EC 9560 – DATA MINING

LAB 04

DARMILA.T

2020/E/027

SEMESTER 07

18th DECEMBER 2024

Data Splitting

```
[66]  1 X.shape
```

```
(2736, 17)
```

```
[69]  1 X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
```

Generated code may be subject to a license | Phat-pham99/Vietnamese_sign_language_recognition_using_MHI_and_CNN

```
1 print(f"X_train shape: {X_train.shape}")
2 print(f"X_val shape: {X_val.shape}")
3 print(f"y_train shape: {y_train.shape}")
4 print(f"y_val shape: {y_val.shape}")
```

```
X_train shape: (2188, 17)
X_val shape: (548, 17)
y_train shape: (2188,)
y_val shape: (548,)
```

Split the train data into train and validation sets, and test data for final evaluation of the model selection.

```
1 X.isnull().sum()
```

	0
Physical-Height	206
Basic_Demos-Age	0
PreInt_EduHx-computerinternet_hoursday	82
Physical-Weight	164
FGC-FGC_CU	817
BIA-BIA_BMI	923
Physical-BMI	209
SDS-SDS_Total_T	211
PAQ_A-Season	0
FGC-FGC_PU	827
BIA-BIA_Frame_num	923
Physical-Systolic_BP	258
FGC-FGC_TL	817
PAQ_C-Season	0
BIA-BIA_FFMI	923
FGC-FGC_SRR_Zone	857
FGC-FGC_SRL_Zone	859

There are still missing values in the feature selection of the training data, but we cannot remove them, as some features have a larger number of missing values. Therefore, I have decided to train the XGBoost model, which can handle missing data effectively.

XGBoost model

The XGBClassifier is used for multi-class classification. The objective='multi:softmax' parameter specifies that we're solving a multi-class classification problem, and num_class specifies the number of unique classes in the target variable y.

```python
model_xgboost = xgb.XGBClassifier(
    objective='multi:softmax',  # Multi-class classification
    num_class=len(y.unique()),  # Number of classes in your target variable
    eval_metric='mlogloss',  # Logarithmic loss for multi-class classification
    random_state=42
)

# Train the model
model_xgboost.fit(X_train, y_train)

# Make predictions
y_pred = model_xgboost.predict(X_val)
y_pred_train = model_xgboost.predict(X_train)

# Evaluate the model's performance on the validation set
accuracy_val = accuracy_score(y_val, y_pred)
accuracy_train = accuracy_score(y_train, y_pred_train)

# Classification report for precision, recall, and F1 score
clf_report = classification_report(y_val, y_pred)

# Confusion matrix
conf_matrix = confusion_matrix(y_val, y_pred)
```

```python
# Print results
print(f"Train Accuracy: {accuracy_train}")
print(f"Validation Accuracy: {accuracy_val}")
print("\nClassification Report:\n", clf_report)
print("\nConfusion Matrix:\n", conf_matrix)
```

```
Train Accuracy: 0.9917733089579525
Validation Accuracy: 0.6167883211678832

Classification Report:
              precision    recall  f1-score   support

         0.0       0.73      0.85      0.78       336
         1.0       0.36      0.37      0.36       131
         2.0       0.19      0.07      0.10        72
         3.0       1.00      0.11      0.20         9

    accuracy                           0.62       548
   macro avg       0.57      0.35      0.36       548
weighted avg       0.58      0.62      0.58       548


Confusion Matrix:
 [[284  41  11   0]
 [ 73  48  10   0]
 [ 29  38   5   0]
 [  2   6   0   1]]
```

Here, the validation and training accuracy deviate significantly, indicating overfitting. To address this issue, I have decided to adjust the parameters of the XGBoost model.

I define the hyperparameters for the XGBClassifier. These parameters are passed when creating the model.

