```

```
/home/filip/py-env/aai-2023-1/lib/python3.9/site-packages/keras/backend.py:5676:
UserWarning: "`binary_crossentropy` received `from_logits=True`, but the
`output` argument was produced by a Sigmoid activation and thus does not
represent logits. Was this intended?
  output, from_logits = _get_logits(
```

[ ]:
```
plt.plot([i for i in range(epochs)], history.history['loss'], history.
 ↪history['binary_accuracy'])
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7f5eb99ba3d0>,
 <matplotlib.lines.Line2D at 0x7f5eb99ba430>]
```



[ ]:
```
dnn_model_dropout_1.evaluate(credit_batches_val)
```

```
5/5 [==============================] - 0s 2ms/step - loss: 0.8764 -
binary_accuracy: 0.7467
```

[ ]:
```
[0.8763694167137146, 0.746666669845581]
```

[ ]:
```
del dnn_model
del history
```

[ ]:
```
# Add L2 weight regularization

def dnn_model(preprocessing_head, inputs):
  body = tf.keras.Sequential([
    layers.Dropout(0.2),
    layers.Dense(64, kernel_regularizer='l2', activation='relu'),
    layers.Dense(32, kernel_regularizer='l2', activation='relu'),
    layers.Dense(16, kernel_regularizer='l2', activation='relu'),
    layers.Dense(1, activation='sigmoid')
  ])

  preprocessed_inputs = preprocessing_head(inputs)
```

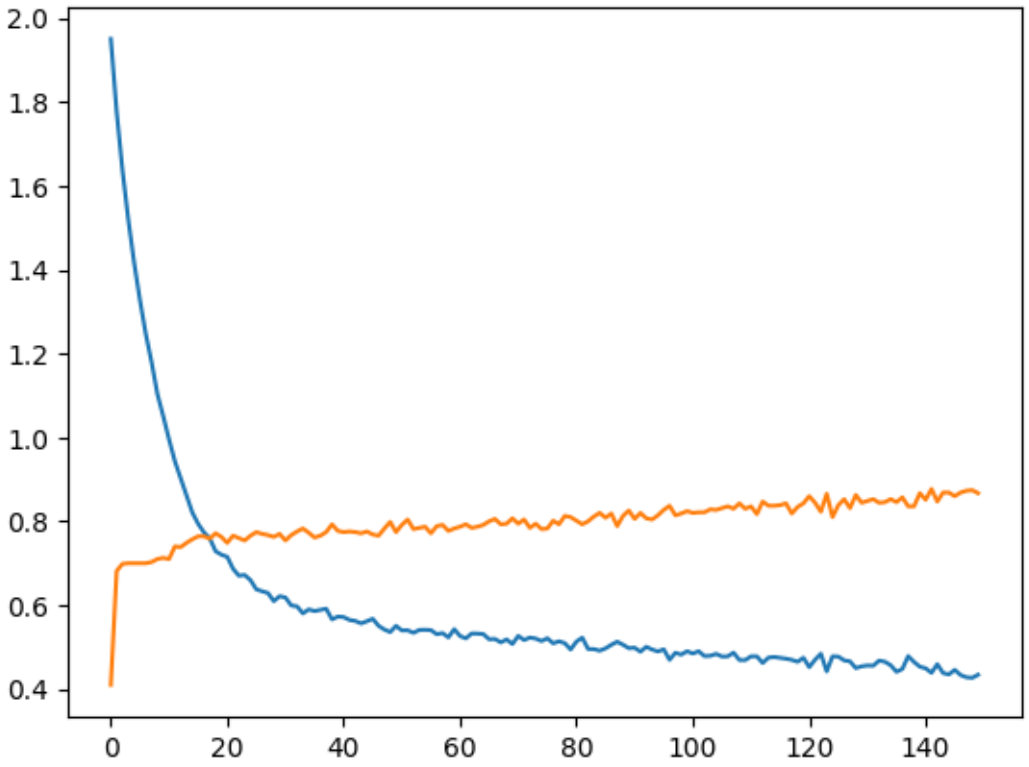
```
  result = body(preprocessed_inputs)
  model = tf.keras.Model(inputs, result)

