Model Development Phase Template

Date	21 June 2024
Team ID	739705
Project Title	Eudaimonia Engine: Machine Learning Delving into Happiness Classification
Maximum Marks	4 Marks

Initial Model Training Code, Model Validation and Evaluation Report

The initial model training code will be showcased in the future through a screenshot. The model validation and evaluation report will include classification reports, accuracy, and confusion matrices for multiple models, presented through respective screenshots.

Initial Model Training Code:

```
#MODEL BUILDING
#TRAINING THE MODEL

from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()

from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()

from sklearn.neighbors import KNeighborsClassifier
log=KNeighborsClassifier()

from sklearn.svm import SVC
svc=SVC()

from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()

from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score

from sklearn.preprocessing import StandardScaler
```

```
[ ] # Separate the independent variables
    x = df.drop(columns='happy',axis=1)

    # Separate the target variable
    y = df['happy']

    from sklearn.model_selection import train_test_split
    x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2, random_state=0)
```

```
[ ] from sklearn.preprocessing import StandardScaler

    # Initialize and fit the scaler
    sc = StandardScaler()
    x_train_scaled = sc.fit_transform(x_train)

    # X_trian = scaler.fit_transform(X_train)
    # X_test = scaler.transform(X_test)
```

```
[ ] #DECISION TREE MODEL

    dt= DecisionTreeClassifier()
    dt.fit(x_train, y_train)

    #Obtain predictions for train and test sets
    y_train_pred = dt.predict(x_train)
    y_test_pred = dt.predict(x_test)

    #Calculate metrics for train and test sets
    train_accuracy = accuracy_score(y_train, y_train_pred)
    train_precision = precision_score(y_train, y_train_pred, average='weighted')
    train_recall = recall_score(y_train, y_train_pred, average='weighted')
    train_f1score = f1_score(y_train, y_train_pred, average='weighted')

    test_accuracy = accuracy_score(y_test, y_test_pred)
    test_precision = precision_score(y_test, y_test_pred, average='weighted')
    test_recall = recall_score(y_test, y_test_pred, average='weighted')
    test_f1score = f1_score(y_test, y_test_pred, average='weighted')

    #Print the metrics
    print("dt-Train Accuracy:", train_accuracy)
    print("dt-Test Accuracy:", test_accuracy)

    print("dt-Train Precision:", train_precision)
    print("dt-Test Precision:", test_precision)

    print("dt-Train Recall:", train_recall)
    print("dt-Test Recall:", test_recall)

    print("dt-Train F1-score:", train_f1score)
    print("dt-Test F1-score:", test_f1score)
```

```
[ ] #KNN MODEL

    log=KNeighborsClassifier()
    log.fit(x_train, y_train)

    #Obtain predictions for train and test sets
    y_train_pred = log.predict(x_train)
    y_test_pred = log.predict(x_test)

    #Calculate metrics for train and test sets
    train_accuracy = accuracy_score(y_train, y_train_pred)
    train_precision = precision_score(y_train, y_train_pred, average='weighted')
    train_recall = recall_score(y_train, y_train_pred, average='weighted')
    train_f1score = f1_score(y_train, y_train_pred, average='weighted')

    test_accuracy = accuracy_score(y_test, y_test_pred)
    test_precision = precision_score(y_test, y_test_pred, average='weighted')
    test_recall = recall_score(y_test, y_test_pred, average='weighted')
    test_f1score = f1_score(y_test, y_test_pred, average='weighted')

    #Print the metrics
    print("log-Train Accuracy:", train_accuracy)
    print("log-Test Accuracy:", test_accuracy)

    print("log-Train Precision:", train_precision)
    print("log-Test Precision:", test_precision)

    print("log-Train Recall:", train_recall)
    print("log-Test Recall:", test_recall)

    print("log-Train F1-score:", train_f1score)
    print("log-Test F1-score:", test_f1score)
```

```
[ ] #RANDOM FOREST MODEL

    # Initialize and fit the scaler
    sc = StandardScaler()
    x_train_scaled = sc.fit_transform(x_train)

    rf = RandomForestClassifier()
    rf.fit(x_train, y_train)

    #Obtain predictions for train and test sets
    y_train_pred = rf.predict(x_train)
    y_test_pred = rf.predict(x_test)

    #Calculate metrics for train and test sets
    train_accuracy = accuracy_score(y_train, y_train_pred)
    train_precision = precision_score(y_train, y_train_pred, average='weighted')
    train_recall = recall_score(y_train, y_train_pred, average='weighted')
    train_f1score = f1_score(y_train, y_train_pred, average='weighted')

    test_accuracy = accuracy_score(y_test, y_test_pred)
    test_precision = precision_score(y_test, y_test_pred, average='weighted')
    test_recall = recall_score(y_test, y_test_pred, average='weighted')
    test_f1score = f1_score(y_test, y_test_pred, average='weighted')

    #Print the metrics
    print("rf-Train Accuracy:", train_accuracy)
    print("rf-Test Accuracy:", test_accuracy)

    print("rf-Train Precision:", train_precision)
    print("rf-Test Precision:", test_precision)

    print("rf-Train Recall:", train_recall)
    print("rf-Test Recall:", test_recall)

    print("rf-Train F1-score:", train_f1score)
    print("rf-Test F1-score:", test_f1score)
```

```
[ ] #SVC MODEL

    svc=SVC()
    svc.fit(x_train,y_train)

    #Obtain predictions for train and test sets
    y_train_pred = svc.predict(x_train)
    y_test_pred = svc.predict(x_test)

    #Calculate metrics for train and test sets
    train_accuracy = accuracy_score(y_train, y_train_pred)
    train_precision = precision_score(y_train, y_train_pred, average='weighted')
    train_recall = recall_score(y_train, y_train_pred, average='weighted')
    train_f1score = f1_score(y_train, y_train_pred, average='weighted')

    test_accuracy = accuracy_score(y_test, y_test_pred)
    test_precision = precision_score(y_test, y_test_pred, average='weighted')
    test_recall = recall_score(y_test, y_test_pred, average='weighted')
    test_f1score = f1_score(y_test, y_test_pred, average='weighted')

    #Print the metrics
    print("svc-Train Accuracy:", train_accuracy)
    print("svc-Test Accuracy:", test_accuracy)

    print("svc-Train Precision:", train_precision)
    print("svc-Test Precision:", test_precision)

    print("svc-Train Recall:", train_recall)
    print("svc-Test Recall:", test_recall)

    print("svc-Train F1-score:", train_f1score)
    print("svc-Test F1-score:", test_f1score)
```

