Intensity Analysis

Welcome to Intensity Analysis Notebook!

Greeting and welcome to this comprehensive notebook that walks you through the model.

Overview

This report presents the findings and analysis of a project aimed at developing a text classification model for classifying emotions based on their intensity levels. The project utilized machine learning techniques to classify textual data into different categories of emotions, including happiness, sadness, and angriness.

The project involved the following key steps:

1. **Data Collection:** The initial step involved collecting a dataset containing textual data labeled with different intensity levels of emotions, including happiness, sadness, and angriness.
2. **Data Preprocessing:** The collected data was preprocessed to clean and prepare it for analysis. This step included removing special characters, converting text to lowercase, and handling missing values.
3. **Feature Engineering:** Text data was transformed into numerical features using TF-IDF vectorization. Feature selection techniques such as SelectKBest were applied to select the most relevant features.
4. **Model Training:** Machine learning models, such as Multinomial Naive Bayes, were trained on the preprocessed data to classify emotions based on their intensity levels.
5. **Model Evaluation:** The trained models were evaluated using performance metrics such as accuracy, precision, recall, and F1-score. Cross-validation techniques were applied to assess model generalization.
6. **Hyperparameter Tuning:** Hyperparameters of the models were fine-tuned using techniques like GridSearchCV to improve model performance.
7. **Deployment:** Once the model was trained and evaluated, it was deployed for real-world applications to classify emotions in unseen textual data.

This code will help you build and evaluate your NLP model effectively.

Dependencies and Imports

```
import pandas as pd
from sagemaker import Session
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, roc_curve, precision_recall_curve, roc_auc_score,
average_precision_score
import matplotlib.pyplot as plt
from sklearn.preprocessing import label_binarize
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV,
cross_val_score
from scipy.stats import uniform
import boto3
import sagemaker
```

Data Loading from Amazon S3

- Created a SageMaker session.
- Specified the S3 bucket name.
- Defined S3 paths for each dataset: happiness, sadness, and angriness.
- Loaded data from S3 into pandas DataFrames.

```
# Create a SageMaker Session
session = Session()

# Specify S3 bucket
bucket_name = 'intensity01'

# Specify S3 paths for your datasets
s3_paths = {
    'happiness': f's3://{bucket_name }/happiness.csv',
    'sadness': f's3://{bucket_name }/sadness.csv',
    'angriness': f's3://{bucket_name}/angriness.csv'
}

# Load data from S3
data = {}
for label, path in s3_paths.items():
    data[label] = pd.read_csv(path)
```

```
/opt/conda/lib/python3.10/site-packages/fsspec/registry.py:275:
UserWarning: Your installed version of s3fs is very old and known to
cause
severe performance issues, see also
https://github.com/dask/dask/issues/10276

To fix, you should specify a lower version bound on s3fs, or
update the current installation.

  warnings.warn(s3_msg)
```

Data Preprocessing

- Iterated over each dataset (happiness, sadness, angriness).
- Removed special characters from the 'content' column using regular expressions.
- Converted text to lowercase to ensure uniformity.
- Handled missing values by dropping rows with missing values.
- Printed the preprocessed data for each dataset.

```python
# Data Preprocessing
for label, df in data.items():
    # Remove special characters
    df['content'] = df['content'].str.replace('[^\w\s]', '')

    # Convert text to lowercase
    df['content'] = df['content'].str.lower()

    # Handle missing values
    df.dropna(inplace=True)  # Remove rows with missing values

    # Print the preprocessed data
    print(f"Preprocessed {label} dataset:")
    print(df.head())
```

```
Preprocessed happiness dataset:
                                             content  intensity
0  wants to know how the hell i can remember word...  happiness
1  love is a long sweet dream & marriage is an al...  happiness
2  the world could be amazing when you are slight...  happiness
3  my secret talent is getting tired without doin...  happiness
4  khatarnaak whatsapp status ever… can\’t talk, ...  happiness
Preprocessed sadness dataset:
                                             content intensity
0  never hurt people who love you a lot, because ...   sadness
1  don’t expect me to tell you what you did wrong...   sadness
2  i preferred walking away than fighting for you...   sadness
3  moving forward in life isn’t the hard part, it...   sadness
4  never cry for anyone in your life, because tho...   sadness
Preprocessed angriness dataset:
```

```
                                                content  intensity
0  sometimes i’m not angry, i’m hurt and there’s ...  angriness
1                    not available for busy people☺  angriness
2  i do not exist to impress the world. i exist t...  angriness
3  everything is getting expensive except some pe...  angriness
4       my phone screen is brighter than my future ☹  angriness
```

