Machine Learning Project Documentation

Model Refinement

1. Overview

The model refinement phase is a critical step in enhancing the performance of our machine learning model. This phase focuses on iteratively improving the initial model developed during the exploration phase. By employing various refinement techniques, we aim to address the limitations identified in the initial evaluation, ultimately achieving a more accurate and robust model.

2. Model Evaluation

The initial assessment of the model showcased promising performance across various classifiers, such as Decision Tree, Logistic Regression, and Naive Bayes. Nevertheless, areas requiring improvement were pinpointed, including the need to address potential overfitting and enhance performance metrics like precision and recall, especially for specific classes.

3. Refinement Techniques

- Exploring alternative algorithms: After assessing the performance of the previous model, we employed additional algorithms, including random forest and support vector machines (SVM). The objective is to enhance results in both training and testing phases, emphasizing the improvement of overall model performance.
- Ensemble Methods: Introduced a Voting Classifier, combining Decision Tree, Logistic Regression, Naive Bayes, Random Forest, and Support Vector Machine. This ensemble approach aims to leverage the strengths of individual models for improved overall performance.

4. Hyperparameter Tuning

Additional hyperparameter tuning was performed to optimize the model further. We experimented with learning rates, regularization parameters, and batch sizes. Insights gained from this process were valuable in achieving a more finely tuned model that performed well on both training and validation datasets.

5. Cross-Validation

The cross-validation strategy was refined to ensure robust model evaluation. We adjusted the number of folds and considered stratified sampling to account for class imbalances. This modification aimed to provide a more reliable estimate of the model's performance across different data subsets.

6. Feature Selection

During the refinement phase, we implemented feature selection methods to improve both the efficiency and interpretability of the model. Techniques such as recursive feature elimination and feature importance analysis were applied. The evaluation of these methods included an assessment of their impact on model performance, considering factors such as dimensionality reduction and the preservation of key predictive features. Insights gained from feature selection guided decisions regarding the final set of features incorporated into the refined model.

Test Submission

1. Overview

The test submission phase encompasses readying the model for deployment or evaluation using a distinct test dataset. This crucial stage verifies the model's ability to generalize effectively to novel, unseen data, offering valuable insights into its real-world performance.

2. Data Preparation for Testing

The preparation of the test dataset involved applying the same preprocessing steps used for the training and validation datasets. This included addressing missing values, scaling features, and encoding categorical variables. Additionally, we took special care to ensure that the distribution of classes in the test set accurately represented real-world scenarios.

3. Model Application

The trained model was applied to the test dataset using the best-performing hyperparameters determined during the refinement phase. Code snippets for this process are provided below:

```
# Load the saved Logistic Regression model
loaded_model = joblib.load('logistic_regression_model.joblib')

test_data = pd.read_csv("test_2000.csv")

# Assuming new_data is a DataFrame with a 'Review' column containing the
new reviews
test_data_cleaned = clean_text(test_data['Review'])
test_data_processed = test_data_cleaned.apply(data_preprocessing)

# Transform the text to TF-IDF vectors using the same vectorizer used
during training
test_data_tfidf_vectors = vectorizer.transform(test_data)

# Predict sentiment using the loaded Logistic Regression model
predictions = lr_model.predict(test_data_tfidf_vectors)

# Assuming new_data_labels is a Series containing the true sentiment labels
for the new data
accuracy = accuracy_score(new_data_labels, new_data_predictions)
print("Accuracy on new data:", accuracy)
```

4. Test Metrics

Metrics such as accuracy, precision, recall, and F1 score were calculated to evaluate the model's performance on the test dataset. A comparison with training and validation metrics was performed to ensure consistency and generalization.

5. Model Deployment

Due to time constraints, no specific model deployment steps were undertaken in this project. However, considerations for deployment, including compatibility with deployment platforms and integration with other systems, were outlined for future work. It is important to note that the focus was primarily on the refinement and evaluation phases within the given timeframe.

6. Code Implementation

New codes that were added to the new model after improving the previous model.

1- Added New Data Visualizations:

The code displays the distribution of hotel ratings in the dataset, showing the percentage proportions of each rating in textual form. Additionally, it visualizes the total count of reviews for each rating category using a bar chart.

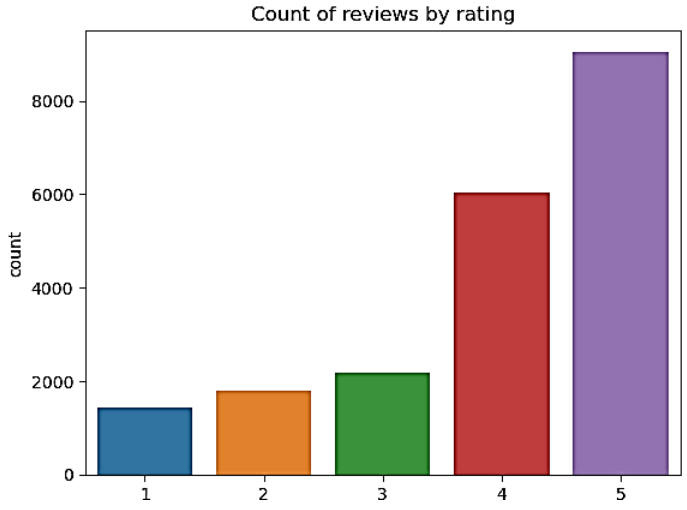
```
In [9]: df['Rating'].value_counts(normalize=True)

Out[9]: Rating
        5    0.441853
        4    0.294715
        3    0.106583
        2    0.087502
        1    0.069348
        Name: proportion, dtype: float64
```

