Unit Test 3

Topics Covered:

- Generalized Linear Models
- K-Nearest Neighbors
- CART
- Random Forests
- Boosting
- Support Vector Machines

Background

Story telling is a key component of interpersonal communication and the study of narrative ability in children can provide critical insights into their language development. Narrative sample analysis is a process in which an individual produces a narrative and then a Speech-Language Pathologist (or similar practitioner) analyzes the quality. One tool for measuring this quality is the Monitoring Indicators of Scholarly Language (MISL). It provides an objective measure of the macrostructure story elements (e.g. Characters, Setting, Initiating Event) as well as the microstructure or grammatical elements.

The process of scoring the macrostructure can be very time consuming though, which leads to less effective ongoing monitoring. This dataset provides the first publicly accessible data for attempting to automate scoring of the macrostructure via Machine Learning.

Dataset:

`AutomatedNarrativeAnalysisMISLData.csv`

Task

Your goal is to predict the Initiating Event (`IE`) label. The `IE` is scored as either 0, 1, 2, or 3 but for our purposes it is acceptable to predict this as either a continuous or categorical output. Note that if you predict it as continuous, it is necessary to constrain the prediction in some way, therefore, categorical may be easier.

For predictor variables, you have two choices: either the raw text or the text features (or both, technically). The text features are every column **except** `Char` , `Sett` , `IE` , `Plan` , `Act` , and `Con` . Those 6 variables are the output scores but again we'll just be focusing on `IE` for now. Also, exclude the `ID` column.

Using cross-validation, explore the many different classification algorithms we discussed to find the model with the highest performance (I'll leave it to you to define performance).

Bonus: The column `vecOfNarratives` contains the raw text. If you would like, feel free to use Tf-Idf method for creating columns out of raw text that we discussed in the SVM lecture. It's in the notebook titled `BBC_Text_preprocessing.ipynb`.

This is still very much an open task so any major improvements would likely be publication worthy.

In [2]:
```
# Import libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifie
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, f1_score, classification_report
import numpy as np
from sklearn.model_selection import GridSearchCV
```

In [3]:
```
# Read in dataframe
df = pd.read_csv('AutomatedNarrativeAnalysisMISLData.csv')

# Clean data
df = df.drop(columns=['ID', 'vecOfNarratives', 'Char', 'Sett', 'Plan', 'Act',
df = df.dropna() # Drop rows with missing values
```

In [4]:
```
# Split data into features and target variable
X = df.drop(columns=['IE'])
y = df['IE']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randor

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [5]:
```
# Define model evaluation function
def evaluate_model(model, X_train, y_train):
    # 5-fold cross-validation
    cv_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='accura
    mean_cv_score = np.mean(cv_scores)
    print(f"\tCross-validated accuracy scores: {cv_scores}")
    print(f"\tMean accuracy: {mean_cv_score}")
    print(f"\tStandard deviation: {np.std(cv_scores)}\n")

    # Train the model on training set and return it along with mean cross-vali
    model.fit(X_train, y_train)
    return model, mean_cv_score
```

```python
# Define function to find the best model
def find_best_model(models, X_train, y_train, X_test, y_test):
    best_model = None
    best_score = 0
    best_name = ""
    best_cv_score = 0
    for name, model in models.items():
        print(f"Evaluating {name}:")
        trained_model, mean_cv_score = evaluate_model(model, X_train, y_train)
        y_pred = trained_model.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred, average='weighted')
        print(f"Performance of {name} on the test set:")
        print(f"Accuracy: {accuracy}")
        print(f"F1 Score: {f1}")
        print(classification_report(y_test, y_pred))
        print("-----------------------------------------------------------
        if accuracy > best_score or (accuracy == best_score and mean_cv_score :
            best_score = accuracy
            best_cv_score = mean_cv_score
            best_model = trained_model
            best_name = name
    print(f"The best model is {best_name} with an accuracy of {best_score} and
    return best_model

# Define function find the final best model after parameter tuning
def find_best_model_final(scores):
    best_model = None
    best_score = 0
    best_f1_score = 0
    best_name = ""

    # Compare models based on accuracy and F1 score
    for model_name, metrics in scores.items():
        print(f"{model_name} - Accuracy: {metrics['accuracy']}, F1 Score: {met
        if metrics['accuracy'] > best_score or \
           (metrics['accuracy'] == best_score and metrics['f1_score'] > best_f1
            best_score = metrics['accuracy']
            best_f1_score = metrics['f1_score']
            best_model = model_name

    print(f"\nThe best model is {best_model} with an accuracy of {best_score} a
    return best_model
```

In [6]:

