Assignment 3 – Shivam Rawat (ssrawat2)

Task 4: Table with the summary of Task 1-4 experiment (the train dataset had 1181 rows and the testing dataset had 937 rows)

Model	Feature space	Micro			Macro			Weighted		
		Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Baseline (Naive Bayes)	5038	0.5571	0.5571	0.5571	0.6212	0.4759	0.4735	0.5998	0.5571	0.5160
Low variance (threshold = 0.005)	490	0.5390	0.5390	0.5390	0.5452	0.5113	0.5212	0.5438	0.5390	0.5356
Low variance (threshold = 0.001)	1596	0.5603	0.5603	0.5603	0.5766	0.5151	0.5271	0.5704	0.5603	0.5495
k-best using mutual information with k = 2000	2000	0.5486	0.5486	0.5486	0.6108	0.4514	0.4417	0.5914	0.5486	0.4927
k-best using mutual information with k = 1000	1000	0.5229	0.5229	0.5229	0.5429	0.4427	0.4284	0.5364	0.5229	0.4760
Lexicon-based feature selection	678	0.5656	0.5656	0.5656	0.5931	0.5001	0.5130	0.5822	0.5656	0.5433

Task 2

Threshold = 0.001:
The number of features is reduced to 1596, which is a moderate reduction. Performance improves slightly compared to the baseline:
Micro F1, Macro F1, and Weighted F1 increased. This suggests that removing low variance features with a less aggressive threshold (0.001) helps retain useful information, leading to better performance.

Threshold = 0.005:
The number of features is reduced significantly to 490, which is a drastic reduction. Performance drops slightly compared to the baseline:
Micro F1 decreased, whereas Macro F1, and Weighted F1 increased. This indicates that a more aggressive threshold (0.005) removes too many features, some of which may be important, leading to a drop in performance.
In conclusion:
More features generally lead to better performance, as seen in the baseline and threshold = 0.001. Too few features (e.g., threshold = 0.005) can harm performance by removing important information. I observed a trade-off between the number of features and performance

Task 3

k = 1000:
The number of features is reduced to 1000. Performance drops significantly compared to the baseline:
Micro F1, Macro F1, and Weighted F1 decreased. This suggests that selecting only 1000 features using mutual information removes too many important features, harming performance.

k = 2000:
The number of features is reduced to 2000. Performance improves compared to k = 1000 but is still lower than the baseline:
Micro F1, Macro F1, and Weighted F1 increased. This indicates that increasing the number of features (from 1000 to 2000) improves performance, but mutual information-based selection may not retain the most informative features for this task.

In comparison with Threshold-Based Models: Threshold = 0.001 (with 1596 features) performed better than both k = 1000 and k = 2000. This suggests that low-variance filtering (threshold = 0.001) is more effective than mutual information-based selection for this task. Threshold = 0.005 (with 490 features) performed better than k = 1000 but worse than k = 2000. This indicates that even with fewer features, threshold-based filtering can outperform mutual information-based selection.

In conclusion:
Higher k values (e.g., k = 2000) improve performance compared to lower k values (e.g., k = 1000). However, mutual information-based selection performs worse than threshold-based filtering for this task. The best-performing model is the one with threshold = 0.001, which achieves the highest Micro F1 (0.5603).

