Analyzing Amazon Text Reviews to Classify Future Review Scores

Introduction/Background

This project develops an algorithm mapping text reviews to a predicted star rating, allowing us to minimize discrepancies between text content and associated star rating in Amazon product reviews, false positive or negative. In looking into book reviews to analyze sentiment, researchers explored various preprocessing techniques as well as extraction using TF-IDF, using KNN, Decision Trees, etc. for classification (Medhat et al., 2017). Additionally, another paper explores supervised learning to separate product reviews into positive/negative categories, with its proposed model achieving 90% accuracy (Rana et al., 2018). A third paper concludes SVM outperforms other models Naive Bayes and Decision Trees in accurately classifying product reviews (Wassan et al., 2021). Our project utilizes the UCSD Dataset on Amazon Reviews, which provides user review data such as rating, title, text., for over 500 million reviews from which we will sample data from.

Link to UCSD Dataset link

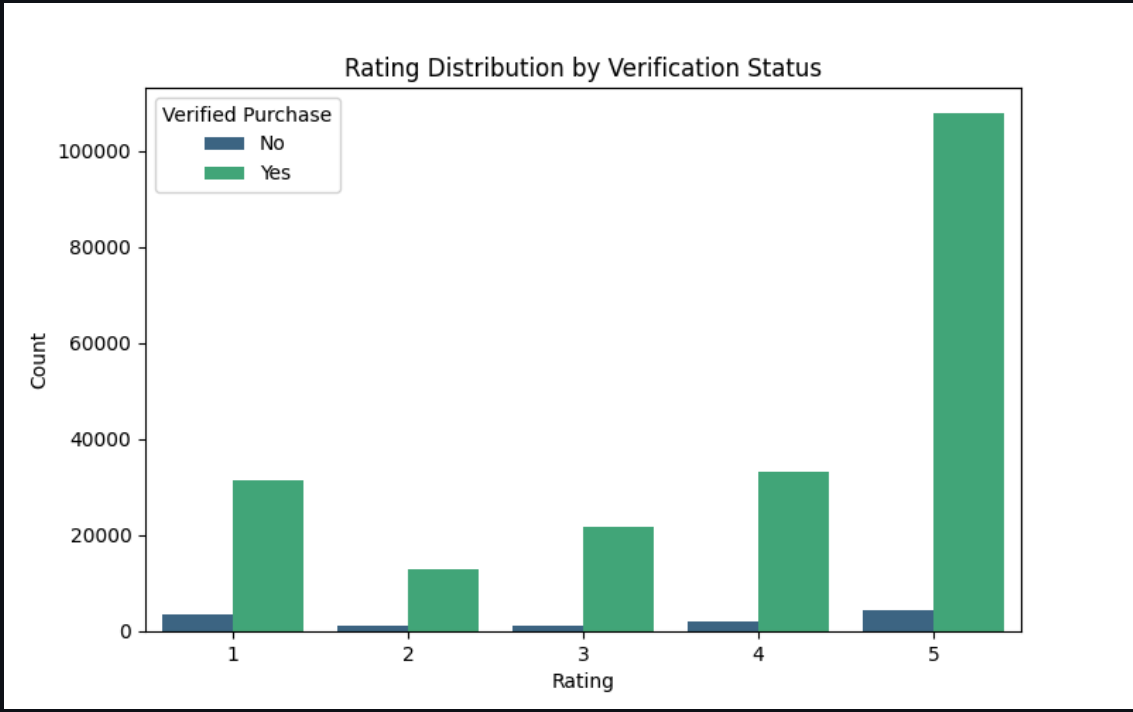
Problem Definition

Amazon product reviews are a deciding factor for purchases, but the presence of individual biases in reviews, specifically accounting for the difference in perception between star rating and text review, make it hard for customers to truly assess product quality. To know the quality of a product, one would have to individually read through many reviews to make a decision, rather than rely on an easy glance at a star rating. We aim to analyze the correlation between star rating and text reviews in order to develop an algorithm to represent a more objective connection between text review and star ratings and develop a system to assign a star rating to any review objectively based on text response.

Dataset:

For this project, we are not going to be able to use the entire Amazon Database link as provided above as that is just too much to handle on a single device and for the calculations we would have to perform. As such, we instead will use a specific dataset from the link, the 'Software' reviews. Here are some visualizations of the results of the database as they would be utilized for our three models.

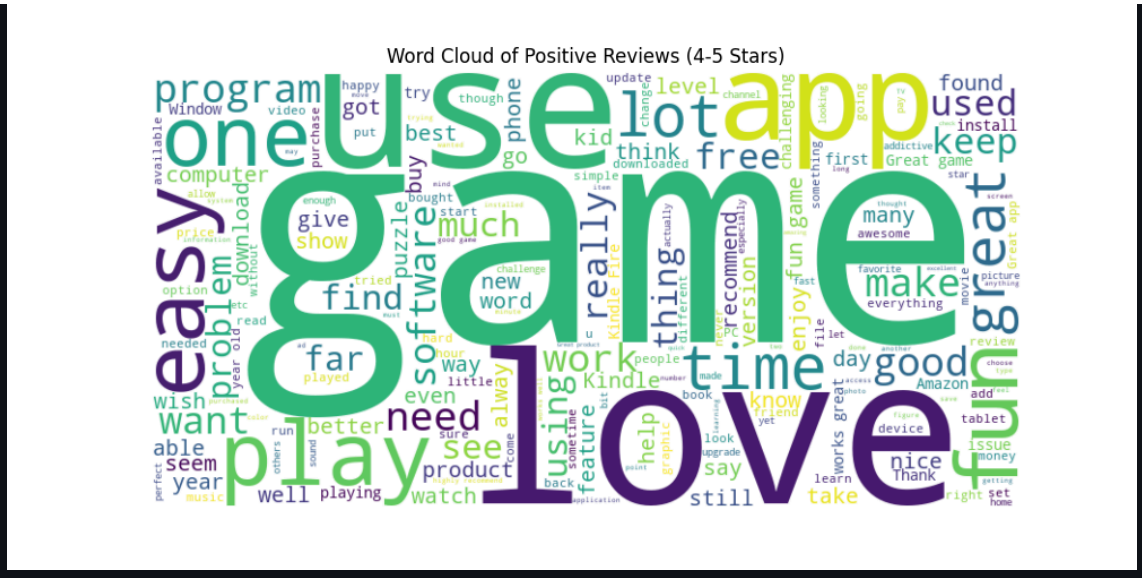
Data Visualization 1 : Rating Distribution



Rating Distribution

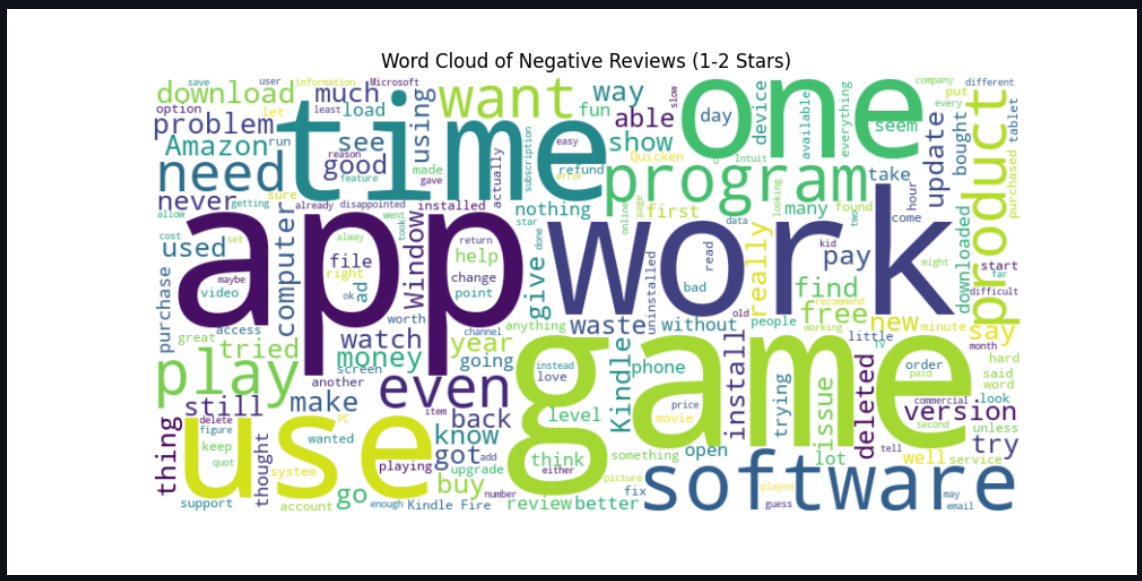
This bar chart shows how the product ratings are distributed between 1-5 stars. Ratings are broken down by whether or not the purchase was verified. Verified purchases are more frequent, especially when it comes to 5-star ratings. This can imply that verified customers leave higher ratings.