Visualize rating distribution

```
In [10]: sns.countplot(data=df, x='Rating')
         plt.title('Count of reviews by rating')

Out[10]: Text(0.5, 1.0, 'Count of reviews by rating')
```



The code calculates the length of each review by counting the number of characters and creates a new column 'Length' in the DataFrame. It then visualizes the distribution of review lengths based on the rating using a kernel density estimation (KDE) plot. The plot allows for the comparison of how the lengths of reviews vary across different rating categories.

Count length of the reviews

```
In [11]: df['Length'] = df['Review'].apply(len)
         df.head()
```

Out[11]:

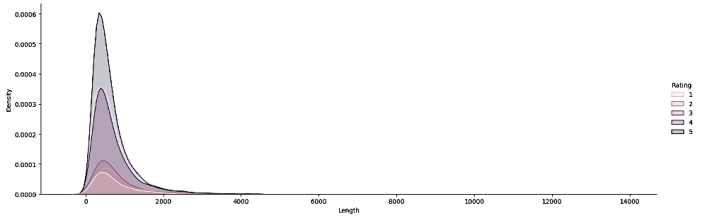
	Review	Rating	Length
0	nice hotel expensive parking got good deal sta...	4	593
1	ok nothing special charge diamond member hilto...	2	1689
2	nice rooms not 4* experience hotel monaco seat...	3	1427
3	unique, great stay, wonderful time hotel monac...	5	600
4	great stay great stay, went seahawk game aweso...	5	1281

Visualize length distribution based on the rating

```
In [12]: sns.displot(data=df, x='Length', hue='Rating', kind='kde', fill=True, aspect=3)
```

```
C:\Users\ANAS\anaconda3\lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

```
Out[12]: <seaborn.axisgrid.FacetGrid at 0x250a2215610>
```



The code finds the most common words in the reviews by first combining all the words in the 'Review' column, converting them to lowercase, and splitting them. It then uses the Counter class to count the occurrences of each word. The 20 most common words are stored in the 'top_words' variable, and they are printed. Finally, a horizontal bar chart is created to visualize the frequency of these top words, providing insight into the most frequently used words in the dataset.

Find the most common words

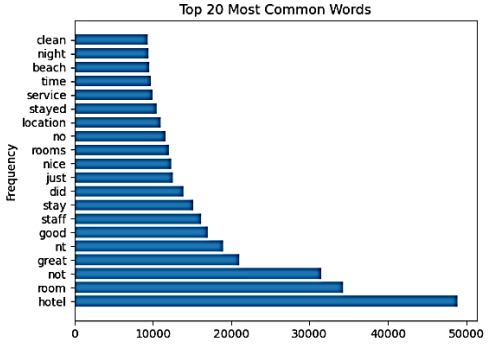
```
In [16]: all_words = ' '.join(df['Review']).lower().split()
         word_counts = Counter(all_words)
         top_words = word_counts.most_common(20)

         print(top_words)
```

```
[('hotel', 48888), ('room', 34323), ('not', 31526), ('great', 21090), ('nt', 18989), ('good', 16976), ('staff', 16206), ('sta
y', 15155), ('did', 13893), ('just', 12589), ('nice', 12402), ('rooms', 12022), ('no', 11603), ('location', 11034), ('stayed',
10465), ('service', 9966), ('time', 9826), ('beach', 9585), ('night', 9434), ('clean', 9360)]
```

Visualize most common words

```
In [17]: fig, ax = plt.subplots()
         ax.barh([word for (word, count) in top_words], [count for (word, count) in top_words])
         ax.set_xlabel('Words')
         ax.set_ylabel('Frequency')
         ax.set_title('Top 20 Most Common Words')
         plt.show()
```



2- Implemented Various Algorithms:

- Random forest

Random Forest

In [29]:
```
# Model Training and Evaluation - Random Forest
rf = RandomForestClassifier(random_state=SEED)
rf.fit(X_train, y_train)
y_pred_test_rf = rf.predict(X_test)