Feature Engineering and Feature Selection

- Combined all datasets (happiness, sadness, angriness) into a single DataFrame.
- Applied TF-IDF Vectorization for feature engineering, converting text data into numerical features.
- Used TfidfVectorizer with a specified maximum number of features (max_features=1000).
- Performed feature selection using SelectKBest with a chi-square test.
- Selected the top 500 features based on their chi-square scores.
- Extracted the selected feature names and printed them for further reference.

```
# Combine data into a single DataFrame
combined_data = pd.concat([
    data['happiness'].assign(label='happiness'),
    data['sadness'].assign(label='sadness'),
    data['angriness'].assign(label='angriness')
], ignore_index=True)

# Feature Engineering: TF-IDF Vectorization
vectorizer = TfidfVectorizer(max_features=1000)
X = vectorizer.fit_transform(combined_data['content'])
y = combined_data['label']

# Feature Selection: SelectKBest with chi-square test
selector = SelectKBest(score_func=chi2, k=500)
X_selected = selector.fit_transform(X, y)

# Get the selected feature names
feature_names = vectorizer.get_feature_names_out()
selected_feature_names = [feature_names[i] for i in
selector.get_support(indices=True)]

# Print selected feature names
print("Selected Features:")
print(selected_feature_names)
```

```
Selected Features:
['able', 'actually', 'afraid', 'again', 'allow', 'alone', 'always',
'am', 'amount', 'anger', 'angry', 'annoying', 'answer', 'any',
'anymore', 'anyone', 'anything', 'apart', 'appreciate', 'are',
'attitude', 'awake', 'away', 'bad', 'balanced', 'balloon', 'based',
```

