Final Report:

This project focuses on binary classification, multiclass classification, and clustering of Amazon product reviews. The dataset includes text reviews and categorical product labels. The main goals were:

- Binary Classification: Predict whether a review belongs to a positive or negative sentiment based on cutoff level
- Multiclass Classification: Predict the product rating on a five-class scale (1-5).
- Clustering: Group reviews into meaningful product category clusters.

I built three different types of classifiers: Logistic Regression (LR), Support Vector Machines (SVM), and Naïve Bayes (NB) using three different vectorization methods:

- TF-IDF (Term Frequency-Inverse Document Frequency)
- HashingVectorizer
- CountVectorizer

I found that, with the exception of cutoff levels 1 and 4 which scored best with HashingVectorizer, TF-IDF was the best feature extraction method for my model in terms of accuracy scores. This is likely because it weighs words based on their importance in distinguishing between different classes, so frequent but uninformative words (such as "the") are down weighted, while rare but meaningful words (such as "refund") receive higher weights. In the context of product review classification, this allows the model to focus on opinionated words that define the sentiment of the review. Each model was tuned using GridSearchCV with 5-fold cross-validation, which selects the best combination of hyperparameters iteratively. Additionally, I implemented SelectKBest to select the k most important features from some models, which increased accuracy by a few points for both the validation and test sets (although selecting too low of a k value caused overfitting). I also performed feature scaling on some LR and SVM models using MaxAbsScaler so that feature values were standardized, finding that this increased accuracy by a few points. The selection of the final models were based on Macro F-1, accuracy, and ROC-AUC scores.

Binary Classification

Best Models & Hyperparameters for Binary Classification

- Cutoff = 1: Logistic Regression (HashingVectorizer), C=10, solver='saga'
    - F1 Macro Score: 0.78
    - Accuracy: .81
    - ROC: .81
- Cutoff = 2: Logistic Regression (TF-IDF) C=0.25, solver='saga'
    - F1 Score: 0.79
    - Accuracy: .79
    - ROC: .79
- Cutoff = 3: Logistic Regression (TF-IDF) C=0.6, solver='saga'
    - F1 = 0.81
    - Accuracy: .83

        - ROC: .83
- Cutoff = 4: Logistic Regression (Hashing) C=5, solver='saga'
    - F1 = 0.80
    - Accuracy: .82
    - ROC: .80

I tested the three feature extraction methods, finding that CountVectorizer was more computationally expensive without providing better accuracy than the other two. TF-IDF and HashingVectorizer provided fairly similar accuracy and Macro F1 scores. For all four cutoff levels, the best model was Logistic Regression (with HashingVectorizer extraction method for cutoff = 1 and cutoff = 4, and tf-idf for cutoff = 2 and cutoff = 3). For all four cutoff levels, the best model was Logistic Regression, with Naïve Bayes performing second best, usually marginally missing the Kaggle targets, and SVM performing the worst. On the test data, my best model had Macro F1 scores of 0.73, 0.78, 0.80, and 0.74 for the four cutoff levels, respectively.

In terms of hyper-parameter tuning for Logistic Regression models, I used the l2 regularizer, as l1 took much more run-time without providing better accuracy. In order to find the optimal C value for each model, I first initialized the param_grid passed into GridSearchCV to contain a range of values (usually .001, .1, 1, 5, 10), then trained the model and reported metrics, and then fine tuned the param_grid to test more tightly bound values to what was first reported as the optimal C value. C is the inverse of the regularization strength, meaning that a higher value leads to less regularization (a more complex model) that fits the data better (although I also had to be careful of overfitting). I found that the best C parameter changed depending on which feature extraction method had been implemented. With the TF-IDF method, the optimal C values were usually in between 0 and 1 (.75 for cutoff=1, .25 for cutoff=2, .6 for cutoff=3, and .4 for cutoff=4), whereas for the HashingVectorizer method the optimal C values were always over 1 (10 for cutoff=1, 4.5 for cutoff=2, 7 for cutoff=3, and 5 for cutoff=4). This is likely because HashingVectorizer doesn't store word frequencies, meaning some features may be weighted smaller due to hash collisions, and a higher C compensates by reducing regularization. TD-IDF, on the other hand, captures word importance, so a lower C helps prevent overfitting. I also found that setting class_weights = "balanced" improved model performance by a few points.

For the SVM models, C controls the soft margin penalty, so a higher C penalizes misclassification more (and thus there is a higher risk of overfitting as C increases). I found that for most models, C was optimal at 1 or 1.5 regardless of feature extraction method. This is likely because a balanced decision boundary avoids overfitting to noise while simultaneously capturing patterns. I used a linear kernel because this tends to work well for high-dimensional text data, and found that other kernels such as rbf performed poorly and significantly increased training time.

