Homework 6

Sophia Godfrey

801149485

Github Link: https://github.com/QueenSophiaLo/Intro-To-ML/tree/main/Homework%206

```
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report, confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
import tensorflow as tf
from tensorflow import keras

layers = tf.keras.layers
```

```
# Reproducibility
seed = 42
np.random.seed(seed)
random.seed(seed)
tf.random.set_seed(seed)
```

Problem 1

Using the diabetes.csv file featured in previous homework assignments, three machine learning models (a Deep Neural Network (DNN), Logistic Regression (LR), and Support Vector Machine (SVM)) will be implemented and their results compared.

```
# Load dataset
df = pd.read_csv('diabetes.csv')

X = df.drop(columns=['Outcome']).values
y = df['Outcome'].values
```

The dataset was divided into an 80% training set and a 20% validation set. The split was performed using stratified sampling to ensure that the ratio of positive (diabetic) to negative (non-diabetic) outcomes was consistent across both partitions. All input features were Standardized using `sklearn.preprocessing.StandardScaler`. The scaler was fitted only on the training data to prevent data leakage and then used to transform both the training and validation sets.

```
# Train/validation split (80% train / 20% validation)
X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.20, random_state=seed, stratify=y
)

# Standardize features (fit on train only)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
```

The model was compiled using the Adam optimizer and binary_crossentropy loss.

```
# Build fully connected neural network with multiple hidden layers
def build_model(input_dim):
    model = keras.Sequential([
        layers.Input(shape=(input_dim,)),
        layers.Dense(64, activation='relu'),
        layers.BatchNormalization(),
        layers.Dropout(0.3),
        layers.Dense(32, activation='relu'),
        layers.BatchNormalization(),
        layers.Dropout(0.2),
        layers.Dense(16, activation='relu'),
        layers.Dense(1, activation='sigmoid')   # binary classification
    ])
    return model

model = build_model(X_train.shape[1])
model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=1e-3),
    loss='binary_crossentropy',
    metrics=['accuracy']
)
model.summary()
```

```
Model: "sequential_2"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ dense_8 (Dense)                 │ (None, 64)             │           576 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_4           │ (None, 64)             │           256 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_4 (Dropout)             │ (None, 64)             │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_9 (Dense)                 │ (None, 32)             │         2,080 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_5           │ (None, 32)             │           128 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_5 (Dropout)             │ (None, 32)             │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_10 (Dense)                │ (None, 16)             │           528 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_11 (Dense)                │ (None, 1)              │            17 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 3,585 (14.00 KB)
 Trainable params: 3,393 (13.25 KB)
 Non-trainable params: 192 (768.00 B)
```

Training was regulated using Early Stopping with patience=15 on the validation loss to restore the best weights and prevent overfitting.

```
# Train the model
# Use EarlyStopping to avoid overfitting and to find a good stopping point.
callbacks = [
    keras.callbacks.EarlyStopping(monitor='val_loss', patience=15, restore_best_weights=True)
]

history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=200,
    batch_size=32,
    callbacks=callbacks,
    verbose=2
)
```

```
Epoch 1/200
20/20 - 4s - 187ms/step - accuracy: 0.5244 - loss: 0.7942 - val_accuracy: 0.7143 - val_loss: 0.6478
Epoch 2/200
20/20 - 0s - 14ms/step - accuracy: 0.6857 - loss: 0.6045 - val_accuracy: 0.7662 - val_loss: 0.5895
Epoch 3/200
20/20 - 0s - 8ms/step - accuracy: 0.7459 - loss: 0.5360 - val_accuracy: 0.7597 - val_loss: 0.5508
Epoch 4/200
20/20 - 0s - 7ms/step - accuracy: 0.7443 - loss: 0.5219 - val_accuracy: 0.7597 - val_loss: 0.5256
Epoch 5/200
20/20 - 0s - 7ms/step - accuracy: 0.7785 - loss: 0.4895 - val_accuracy: 0.7597 - val_loss: 0.5084
Epoch 6/200
20/20 - 0s - 8ms/step - accuracy: 0.7476 - loss: 0.5030 - val_accuracy: 0.7532 - val_loss: 0.4998
Epoch 7/200
20/20 - 1s - 31ms/step - accuracy: 0.7606 - loss: 0.4878 - val_accuracy: 0.7273 - val_loss: 0.4940
Epoch 8/200
20/20 - 0s - 7ms/step - accuracy: 0.7720 - loss: 0.4763 - val_accuracy: 0.7403 - val_loss: 0.4866
Epoch 9/200
20/20 - 0s - 7ms/step - accuracy: 0.7818 - loss: 0.4684 - val_accuracy: 0.7597 - val_loss: 0.4768
Epoch 10/200
20/20 - 0s - 7ms/step - accuracy: 0.7932 - loss: 0.4599 - val_accuracy: 0.7532 - val_loss: 0.4725
Epoch 11/200
20/20 - 1s - 29ms/step - accuracy: 0.7899 - loss: 0.4580 - val_accuracy: 0.7597 - val_loss: 0.4725
Epoch 12/200
```