'basically', 'beautiful', 'because', 'bed', 'been', 'begin', 'begun',
'being', 'believing', 'better', 'beyond', 'big', 'birth', 'bitches',
'blankly', 'bleakness', 'blessings', 'bliss', 'block', 'boy', 'break',
'breaking', 'breakup', 'bright', 'broke', 'broken', 'burst',
'business', 'but', 'buy', 'calm', 'can', 'cannot', 'cant', 'capacity',
'care', 'cares', 'caring', 'change', 'changed', 'cheaper', 'cheat',
'cheating', 'cheerful', 'children', 'choke', 'close', 'closer',
'clouds', 'colors', 'complains', 'complete', 'condition', 'cries',
'crumpled', 'cry', 'crying', 'curious', 'damage', 'danger', 'day',
'death', 'decide', 'decision', 'deep', 'deeper', 'deepest', 'deeply',
'definitely', 'definition', 'delete', 'deleted', 'depressed',
'depressing', 'describe', 'deserves', 'destroy', 'did', 'die', 'died',
'difference', 'direction', 'disappearing', 'disappointment', 'doesnt',
'don', 'dont', 'drinking', 'during', 'dying', 'each', 'easier',
'easily', 'eat', 'eight', 'embedded', 'empty', 'energy', 'english',
'enjoy', 'enough', 'entertain', 'even', 'eventually', 'every',
'everyday', 'everyone', 'everything', 'evil', 'ex', 'exactly',
'except', 'exit', 'expectation', 'expensive', 'experience',
'experiencing', 'expired', 'explain', 'explaining', 'eyes', 'face',
'facebook', 'fact', 'failed', 'failure', 'fall', 'falling', 'far',
'fast', 'faster', 'fear', 'feel', 'feelings', 'feels', 'fightin',
'fighting', 'finger', 'finished', 'flat', 'flower', 'food', 'for',
'forces', 'forget', 'forgetting', 'forgive', 'forgiven',
'forgiveness', 'forgotten', 'form', 'friend', 'from', 'frustrated',
'frustration', 'get', 'getting', 'girls', 'given', 'go', 'goes',
'granted', 'guess', 'guys', 'had', 'hands', 'happen', 'happened',
'happens', 'happiness', 'happy', 'hard', 'hating', 'head', 'heal',
'heart', 'heartbreak', 'hearts', 'heavy', 'her', 'hey', 'him',
'himself', 'home', 'hope', 'how', 'hug', 'hurt', 'hurting', 'hurts',
'if', 'ignore', 'important', 'increases', 'indicates', 'inner',
'inside', 'instead', 'is', 'it', 'its', 'journey', 'joy', 'judge',
'just', 'karma', 'keeping', 'key', 'kill', 'kind', 'kiss', 'knives',
'know', 'knowing', 'left', 'letter', 'lie', 'lied', 'like', 'likes',
'limits', 'little', 'll', 'loneliest', 'loneliness', 'losing', 'lots',
'love', 'loved', 'lying', 'mad', 'main', 'make', 'makes', 'man',
'manage', 'management', 'manager', 'map', 'me', 'meaning', 'meant',
'measure', 'memories', 'mental', 'messed', 'middle', 'mind', 'minds',
'minute', 'mirror', 'misery', 'miss', 'missing', 'mood', 'more',
'morning', 'most', 'mouth', 'mouths', 'move', 'moving', 'much',
'music', 'my', 'name', 'needed', 'negative', 'never', 'new', 'nicest',
'night', 'nightmare', 'no', 'nobody', 'nothing', 'noticed', 'now',
'number', 'off', 'okay', 'once', 'ones', 'online', 'open', 'opens',
'order', 'other', 'our', 'out', 'own', 'page', 'paid', 'pain',
'painful', 'papers', 'part', 'people', 'person', 'personality',
'persons', 'phrase', 'pick', 'pieces', 'pissed', 'pissing', 'please',
'plot', 'poison', 'positive', 'power', 'prefect', 'pretty',
'promised', 'prudent', 'pulled', 'punch', 'punished', 'pushing',
'put', 'quality', 'quick', 'quiet', 'rain', 're', 'react', 'read',
'real', 'realize', 'realizing', 'red', 'regret', 'rejected',

```
'relationships', 'relieved', 'remembered', 'remembering', 'replaces',
'response', 'return', 'revenge', 'ridiculous', 'sad', 'saddest',
'sadness', 'say', 'says', 'seconds', 'secret', 'seem', 'seldom',
'selfish', 'set', 'shame', 'she', 'shit', 'short', 'shuts', 'sick',
'silence', 'sin', 'sixty', 'skills', 'smaller', 'smile', 'smiling',
'so', 'solve', 'some', 'someday', 'someone', 'something', 'sometimes',
'somewhere', 'soon', 'sorry', 'speak', 'speed', 'spread', 'stare',
'status', 'still', 'stop', 'stopped', 'stupidity', 'success', 'sucks',
'suddenly', 'surprised', 'taken', 'talking', 'tears', 'technically',
'tell', 'telling', 'temper', 'texts', 'than', 'thanks', 'that', 'the',
'their', 'they', 'things', 'think', 'thinking', 'thought', 'thousand',
'throw', 'tired', 'to', 'together', 'tomorrow', 'tongue', 'too',
'touched', 'toughest', 'transmuted', 'treat', 'treated', 'tried',
'true', 'trust', 'truth', 'trying', 'turn', 'turns', 'typing',
'understand', 'upset', 'use', 'used', 'using', 've', 'waiting',
'walk', 'wanna', 'was', 'wasn', 'waste', 'wasted', 'watch', 'water',
'we', 'weak', 'whatsapp', 'when', 'whenever', 'while', 'who', 'why',
'wife', 'will', 'win', 'winning', 'wish', 'woman', 'won', 'words',
'work', 'works', 'worse', 'worst', 'wouldn', 'wrong', 'xa0',
'xa0angry', 'xa0complicated', 'xa0sad', 'years', 'yet', 'you', 'your']
```

Train-Test Split

- Utilized train_test_split function from sklearn.model_selection module to split the dataset into training and testing sets.
- Specified the test size to be 20% of the entire dataset and set a random state for reproducibility.
- Printed the shapes of the training and testing sets to verify the split.