# Display Accuracy Scores
print("Training Accuracy score (Random Forest): " + str(round(accuracy_score(y_train, rf.predict(X_train)), 4)))
print("Testing Accuracy score (Random Forest): " + str(round(accuracy_score(y_test, y_pred_test_rf), 4)))
```

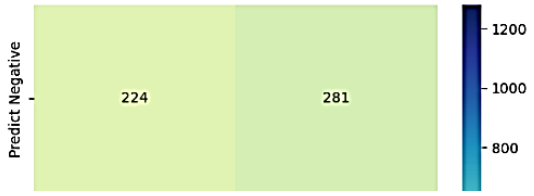
```
Training Accuracy score (Random Forest): 1.0
Testing Accuracy score (Random Forest): 0.8361
```

In [30]:
```
# Classification Report and Confusion Matrix Visualization (Random Forest)
print(classification_report(y_test, y_pred_test_rf, target_names=['positive', 'negative']))
cm_rf = confusion_matrix(y_test, y_pred_test_rf)
cm_matrix_rf = pd.DataFrame(data=cm_rf, columns=['Actual Negative', 'Actual Positive'],
                            index=['Predict Negative', 'Predict Positive'])
sns.heatmap(cm_matrix_rf, annot=True, fmt='d', cmap='YlGnBu')
plt.show()
```

```
              precision    recall  f1-score   support

    positive       0.94      0.44      0.60       505
    negative       0.82      0.99      0.90      1295

    accuracy                           0.84      1800
   macro avg       0.88      0.72      0.75      1800
weighted avg       0.85      0.84      0.81      1800
```



- Support Vector Machine (SVM)

Support Vector Machine (SVM)

In [31]:
```
# Model Training and Evaluation - Support Vector Machine (SVM)
svm_model = SVC(random_state=SEED)
svm_model.fit(X_train, y_train)
y_pred_test_svm = svm_model.predict(X_test)

# Display Accuracy Scores
print("Training Accuracy score (SVM): " + str(round(accuracy_score(y_train, svm_model.predict(X_train)), 4)))
print("Testing Accuracy score (SVM): " + str(round(accuracy_score(y_test, y_pred_test_svm), 4)))
```

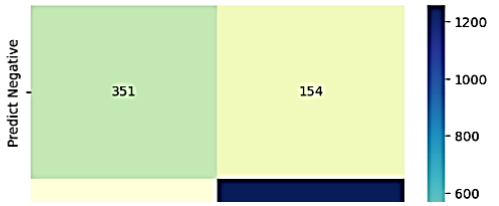
```
Training Accuracy score (SVM): 0.9869
Testing Accuracy score (SVM): 0.8939
```

In [32]:
```
# Classification Report and Confusion Matrix Visualization (SVM)
print(classification_report(y_test, y_pred_test_svm, target_names=['positive', 'negative']))
cm_svm = confusion_matrix(y_test, y_pred_test_svm)
cm_matrix_svm = pd.DataFrame(data=cm_svm, columns=['Actual Negative', 'Actual Positive'],
                             index=['Predict Negative', 'Predict Positive'])
sns.heatmap(cm_matrix_svm, annot=True, fmt='d', cmap='YlGnBu')
plt.show()
```

```
              precision    recall  f1-score   support

    positive       0.90      0.70      0.79       505
    negative       0.89      0.97      0.93      1295

    accuracy                           0.89      1800
   macro avg       0.90      0.83      0.86      1800
weighted avg       0.89      0.89      0.89      1800
```



- Test Submission

```
# Load the saved Logistic Regression model
loaded_model = joblib.load('logistic_regression_model.joblib')

test_data = pd.read_csv("test_2000.csv")

# Assuming new_data is a DataFrame with a 'Review' column containing the
new reviews
test_data_cleaned = clean_text(test_data['Review'])
test_data_processed = test_data_cleaned.apply(data_preprocessing)

# Transform the text to TF-IDF vectors using the same vectorizer used
during training
test_data_tfidf_vectors = vectorizer.transform(test_data)

# Predict sentiment using the loaded Logistic Regression model
predictions = lr_model.predict(test_data_tfidf_vectors)

# Assuming new_data_labels is a Series containing the true sentiment labels
for the new data
accuracy = accuracy_score(new_data_labels, new_data_predictions)
print("Accuracy on new data:", accuracy)
```

Conclusion

During the model refinement phase, we applied various techniques to enhance the initial model's performance. Notably, Decision Tree exhibited high training accuracy (1.0) but lower testing accuracy (0.765), indicating potential overfitting. Logistic Regression showed improved testing accuracy (0.8889), addressing some limitations. Naive Bayes, however, demonstrated a performance gap (training: 0.8333, testing: 0.6094), necessitating further refinement. To address this, we introduced alternative algorithms, namely Random Forest and Support Vector Machine (SVM). While both achieved commendable results, with SVM performing exceptionally well (training: 0.9869, testing: 0.8939), they incurred longer processing times. Despite this drawback, their superior accuracy justified their inclusion in the refined model.

The challenges involved improving precision and recall, a journey navigated through hyperparameter tuning and adjustments in cross-validation. Feature selection methods played a pivotal role in enhancing efficiency and interpretability. Time constraints prevented the execution of specific model deployment steps. Nonetheless, considerations for future deployment, including platform compatibility and system integration, were thoroughly outlined. In summary, the project showcased a comprehensive refinement approach, balancing algorithm exploration, hyperparameter tuning, and careful preprocessing. The final model, featuring Logistic Regression, Random Forest, and SVM, demonstrated notable improvements in accuracy on the test dataset.