For the Naive Bayes models, I tuned the alpha parameter which is a smoothing parameter. A smaller value leads to less smoothing meaning the model relies on word frequencies heavily and there is a greater risk of overfitting. The optimal alpha for models were usually around 0.05.

Multiclass Classification

Best model & hyperparameters for multiclass:

- Logistic Regression (TF-IDF), C = 0.45, penalty= L2, and solver=Saga

        - F1 Score: .48
        - Accuracy: .49
        - ROC: .80

For multiclass classification, I extended the models to predict ratings from 1-5. The best-performing model was Logistic Regression with TF-IDF. Naive Bayes also performed well (F1 score .46), followed by SVM (F1 score of .45). The confusion matrix reveals some misclassification, with some overlap between classes, likely due to class imbalances or overlapping feature distributions. The ROC curve indicates that while some classes achieve strong separability, others struggle, particularly in the mid-range FPR.

Compared to binary classification with different cutoff levels, multinomial classification performed worse. This suggests that certain class distinctions are more effectively captured through binary approaches rather than a direct multinomial framework. The drop in performance may be due to high class overlap in the feature space, increased model complexity leading to difficulties in convergence, and potential feature sparsity issues with TF-IDF when applied to a multiclass setup (although TF-IDF models performed better than HashingVectorizer).

Clustering

Best Spectral Model Scores:

- Spectral Clustering: k = 2
    - Silhouette Score = 0.55

Best K-Means Model Scores:

- With Dimensionality Reduction:
    - k = 2
        - Silhouette Score = 0.64
- Without Dimensionality Reduction:
    - k = 9
        - Silhouette Score = .01, Rand score = .72

Clustering was performed on product reviews to group them based on text similarity. The primary goal was to form clusters that aligned with product categories. Performance was measured using silhouette score to measure how well-separated the clusters are as well as rand score to measure how well the clustering matches the true product categories. I implemented two clustering methods, as I wanted to explore how they handled this dataset differently:

- Spectral Clustering (graph-based approach)
    - Spectral Clustering is known to handle high-dimensional text data better than K-Means.
    - I used TF-IDF feature extraction and SVD (singular value decomposition) for improved separation.
    - The best performance was k = 2, achieving a Silhouette Score of 0.55, suggesting moderate separation but not highly distinct clusters.
- K-Means (centroid-based clustering)

        - I used TF-IDF vectorization with bigram inclusion (ngram_range=(1,2)) to capture contextual meaning.
        - I tested different values of k (from 2 to 10) to find the optimal cluster count, and evaluated each k using silhouette score and rand score.

To explore how dimensionality effects clustering, I applied Truncated SVD (a type of Latent Semantic Analysis (LSA)) to reduce feature dimensionality, revealing a trade-off between silhouette score and rand score. Higher dimensionality (without SVD) produced very low silhouette scores (0.01), which suggests poor separation between clusters, meaning that reviews in different clusters are not well-distinguished. However, the high rand score (0.72) indicates that the clusters still aligned well with actual product categories. Reduced dimensionality (SVD) improved silhouette scores but significantly lowered rand scores, suggesting clearer cluster separation but less category alignment. K-Means without SVD is the best model in the practical sense due to its better real-world category alignment. The results indicate that text alone is somewhat effective for clustering products, but further improvements could be made in the future by including additional features (such as rating, verification, image, etc) to improve cluster alignment.

Comparison:

- K-Means outperformed Spectral Clustering in terms of Silhouette Score
- Spectral Clustering resulted in less distinct clusters (Silhouette Score = 0.55), whereas K-Means provided better separation.

Future Improvements

To expand the scope of the project in the future, I am interested in exploring deep learning models. While traditional machine learning models like Logistic Regression, SVM, and Naïve Bayes performed well, deep learning approaches could further enhance classification by capturing semantic relationships in text. Long Short-Term Memory (a type of RNN) networks are effective for text classification as they capture long-range dependencies, meaning they could handle context-dependent word meaning. CNNs are also known to perform well on text, extracting local n-gram patterns instead of treating text as sequential data. This could be particularly useful for binary sentiment analysis with phrase-based shifts being crucial. I am also interested in alternative vectorization techniques,as TF-IDF still has limitations such as ignoring word order and context. Additionally, incorporating additional metadata (such as review length, helpfulness scores, and verification) could improve cluster separation and alignment with real-world product categories.