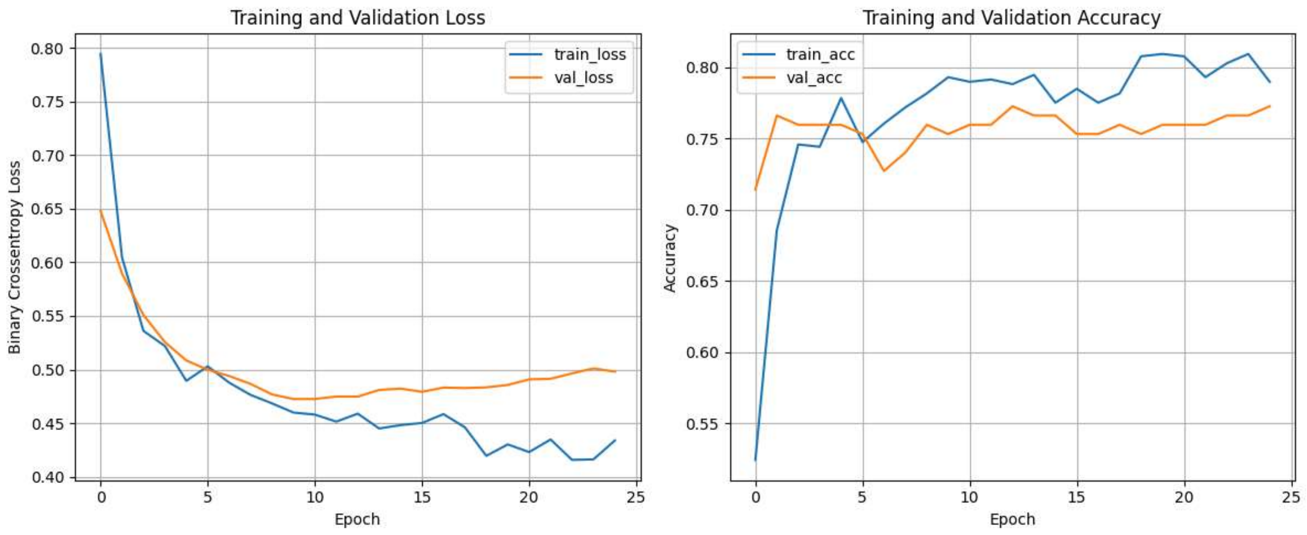
```
20/20 - 0s - 14ms/step - accuracy: 0.7915 - loss: 0.4514 - val_accuracy: 0.7597 - val_loss: 0.4748
Epoch 13/200
20/20 - 0s - 7ms/step - accuracy: 0.7883 - loss: 0.4589 - val_accuracy: 0.7727 - val_loss: 0.4748
Epoch 14/200
20/20 - 1s - 30ms/step - accuracy: 0.7948 - loss: 0.4450 - val_accuracy: 0.7662 - val_loss: 0.4810
Epoch 15/200
20/20 - 0s - 10ms/step - accuracy: 0.7752 - loss: 0.4482 - val_accuracy: 0.7662 - val_loss: 0.4822
Epoch 16/200
20/20 - 0s - 7ms/step - accuracy: 0.7850 - loss: 0.4502 - val_accuracy: 0.7532 - val_loss: 0.4793
Epoch 17/200
20/20 - 0s - 7ms/step - accuracy: 0.7752 - loss: 0.4584 - val_accuracy: 0.7532 - val_loss: 0.4832
Epoch 18/200
20/20 - 0s - 7ms/step - accuracy: 0.7818 - loss: 0.4461 - val_accuracy: 0.7597 - val_loss: 0.4827
Epoch 19/200
20/20 - 0s - 8ms/step - accuracy: 0.8078 - loss: 0.4195 - val_accuracy: 0.7532 - val_loss: 0.4834
Epoch 20/200
20/20 - 0s - 7ms/step - accuracy: 0.8094 - loss: 0.4302 - val_accuracy: 0.7597 - val_loss: 0.4856
Epoch 21/200
20/20 - 0s - 7ms/step - accuracy: 0.8078 - loss: 0.4231 - val_accuracy: 0.7597 - val_loss: 0.4908
Epoch 22/200
20/20 - 0s - 7ms/step - accuracy: 0.7932 - loss: 0.4348 - val_accuracy: 0.7597 - val_loss: 0.4914
Epoch 23/200
20/20 - 0s - 7ms/step - accuracy: 0.8029 - loss: 0.4158 - val_accuracy: 0.7662 - val_loss: 0.4964
Epoch 24/200
20/20 - 0s - 7ms/step - accuracy: 0.8094 - loss: 0.4162 - val_accuracy: 0.7662 - val_loss: 0.5010
Epoch 25/200
20/20 - 0s - 7ms/step - accuracy: 0.7899 - loss: 0.4339 - val_accuracy: 0.7727 - val_loss: 0.4980
```

```python
# Plot training & validation loss and accuracy
plt.figure(figsize=(12,5))

plt.subplot(1,2,1)
plt.plot(history.history['loss'], label='train_loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Binary Crossentropy Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.grid(True)

plt.subplot(1,2,2)
plt.plot(history.history['accuracy'], label='train_acc')
plt.plot(history.history['val_accuracy'], label='val_acc')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```



The performance of all classification algorithms (Neural Network, Logistic Regression, and SVM) was evaluated using four metrics.

1. **Accuracy** provides the general measure of model correctness by considering the number of sucessful diabetes diagnoses.
2. **Precision** measures the model's exactness, indicating how often a patient predicted to have diabetes is actually diabetic, thereby minimizing false alarms (False Positives).
3. **Recall** (or Sensitivity) measures the ability of a model to identify all relevant instances within a dataset. High Recall is important when classifying medical ailments (like diabetes in this scenario) even if there are some false positives. This is becuase a false negative (failing to identify diabetes) can have severe consequences for patients.
4. **F1 Score** is used to provide a single, balanced measure of a model's robustness, and is the average of Precision and Recall.