Data Visualization 2 : Word Cloud



Positive Word Cloud

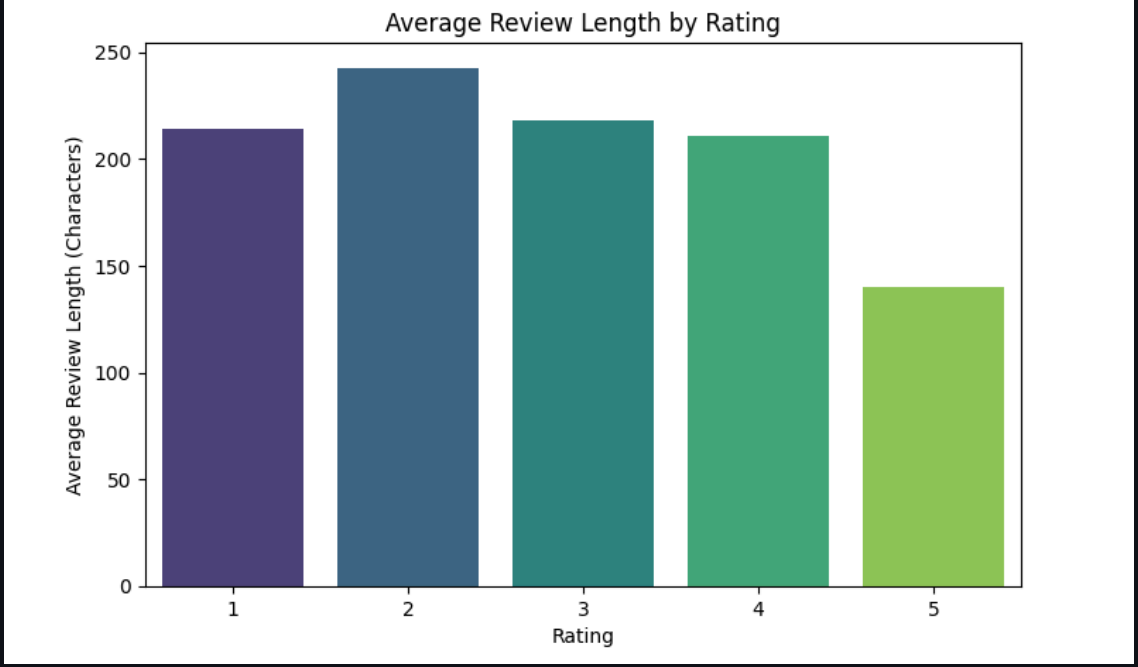
This word cloud shows the most common words in positive 4-5 star reviews. The words that are common are words like love, easy, and game.



Negative Word Cloud

This word cloud shows the most common words in negative 1-2 star reviews. The words that are most common are problem, time, and work.

Data Visualization 3 : Review Length By Rating



Review Length By Rating

This shows the average length of each review based on star rating. 4-star reviews are generally longer while 5-star reviews are a bit shorter. Moderately positive feedback accompanies longer commentary.

Model 1 - Naive Bayes

Data Preprocessing

For data cleaning, reviews that had less than 10 words and all columns besides rating, helpful vote, title, text, and verified purchase were removed. HTML breaks were replaced with empty strings for better formatting.

```python
import pandas as pd
import numpy as np

# Paths
in_path = "data/Software.jsonl"  # Place data here or change the path variable
out_path = "clean_data/software_clean.csv"

# Parameters
testing = False # CHANGE THIS TO TRUE MANUALLY ENTER PARAMETERS
max_entries = 250000
min_text_len = 10
good_columns = ["rating", "helpful_vote", "title", "text", "verified_purchase"]
if testing:
    max_entries = input("Enter the max number of entries to read from the dataset: ")
    min_text_len = input("Enter the min length of a review required: ")

df = pd.read_json(in_path, lines=True, nrows=max_entries)

df.dropna(subset=good_columns, inplace=True)

mask = df['text'].str.len() > min_text_len
df = df[mask]

mask = df['helpful_vote'] >= np.floor(df['helpful_vote'].median())
df = df[mask]

df['text'] = df['text'].str.replace('<br />', '')

df = df[good_columns]

df.to_csv(out_path, index=False)
```

clean_data.py

For feature reduction, PCA was used with n_components=0.1 to ensure that 10% of the variance was kept from the original data to prevent overfitting, as many of the words from the reviews are noisy.

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
from sklearn.decomposition import PCA, TruncatedSVD
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from scipy.sparse import csr_matrix



#the first part is the same as the naive_baiyes.py (this is needed to convert the words to vectors)
data = pd.read_csv('data/software_clean.csv')
data['combined_text'] = data['title'].fillna('') + ' ' + data['text']
X = data['combined_text']
y = data['rating']

print("this is pre-vectorized:", data[:2])
vectorizer = TfidfVectorizer(max_features=5000)  # Limit to top 5000 words
X_tfidf = vectorizer.fit_transform(X)
print("prePCA", X_tfidf.shape)
print("this is what prePCA looks like", X_tfidf[:2])

#doing reduction to 100 features-> originally got error about using sparse input, had to convert to array but not great solution
pca = PCA(n_components= 0.1, svd_solver='full')
pca.fit(X_tfidf.toarray())
newX = pca.transform(X_tfidf.toarray())



# got an error running here becaus
# e "Negative values in data passed to MultinomialNB" but PCA can get negative values so tried
# fixing using min max scalar
scalar = MinMaxScaler()
scalar.fit(newX)
newX = scalar.transform(newX)
# print("first two lines of before csr x", newX[:2])

# newX = csr_matrix(newX)
# print("first two lines of after csr x", newX[:2])


# added the columns of newX first, then the rating, converted to csv
# newfile = pd.DataFrame.sparse.from_spmatrix(newX)
newfile = pd.DataFrame(newX)
# newfile['reduced_text'] = newX
newfile['rating'] = y
newfile.to_csv('data/software_reduction.csv',index=False)

print("this is newfile", newfile.head(10))
# print("shape of pca", newX.shape)
```

feature_reduction.py

Naive Bayes Algorithm

Naive Bayes works very well with high-dimensional data, which is typical of text reviews transformed into feature vectors. When we assume conditional independence between features, this algorithm is able to handle thousands of features extracted from texts efficiently. Naive Bayes is also fast and tends to yield good results for text classification tasks like sentimental analysis.

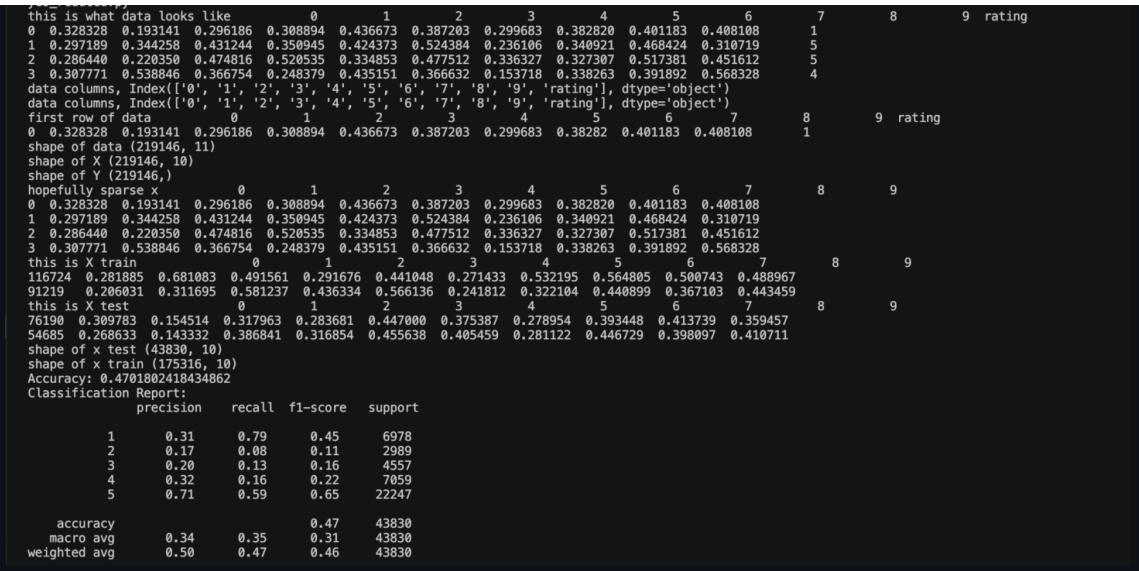
Gaussian Naive Bayes assumes a normal distribution of features and can perform reasonably well on various data types, including real-valued features, making it suitable for our project.

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from scipy.sparse import csr_matrix


# Load the data
data = pd.read_csv('data/software_reduction.csv')
print("this is what data looks like", data[:4])
print("data columns,", data.columns)
# if 'rating' not in data.columns:
#     data = data.rename(columns={data.columns[0]:'rating'})
print("data columns,", data.columns)
print("first row of data", data[:1])

# # Combine title and text for the Naive Bayes model
# data['combined_text'] = data['title'].fillna('') + ' ' + data['text']

# Split the data into features and target variable
# X = data['combined_text']
# y = data['rating']
# added the reduced vector columns of combined_text first, then rating is the last column
print("shape of data", data.shape)

X = data.iloc[:,:-1]
print("shape of X", X.shape)
y = data['rating']
print("shape of Y", y.shape)

#X = csr_matrix(X)
print("hopefully sparse x", X[:4])

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

X_train = X_train.iloc[:,:-1]
newfile = pd.DataFrame(X_train)
newfile['rating'] = y
newfile.to_csv('data/software_reduction2.csv',index=False)

# # Convert text to TF-IDF features
# Don't need bc this is taken care of by feature reduction -> already vectorized
# vectorizer = TfidfVectorizer(max_features=5000)  # Limit to top 5000 words
# X_train_tfidf = vectorizer.fit_transform(X_train)
# X_test_tfidf = vectorizer.transform(X_test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the Naive Bayes classifier

print("this is X train", X_train[:2])
print("this is X test", X_test[:2])
print("shape of x test", X_test.shape)
print("shape of x train", X_train.shape)



nb_classifier = GaussianNB() #max_iter=1000
nb_classifier.fit(X_train, y_train)
# nb_classifier.fit(X_train_tfidf, y_train)      can directly use X_train

# Predict and evaluate
# y_pred = nb_classifier.predict(X_test_tfidf)        can directly use x_test

y_pred = nb_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print("Classification Report:\n", report)
```

