Experiment 5

Problem Statement:

To build an advance ANN classification model for churn modelling data with:

- a. Cross Validation
- b. Grid Search
- c. Checkpoint

GitHub & Google Colab Links:

GitHub Link: https://github.com/piyush-gambhir/ncu-lab-manual-and-end-semester-projects/blob/main/NCU-CSL312%20-%20DL%20-%20Lab%20Manual/Experiment%205/Experiment%205.ipynb

Google Colab Link:

Installing Dependencies:

In [ ]:
```
! pip install tabulate numpy pandas matplotlib seaborn
```

```
Requirement already satisfied: tabulate in c:\users\mainp\appdata\local\programs\python\python311\lib\site-packa
ges (0.9.0)
Requirement already satisfied: numpy in c:\users\mainp\appdata\local\programs\python\python311\lib\site-packages
(1.26.4)
Requirement already satisfied: pandas in c:\users\mainp\appdata\local\programs\python\python311\lib\site-package
s (2.2.2)
Requirement already satisfied: matplotlib in c:\users\mainp\appdata\local\programs\python\python311\lib\site-pac
kages (3.8.4)
Requirement already satisfied: seaborn in c:\users\mainp\appdata\local\programs\python\python311\lib\site-packag
es (0.13.2)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\mainp\appdata\local\programs\python\python311\
lib\site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\mainp\appdata\local\programs\python\python311\lib\site-p
ackages (from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\mainp\appdata\local\programs\python\python311\lib\site
-packages (from pandas) (2024.1)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\mainp\appdata\local\programs\python\python311\lib\si
te-packages (from matplotlib) (1.2.1)
Requirement already satisfied: cycler>=0.10 in c:\users\mainp\appdata\local\programs\python\python311\lib\site-p
ackages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\mainp\appdata\local\programs\python\python311\lib\s
ite-packages (from matplotlib) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\mainp\appdata\local\programs\python\python311\lib\s
ite-packages (from matplotlib) (1.4.5)
Requirement already satisfied: packaging>=20.0 in c:\users\mainp\appdata\local\programs\python\python311\lib\sit
e-packages (from matplotlib) (24.0)
Requirement already satisfied: pillow>=8 in c:\users\mainp\appdata\local\programs\python\python311\lib\site-pack
ages (from matplotlib) (10.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\mainp\appdata\local\programs\python\python311\lib\si
te-packages (from matplotlib) (3.1.2)
Requirement already satisfied: six>=1.5 in c:\users\mainp\appdata\local\programs\python\python311\lib\site-packa
ges (from python-dateutil>=2.8.2->pandas) (1.16.0)
```

Code

In [ ]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.base import BaseEstimator, ClassifierMixin
```

In [ ]:
```
# Load the dataset
data = pd.read_csv("./churn_modelling.csv")

# Drop the columns that are not needed for modeling
```

```
data = data.drop(['RowNumber', 'CustomerId', 'Surname'], axis=1)

# Separate features and target variable
X = data.drop('Exited', axis=1)
y = data['Exited']

# Preprocessing for numeric columns: scale numeric features
numeric_features = X.select_dtypes(
    include=['int64', 'float64']).columns.difference(['HasCrCard', 'IsActiveMember'])
numeric_transformer = StandardScaler()

# Preprocessing for categorical columns: one-hot encode categorical features
categorical_features = ['Geography', 'Gender']
categorical_transformer = OneHotEncoder(drop='first')

# Create the preprocessing pipeline
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])
```

In [ ]:
```
# Define the Keras Classifier Wrapper


class KerasClassifierWrapper(BaseEstimator, ClassifierMixin):
    def __init__(self, neurons=64):
        self.neurons = neurons
        self.model = None

    def fit(self, X, y, **kwargs):
        def create_model():
            model = Sequential()
            model.add(Dense(self.neurons, activation='relu',
                      input_shape=(X.shape[1],)))
            model.add(Dropout(0.2))
            model.add(Dense(self.neurons, activation='relu'))
            model.add(Dropout(0.2))
            model.add(Dense(1, activation='sigmoid'))
            model.compile(optimizer='adam',
                          loss='binary_crossentropy', metrics=['accuracy'])
            return model

        self.model = create_model()
        self.model.fit(X, y, **kwargs)
        return self

    def predict(self, X, **kwargs):
        return (self.model.predict(X, **kwargs) > 0.5).astype("int32")

    def score(self, X, y, **kwargs):
        _, accuracy = self.model.evaluate(X, y, **kwargs)
        return accuracy

    def get_params(self, deep=True):
        return {'neurons': self.neurons}

    def set_params(self, **parameters):
        for parameter, value in parameters.items():
            setattr(self, parameter, value)
        return self
```

In [ ]:
```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

# Set up a pipeline that includes preprocessing and the estimator
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('classifier', KerasClassifierWrapper())])

# Hyperparameter grid
param_grid = {
    'classifier__neurons': [32, 64, 128],
}

# Grid search setup
grid = GridSearchCV(pipeline, param_grid, cv=3)

# Perform the grid search
grid_result = grid.fit(X_train, y_train)

# Evaluate the model
print("Best parameters found: ", grid_result.best_params_)
```

```
print("Best accuracy found: ", grid_result.best_score_)

best_model = grid_result.best_estimator_
X_test_transformed = best_model.named_steps['preprocessor'].transform(X_test)
test_accuracy = best_model.named_steps['classifier'].score(
    X_test_transformed, y_test)
print(f"Test Accuracy: {test_accuracy:.4f}")
```

```
c:\Users\mainp\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\layers\core\dense.py:86: User
Warning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer usin
g an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
167/167 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.7645 - loss: 0.5414
84/84 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7920 - loss: 0.4683
c:\Users\mainp\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\layers\core\dense.py:86: User
Warning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer usin
g an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
167/167 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.7094 - loss: 0.5776
84/84 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7811 - loss: 0.4674
c:\Users\mainp\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\layers\core\dense.py:86: User
Warning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer usin
g an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
167/167 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.6786 - loss: 0.6017
84/84 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8058 - loss: 0.4477
c:\Users\mainp\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\layers\core\dense.py:86: User
Warning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer usin
g an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
167/167 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.7945 - loss: 0.5081
84/84 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8012 - loss: 0.4535
c:\Users\mainp\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\layers\core\dense.py:86: User
Warning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer usin
g an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
167/167 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.7771 - loss: 0.5325
84/84 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7934 - loss: 0.4441
c:\Users\mainp\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\layers\core\dense.py:86: User
Warning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer usin
g an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
167/167 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.7644 - loss: 0.5186
84/84 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8148 - loss: 0.4228
c:\Users\mainp\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\layers\core\dense.py:86: User
Warning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer usin
g an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
167/167 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.7817 - loss: 0.4978
84/84 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8183 - loss: 0.4240
c:\Users\mainp\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\layers\core\dense.py:86: User
Warning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer usin
g an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
167/167 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7870 - loss: 0.4995
84/84 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7992 - loss: 0.4358
c:\Users\mainp\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\layers\core\dense.py:86: User
Warning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer usin
g an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
167/167 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.7589 - loss: 0.5112
84/84 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8237 - loss: 0.4139
c:\Users\mainp\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\layers\core\dense.py:86: User
Warning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer usin
g an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.7720 - loss: 0.5011
Best parameters found:  {'classifier__neurons': 128}
Best accuracy found:  0.8147505720456442
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8328 - loss: 0.3904
Test Accuracy: 0.8415
```