  model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
                metrics=tf.keras.metrics.BinaryAccuracy(),
                optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005))
  return model

dnn_model_l2 = dnn_model(credit_preprocessing_train, train_inputs)
```

[ ]:
```
epochs = 150
history = dnn_model_l2.fit(credit_batches_train, epochs=epochs, verbose=0)
```

```
/home/filip/py-env/aai-2023-1/lib/python3.9/site-packages/keras/backend.py:5676:
UserWarning: "`binary_crossentropy` received `from_logits=True`, but the
`output` argument was produced by a Sigmoid activation and thus does not
represent logits. Was this intended?
  output, from_logits = _get_logits(
```

[ ]:
```
plt.plot([i for i in range(epochs)], history.history['loss'], history.
  ↪history['binary_accuracy'])
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7f5eb810f700>,
 <matplotlib.lines.Line2D at 0x7f5eb810f760>]
```



[ ]:
```
dnn_model_l2.evaluate(credit_batches_val)
```

```
5/5 [==============================] - 0s 3ms/step - loss: 0.6005 -
binary_accuracy: 0.7800
```

[ ]:
```
[0.6004582643508911, 0.7799999713897705]
```

[ ]:
```
dnn_model_l2.evaluate(credit_batches_test)
```

```
5/5 [==============================] - 0s 2ms/step - loss: 0.6342 -
binary_accuracy: 0.7333
```

[ ]:
```
[0.6341845393180847, 0.7333333492279053]
```

6 Model Performance Comparison

[ ]:
```
def acc(model, X, y):
    y_pred = model.predict(X)
    acc = accuracy_score(y, y_pred)
    return acc

def f1(model, X, y):
    y_pred = model.predict(X)
    f1 = f1_score(y, y_pred)
    return f1

# Accuracy
rf_acc = acc(rf_best, X_test, Y_test)
rf_os_acc = acc(rf_os_best, X_test, Y_test)
rf_os_smoten_acc = acc(rf_os_smoten_best, X_test, Y_test)
rf_us_acc = acc(rf_us_best, X_test, Y_test)
rf_ss_acc = acc(rf_ss_best, X_test_SS, y_test_SS)

lr_acc = acc(lr, X_test, Y_test)
lr_ss_acc = acc(lr_SS, X_test_SS, y_test_SS)
lr_smoten_acc = acc(lr_smoten, X_test, Y_test)
lr_cv_acc = acc(lr_cv, X_test, Y_test)

dnn_acc = dnn_model_1.evaluate(credit_batches_test,␣
 ↪return_dict=True)['binary_accuracy']
dnn_relu_sigmoid_acc = dnn_model_relu_sigmoid.evaluate(credit_batches_test,␣
 ↪return_dict=True)['binary_accuracy']
dnn_dropout_1_acc = dnn_model_dropout_1.evaluate(credit_batches_test,␣
 ↪return_dict=True)['binary_accuracy']
dnn_dropouts_acc = dnn_model_dropouts.evaluate(credit_batches_test,␣
 ↪return_dict=True)['binary_accuracy']
```

```
dnn_l2_acc = dnn_model_l2.evaluate(credit_batches_test,␣
 ↪return_dict=True)['binary_accuracy']

# F1
rf_f1 = f1(rf_best, X_test, Y_test)
rf_os_f1 = f1(rf_os_best, X_test, Y_test)
rf_os_smoten_f1 = f1(rf_os_smoten_best, X_test, Y_test)
rf_us_f1 = f1(rf_us_best, X_test, Y_test)
rf_ss_f1 = f1(rf_ss_best, X_test_SS, y_test_SS)

lr_f1 = f1(lr, X_test, Y_test)
lr_ss_f1 = f1(lr_SS, X_test_SS, y_test_SS)
lr_smoten_f1 = f1(lr_smoten, X_test, Y_test)
lr_cv_f1 = f1(lr_cv, X_test, Y_test)
```