```
# Predictions and metrics for neural network
y_val_pred_proba = model.predict(X_val).ravel()
y_val_pred = (y_val_pred_proba >= 0.5).astype(int)

def print_metrics(y_true, y_pred, name='Model'):
    acc = accuracy_score(y_true, y_pred)
    prec = precision_score(y_true, y_pred, zero_division=0)
    rec = recall_score(y_true, y_pred, zero_division=0)
    f1 = f1_score(y_true, y_pred, zero_division=0)
    print(f"--- {name} ---")
    print(f"Accuracy : {acc:.4f}")
    print(f"Precision: {prec:.4f}")
    print(f"Recall   : {rec:.4f}")
    print(f"F1 score : {f1:.4f}")
    print("Confusion matrix:")
    print(confusion_matrix(y_true, y_pred))
    print("\nClassification report:")
    print(classification_report(y_true, y_pred, zero_division=0))
    return {'accuracy': acc, 'precision': prec, 'recall': rec, 'f1': f1}

nn_metrics = print_metrics(y_val, y_val_pred, name='Neural Network (Dense)')
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
--- Neural Network (Dense) ---
Accuracy : 0.7532
Precision: 0.7000
Recall   : 0.5185
F1 score : 0.5957
Confusion matrix:
[[88 12]
 [26 28]]

Classification report:
              precision    recall  f1-score   support

           0       0.77      0.88      0.82       100
           1       0.70      0.52      0.60        54

    accuracy                           0.75       154
   macro avg       0.74      0.70      0.71       154
weighted avg       0.75      0.75      0.74       154
```

```
# Logistic Regression baseline
logreg = LogisticRegression(max_iter=1000, random_state=seed)
logreg.fit(X_train, y_train)
y_lr = logreg.predict(X_val)
lr_metrics = print_metrics(y_val, y_lr, name='Logistic Regression')
```

```
--- Logistic Regression ---
Accuracy : 0.7143
Precision: 0.6087
Recall   : 0.5185
F1 score : 0.5600
Confusion matrix:
[[82 18]
 [26 28]]

Classification report:
              precision    recall  f1-score   support

           0       0.76      0.82      0.79       100
           1       0.61      0.52      0.56        54

    accuracy                           0.71       154
   macro avg       0.68      0.67      0.67       154
weighted avg       0.71      0.71      0.71       154
```

```
# Support Vector Machine baseline (RBF kernel)
svc = SVC(kernel='rbf', probability=True, random_state=seed)
svc.fit(X_train, y_train)
y_svc = svc.predict(X_val)
svc_metrics = print_metrics(y_val, y_svc, name='SVM (RBF)')
```

```
--- SVM (RBF) ---
Accuracy : 0.7532
Precision: 0.6600
Recall   : 0.6111
F1 score : 0.6346
Confusion matrix:
[[83 17]
 [21 33]]

Classification report:
              precision    recall  f1-score   support

           0       0.80      0.83      0.81       100
           1       0.66      0.61      0.63        54

    accuracy                           0.75       154
   macro avg       0.73      0.72      0.72       154
weighted avg       0.75      0.75      0.75       154
```

```
summary = pd.DataFrame([nn_metrics, lr_metrics, svc_metrics], index=['NeuralNet','Logistic','SVM'])
print("\nSummary comparison (validation set):")
print(summary)
```

```

Summary comparison (validation set):
           accuracy  precision    recall        f1
NeuralNet  0.753247   0.700000  0.518519  0.595745
Logistic   0.714286   0.608696  0.518519  0.560000
SVM        0.753247   0.660000  0.611111  0.634615
```

```
# Confusion Matrix comparison
def calculate_confusion_metrics(y_true, y_pred):
    """Calculates TN, FP, FN, TP from the confusion matrix."""
    # confusion_matrix returns: [[TN, FP], [FN, TP]]
    tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
    return {'TN (True Negative)': tn,
            'FP (False Positive)': fp,
            'FN (False Negative)': fn,
            'TP (True Positive)': tp}

# Calculate raw confusion matrix components for each model
# These use the predictions (y_val_pred, y_lr, y_svc) calculated above
nn_cm_metrics = calculate_confusion_metrics(y_val, y_val_pred)
lr_cm_metrics = calculate_confusion_metrics(y_val, y_lr)
svc_cm_metrics = calculate_confusion_metrics(y_val, y_svc)

# Create the comparison DataFrame
comparison_data = [
    {'Model': 'Neural Network (Dense)', **nn_cm_metrics},
    {'Model': 'Logistic Regression', **lr_cm_metrics},
    {'Model': 'SVM (RBF)', **svc_cm_metrics},
]

df_comparison_cm = pd.DataFrame(comparison_data)

print("\n\n--- Model Comparison: Raw Confusion Matrix Component Counts ---")
# Display the table, setting the Model column as the index
df_comparison_cm.set_index('Model', inplace=True)
print(df_comparison_cm)
```

```


--- Model Comparison: Raw Confusion Matrix Component Counts ---
                        TN (True Negative)  FP (False Positive)  \
Model
Neural Network (Dense)                  88                   12
Logistic Regression                     82                   18
SVM (RBF)                               83                   17

                        FN (False Negative)  TP (True Positive)
```

```
Model
Neural Network (Dense)                  26              28
Logistic Regression                     26              28
SVM (RBF)                               21              33
```

The Deep Neural Network (DNN) performed well in overall accuracy (0.7532) and achieved the highest Precision (0.7000). However, the DNN struggled significantly with Recall (0.5185), which resulted in 26 False Negatives (missed diagnoses). This suggests that while the DNN was careful about predicting the positive class (high precision), it was too conservative, missing nearly half of the actual diabetic patients. Because diagnostic safety requires minimizing False Negatives, Recall and F1 Score are the most important metrics. The Support Vector Machine (RBF) model proved to be the most reliable classifier, achieving the highest F1 Score (0.6346) and the highest Recall (0.6111), with the lowest count of False Negatives (21). The Support Vector Machine gave the most clinically reliable performance by reducing the number of missed diabetic cases. These results show that on this relatively small dataset, a well-tuned classical model like SVM can still outperform a standard deep learning architecture.