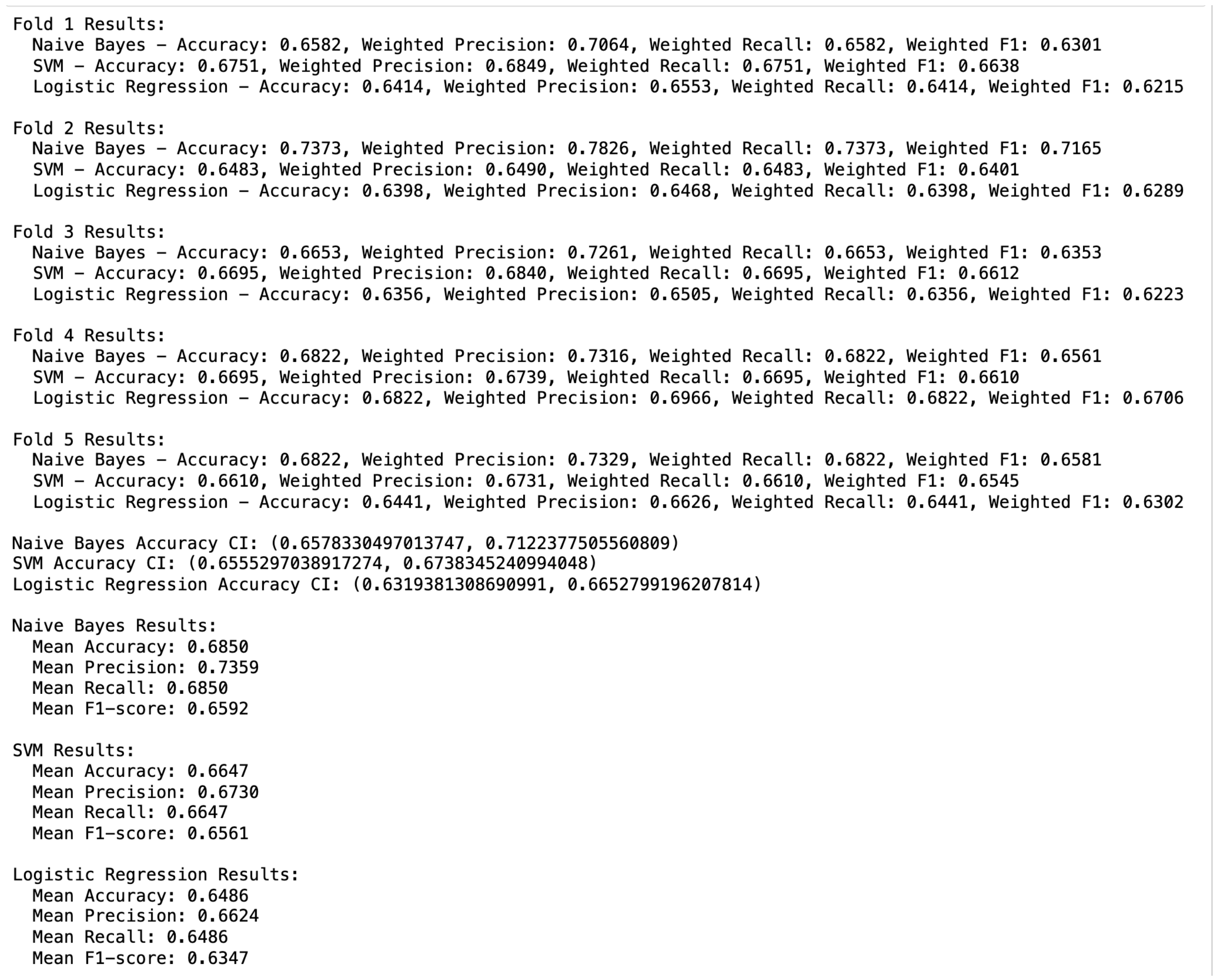
Task 4

The lexicon-based model uses only 678 features, which is significantly fewer than the baseline (5038 features) and even fewer than the threshold 0.001 model (1596 features). Despite using fewer features, the lexicon-based model decently performed as compared to all other models, demonstrating that lexicon-based features are highly informative for this classification task.
The model performs well on the neutral class (highest F1-score of 0.6419) but struggles with the positive class (lowest F1-score of 0.3881). This is likely due to the imbalanced nature of the dataset, where the neutral class has the most support (449 instances), while the positive class has the least (228 instances).

Task 5
Every feature selection technique has advantages and disadvantages. All characteristics are used in the baseline method, which preserves all information but runs the risk of overfitting, high computing costs, and decreased interpretability because of its high dimensionality. By eliminating uninformative features, low-variance filtering lowers dimensionality and improves efficiency and interpretability; a threshold of 0.001 works well (Micro F1 = 0.5603). A higher criterion, such as 0.005, however, may remove too many features, which would impair performance, and it ignores how relevant a feature is to the goal. Although mutual information may enhance performance by choosing features according to their significance to the goal, it performs worse than threshold-based and lexicon-based approaches. This was unexpected for k = 2000 but expected for a low k value of 1000. It might also overlook intricate feature-target interactions. Lexicon-based selection is notable because it uses domain-specific information to achieve the best performance (Micro F1 = 0.5656) with just 678 features. It does, however, rely on a carefully selected lexicon, which isn't always accessible, and it might miss uncommon or domain-specific phrases that aren't included.