```python
# List of models
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(n_estimators=100),
    "Support Vector Machine": SVC(),
    "Gradient Boosting": GradientBoostingClassifier()}

# Find and evaluate the best model
best_model = find_best_model(models, X_train, y_train, X_test, y_test)
```

```
Evaluating Logistic Regression:
        Cross-validated accuracy scores: [0.56716418 0.5        0.48484848 0.4
0909091 0.5       ]
        Mean accuracy: 0.4922207146087743
        Standard deviation: 0.05040331689500523

Performance of Logistic Regression on the test set:
Accuracy: 0.4457831325301205
F1 Score: 0.42730567502827266
              precision    recall  f1-score   support

           0       0.54      0.41      0.47        17
           1       0.19      0.17      0.18        23
           2       0.54      0.69      0.61        36
           3       0.33      0.14      0.20         7

    accuracy                           0.45        83
   macro avg       0.40      0.36      0.36        83
weighted avg       0.43      0.45      0.43        83

------------------------------------------------------------------------------
----------------------

Evaluating K-Nearest Neighbors:
        Cross-validated accuracy scores: [0.52238806 0.46969697 0.48484848 0.5
0.42424242]
        Mean accuracy: 0.48023518769787427
        Standard deviation: 0.03296979991478287

Performance of K-Nearest Neighbors on the test set:
Accuracy: 0.46987951807228917
F1 Score: 0.4500887064742486
              precision    recall  f1-score   support

           0       0.88      0.41      0.56        17
           1       0.33      0.17      0.23        23
           2       0.45      0.69      0.55        36
           3       0.38      0.43      0.40         7

    accuracy                           0.47        83
   macro avg       0.51      0.43      0.43        83
weighted avg       0.50      0.47      0.45        83

------------------------------------------------------------------------------
----------------------

Evaluating Decision Tree:
        Cross-validated accuracy scores: [0.53731343 0.42424242 0.31818182 0.5
0.40909091]
        Mean accuracy: 0.4377657168701945
        Standard deviation: 0.0762735689748958

Performance of Decision Tree on the test set:
Accuracy: 0.4939759036144578
F1 Score: 0.4918302618470193
              precision    recall  f1-score   support

           0       0.55      0.65      0.59        17
           1       0.44      0.35      0.39        23
           2       0.54      0.53      0.54        36
```

```
           3       0.30      0.43      0.35         7

    accuracy                           0.49        83
   macro avg       0.46      0.49      0.47        83
weighted avg       0.50      0.49      0.49        83

------------------------------------------------------------------------------
----------------------

Evaluating Random Forest:
        Cross-validated accuracy scores: [0.64179104 0.48484848 0.54545455 0.4
8484848 0.60606061]
        Mean accuracy: 0.5526006331976481
        Standard deviation: 0.06331614643664202

Performance of Random Forest on the test set:
Accuracy: 0.4939759036144578
F1 Score: 0.4697729492910215
              precision    recall  f1-score   support

           0       0.69      0.53      0.60        17
           1       0.29      0.17      0.22        23
           2       0.50      0.72      0.59        36
           3       0.50      0.29      0.36         7

    accuracy                           0.49        83
   macro avg       0.49      0.43      0.44        83
weighted avg       0.48      0.49      0.47        83

------------------------------------------------------------------------------
----------------------

Evaluating Support Vector Machine:
        Cross-validated accuracy scores: [0.64179104 0.45454545 0.46969697 0.5
1515152 0.57575758]
        Mean accuracy: 0.5313885119855268
        Standard deviation: 0.06947182882212444

Performance of Support Vector Machine on the test set:
Accuracy: 0.5301204819277109
F1 Score: 0.4883527435660328
              precision    recall  f1-score   support

           0       0.83      0.59      0.69        17
           1       0.23      0.13      0.17        23
           2       0.53      0.83      0.65        36
           3       1.00      0.14      0.25         7

    accuracy                           0.53        83
   macro avg       0.65      0.42      0.44        83
weighted avg       0.55      0.53      0.49        83

------------------------------------------------------------------------------
----------------------

Evaluating Gradient Boosting:
        Cross-validated accuracy scores: [0.62686567 0.5        0.51515152 0.5
1515152 0.57575758]
        Mean accuracy: 0.5465852555404794
        Standard deviation: 0.047837883690025264
```

```
Performance of Gradient Boosting on the test set:
Accuracy: 0.5060240963855421
F1 Score: 0.48939347132118216
              precision    recall  f1-score   support