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_selected, y,
test_size=0.2, random_state=42)

# Print the shapes of the training and testing sets
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (1631, 500)
Shape of X_test: (408, 500)
Shape of y_train: (1631,)
Shape of y_test: (408,)
```

Model Training and Evaluation

- Initialized the Multinomial Naive Bayes classifier.
- Trained the classifier using the training data.
- Made predictions on the test set.

- Evaluated the model's performance using accuracy score.

```
# Initialize the Multinomial Naive Bayes classifier
classifier = MultinomialNB()

# Train the classifier
classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.7377450980392157
```

Model Evaluation Metrics

Classification Report:

- Generated a classification report containing precision, recall, F1-score, and support for each class.

Confusion Matrix:

- Calculated and printed the confusion matrix to visualize the performance of the classifier.

```
# Classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

   angriness       0.71      0.82      0.76       129
   happiness       0.81      0.59      0.68       156
     sadness       0.71      0.84      0.77       123

    accuracy                           0.74       408
   macro avg       0.74      0.75      0.74       408
weighted avg       0.75      0.74      0.73       408
```

```
Confusion Matrix:
[[106  10  13]
 [ 35  92  29]
 [  9  11 103]]
```

Hyperparameter Tuning using GridSearchCV

Best Hyperparameters:

- Identified the best hyperparameters using GridSearchCV, which yielded the best performance.

Model Evaluation Metrics:

- Calculated accuracy, precision, recall, and F1-score to evaluate the best model's performance.

Classification Report:

- Provided a detailed classification report containing precision, recall, F1-score, and support for each class.

Confusion Matrix:

- Generated a confusion matrix to visualize the performance of the best model.

```
# Define hyperparameters to tune
param_grid = {
    'alpha': [0.1, 0.5, 1.0],
}

# Initialize Multinomial Naive Bayes classifier
classifier = MultinomialNB()

# Perform GridSearchCV for hyperparameter tuning
grid_search = GridSearchCV(classifier, param_grid, cv=5,
scoring='accuracy')
best_model = grid_search.fit(X_train, y_train)

# Print best hyperparameters
print("Best Hyperparameters:", best_model.best_params_)

# Predictions on test set using the best model
y_pred = best_model.predict(X_test)

# Evaluate the best model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Classification report
print("\nClassification Report:")
```

```
print(classification_report(y_test, y_pred))

# Confusion matrix
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Best Hyperparameters: {'alpha': 0.5}
Accuracy: 0.7426470588235294

Classification Report:
              precision    recall  f1-score   support

   angriness       0.72      0.81      0.76       129
   happiness       0.81      0.62      0.70       156
     sadness       0.72      0.84      0.77       123

    accuracy                           0.74       408
   macro avg       0.75      0.75      0.74       408
weighted avg       0.75      0.74      0.74       408


Confusion Matrix:
[[104  12  13]
 [ 32  96  28]
 [  9  11 103]]
```

Hyperparameter Tuning using RandomizedSearchCV

Best Hyperparameters:

- Utilized RandomizedSearchCV to identify the best hyperparameters, which resulted in improved performance.

Cross-Validation:

- Evaluated the best model's performance using cross-validation to ensure robustness and reliability.

Mean Cross-Validation Score:

- Calculated the mean cross-validation score to assess the overall performance of the best model.

```
# Define the hyperparameter grid
param_dist = {'alpha': uniform(0.1, 1.0)}

# Initialize the Multinomial Naive Bayes classifier
classifier = MultinomialNB()

# Initialize RandomizedSearchCV
random_search = RandomizedSearchCV(classifier,
```

```
param_distributions=param_dist, n_iter=10, cv=5, scoring='accuracy',
random_state=42)

# Perform RandomizedSearchCV
random_search.fit(X_train, y_train)

# Print the best hyperparameters
print("Best Hyperparameters:", random_search.best_params_)

# Evaluate the best model using cross-validation
best_model = random_search.best_estimator_
cv_scores = cross_val_score(best_model, X_train, y_train, cv=5)
print("Cross-Validation Scores:", cv_scores)
print("Mean CV Score:", cv_scores.mean())
```

```
Best Hyperparameters: {'alpha': 0.47454011884736247}
Cross-Validation Scores: [0.77981651 0.79141104 0.71472393 0.7607362
0.7607362 ]
Mean CV Score: 0.7614847751449316
```

Model Evaluation

Accuracy:

- The accuracy of the best model on the test set is calculated to assess its overall performance.

Classification Report:

- A detailed classification report is generated to provide insights into the precision, recall, and F1-score for each class.

Confusion Matrix:

- The confusion matrix is generated to visualize the performance of the model in terms of true positive, false positive, true negative, and false negative predictions.