Combining Feature Selection Strategies to Improve Performance:
a. Sequentially apply feature selection methods. For example: First, use low variance filtering to remove uninformative features. Then, apply mutual information to select the most relevant features from the reduced set. Finally, add lexicon-based features to capture domain-specific information.
b. Hybrid Approach: Combine lexicon-based features with low-variance filtering or mutual information-based features. For example: First, use lexicon-based features to capture domain-specific information. Then, apply low-variance filtering or mutual information to further refine the feature set. This approach may not be feasible in this example as the no. of lexicon-based features were very less.

Task 7,8, 9

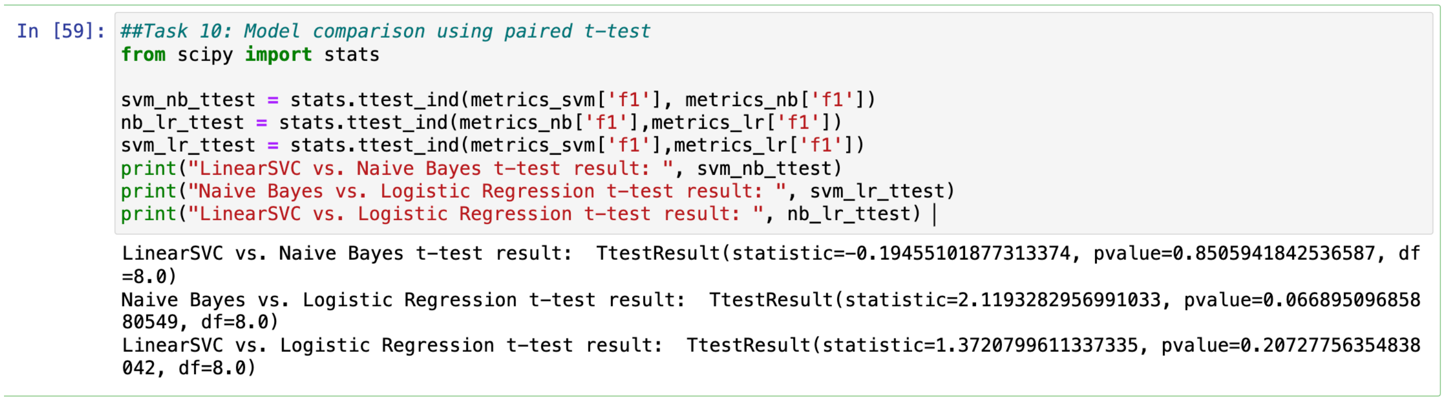


```
Fold 1 Results:
  Naive Bayes - Accuracy: 0.6582, Weighted Precision: 0.7064, Weighted Recall: 0.6582, Weighted F1: 0.6301
  SVM - Accuracy: 0.6751, Weighted Precision: 0.6849, Weighted Recall: 0.6751, Weighted F1: 0.6638
  Logistic Regression - Accuracy: 0.6414, Weighted Precision: 0.6553, Weighted Recall: 0.6414, Weighted F1: 0.6215

Fold 2 Results:
  Naive Bayes - Accuracy: 0.7373, Weighted Precision: 0.7826, Weighted Recall: 0.7373, Weighted F1: 0.7165
  SVM - Accuracy: 0.6483, Weighted Precision: 0.6490, Weighted Recall: 0.6483, Weighted F1: 0.6401
  Logistic Regression - Accuracy: 0.6398, Weighted Precision: 0.6468, Weighted Recall: 0.6398, Weighted F1: 0.6289

Fold 3 Results:
  Naive Bayes - Accuracy: 0.6653, Weighted Precision: 0.7261, Weighted Recall: 0.6653, Weighted F1: 0.6353
  SVM - Accuracy: 0.6695, Weighted Precision: 0.6840, Weighted Recall: 0.6695, Weighted F1: 0.6612
  Logistic Regression - Accuracy: 0.6356, Weighted Precision: 0.6505, Weighted Recall: 0.6356, Weighted F1: 0.6223

Fold 4 Results:
  Naive Bayes - Accuracy: 0.6822, Weighted Precision: 0.7316, Weighted Recall: 0.6822, Weighted F1: 0.6561
  SVM - Accuracy: 0.6695, Weighted Precision: 0.6739, Weighted Recall: 0.6695, Weighted F1: 0.6610
  Logistic Regression - Accuracy: 0.6822, Weighted Precision: 0.6966, Weighted Recall: 0.6822, Weighted F1: 0.6706

Fold 5 Results:
  Naive Bayes - Accuracy: 0.6822, Weighted Precision: 0.7329, Weighted Recall: 0.6822, Weighted F1: 0.6581
  SVM - Accuracy: 0.6610, Weighted Precision: 0.6731, Weighted Recall: 0.6610, Weighted F1: 0.6545
  Logistic Regression - Accuracy: 0.6441, Weighted Precision: 0.6626, Weighted Recall: 0.6441, Weighted F1: 0.6302

Naive Bayes Accuracy CI: (0.6578330497013747, 0.7122377505560809)
SVM Accuracy CI: (0.6555297038917274, 0.6738345240994048)
Logistic Regression Accuracy CI: (0.6319381308690991, 0.6652799196207814)

Naive Bayes Results:
  Mean Accuracy: 0.6850
  Mean Precision: 0.7359
  Mean Recall: 0.6850
  Mean F1-score: 0.6592

SVM Results:
  Mean Accuracy: 0.6647
  Mean Precision: 0.6730
  Mean Recall: 0.6647
  Mean F1-score: 0.6561

Logistic Regression Results:
  Mean Accuracy: 0.6486
  Mean Precision: 0.6624
  Mean Recall: 0.6486
  Mean F1-score: 0.6347
```

