Model Development Phase Template

Date	15 November 2024
TeamID	739909
Project Name	Unlocking the Minds: Analyzing Mental Health with NLP
Maximum Marks	10 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 (5 Marks):

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, f1_score

# Vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(x_train)  # Apply fit_transform to training data
X_test_tfidf = vectorizer.transform(x_test)  # Apply transform to test data

# Create and train the decision tree classifier
Dt = DecisionTreeClassifier()
Dt.fit(X_train_tfidf, y_train)  # Train the model on the vectorized data

# Make predictions
y_pred = Dt.predict(X_test_tfidf)
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, f1_score

# Vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(x_train)  # Apply fit_transform to training data
X_test_tfidf = vectorizer.transform(x_test)  # Apply transform to test data

# Create and train the random forest classifier
rf = RandomForestClassifier()
rf.fit(X_train_tfidf, y_train)  # Train the model on the vectorized data

# Make predictions
y_pred = rf.predict(X_test_tfidf)
```

```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, f1_score

# Vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(x_train)  # Apply fit_transform to training data
X_test_tfidf = vectorizer.transform(x_test)  # Apply transform to test data

# Create and train the AdaBoost classifier
ab = AdaBoostClassifier()
ab.fit(X_train_tfidf, y_train)  # Train the model on the vectorized data

# Make predictions
y_pred = ab.predict(X_test_tfidf)
```

```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, f1_score

# Vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(x_train)  # Apply fit_transform to training data
X_test_tfidf = vectorizer.transform(x_test)  # Apply transform to test data

# Create and train the Gradient Boosting classifier
gb = GradientBoostingClassifier()
gb.fit(X_train_tfidf, y_train)  # Train the model on the vectorized data

# Make predictions
y_pred = gb.predict(X_test_tfidf)
```

```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, f1_score

# Vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(x_train)  # Apply fit_transform to training data
X_test_tfidf = vectorizer.transform(x_test)  # Apply transform to test data

# Create and train the Logistic Regression classifier
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train_tfidf, y_train)  # Train the model on the vectorized data

# Make predictions
y_pred = lr.predict(X_test_tfidf)
```

```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, f1_score

# Vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(x_train)  # Apply fit_transform to training data
X_test_tfidf = vectorizer.transform(x_test)  # Apply transform to test data

# Create and train the Support Vector Classifier (SVC)
sv = SVC()
sv.fit(X_train_tfidf, y_train)  # Train the model on the vectorized data

# Make predictions
y_pred = sv.predict(X_test_tfidf)
```

Model Validation and Evaluation Report (5 Marks):

Model	Classification Report	F1 Score	Confusion Matrix
Decision Tree Classfier	precision recall f1-score support 0 0.82 0.83 0.82 4271 1 0.82 0.81 0.81 4121 accuracy 0.82 8392 macro avg 0.82 0.82 0.82 8392 weighted avg 0.82 0.82 0.82 8392 0.8169685414680649	81%	conf_matrix = confusion_matrix(y_test, y_pred) print("Decision Tree Confusion Matrix:\n", conf_matrix) Decision Tree Confusion Matrix: [[3524 747] [789 3332]]
Random Forest Classifier	precision recall f1-score support 0 0.88 0.90 0.89 4271 1 0.89 0.87 0.88 4121 accuracy 0.88 8392 macro avg 0.88 0.88 0.88 8392 weighted avg 0.88 0.88 0.88 8392 0.8831029551954243	88%	conf_matrix = confusion_matrix(y_test, y_pred) print("Random Forest Confusion Matrix:\n", conf_matrix) Random Forest Confusion Matrix: [[3836 435] [546 3575]]
Adaboost Classifier	precision recall f1-score support 0 0.84 0.91 0.87 4271 1 0.90 0.82 0.86 4121 accuracy 0.87 8392 macro avg 0.87 0.87 0.87 8392 weighted avg 0.87 0.87 0.87 8392 0.8666587225929456	86%	conf_matrix = confusion_matrix(y_test, y_pred) print("AdaBoost Confusion Matrix:\n", conf_matrix) AdaBoost Confusion Matrix: [[3906 365] [754 3367]]
Gradient Boosting Classifier	precision recall f1-score support 0 0.83 0.91 0.87 4271 1 0.90 0.81 0.85 4121 accuracy 0.86 8392 macro avg 0.87 0.86 0.86 8392 weighted avg 0.86 0.86 0.86 8392 0.8609389895138226	86%	conf_matrix = confusion_matrix(y_test, y_pred) print("Gradient Boosting Confusion Matrix:\n", conf_matrix) Gradient Boosting Confusion Matrix: [[3900 371] [796 3325]]
Logistic Regression	precision recall f1-score support 0 0.90 0.93 0.91 4271 1 0.93 0.89 0.91 4121 accuracy 0.91 8392 macro avg 0.91 0.91 0.91 8392 weighted avg 0.91 0.91 0.91 8392 0.911582459485224	91%	conf_matrix = confusion_matrix(y_test, y_pred) print("Logistic Regression Confusion Matrix:\n", conf_matrix) Logistic Regression Confusion Matrix: [[3975 296] [446 3675]]
Support Vector Classifier	precision recall f1-score support 0 0.90 0.94 0.92 4271 1 0.93 0.90 0.91 4121 accuracy 0.92 8392 macro avg 0.92 0.92 0.92 8392 weighted avg 0.92 0.92 0.92 8392	92%	conf_matrix = confusion_matrix(y_test, y_pred) print("Support Vector Classifier Confusion Matrix:\n", conf_matrix) Support Vector Classifier Confusion Matrix: [[3995 276] [422 3699]]

Model	Summary	Training and Validation Performance Metrics
Decision Tree Classfier	Decision Tree Classifier is particularly useful for “Unlocking the Minds: Analyzing Mental Health with NLP” due to its interpretability and ability to handle both categorical and numerical data, making it effective for understanding key decision points. It also excels in capturing non-linear patterns, which are often present in mental health-related text data.	-
Random Forest Classifier	Random Forest Classifier is well-suited for “Unlocking the Minds: Analyzing Mental Health with NLP” due to its ability to handle large feature spaces efficiently and its robustness against overfitting through ensemble averaging, making it ideal for text data with complex patterns.	-
Adaboost Classifier	AdaBoost Classifier enhances text prediction in “Unlocking the Minds: Analyzing Mental Health with NLP” by combining weak learners into a strong model, excelling in handling imbalanced data and improving classification accuracy through adaptive boosting techniques.	-
Gradient Boosting Classifier	Gradient Boosting Classifier excels in mental health text analysis within “Unlocking the Minds” due to its ability to build robust models by combining weak learners, effectively capturing complex patterns in data. Its iterative approach minimizes error and enhances prediction accuracy, making it ideal for nuanced NLP tasks.	-
Logistic Regression	Logistic Regression is ideal for “Unlocking the Minds: Analyzing Mental Health with NLP” due to its simplicity, interpretability, and effectiveness in handling binary or multiclass text classification tasks. It leverages the sigmoid function to model probabilities, making it suitable for distinguishing subtle patterns in mental health-related text data.	-
Support Vector Classifier	SVM is particularly effective for mental health text analysis in “Unlocking the Minds: Analyzing Mental Health with NLP” due to its ability to handle high-dimensional textual data and classify complex patterns using kernel-based transformations. Its robustness to overfitting ensures accurate predictions, even with limited labeled data.