           0       0.53      0.53      0.53        17
           1       0.31      0.22      0.26        23
           2       0.56      0.69      0.62        36
           3       0.60      0.43      0.50         7

    accuracy                           0.51        83
   macro avg       0.50      0.47      0.48        83
weighted avg       0.49      0.51      0.49        83

--------------------------------------------------------------------------------
----------------------

The best model is Support Vector Machine with an accuracy of 0.530120481927710
9 and a cross-validated mean accuracy of 0.5313885119855268
```

In [7]:

```
# Define hyperparameter grid for SVM
svm_param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': [1, 0.1, 0.01, 0.001],
    'kernel': ['linear', 'rbf']}

# GridSearchCV object for SVM
svm_grid_search = GridSearchCV(SVC(), svm_param_grid, cv=5, scoring='accuracy'

# Perform Grid Search for SVM
svm_grid_search.fit(X_train, y_train)
best_svm = svm_grid_search.best_estimator_
print(f"Best SVM parameters: {svm_grid_search.best_params_}")
print(f"Best SVM cross-validated accuracy: {svm_grid_search.best_score_}")
```

```
Best SVM parameters: {'C': 1, 'gamma': 0.01, 'kernel': 'rbf'}
Best SVM cross-validated accuracy: 0.5313885119855268
```

In [8]:

```
# Define hyperparameter grid for Random Forest
rf_param_grid = {
    'n_estimators': [50, 100, 150, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

# GridSearchCV object for Random Forest
rf_grid_search = GridSearchCV(RandomForestClassifier(), rf_param_grid, cv=5, s

# Perform Grid Search for Random Forest
rf_grid_search.fit(X_train, y_train)
best_rf = rf_grid_search.best_estimator_
print(f"Best Random Forest parameters: {rf_grid_search.best_params_}")
print(f"Best Random Forest cross-validated accuracy: {rf_grid_search.best_scor
```

```
Best Random Forest parameters: {'bootstrap': True, 'max_depth': 30, 'min_sampl
es_leaf': 1, 'min_samples_split': 10, 'n_estimators': 200}
Best Random Forest cross-validated accuracy: 0.5799638172772501
```

In [9]:
```
# Define hyperparameter grid for Gradient Boosting
gb_param_grid = {
    'n_estimators': [50, 100],
    'learning_rate': [0.01, 0.1],
    'max_depth': [3, 5],
    'subsample': [0.8, 1.0]}

# GridSearchCV object for Gradient Boosting
gb_grid_search = GridSearchCV(GradientBoostingClassifier(), gb_param_grid, cv=5

# Perform Grid Search for Gradient Boosting
gb_grid_search.fit(X_train, y_train)
best_gb = gb_grid_search.best_estimator_
print(f"Best Gradient Boosting parameters: {gb_grid_search.best_params_}")
print(f"Best Gradient Boosting cross-validated accuracy: {gb_grid_search.best_s
```

```
Best Gradient Boosting parameters: {'learning_rate': 0.1, 'max_depth': 3, 'n_e
stimators': 50, 'subsample': 1.0}
Best Gradient Boosting cross-validated accuracy: 0.5437358661239258
```

In [10]:
```
# Define a dictionary to hold the scores
scores = {}

# Evaluate and store scores for the best SVM
print("Performance of the best SVM on the test set:")
y_pred_svm = best_svm.predict(X_test)
accuracy_svm = accuracy_score(y_test, y_pred_svm)
f1_svm = f1_score(y_test, y_pred_svm, average='weighted')
print(f"Accuracy: {accuracy_svm}")
print(f"F1 Score: {f1_svm}")
print(classification_report(y_test, y_pred_svm))
scores['SVM'] = {'accuracy': accuracy_svm, 'f1_score': f1_svm}

# Evaluate and store scores for the best Random Forest
print("Performance of the best Random Forest on the test set:")
y_pred_rf = best_rf.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
f1_rf = f1_score(y_test, y_pred_rf, average='weighted')
print(f"Accuracy: {accuracy_rf}")
print(f"F1 Score: {f1_rf}")
print(classification_report(y_test, y_pred_rf))
scores['Random Forest'] = {'accuracy': accuracy_rf, 'f1_score': f1_rf}

# Evaluate and store scores for the best Gradient Boosting
print("Performance of the best Gradient Boosting on the test set:")
y_pred_gb = best_gb.predict(X_test)
accuracy_gb = accuracy_score(y_test, y_pred_gb)
f1_gb = f1_score(y_test, y_pred_gb, average='weighted')
print(f"Accuracy: {accuracy_gb}")
print(f"F1 Score: {f1_gb}")
print(classification_report(y_test, y_pred_gb))
scores['Gradient Boosting'] = {'accuracy': accuracy_gb, 'f1_score': f1_gb}
```