1. **Accuracy Confidence Interval**:
   Naive Bayes: (0.6578, 0.7122)
   SVM (LinearSVC): (0.6555, 0.6738)
   Logistic Regression: (0.6319, 0.6653)
2. **Interpretation of what the accuracy confidence interval indicates**:
   The confidence interval (CI) provides a range within which the true accuracy of the model is expected to lie, with a certain level of confidence (95% in this assignment). A narrower CI indicates more consistent performance across folds, while a wider CI suggests greater variability.
   With the broadest CI, Naive Bayes exhibited greater performance variability between folds. But out of the three models, its mean accuracy was the highest. Although SVM's mean accuracy is marginally lower than Naive Bayes', it's very narrow CI indicates consistent performance. Of the three models, logistic regression has the lowest mean accuracy but the narrowest confidence interval (CI), suggesting steady performance.

3. **Which of the trained models from cross-validation would you evaluate on the test set and why?**
   Naive Bayes is the best-performing model based on cross-validation results: It has the highest mean accuracy (0.6850) and highest mean F1-score (0.6592). Despite having a wider confidence interval, its overall performance is superior to SVM and Logistic Regression. SVM is a close second, with consistent performance (narrow CI) and slightly lower mean accuracy. Logistic Regression performs the worst in terms of mean accuracy and F1-score, though it has the most stable performance (narrowest CI). Due to it's highest mean accuracy and F1-score, indicating better overall performance I'll evaluate Naive Bayes on the test set.

Task 10



```
In [59]: ##Task 10: Model comparison using paired t-test
         from scipy import stats

         svm_nb_ttest = stats.ttest_ind(metrics_svm['f1'], metrics_nb['f1'])
         nb_lr_ttest = stats.ttest_ind(metrics_nb['f1'],metrics_lr['f1'])
         svm_lr_ttest = stats.ttest_ind(metrics_svm['f1'],metrics_lr['f1'])
         print("LinearSVC vs. Naive Bayes t-test result: ", svm_nb_ttest)
         print("Naive Bayes vs. Logistic Regression t-test result: ", svm_lr_ttest)
         print("LinearSVC vs. Logistic Regression t-test result: ", nb_lr_ttest)
```

```
LinearSVC vs. Naive Bayes t-test result:  TtestResult(statistic=-0.19455101877313374, pvalue=0.8505941842536587, df
=8.0)
Naive Bayes vs. Logistic Regression t-test result:  TtestResult(statistic=2.1193282956991033, pvalue=0.066895096858
80549, df=8.0)
LinearSVC vs. Logistic Regression t-test result:  TtestResult(statistic=1.3720799611337335, pvalue=0.20727756354838
042, df=8.0)
```

