IT3071 – Machine Learning and Optimization Methods

Lab Sheet 04

Part A

01) Import the dataset

- Import required libraries for data handling and deep learning (Ex- pandas, numpy, sklearn utilities, tensorflow/keras).
- Load the CSV into a DataFrame and separate features **X** and label **y**.
- Display:
  - Dataset shape
  - First few rows of **X**
  - Class balance (counts of 0 and 1)

02) Split (and scale) the data

- Split into train/test sets (use a fixed random state).
- Explain why feature scaling is recommended for MLPs; apply **StandardScaler** correctly (fit on train only; transform both train and test).
- Set seeds for numpy and tensorflow to improve reproducibility.

03) Create the Keras MLP Classifier model

- Build a **Sequential** model with this architecture (to mirror your sklearn setup):
  - Input: shape = (n_features,)
  - Dense(5, activation='relu')
  - Dense(3, activation='relu')
  - Dense(1, activation='sigmoid') (binary output)
- Choose loss and metrics appropriate for **binary** classification and justify your choice.
- Choose an optimizer (Ex- Adam) and state your learning-rate rationale.

04) Compile and train

- Compile the model specifying **optimizer**, **loss**, and **metrics**.

- Train with suitable **epochs**, **batch_size**, **validation_split**, and **verbose**.

05) Evaluate classification performance

- Evaluate on the **test** set and report **accuracy**.
- Convert predicted probabilities to class labels using a **0.5** threshold and show a few predictions.
- Also compute and discuss: **confusion matrix**, **precision**, **recall**, **F1-score** (why they matter for imbalance).

Part B

01) Import the dataset

- Import required libraries (Ex- pandas, numpy, sklearn utilities, tensorflow/keras).
- Load the CSV; separate features **X** and target **y**.
- Display:
    - Dataset shape
    - First few rows of **X**
    - First few values of **y**

02) Split (and scale) the data

- Split into train/test sets (use a fixed random state).
- Explain why feature scaling is helpful for MLP regressors; apply **StandardScaler** correctly.
- Set seeds for reproducibility.

03) Create the Keras MLP Regressor model

- Build a **Sequential** model that mirrors your earlier hidden sizes (3, 2) with **either**:
    - **Linear** (identity) hidden activations to exactly mirror the earlier sheet, **or**
    - **ReLU** hidden activations for potentially better performance.
- Example (linear/identity mirror):
    - Input: shape = (n_features,)
    - Dense(3, activation=None)

  - Dense(2, activation=None)
  - Dense(1, activation=None) (linear output)
- Select loss/metrics for regression (Ex- **MSE**, **RMSE**) and justify.

04) Compile and train

- Compile with appropriate optimizer (Ex- **Adam**) and loss (**'mse'**).
- Train with chosen **epochs**, **batch_size**, **validation_split**, **verbose**.

05) Evaluate regression performance

- Evaluate on the **test** set and report **RMSE** (or compute via predictions).
- Additionally compute **MSE** and **MAE**; interpret what lower values mean in context.

Import the libraries

In [ ]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
import tensorflow as tf
from tensorflow import keras
```

Part A

Import data

In [ ]:
```
data = pd.read_csv("diabetes.CSV")
X = data.iloc[:, :8].values
y = data.iloc[:, 8].values
```

Training and testing data

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, random_state=0, stratify=y)
```

Standardizing

In [ ]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test  = scaler.transform(X_test)
```

Model architecture

In [ ]:
```
model = keras.Sequential([
    keras.layers.Input(shape=(X_train.shape[1],)),
    keras.layers.Dense(5, activation="relu"),
    keras.layers.Dense(3, activation="relu"),
    keras.layers.Dense(1, activation="sigmoid")
])
```

Compiling the model

In [ ]:
```
model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"]
```

Training

In [ ]:
```
history = model.fit(
    X_train, y_train,
    epochs=100,
    batch_size=32,
    validation_split=0.2,
    verbose=1
)
```

Validating

In [ ]:
```
loss, acc = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Accuracy: {acc:.4f}")
```

In [ ]:
```
y_pred_prob = model.predict(X_test).ravel()
y_pred = (y_pred_prob >= 0.5).astype(int)
print(y_pred[:10])
```

Part B

In [ ]:
```
data = pd.read_csv("Boston.CSV")
X = data.iloc[:, :12].values
y = data.iloc[:, 12].values
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, random_state=0)
```

In [ ]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test  = scaler.transform(X_test)
```

In [ ]:
```
reg_model = keras.Sequential([
    keras.layers.Input(shape=(X_train.shape[1],)),
    keras.layers.Dense(3, activation=None),
    keras.layers.Dense(2, activation=None),
    keras.layers.Dense(1, activation=None)
])
```

In [ ]:
```
reg_model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.01),
                  loss="mse",
                  metrics=[keras.metrics.RootMeanSquaredError()])
```

In [ ]:
```
reg_history = reg_model.fit(
    X_train, y_train,
    epochs=300,
    batch_size=32,
    validation_split=0.2,
    verbose=0
)
```

In [ ]:
```
rmse = reg_model.evaluate(X_test, y_test, verbose=0)[1]
print(f"Test RMSE: {rmse:.4f}")
```

In [ ]:
```
y_pred = reg_model.predict(X_test).ravel()
print(np.sqrt(mean_squared_error(y_test, y_pred)))
```