naive_bayes_reduced.py

Result & Discussion

Result of Naive Bayes Calculation

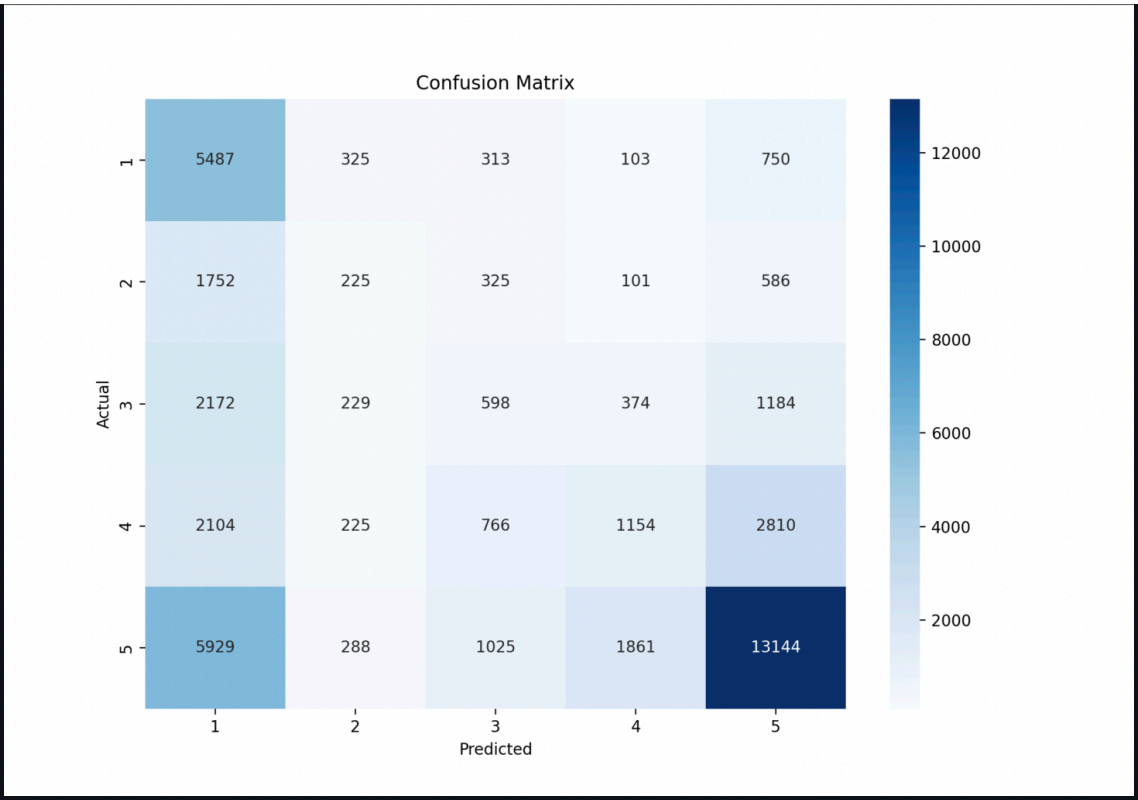


```
this is what data looks like          0         1         2         3         4         5         6         7         8         9  rating
0  0.328328  0.193141  0.296186  0.308894  0.436673  0.387203  0.299683  0.382820  0.401183  0.408108       1
1  0.297189  0.344258  0.431244  0.350945  0.424373  0.524384  0.236106  0.340921  0.468424  0.310719       5
2  0.286440  0.220350  0.474816  0.520535  0.334853  0.477512  0.336327  0.327307  0.517381  0.451612       5
3  0.307771  0.538846  0.366754  0.248379  0.435151  0.366632  0.153718  0.338263  0.391892  0.568328       4
data columns, Index(['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'rating'], dtype='object')
data columns, Index(['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'rating'], dtype='object')
first row of data          0         1         2         3         4         5         6         7         8         9  rating
0  0.328328  0.193141  0.296186  0.308894  0.436673  0.387203  0.299683  0.38282  0.401183  0.408108       1
shape of data (219146, 11)
shape of X (219146, 10)
shape of Y (219146,)
hopefully sparse x          0         1         2         3         4         5         6         7         8         9
0  0.328328  0.193141  0.296186  0.308894  0.436673  0.387203  0.299683  0.382820  0.401183  0.408108
1  0.297189  0.344258  0.431244  0.350945  0.424373  0.524384  0.236106  0.340921  0.468424  0.310719
2  0.286440  0.220350  0.474816  0.520535  0.334853  0.477512  0.336327  0.327307  0.517381  0.451612
3  0.307771  0.538846  0.366754  0.248379  0.435151  0.366632  0.153718  0.338263  0.391892  0.568328
this is X train                0         1         2         3         4         5         6         7         8         9
116724  0.281885  0.681083  0.491561  0.291676  0.441048  0.271433  0.532195  0.564805  0.500743  0.488967
91219   0.206031  0.311695  0.581237  0.436334  0.566136  0.241812  0.322104  0.440899  0.367103  0.443459
this is X test                0         1         2         3         4         5         6         7         8         9
76190  0.309783  0.154514  0.317963  0.283681  0.447000  0.375387  0.278954  0.393448  0.413739  0.359457
54685  0.268633  0.143332  0.386841  0.316854  0.455638  0.405459  0.281122  0.446729  0.398097  0.410711
shape of x test (43830, 10)
shape of x train (175316, 10)
Accuracy: 0.4701802418434862
Classification Report:
              precision    recall  f1-score   support

           1       0.31      0.79      0.45      6978
           2       0.17      0.08      0.11      2989
           3       0.20      0.13      0.16      4557
           4       0.32      0.16      0.22      7059
           5       0.71      0.59      0.65     22247

    accuracy                           0.47     43830
   macro avg       0.34      0.35      0.31     43830
weighted avg       0.50      0.47      0.46     43830
```

Calculation using Reduction Model

In using our model for Naive Bayes calculation, this is the generation of the output of the calculation. The model visualizations above give a good indication as to what each value means in the final output. The accuracy of around 47% is definitely low. However, much of that can be justified from the inaccuracy of humans in generating objective rating star responses for their reviews, often going to extremities of love or hate (5 or 1) for products that might have more in-the-middle star ratings on an objective scale. We hope to improve our model and have much stronger accuracy in future cases and with other model systems.

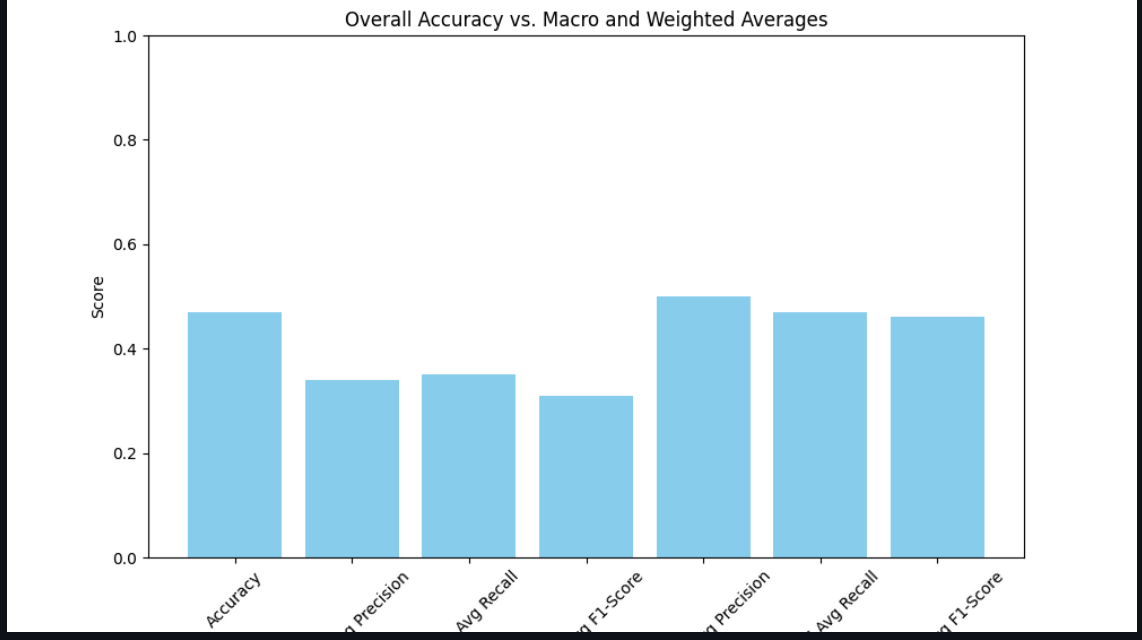
Visual of Naive Bayes Confusion Matrix



Confusion Matrix

The confusion matrix generated from the Naive Bayes function shows the performance of a classification model on the five classes, each class being the star rating system of 1-5, with the actual classes on the vertical axis and predicted classes on the horizontal axis. The model seems to perform best for class 5, with 13,144 correct prediction, as shown from its dark blue box. However, the other classes, especially 3 and 4, are much more muffled in their calcuations with a large number of misclassifications, with class 3 instances being misclassified as class 1, 4, or 5 very often. This could suggest that while the model is good at extremities, such as 1 and 5, that the distinguishing between interior ratings is much more difficult, which could reflect a lot of general user's rating intentions, often going to extremities when rating rather than objective classifications. This would need some potential improvement.