Results

Alternative Hypotheses:

LinearSVC vs. Naive Bayes: LinearSVC performs worse than Naive Bayes (LinearSVC < Naive Bayes).
LinearSVC vs. Logistic Regression: LinearSVC performs worse than Logistic Regression (LinearSVC < Logistic Regression).
Naive Bayes vs. Logistic Regression: Naive Bayes performs better than Logistic Regression (Naive Bayes > Logistic Regression).

LinearSVC vs. Naive Bayes: The p-value (0.8506) is much greater than the significance level (alpha = 0.05).
Conclusion: We fail to reject the null hypothesis. There is no significant difference in performance between LinearSVC and Naive Bayes.

LinearSVC vs. Logistic Regression: The p-value (0.2073) is greater than the significance level (alpha = 0.05).
Conclusion: We fail to reject the null hypothesis. There is no significant difference in performance between LinearSVC and Logistic Regression.

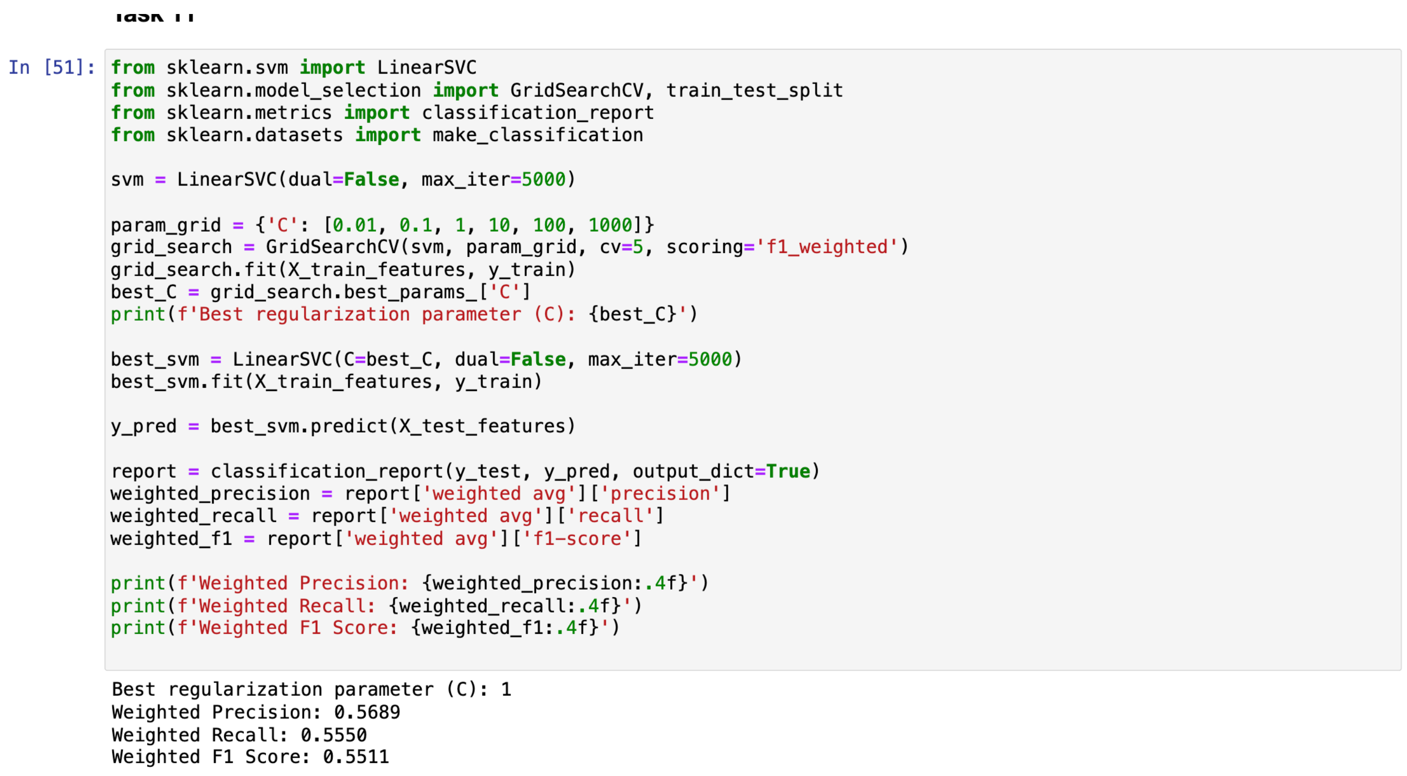
Naive Bayes vs. Logistic Regression: The p-value (0.0669) is slightly greater than the significance level (alpha = 0.05).
Conclusion: We fail to reject the null hypothesis. There is no significant difference in performance between Naive Bayes and Logistic Regression.