- max_depth: Maximum depth of a tree, controlling overfitting.
- n_estimators: Number of boosting rounds (trees).
- learning_rate: Step size shrinkage to prevent overfitting.
- subsample: Fraction of samples used for fitting each tree.
- colsample_bytree: Fraction of features to use when building each tree.

```python
# Set model parameters for XGBoost
params = {
    'max_depth': 3,
    'n_estimators': 202,
    'learning_rate': 0.07956777025142073,
    'subsample': 0.8197358255094112,
    'colsample_bytree': 0.645036755035947
}
# Initialize the XGBoost classifier with the given parameters
clf = xgb.XGBClassifier(**params)
```

Prints the **quadratic weighted kappa** scores for each fold of cross-validation and their mean.

```python
# Define the custom quadratic weighted kappa scorer
def quadratic_kappa(y_true, y_pred):
    return cohen_kappa_score(y_true, y_pred, weights='quadratic')

# Create the kappa scorer
kappa_scorer = make_scorer(quadratic_kappa)

# Perform cross-validation and evaluate using the quadratic weighted kappa score
scores = cross_val_score(clf, X_train, y_train, cv=5, scoring=kappa_scorer)

# Print the results: QWK scores for each fold and the mean QWK score
print("QWK Scores:", scores)
print("Mean QWK Score:", np.mean(scores))
```

```
QWK Scores: [0.29320382 0.4135213  0.31642188 0.34673274 0.30450232]
Mean QWK Score: 0.3348764135666515
```

This function calculates the **quadratic weighted kappa** score, which is a metric used to evaluate the agreement between two raters (between the true and predicted values).

Ensures that each fold of the cross-validation maintains the same proportion of classes as the original dataset (for imbalanced class distributions).

```
[100]  1 clf.fit(X_train, y_train)
       2
       3 y_pred_train = clf.predict(X_train)
       4 y_pred_val = clf.predict(X_val)
       5
       6 accuracy_train = accuracy_score(y_train, y_pred_train)
       7 accuracy_val = accuracy_score(y_val, y_pred_val)
       8 clf_report_val = classification_report(y_val, y_pred_val)
       9 conf_matrix_val = confusion_matrix(y_val, y_pred_val)
```

```
1 # Print the results
2 print("\nTrain Accuracy:", accuracy_train)
3 print("Validation Accuracy:", accuracy_val)
4 print("\nValidation Classification Report:")
5 print(clf_report_val)
6 print("\nValidation Confusion Matrix:")
7 print(conf_matrix_val)
```

```
Train Accuracy: 0.779707495429616
Validation Accuracy: 0.6186131386861314

Validation Classification Report:
              precision    recall  f1-score   support

         0.0       0.72      0.87      0.79       336
         1.0       0.32      0.29      0.31       131
         2.0       0.32      0.11      0.16        72
         3.0       1.00      0.11      0.20         9

    accuracy                           0.62       548
   macro avg       0.59      0.35      0.36       548
weighted avg       0.58      0.62      0.58       548


Validation Confusion Matrix:
[[292  39   5   0]
 [ 82  38  11   0]
 [ 31  33   8   0]
 [  0   7   1   1]]
```

Here, we can observe that the training accuracy has been reduced, and the gap between the training and validation accuracies is smaller. Additionally, this method allows us to see the importance of each feature. This approach is particularly useful for evaluating multi-class classification models and understanding the relative importance of each feature in making predictions.

```python
# Get and sort feature importances
feature_imp = pd.Series(clf.feature_importances_, index=X.columns).sort_values(ascending=False)

# Display the feature importances
print("\nFeature Importances:")
print(feature_imp)
```

```
Feature Importances:
Basic_Demos-Age                          0.107312
PreInt_EduHx-computerinternet_hoursday   0.105591
SDS-SDS_Total_T                          0.074181
Physical-Height                          0.066824
Physical-Weight                          0.064558
FGC-FGC_CU                               0.062438
Physical-BMI                             0.051492
FGC-FGC_PU                               0.051205
FGC-FGC_SRR_Zone                         0.050598
PAQ_C-Season                             0.049476
BIA-BIA_BMI                              0.047777
PAQ_A-Season                             0.047720
BIA-BIA_FFMI                             0.046973
Physical-Systolic_BP                     0.045807
BIA-BIA_Frame_num                        0.044509
FGC-FGC_TL                               0.042335
FGC-FGC_SRL_Zone                         0.041204
dtype: float32
```

Here, age and computer and internet usage are the most influential features in classifying the level of problematic internet use among children and adolescents.