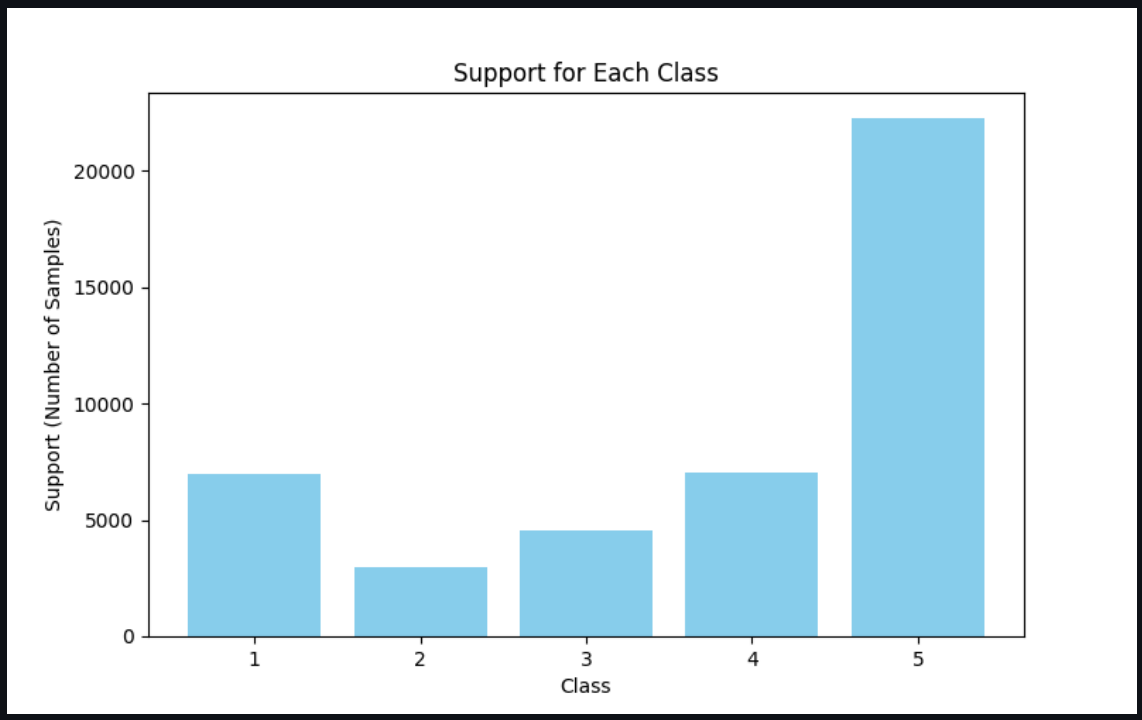
Model Visualization 1: Overall Accuracy vs. Macro and Weighted Averages



Overall vs Macro & Weighted

This chart compares the overall accuracy of the Amazon review model with its macro and weighted average scores for precision, recall, and F1-score. The model achieves slight accuracy, while the macro and weighted averages provide insights into how well it performs across all classes. It can be shown that there is potential improvement in class-level performance.

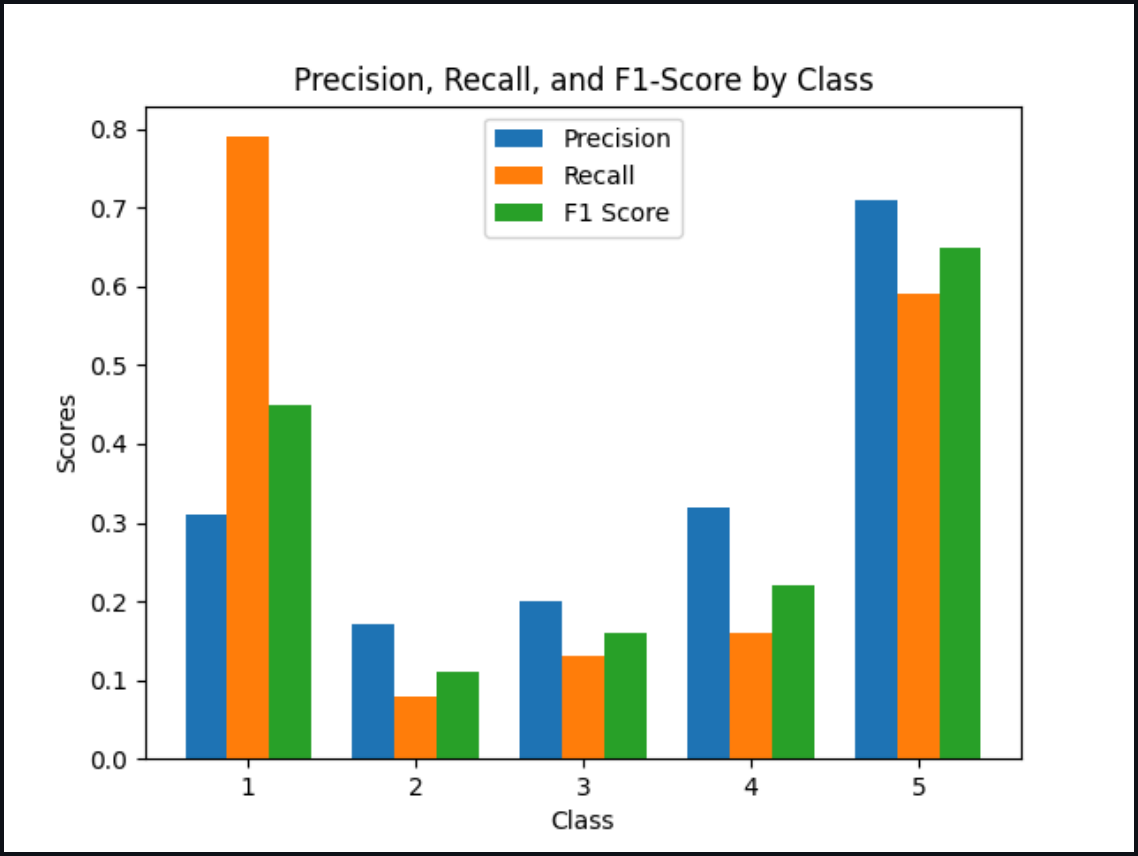
Model Visualization 2: Support For Each Class



Support

This model shows the support (number of samples) for each class in the Amazon review dataset, every class referring to the star rating from 1-5 stars. The distribution shows a significant imbalance, with the majority of the samples falling under 5 stars, while the classes 1-4 have far fewer samples.

Model Visualization 3: Precision, Recall, and F1-Score by Class



Precision, Recall, F1-Scores

This graph shows the precision, recall, and F1-score for each class in the Amazon review model, specifically showing off performance metrics varying significantly across different ratings. The model achieves highest precision, recall, and F1-score for classes 1 and 5, while the in between show lower performance, which can reflect on the challenges of predicting ratings in these classes as shown above.

Final Observations

We observed that increasing the number of features retained by PCA actually lowered the accuracy, likely due to it adding noise to the model. Reducing dimensionality also helped reduce features that were highly specialized and unlikely to be useful to the model in predicting the rating

Our model performed less accurately than what we wanted, but not too poorly. Compared to other datasets we tested on, the software dataset had a significantly lower accuracy, possibly due to the circumstances surrounding online software purchases. Many of the four and three star reviews also use positive language and do not specify exactly what caused the lowered stars. There is also a large class imbalance as a majority of the reviews were 5 stars, around 4 times as many as the number of 2 stars. We noted that the accuracy of the model improved when PCA was limited to fewer features, demonstrating that a majority of the language used in the reviews is “noise” and does not help discern what the star rating is. Software often has descriptions that are unique to the product and therefore may not be helpful in training the model.

The next step is to implement a model that is better suited for natural language applications. We could use a cross-validation method such as k-fold to ensure that the model isn’t overfitting and will perform accurately for all the data. Additionally, we noted that using models such as logistic regression and SVM also resulted in higher accuracy. This could be due to the “independent” assumption made in Naive Bayes as the order of the words in the reviews and the relationships between them matters.

Model 2 - SVM

Data Preprocessing

For data cleaning, in addition to what we had before, we improved the filtering of our data by getting the reviews with a helpfulness votes above the median.

```python
import pandas as pd
import numpy as np

# Paths
in_path = "data/Software.jsonl"  # Place data here or change the path variable
out_path = "clean_data/software_clean.csv"

# Parameters
testing = False # CHANGE THIS TO TRUE MANUALLY ENTER PARAMETERS
max_entries = 250000
min_text_len = 10
good_columns = ["rating", "helpful_vote", "title", "text", "verified_purchase"]
if testing:
    max_entries = input("Enter the max number of entries to read from the dataset: ")
    min_text_len = input("Enter the min length of a review required: ")

df = pd.read_json(in_path, lines=True, nrows=max_entries)

df.dropna(subset=good_columns, inplace=True)

mask = df['text'].str.len() > min_text_len
df = df[mask]

mask = df['helpful_vote'] >= np.floor(df['helpful_vote'].median())
df = df[mask]

df['text'] = df['text'].str.replace('<br />', '')

df = df[good_columns]

df.to_csv(out_path, index=False)
```

clean_data.py

For feature reduction, we utilized a standard scalar since it's important for SVM in particular.

SVM

SVM is useful since it can handle both linear and nonlinear classification tasks. We started with using a Linear SVC, and it ended up working great.