```
# Predictions on test set
y_pred_test = best_model.predict(X_test)

# Calculate accuracy score
accuracy = accuracy_score(y_test, y_pred_test)
print("Accuracy:", accuracy)

# Generate classification report
print("Classification Report:")
print(classification_report(y_test, y_pred_test))

# Generate confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_test))
```

```
Accuracy: 0.7401960784313726
Classification Report:
              precision    recall  f1-score   support

   angriness       0.72      0.81      0.76       129
   happiness       0.80      0.62      0.70       156
     sadness       0.71      0.83      0.77       123

    accuracy                           0.74       408
   macro avg       0.74      0.75      0.74       408
weighted avg       0.75      0.74      0.74       408

Confusion Matrix:
[[104  12  13]
 [ 32  96  28]
 [  9  12 102]]
```

Model Deployment with Amazon SageMaker

Initialize SageMaker Session:

- Initialize the SageMaker session to interact with SageMaker services.

Specify IAM Role:

- Specify the IAM role ARN used to give SageMaker access to your data and resources.

Define Endpoint Name:

- Create a unique name for the endpoint where the deployed model will be hosted.

Deploy the Model:

- Deploy the trained model to Amazon SageMaker with specified configurations such as instance count, instance type, and endpoint name.

```
# Initialize the SageMaker session
session = sagemaker.Session()

# Specify the IAM role arn used to give SageMaker access to data
role = "sagemaker-role-arn"

# Create a unique name for the endpoint
endpoint_name = "endpoint-name"

# Deploy the model to Amazon SageMaker
predictor = None
try:
    # Deploy the model
    predictor = model.deploy(
        initial_instance_count=1,
```

```
        instance_type="ml.m4.xlarge",
        endpoint_name=endpoint_name
    )
    print("Model deployed successfully!")
except Exception as e:
    print("Deployment failed:", e)
```

Project Conclusion:

In this project, we aimed to develop a text classification model to classify text data into different intensity levels of emotions such as happiness, sadness, and angriness. Here's a summary of the key steps and findings of the project:

1. **Data Collection:** We collected text data for different emotions from various sources and stored them in separate CSV files.
2. **Data Preprocessing:** The collected data underwent preprocessing steps including removing special characters, converting text to lowercase, and handling missing values.
3. **Feature Engineering:** We used TF-IDF vectorization to convert the text data into numerical features, which represent the importance of words in the documents.
4. **Feature Selection:** To select the most relevant features, we employed SelectKBest with the chi-square test to reduce the dimensionality of the feature space.
5. **Model Training:** We trained a Multinomial Naive Bayes classifier using the preprocessed and selected features.
6. **Model Evaluation:** The trained model was evaluated using metrics such as accuracy, precision, recall, and F1-score. Additionally, confusion matrices were generated to analyze the performance of the model for each emotion class.
7. **Hyperparameter Tuning:** We explored hyperparameter tuning using techniques like GridSearchCV and RandomizedSearchCV to optimize the model's performance.
8. **Deployment:** Finally, we deployed the trained model on Amazon SageMaker to make it available for inference.

Conclusion: The developed text classification model demonstrated promising results in accurately classifying text data into different emotion intensity levels. Through systematic preprocessing, feature engineering, and model training, we were able to build a robust model capable of understanding and classifying emotions in textual data. The deployment of the model on Amazon SageMaker provides a scalable and efficient solution for real-time inference, making it suitable for various applications such as sentiment analysis, customer feedback analysis, and social media monitoring.

Thank You

Riddhi Sharma - Data Scientist