Handling the missing values

```python
from sklearn.impute import KNNImputer

# KNN imputation for numeric features
knn_imputer = KNNImputer(n_neighbors=5)
X[['Physical-Height', 'Physical-Weight',
    'FGC-FGC_CU', 'BIA-BIA_BMI', 'Physical-BMI', 'SDS-SDS_Total_T',
    'FGC-FGC_PU', 'Physical-Systolic_BP', 'FGC-FGC_TL',
    'BIA-BIA_FFMI']] = knn_imputer.fit_transform(X[['Physical-Height', 'Physical-Weight', 'FGC-FGC_CU', 'BIA-BIA_BMI', 'Physical-BMI', 'SDS-SDS_Total_T',
                                                    'FGC-FGC_PU', 'Physical-Systolic_BP',
                                                    'FGC-FGC_TL', 'BIA-BIA_FFMI']])

# Impute missing values in categorical features with the mode (most frequent value)
imputer = SimpleImputer(strategy='most_frequent')
X[['PAQ_A-Season', 'PAQ_C-Season', 'PreInt_EduHx-computerinternet_hoursday',
   'FGC-FGC_SRR_Zone', 'FGC-FGC_SRL_Zone', 'BIA-BIA_Frame_num']] = imputer.fit_transform(X[['PAQ_A-Season', 'PAQ_C-Season' ,'PreInt_EduHx-computerinternet_hoursday',
                                                                                           'FGC-FGC_SRR_Zone', 'FGC-FGC_SRL_Zone', 'BIA-BIA_Frame_num']])
```

Here, I use KNN imputation for numerical features and mode imputation for categorical features.

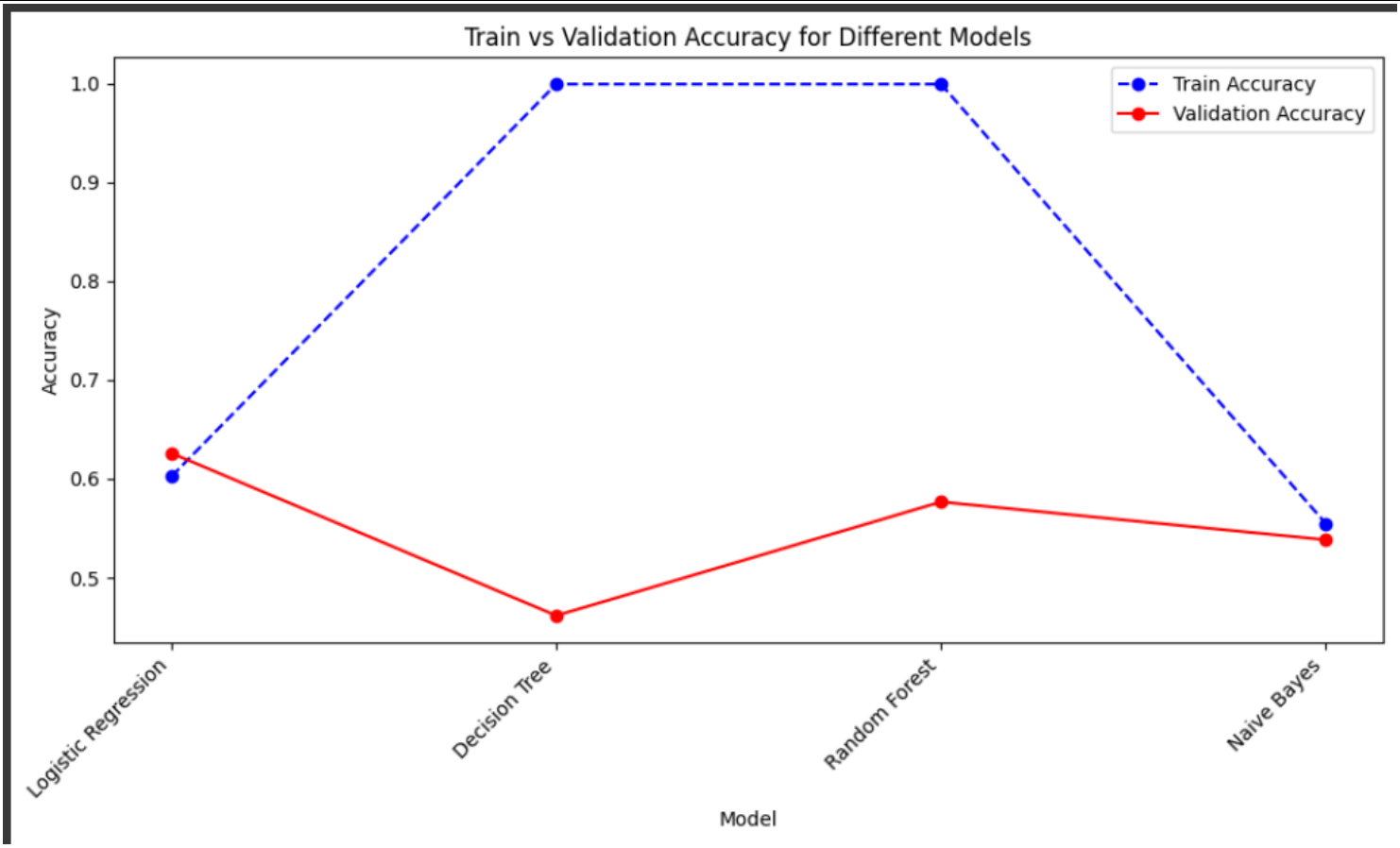
Train the model for different algorithms

Here, I used the following algorithms to build the model:

- Logistic Regression with a maximum of 1000 iterations to ensure convergence.
- Decision Tree Classifier.
- Random Forest Classifier.
- Naive Bayes Classifier (GaussianNB).

```python
# Initialize the models
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'Naive Bayes': GaussianNB(),
}

results = {
    'Model': [],
    'Train Accuracy': [],
    'Validation Accuracy': [],
    'Classification Report': [],
    'Confusion Matrix': []
}

# Loop through each model, fit it, and evaluate
for name, model in models.items():
    model.fit(X_train, y_train)

    y_pred = model.predict(X_val)
    y_pred_train = model.predict(X_train)

    accuracy_train = accuracy_score(y_train, y_pred_train)
    accuracy_val = accuracy_score(y_val, y_pred)

    clf_report = classification_report(y_val, y_pred, output_dict=True)

    # Store metrics in the results dictionary
    results['Model'].append(name)
    results['Train Accuracy'].append(accuracy_train)
    results['Validation Accuracy'].append(accuracy_val)
    results['Confusion Matrix'].append(confusion_matrix(y_val, y_pred))
    results['Classification Report'].append(clf_report)
```

```python
results_df = pd.DataFrame(results)

# Plotting the results for training and validation accuracy
plt.figure(figsize=(10,6))
plt.plot(results_df['Model'], results_df['Train Accuracy'], label='Train Accuracy', marker='o', linestyle='--', color='blue')
plt.plot(results_df['Model'], results_df['Validation Accuracy'], label='Validation Accuracy', marker='o', linestyle='-', color='red')
plt.xticks(rotation=45, ha='right')
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.title('Train vs Validation Accuracy for Different Models')
plt.legend()
plt.tight_layout()
plt.show()

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



Here, we can see the difference between each model. Logistic Regression and Naive Bayes provide the most balanced models, while Decision Tree and Random Forest models exhibit overfitting.

The evaluation matrix of models:

```python
# Loop over the 'results' dictionary keys and print the results in a readable format
for i in range(len(results['Model'])):
    print(f"\n{'-'*50}")
    print(f"Model: {results['Model'][i]}")
    print(f"{'-'*50}")

    print(f"Accuracy on Training Data: {results['Train Accuracy'][i]:.4f}")
    print(f"Accuracy on Validation Data: {results['Validation Accuracy'][i]:.4f}")

    print("\nClassification Report:\n", results['Classification Report'][i])

    print("\nConfusion Matrix:")
    print(results['Confusion Matrix'][i])

    print(f"{'-'*50}\n")

```

```
--------------------------------------------------
Model: Logistic Regression
--------------------------------------------------
Accuracy on Training Data: 0.6024
Accuracy on Validation Data: 0.6259

Classification Report:
 {'0.0': {'precision': 0.6951501154734411, 'recall': 0.8958333333333334, 'f1-score': 0.7828348504551366, 'support': 336.0}, '1.0': {'precision': 0.34177215189873417, 'recall': 0.20610687022900764, 'f1-score': 0.2571428571428571, 'suppor

Confusion Matrix:
[[301  28   7   0]
 [ 93  27  11   0]
 [ 38  19  14   1]
 [  1   5   2   1]]
--------------------------------------------------


--------------------------------------------------
Model: Decision Tree
--------------------------------------------------
Accuracy on Training Data: 0.9991
Accuracy on Validation Data: 0.4617

Classification Report:
 {'0.0': {'precision': 0.6666666666666666, 'recall': 0.5952380952380952, 'f1-score': 0.6289308176100629, 'support': 336.0}, '1.0': {'precision': 0.23178807947019867, 'recall': 0.26717557251908397, 'f1-score': 0.24822695035460993, 'suppo

Confusion Matrix:
[[200  88  47   1]
 [ 70  35  25   1]
 [ 28  26  14   4]
 [  2   2   1   4]]
--------------------------------------------------
```

```
--------------------------------------------------
Model: Random Forest
--------------------------------------------------
Accuracy on Training Data: 0.9991
Accuracy on Validation Data: 0.5766

Classification Report:
 {'0.0': {'precision': 0.6907730673316709, 'recall': 0.8244047619047619, 'f1-score': 0.7516960651289009, 'support': 336.0}, '1.0': {'precision': 0.2689075630252101, 'recall': 0.24427480916030533, 'f1-score': 0.256, 'support': 131.0}, '2.0

Confusion Matrix:
[[277  53   6   0]
 [ 85  32  14   0]
 [ 38  27   7   0]
 [  1   7   1   0]]
--------------------------------------------------


--------------------------------------------------
Model: Naive Bayes
--------------------------------------------------
Accuracy on Training Data: 0.5548
Accuracy on Validation Data: 0.5383

Classification Report:
 {'0.0': {'precision': 0.7070422535211267, 'recall': 0.7470238095238095, 'f1-score': 0.7264833574529667, 'support': 336.0}, '1.0': {'precision': 0.2689075630252101, 'recall': 0.24427480916030533, 'f1-score': 0.256, 'support': 131.0}, '2

Confusion Matrix:
[[251  61  15   9]
 [ 75  32  10  14]
 [ 29  23   6  14]
 [  0   3   0   6]]
--------------------------------------------------
```

Model	Train Accuracy	Validation Accuracy
XGBoost	0.779	0.618
Logistic Regression	0.6024	0.6259
Decision Tree	0.9991	0.4617
Random Forest	0.9991	0.5766
Naïve Bayes	0.5548	0.5383

Analysis:

- **XGBoost** shows a good balance between training accuracy (0.779) and validation accuracy (0.618). It indicates that the model has learned well from the training data and generalizes moderately to unseen data.
- **Logistic Regression** has lower training accuracy (0.6024) but performs better on the validation set (0.6259) compared to other models. This suggests that logistic regression is less prone to overfitting, and it generalizes better on unseen data.
- ***Decision Tree*** *and* ***Random Forest*** *both show extremely high training accuracy (0.9991), which suggests they are overfitting the training data. Their validation accuracies (0.4617 and 0.5766, respectively) are much lower, indicating poor generalization.*
- ***Naïve Bayes*** *has low training and validation accuracy (0.5548 and 0.5383, respectively), indicating that this model is not performing well for the given problem.*

In this case, **Logistic Regression** had missing values imputed, while **XGBoost** was trained without imputing missing values.

Best Model: XGBoost

- **Why XGBoost?**
    - **Robustness to Missing Data**: XGBoost's ability to handle missing values natively and still achieve relatively high performance (especially the validation accuracy of 0.618) makes it an excellent model in this case, especially given that the missing values were not imputed before training.
    - **High Validation Accuracy**: Although Logistic Regression showed a slightly higher validation accuracy (0.6259), the fact that XGBoost performed well without imputation is a strong point in its favor, as it demonstrates its robustness and flexibility in real-world scenarios where missing data might be common.
    - **Higher Training Accuracy**: XGBoost also has higher training accuracy compared to Logistic Regression, indicating that it has learned better patterns from the data.

Conclusion

While Logistic Regression performs well after imputing missing values, XGBoost stands out as the better choice due to its robustness in missing data, better handling of the feature interactions, and higher training accuracy. Given that XGBoost was able to achieve high validation accuracy without any imputation, it is likely to be more effective in scenarios with missing data, making it the best model overall in this case.