```
5/5 [==============================] - 0s 2ms/step - loss: 0.5276 -
binary_accuracy: 0.6800
5/5 [==============================] - 0s 2ms/step - loss: 1.7848 -
binary_accuracy: 0.7467
5/5 [==============================] - 0s 2ms/step - loss: 0.8073 -
binary_accuracy: 0.7933
5/5 [==============================] - 0s 2ms/step - loss: 0.5365 -
binary_accuracy: 0.7067
5/5 [==============================] - 0s 2ms/step - loss: 0.6342 -
binary_accuracy: 0.7333
```

[ ]:
```
# Creating tabular format for better comparison
tbl=pd.DataFrame()
tbl['Model']=pd.Series(['RF',
                        'RF OverSampling',
                        'RF SMOTEN',
                        'RF UnderSampling',
                        'RF Stratified',
                        'LR',
                        'LR Stratified',
                        'LR SMOTEN',
                        'LR Cross-validation',
                        'DNN',
                        'DNN ReLU',
                        'DNN Big Dropout',
                        'DNN Small Dropout',
                        'DNN L2 reg.'])
tbl['Accuracy']=pd.Series([rf_acc,
                           rf_os_acc,
                           rf_os_smoten_acc,
                           rf_us_acc,
                           rf_ss_acc,
```

```
                            lr_acc,
                            lr_ss_acc,
                            lr_smoten_acc,
                            lr_cv_acc,
                            dnn_acc,
                            dnn_relu_sigmoid_acc,
                            dnn_dropout_1_acc,
                            dnn_dropouts_acc,
                            dnn_l2_acc])
tbl['F1_Score']=pd.Series([rf_f1,
                            rf_os_f1,
                            rf_os_smoten_f1,
                            rf_us_f1,
                            rf_ss_f1,
                            lr_f1,
                            lr_ss_f1,
                            lr_smoten_f1,
                            lr_cv_f1])
tbl.sort_values('Accuracy', ascending=False)
```

[ ]:

	Model	Accuracy	F1_Score
2	RF SMOTEN	0.975000	0.982206
11	DNN Big Dropout	0.793333	NaN
1	RF OverSampling	0.770000	0.839161
10	DNN ReLU	0.746667	NaN
0	RF	0.740000	0.832258
3	RF UnderSampling	0.735000	0.792157
13	DNN L2 reg.	0.733333	NaN
5	LR	0.730000	0.823529
7	LR SMOTEN	0.730000	0.798507
8	LR Cross-validation	0.730000	0.821192
4	RF Stratified	0.706667	0.813559
12	DNN Small Dropout	0.706667	NaN
9	DNN	0.680000	NaN
6	LR Stratified	0.660000	0.777293

[ ]:
```
# Best model on the basis of F1 Score
tbl.sort_values('F1_Score', ascending=False)
```

[ ]:

	Model	Accuracy	F1_Score
2	RF SMOTEN	0.975000	0.982206
1	RF OverSampling	0.770000	0.839161
0	RF	0.740000	0.832258
5	LR	0.730000	0.823529
8	LR Cross-validation	0.730000	0.821192
4	RF Stratified	0.706667	0.813559
7	LR SMOTEN	0.730000	0.798507

```
3      RF UnderSampling  0.735000  0.792157
6         LR Stratified  0.660000  0.777293
9                   DNN  0.680000       NaN
10              DNN ReLU  0.746667       NaN
11       DNN Big Dropout  0.793333       NaN
12    DNN Small Dropout  0.706667       NaN
13          DNN L2 reg.  0.733333       NaN
```

[ ]: