Machine Learning Project

Models used in the project:

- K-Nearest Neighbors (KNN)
- Linear Regression
- Support Vector Machine (SVM)
- Neural Network
- Logistic Regression

Applying Machine learning models on such a huge dataset makes it fair to compare the most used ones in terms of accuracy, precision, recall, F1-score, and confusion matrix.

By:

Ahmed Kahla
s-ahmed.kahla@zewailcity.edu.eg

Dataset:
https://www.kaggle.com/datasets/gaurav2022/mobile-health/data

First of all, is reading the data and specifying a suitable subset of it due to its huge records, also specifying the features and the label for model training and testing purposes. I divided the dataset into X_train and y_train and their test data according to this line of code :

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

KNN model :

Knn model achieved great results in terms of evaluation metrics and also it achieved a very short time to finish training (31.6 seconds).
Note: Knn model without cross-validation process takes only 7 seconds.
The following results show the power of knn model on the dataset provided:

```
Cross-validation Scores: [0.99916718 0.99925972 0.99917745 0.99935225 0.99922886]
Mean CV Accuracy: 0.9992370918333583
Test Set Accuracy: 0.999317293851532
Precision: 0.9987017668045453
Recall: 0.9986210985329299
F1-score: 0.9986611681346632
```

And the following is a screenshot of the knn model code:

KNN

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

knn = KNeighborsClassifier(n_neighbors=5)
cv_scores = cross_val_score(knn, X_train, y_train, cv=5)
print("Cross-validation Scores:", cv_scores)
print("Mean CV Accuracy:", cv_scores.mean())
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred,average = 'macro')
f1 = f1_score(y_test, y_pred,average = 'macro')

print("Test Set Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

```
Cross-validation Scores: [0.99916718 0.99925972 0.99917745 0.99935225 0.99922886]
Mean CV Accuracy: 0.9992370918333583
Test Set Accuracy: 0.999317293851532
Precision: 0.9987017668045453
Recall: 0.9986210985329299
F1-score: 0.9986611681346632
```

Neural Network:

Neural network model has also achieved good results in terms of accuracy and recall but not too good in F1-score and precision as it achieved only 78% precision and 84% F1-score. In terms of time, the model finished after 10 epochs in 7.15 minutes as shown in the screenshot below.
The following is the results of the NN model:

```
Precision: 0.7861964883364931
Recall: 0.9224419304387858
F1-score: 0.8463493781005482
Accuracy: 0.9065844129138392
```

The screenshot below shows the code used to provide the NN model

```
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
model = Sequential([
    Dense(256, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(13, activation='softmax')
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=32)
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
precision = precision_score(y_test, y_pred_classes, average='macro')
recall = recall_score(y_test, y_pred_classes, average='macro')
f1 = f1_score(y_test, y_pred_classes, average='macro')
accuracy = accuracy_score(y_test, y_pred_classes)

print('Precision:', precision)
print('Recall:', recall)
print('F1-score:', f1)
print('Accuracy:', accuracy)
conf_matrix = confusion_matrix(y_test, y_pred_classes)
print('Confusion Matrix:')
print(conf_matrix)
```

✓ 7m 15.0s

Linear Regression:

Linear regression is marked by its simplicity of coding and its fast time of running

On this dataset, the model ran only in 0.7 seconds and got a 10.19 mean square error as shown below

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
model = LinearRegression()
model.fit(X_train, y_train)
predictions = model.predict(X_test)
mse = mean_squared_error(y_test, predictions)
print("Mean Squared Error:", mse)
```

✓ 0.7s

```
Mean Squared Error: 10.197917791195172
```

Logistic regression:

It noticed that logistic regression is not suitable for this dataset as it achieved bad results in such evaluation metrics as F1-score, precision, and recall. But in accuracy, it wasn't so bad as it achieved 72% accuracy. Several factors have affected the time of running as cross-validation to choose the best hyperparameters as it ran in 28.31 minutes. The following shows the results of the logistic regression model :

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
Best Hyperparameters: {'C': 0.001, 'penalty': 'l2', 'solver': 'saga'}
Accuracy: 0.7304133251079581
Precision: 0.22822943136404342
Recall: 0.1856961180456437
F1-score: 0.1836104396991377
```

The following is a screenshot of the code used

```python
from sklearn.model_selection import GridSearchCV
param_grid = {
    'penalty': ['l1', 'l2'],
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'solver': ['liblinear', 'saga']
}
model = LogisticRegression()
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='accuracy', verbose=1, n_jobs=-1)
grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

precision = precision_score(y_test, y_pred, average="macro")
recall = recall_score(y_test, y_pred,average="macro")
f1 = f1_score(y_test, y_pred,average="macro")

print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)

conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

✓ 28m 31.1s

SVM:

The SVM model was very simple in coding comparison to other models, the screenshot shows the code of the SVM

```python
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
svm_classifier = SVC(kernel='rbf')

svm_classifier.fit(X_train, y_train)

y_pred = svm_classifier.predict(X_test)

precision = precision_score(y_test, y_pred, average="macro")
recall = recall_score(y_test, y_pred,average="macro")
f1 = f1_score(y_test, y_pred,average="macro")
accuracy = accuracy_score(y_test, y_pred)

print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("Accuracy:", accuracy)

# cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
# cv_scores = cross_val_score(svm_classifier, X, y, cv=cv, scoring='accuracy')
# print("Cross-Validation Scores:", cv_scores)
# print("Mean CV Accuracy:", np.mean(cv_scores))
```

In terms of results and time : It still running