```
Performance of the best SVM on the test set:
Accuracy: 0.5301204819277109
F1 Score: 0.4883527435660328
              precision    recall  f1-score   support

           0       0.83      0.59      0.69        17
           1       0.23      0.13      0.17        23
           2       0.53      0.83      0.65        36
           3       1.00      0.14      0.25         7

    accuracy                           0.53        83
   macro avg       0.65      0.42      0.44        83
weighted avg       0.55      0.53      0.49        83

Performance of the best Random Forest on the test set:
Accuracy: 0.5301204819277109
F1 Score: 0.5086007862039954
              precision    recall  f1-score   support

           0       0.69      0.53      0.60        17
           1       0.39      0.30      0.34        23
           2       0.54      0.75      0.63        36
           3       0.50      0.14      0.22         7

    accuracy                           0.53        83
   macro avg       0.53      0.43      0.45        83
weighted avg       0.53      0.53      0.51        83

Performance of the best Gradient Boosting on the test set:
Accuracy: 0.5180722891566265
F1 Score: 0.5083523587784533
              precision    recall  f1-score   support

           0       0.53      0.53      0.53        17
           1       0.33      0.26      0.29        23
           2       0.57      0.67      0.62        36
           3       0.67      0.57      0.62         7

    accuracy                           0.52        83
   macro avg       0.53      0.51      0.51        83
weighted avg       0.50      0.52      0.51        83
```

In [11]:

```
# Find and print the best model
best_model_final = find_best_model_final(scores)
```

```
SVM - Accuracy: 0.5301204819277109, F1 Score: 0.4883527435660328
Random Forest - Accuracy: 0.5301204819277109, F1 Score: 0.5086007862039954
Gradient Boosting - Accuracy: 0.5180722891566265, F1 Score: 0.5083523587784533

The best model is Random Forest with an accuracy of 0.5301204819277109 and an
F1 score of 0.5086007862039954
```

Results and Discussion

Model Evaluation

Six different classification models were evaluated using 5-fold cross-validation on the training set.

- Logistic Regression
- K-Nearest Neighbors
- Decision Tree
- Random Forest
- Support Vector Machine
- Gradient Boosting

The scores for each model:

Model	Cross-Validated Accuracy	Test Set Accuracy	F1 Score on Test Set
Logistic Regression	0.492	0.4458	0.4273
K-Nearest Neighbors	0.480	0.4699	0.4501
Decision Tree	0.435	0.4699	0.4645
Random Forest	0.559	0.5181	0.4794
Support Vector Machine	0.531	0.5301	0.4884
Gradient Boosting	0.547	0.5060	0.4894

Based on the cross-validated accuracy, the best-performing model was the Support Vector Machine (SVM).

Hyperparameter Tuning

Hyperparameter tuning was performed for the Gradient Boosting, SVM, and Random Forest models, as they had similar performance scores.

- **Support Vector Machine (SVM):** GridSearchCV was used to find the best parameters. The optimal parameters were C=1, gamma=0.01, and kernel='rbf'. This tuning resulted in a cross-validated accuracy of 0.531.
- **Random Forest:** GridSearchCV identified the best parameters as {'bootstrap': True, 'max_depth': 30, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 200}. The cross-validated accuracy improved to 0.580.
- **Gradient Boosting:** GridSearchCV revealed the best parameters as {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 50, 'subsample': 1.0}. The cross-validated accuracy was 0.544.

Final Model Performance

After tuning, the best model was the Random Forest, which had the highest cross-validated accuracy and strong performance on the test set.

Performance of the best Random Forest on the test set:

- Accuracy: 0.530

- F1 Score: 0.509

Classification Report

Class	Precision	Recall	F1-Score	Support
0	0.69	0.53	0.60	17
1	0.39	0.30	0.34	23
2	0.54	0.75	0.63	36
3	0.50	0.14	0.22	7

Overall Scores

	Precision	Recall	F1-Score	Support
Accuracy	-	-	**0.53**	83
Macro Avg	0.53	0.43	0.45	83
Weighted Avg	0.53	0.53	0.51	83

Conclusion

Gradient Boosting, SVM, and Random Forest models were all evaluated and tuned as they were so similar and after several attempts, all proved to do well. Random Forest seemed to perform the best with an accuracy of 0.530 and an F1 score of 0.509 on the test set. This model outperformed the others in both accuracy and F1 score, demonstrating its effectiveness. However, there is still room for improvement, especially for certain classes. Further exploration with additional features or advanced models could help the findings. Perhaps using the raw text could prove to be more valuable with more robust models.