Overall Model Performance: According to the t-test results, no model is noticeably better than the others. Even if the difference is not statistically significant, Naive Bayes may perform somewhat better in practice because it has the greatest mean F1-score (as demonstrated by previous results). There is no discernible difference in the performance of LinearSVC and Logistic Regression.

Evidence Supporting Interpretation:

The p-values for all comparisons are greater than alpha = 0.05, indicating that the observed differences in performance are not statistically significant. The t-statistics are relatively small, further supports the lack of significant differences between the models.

Task 11



```
In [51]: from sklearn.svm import LinearSVC
         from sklearn.model_selection import GridSearchCV, train_test_split
         from sklearn.metrics import classification_report
         from sklearn.datasets import make_classification

         svm = LinearSVC(dual=False, max_iter=5000)

         param_grid = {'C': [0.01, 0.1, 1, 10, 100, 1000]}
         grid_search = GridSearchCV(svm, param_grid, cv=5, scoring='f1_weighted')
         grid_search.fit(X_train_features, y_train)
         best_C = grid_search.best_params_['C']
         print(f'Best regularization parameter (C): {best_C}')

         best_svm = LinearSVC(C=best_C, dual=False, max_iter=5000)
         best_svm.fit(X_train_features, y_train)

         y_pred = best_svm.predict(X_test_features)

         report = classification_report(y_test, y_pred, output_dict=True)
         weighted_precision = report['weighted avg']['precision']
         weighted_recall = report['weighted avg']['recall']
         weighted_f1 = report['weighted avg']['f1-score']

         print(f'Weighted Precision: {weighted_precision:.4f}')
         print(f'Weighted Recall: {weighted_recall:.4f}')
         print(f'Weighted F1 Score: {weighted_f1:.4f}')

         Best regularization parameter (C): 1
         Weighted Precision: 0.5689
         Weighted Recall: 0.5550
         Weighted F1 Score: 0.5511
```

Best regularization parameter (C): 1
Weighted Precision: 0.5689
Weighted Recall: 0.5550
Weighted F1 Score: 0.5511

The regularization parameter C balances the trade-off between achieving low training error and maintaining a simple model to prevent overfitting. A smaller C (e.g., 0.01 or 0.1) increases regularization, which can lead to underfitting by penalizing the model too heavily and limiting its ability to capture data patterns. On the other hand, a larger C (e.g., 10, 100, or 1000) reduces regularization, risking overfitting as the model may fit the training data too closely, capturing noise instead of generalizable trends. It seems C = 1 struck a good balance, allowing the model to fit the training data well while generalizing effectively to unseen data (test set).

Task 12:

As discussed in last week's class as well, building strong and dependable text classification models requires the use of model selection and comparison strategies including t-tests, cross-validation, and hyperparameter tuning.

By ensuring that the model performs consistently across several data subsets, cross-validation lowers the possibility of overfitting. By fine-tuning the model to balance bias and variance, hyperparameter tuning—such as determining the ideal regularization parameter C—improves generalization to unknown data. To verify that observed performance differences are substantial and not the result of chance, paired t-tests offer statistical support for model comparisons. When combined, these methods make it possible to choose the model that performs the best and guarantee its dependability in practical applications.