Users > divyaramaswamy > Desktop > ML > ML-Amazon-Reviews-Project-main > svm.py > ...

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from scipy.sparse import csr_matrix
from sklearn import svm

# Load the data
data = pd.read_csv('data/svm_software_reduction.csv')
print("this is what data looks like", data[:4])
print("data columns,", data.columns)
print("data columns,", data.columns)
print("first row of data", data[:1])
# Split the data into features and target variable
# added the reduced vector columns of combined_text first, then rating is the last column
print("shape of data", data.shape)
X = data.iloc[:,:-1]
print("shape of X", X.shape)
y = data['rating']
print("shape of Y", y.shape)

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

# X_train = X_train.iloc[:,:-1]
# newfile = pd.DataFrame(X_train)
# newfile['rating'] = y

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the Naive Bayes classifier

nb_classifier = svm.LinearSVC() #max_iter=1000
nb_classifier.fit(X_train, y_train)
# nb_classifier.fit(X_train_tfidf, y_train)      can directly use X_train

# Predict and evaluate
# y_pred = nb_classifier.predict(X_test_tfidf)         can directly use x_test

y_pred = nb_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print("Classification Report:\n", report)

```

svm_reduced.py

Result & Discussion

Result of SVM Calculation

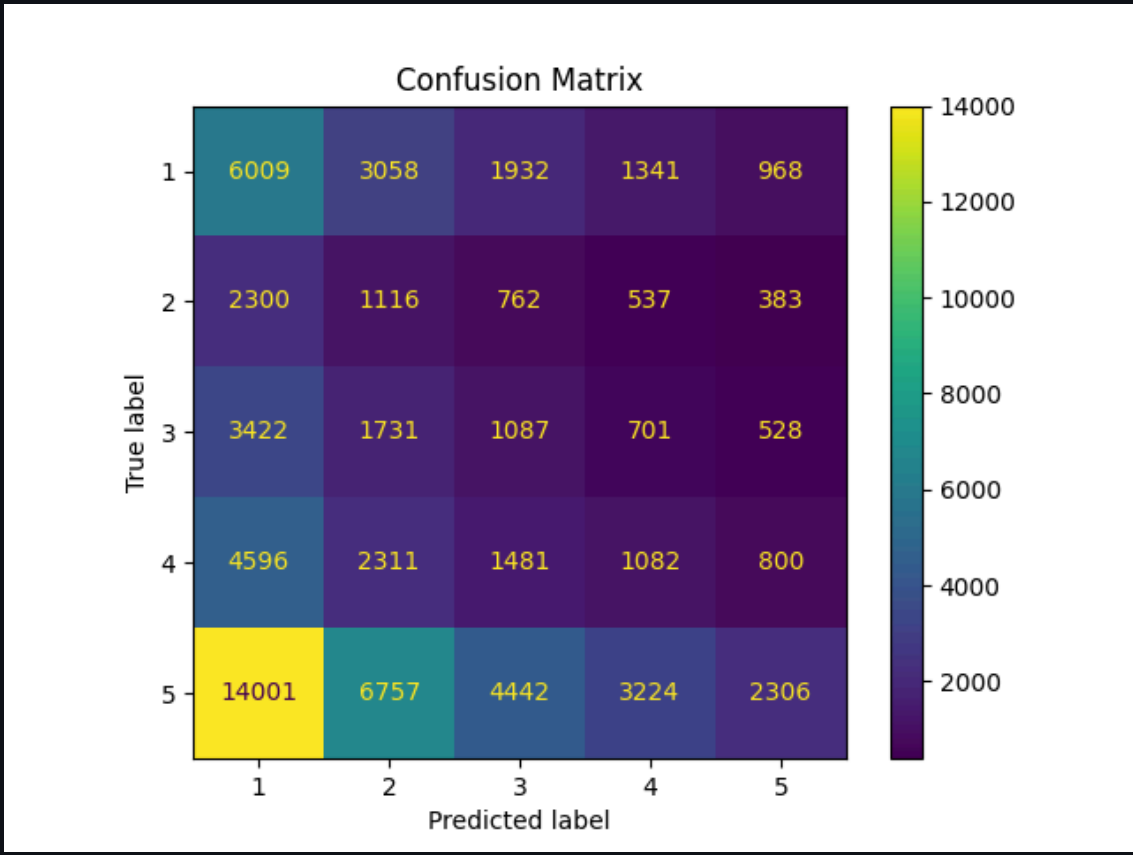
```
Accuracy: 0.6705224731918777
Classification Report:
              precision    recall  f1-score   support

           1       0.59      0.75      0.66      6978
           2       0.72      0.14      0.24      2989
           3       0.60      0.28      0.38      4557
           4       0.58      0.23      0.33      7059
           5       0.71      0.94      0.81     22247