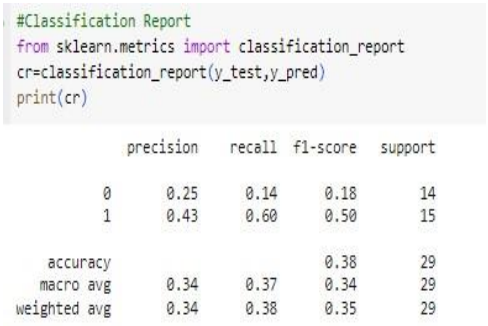
```
[ ] #LOGISTIC MODEL

    lr=LogisticRegression()
    lr.fit(x_train,y_train)

    #Obtain predictions for train and test sets
    y_train_pred = lr.predict(x_train)
    y_test_pred = lr.predict(x_test)

    #Calculate metrics for train and test sets
    train_accuracy = accuracy_score(y_train, y_train_pred)
    train_precision = precision_score(y_train, y_train_pred, average='weighted')
    train_recall = recall_score(y_train, y_train_pred, average='weighted')
    train_f1score = f1_score(y_train, y_train_pred, average='weighted')

    test_accuracy = accuracy_score(y_test, y_test_pred)
    test_precision = precision_score(y_test, y_test_pred, average='weighted')
    test_recall = recall_score(y_test, y_test_pred, average='weighted')
    test_f1score = f1_score(y_test, y_test_pred, average='weighted')

    #Print the metrics
    print("lr-Train Accuracy:", train_accuracy)
    print("lr-Test Accuracy:", test_accuracy)

    print("lr-Train Precision:", train_precision)
    print("lr-Test Precision:", test_precision)

    print("lr-Train Recall:", train_recall)
    print("lr-Test Recall:", test_recall)

    print("lr-Train F1-score:", train_f1score)
    print("lr-Test F1-score:", test_f1score)
```

Model Validation and Evaluation Report:

Model	Classification Report	F1 Score	Confusion Matrix

Decision Tree Model	[63] `#Classification Report`
	`from sklearn.metrics import classification_report`
	`cr=classification_report(y_test,y_pred)`
	`print(cr)`
	
	precision recall f1-score support
	0 0.73 0.57 0.64 14
	1 0.67 0.80 0.73 15
	accuracy 0.69 29
	macro avg 0.70 0.69 0.68 29
	weighted avg 0.70 0.69 0.69 29
		69%	[65] `#Confusion Matrix`
			`from sklearn.metrics import confusion_matrix`
			`cm=confusion_matrix(y_test,y_pred)`
			`print(cm)`
			
			[[8 6]
			[3 12]]

Random Forest Model	[67] #Classification Report from sklearn.metrics import classification_report cr=classification_report(y_test,y_pred) print(cr) precision recall f1-score support 0 0.56 0.36 0.43 14 1 0.55 0.73 0.63 15 accuracy 0.55 29 macro avg 0.55 0.55 0.53 29 weighted avg 0.55 0.55 0.54 29	55%	[68] #Confusion Matrix from sklearn.metrics import confusion_matrix cm=confusion_matrix(y_test,y_pred) print(cm) [[5 9] [4 11]]
KNN	71] #Classification Report from sklearn.metrics import classification_report cr=classification_report(y_test,y_pred) print(cr) precision recall f1-score support 0 0.25 0.14 0.18 14 1 0.43 0.60 0.50 15 accuracy 0.38 29 macro avg 0.34 0.37 0.34 29 weighted avg 0.34 0.38 0.35 29	38%	[72] #Confusion Matrix from sklearn.metrics import confusion_matrix cm=confusion_matrix(y_test,y_pred) print(cm) [[2 12] [6 9]]
SVC	[ ] #Classification Report from sklearn.metrics import classification_report cr=classification_report(y_test,y_pred) print(cr) precision recall f1-score support 0 0.25 0.14 0.18 14 1 0.43 0.60 0.50 15 accuracy 0.38 29 macro avg 0.34 0.37 0.34 29 weighted avg 0.34 0.38 0.35 29	38%	[ ] #Confusion Matrix from sklearn.metrics import confusion_matrix cm=confusion_matrix(y_test,y_pred) print(cm) [[2 12] [6 9]]

Logistic Regression Model		38%