    accuracy                           0.67     43830
   macro avg       0.64      0.47      0.48     43830
weighted avg       0.66      0.67      0.62     43830
```

Calculation using Reduction Model

In using our model for Naive Bayes calculation, this is the generation of the output of the calculation. The model visualizations above give a good indication as to what

each value means in the final output. The accuracy of around 67% is not bad. As mentioned before, this could be because responses are tending towards extremes (1 or 5 stars).

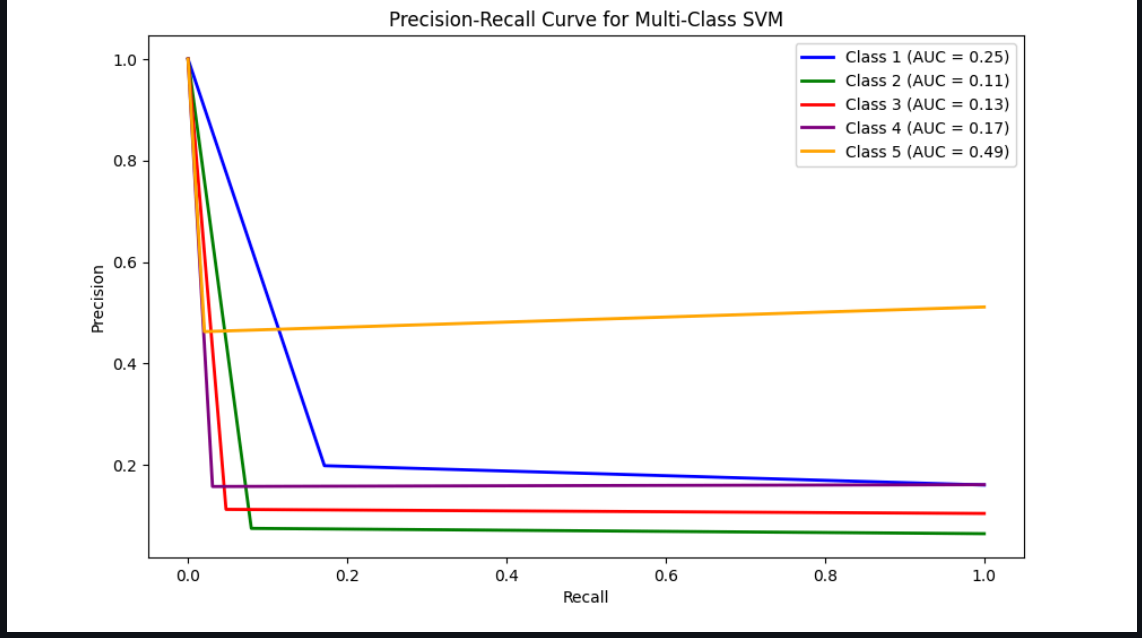
Visual of SVM Confusion Matrix



Confusion Matrix

The confusion matrix generated from the SVM function shows the performance of a classification model on the five classes, each class being the star rating system of 1-5, with the actual classes on the vertical axis and predicted classes on the horizontal axis. Interestingly, the model predicted class 5 reviews as class 1 the most. This may be because there were many 5 star reviews, and the model strongly preferred labeling reviews as class 1 (as seen in the first column.

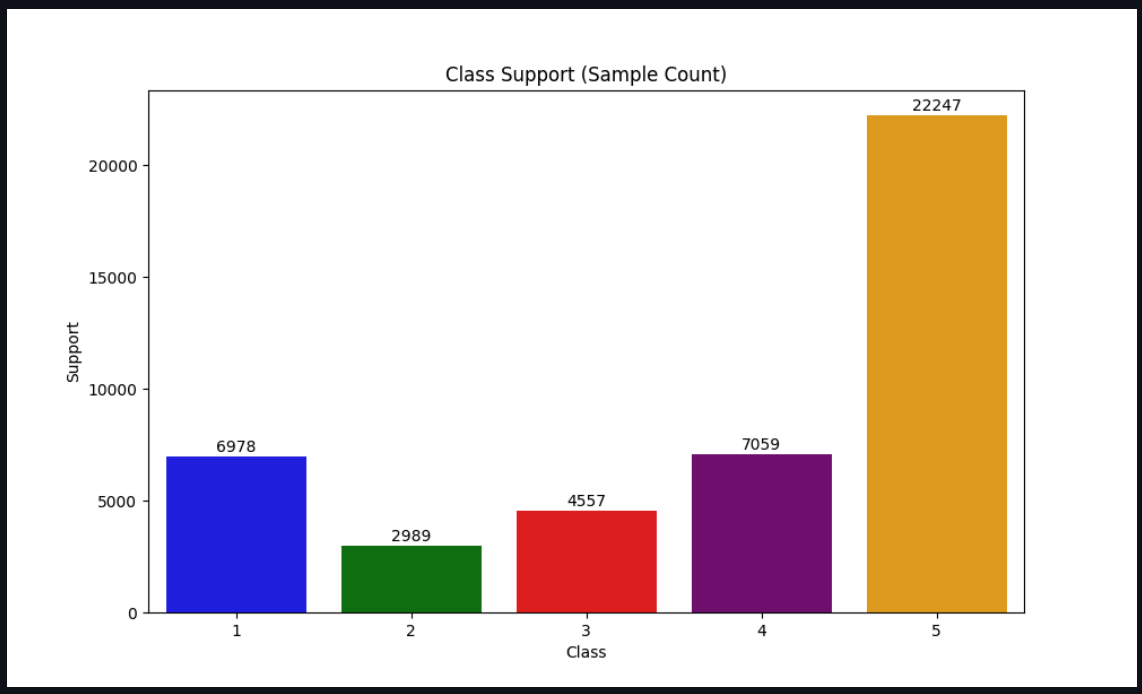
Model Visualization 1: Precision vs Recall for each class



Precision vs Recall per class

This chart compares the precision and recall for each class. We see here that class 5 performed the best, followed by class 4. Classes 1-3 perform very similarly to each other.

Model Visualization 2: Support For Each Class

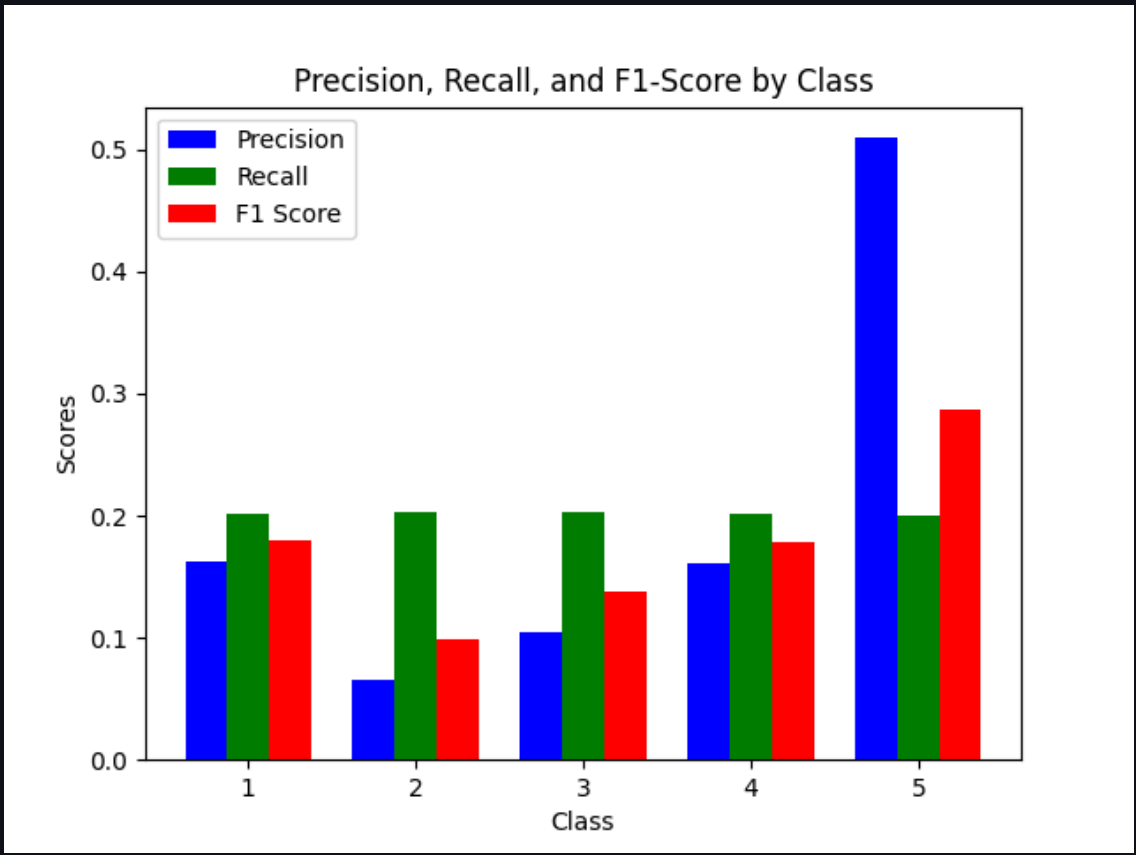


Support

This model shows the support (number of samples) for each class in the Amazon review dataset, every class referring to the star rating from 1-5 stars. The distribution

shows a significant imbalance, with the majority of the samples falling under 5 stars, while the classes 1-4 have far fewer samples.

Model Visualization 3: Precision, Recall, and F1-Score by Class



Precision, Recall, F1-Scores

This graph shows the precision, recall, and F1-score for each class in the Amazon review model, specifically showing off performance metrics varying significantly across different ratings. The model achieves highest precision, recall, and F1-score for class 5. However, we would've expected class 1 to also be high, as we saw in the Naive Bayes visualization.

Final Observations

We observed that increasing the number of features retained by PCA actually lowered the accuracy, likely due to it adding noise to the model. Reducing dimensionality also helped reduce features that were highly specialized and unlikely to be useful to the model in predicting the rating

Our model performed less accurately than what we wanted, but not too poorly. Compared to other datasets we tested on, the software dataset had a significantly lower accuracy, possibly due to the circumstances surrounding online software

purchases. Many of the four and three star reviews also use positive language and do not specify exactly what caused the lowered stars. There is also a large class imbalance as a majority of the reviews were 5 stars, around 4 times as many as the number of 2 stars. We noted that the accuracy of the model improved when PCA was limited to fewer features, demonstrating that a majority of the language used in the reviews is “noise” and does not help discern what the star rating is. Software often has descriptions that are unique to the product and therefore may not be helpful in training the model.

The next step is to implement a model that is better suited for natural language applications. We could use a cross-validation method such as k-fold to ensure that the model isn’t overfitting and will perform accurately for all the data. Additionally, we noted that using models such as logistic regression and SVM also resulted in higher accuracy. This could be due to the “independent” assumption made in Naive Bayes as the order of the words in the reviews and the relationships between them matters.

The SVM model has an accuracy of 67%. This makes sense as multi-class classification is more difficult. The model worked best on 5-star and 1-star reviews, as the F1 scores were 0.81 and 0.66, respectively. This is as expected since extreme reviews generally will have more similar words and phrases associated with them. The SVM model struggled with 2-3 star reviews which is most likely because they are a blend of sentiment.

The model performed better than the Naive Bayes model by a decent margin. Additionally, although class 1 didn't improve much, class 2-4 improved a great deal. This is significant as class 2-4 are very similar, so distinguishing them can be a challenge. One strange occurence was the confusion matrix predicting class 1 as class 5, but this could be resolved with more data or more training.

In the future iterations of the model, we may consider feeding more data to help with the 2-4 star reviews. More data can help create help clear up confusion between them. Additionally, the SVM model can be improved by hyperparameter tuning. Doing so does not require much work, but could improve it a surprising ammount since SVM is one-vs-all structured.

Model 3 - KNN

Data Preprocessing

For KNN, we refined our data preprocessing by focusing on reviews with helpfulness votes above the median and changing verified_purchase to binary format to increase model performance.

```python
import pandas as pd
import numpy as np

in_path = "data/Software.jsonl"
out_path = "clean_data/knn_software_clean.csv"

testing = False  # Set this to True for manual parameter input
max_entries = 250000
min_text_len = 10
good_columns = ["rating", "helpful_vote", "title", "text", "verified_purchase"]


df = pd.read_json(in_path, lines=True, nrows=max_entries)
df.dropna(subset=good_columns, inplace=True)
df = df[df['text'].str.len() > min_text_len]

# Filter rows where 'helpful_vote' is greater than or equal to the median
median_votes = np.floor(df['helpful_vote'].median())
df = df[df['helpful_vote'] >= median_votes]
df['text'] = df['text'].str.replace('<br />', '', regex=False)
df = df[good_columns]

df['verified_purchase'] = df['verified_purchase'].apply(lambda x: 1 if x else 0)

df.to_csv(out_path, index=False)
```

knn_data_cleaning.py

For feature reduction, we applied TF-IDF vectorization. Then, PCA to reduce the feature dimensions to 100. Finally, we scaled reduced features to make it compatible with KNN

```python
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Load the data
data = pd.read_csv('clean_data/knn_software_clean.csv')
data['combined_text'] = data['title'].fillna('') + ' ' + data['text']
X = data['combined_text']
y = data['rating']

# Vectorize the text using TF-IDF
vectorizer = TfidfVectorizer(max_features=5000)  # Limit to top 5000 words
X_tfidf = vectorizer.fit_transform(X)

print("TF-IDF vectorized shape:", X_tfidf.shape)

# Reduction to 100 features
pca = PCA(n_components= 100)
pca.fit(X_tfidf.toarray())
X_reduced = pca.transform(X_tfidf.toarray())

print("Reduced shape:", X_reduced.shape)

# Scale the reduced features (important for SVM)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_reduced)

# Save the reduced features for further use
reduced_data = pd.DataFrame(X_scaled)
reduced_data['rating'] = y
reduced_data.to_csv('data/knn_software_reduction.csv', index=False)

print("Sample of reduced data:", reduced_data.head())
```

knn_feature_reduction.py

KNN

KNN (K-Nearest Neighbors) is useful as it utilizes similarity in between data points to classify star ratings based on the text content. This allows KNN to capture nuanced relationships in the dataset (subtle differences in sentiment or language patterns across star ratings).

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import numpy as np

data = pd.read_csv('data/knn_software_reduction.csv')

X = data.drop(columns=['rating'])
y = data['rating']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

k = 5
knn = KNeighborsClassifier(n_neighbors=k)

knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

print("KNN Classification Report:")
print(classification_report(y_test, y_pred))

print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

knn_implementation.py

Result & Discussion

Result of KNN Calculation

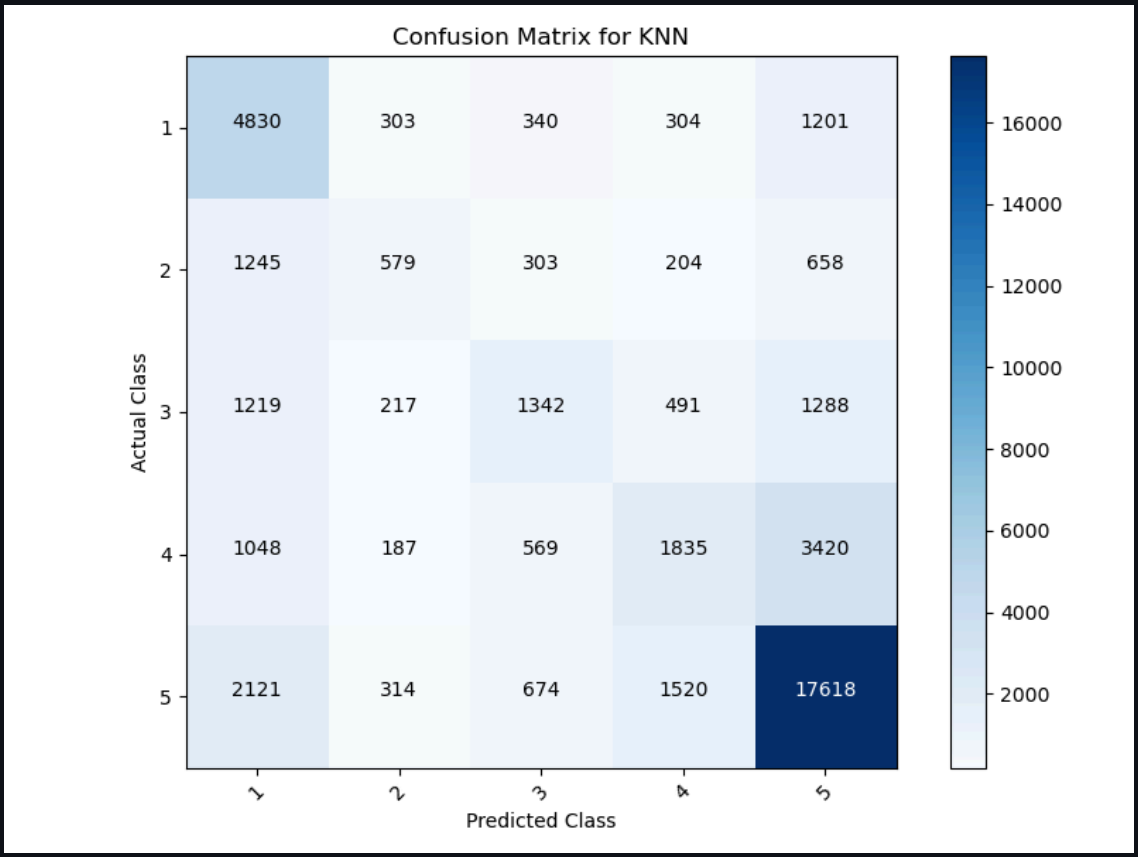
```
KNN Classification Report:
              precision    recall  f1-score   support

           1       0.46      0.69      0.55      6978
           2       0.36      0.19      0.25      2989
           3       0.42      0.29      0.34      4557
           4       0.42      0.26      0.32      7059
           5       0.73      0.79      0.76     22247

    accuracy                           0.60     43830
   macro avg       0.48      0.45      0.45     43830
weighted avg       0.58      0.60      0.58     43830

Accuracy Score: 0.5978553502167465
```

Calculation using KNN Model

For our KNN model, the generated output provides insights into how well the algorithm performs across different classes. The model visualizations below help interpret the meaning of each value in the final output. KNN got an accuracy of around 60%. This is an improvement from before, but it is till not great. This reflects challenges in predicting amazon review ratings objectively. Since human ratings lean

to extreme ratings, this adds noise to the dataset. Moving forward, we could try to further enhance accuracy and reliability with other models

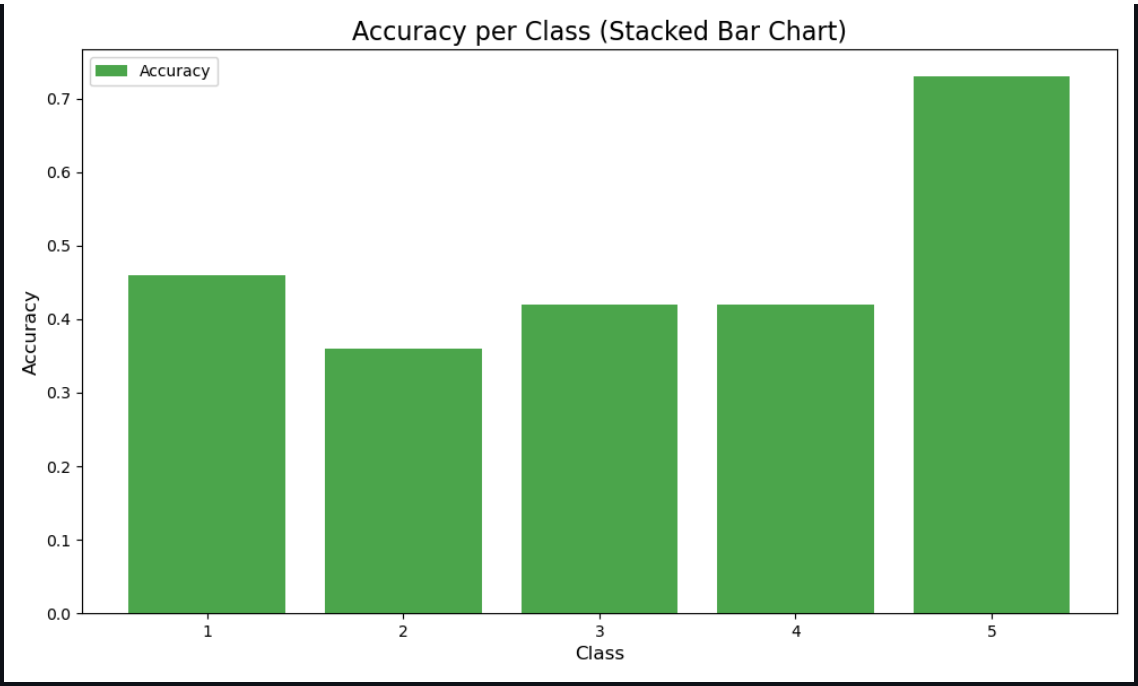
Model Visualization: Confusion Matrix for KNN)



Confusion Matrix

Evaluating the Confusion Matrix from KNN suggests that class 5 (strongly positive reviews) has the most number of correct predictions (17,618). The Confusion Matrix also suggests that class 2 reviews (2 star review) are often misclassified as class 1 (1 star reviews) as there were 1,245 misclassified reviews. There is a bias towards positive reviews and confusion between minority classes with mixed and negative reviews. There might be need for class weighting or resampling because of the large class imbalance. This would improve performance for underrepresented classes, which is necessary for Amazon review analysis because it is important to capture positive and negative sentiments to comprehensively classify customer sentiment.

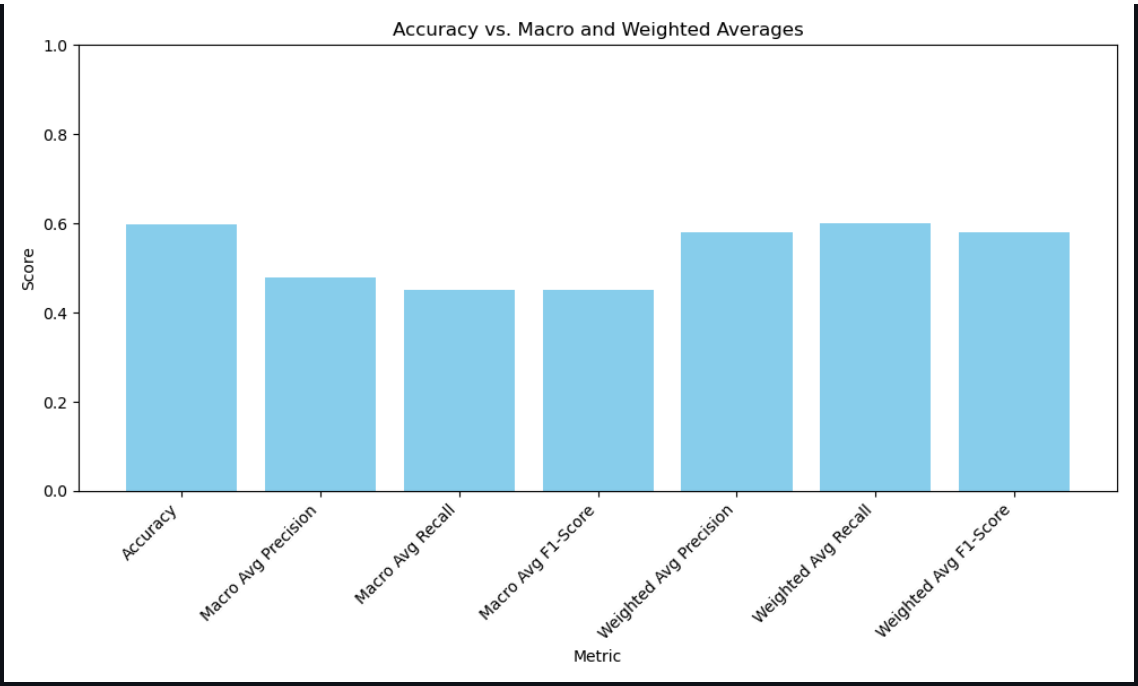
Model Visualization: Accuracy Per Class (Stacked Bar Chart)



Accuracy Per Class

This chart shows the accuracy distribution based on class, helping to visualize the different variations in accuracy for the model based on class. The accuracy is highest for Class 5, which (highly positive reviews), meaning that the KNN model is very accurately predicting strongly positive sentiments. On the other hand, the lowest accuracy belongs to Class 2 (neutral and mixed reviews) meaning that KNN was nut successful in distinguishing features of 2-star reviews. This makes sense as clear positive reviews tend to be easier to identify based on keywords, but the model has scope for improvement in predicting more nuanced feedback that has mixed feelings.

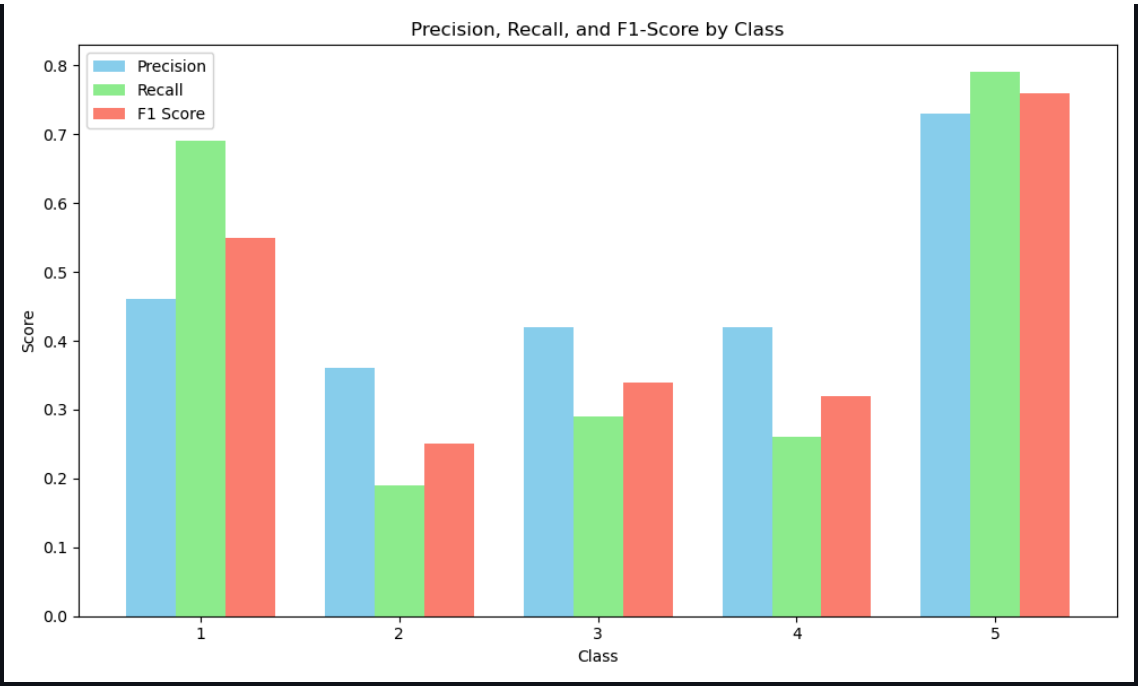
Model Visualization: Accuracy VS Macro and Weighted Averages



Accuracy VS Macro and Weighted Averages

This bar chart shows the overall accuracy, macro-averaged, and weighted-averaged score, and we see that weighted metrics surpass the macro-averaged scores. This shows that the model does better with majority classes and positive reviews, but not so well with negative and neutral reviews. The model favors classes with larger support and positive years, but the macro averages show broader performance across all 5 classes. F1 scores suggest that there is a good balance between recall and precision, but there is imbalance in class sizes, leading to different imbalances when predicting for each of the different classes.

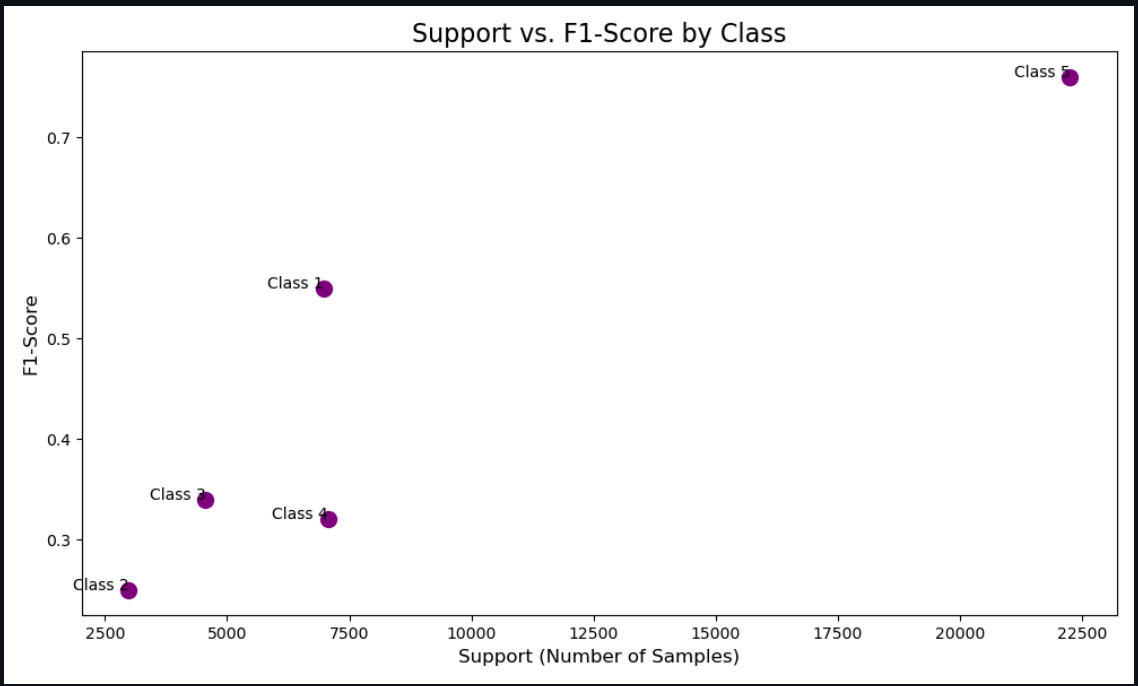
Model Visualization: Precision, Recall, and F1 Score By Class



Precision, Recall, and F1 Score

This graph demonstrates that the model achieves best performance with 5-star-reviews as the F1-score is 0.77 and there was high precision and recall (0.74 and 0.81 respectively). This shows again that our model is successful at classifying strong-positive reviews. 2-star class-2 reviews have the weakest F1 score (0.27) and low recall and precision, potentially due to frequent misclassifications. This indicates that the model performs worse when there is mixed sentiment involved in text reviews common in minority classes like 1 & 2 star reviews.

Model Visualization: Support VS F1 Score by Class



Support VS F1 Score by Class

There is a positive relationship with support and F1 score. Class 5 has the highest support and best F1 score (22,000 and 0.77 respectively). Class 2 has low support and also lowest F1-Score (2,000 and 0.27 respectively). This is likely due to the slightly imbalance dataset on model performance which skews sentiment analysis and might lead to a misinterpretation fo customer opinions. Enhancing model sensitivity for 1-star and 2-star reviews can improve the model accuracy for all types of reviews.

Final Observations

What do these (quantitative scoring metrics) mean?: The KNN model has an accuracy of 60%. This is reasonable for the context because there are obvious challenges associated with predicting ratings solely based off review text. The KNN model worked best on 5-star reviews, as the F1 score was 0.76. This is logical because 5-star reviews are the most distinct in tone and sentiment, making it easier for the model to classify these reviews. They are also the most abundant. The KNN model struggled with 2-star reviews as the F1-score was only 0.25 ,likely because there is mixed language and sentiment in this group of reviews, making it harder to classify and distinguish. The dataset imbalance might have also added to this as there were fewer reviews.

Why did they perform well or poorly?: Dimensionality reduction with PCA allowed us to take into account the high dimension TF-IDF vectors as we only retained the most meaningful 100 features. Even after PCA, though, KNN relying on distance metrics only makes it sensitive to noise which might have caused ambiguity in classifications due to the nuanced language of reviews. There is confusion in classes with low F1 scores, which is explained by positive but slightly upset/critical 4-star reviews or slightly negative 3-star reviews sharing the same words/sentiment usage.

What might need to be tweaked to improve the performance next time?: In future iterations of the model: first, we can balance the dataset by reweighting underrepresented classes (2- or 3-star reviews). Next, we can use metrics for distance specifically suited for text data (eg. cosine similarity) to fully understand the nuanced semantics of Amazon reviews. Exploring models specific to NLP tasks like transformers can increase the accuracy by addressing limitations.

Comparison of Models

SVM was our most accurate model with 67% accuracy, followed by KNN at 60% accuracy, and naive bayes had the lowest accuracy at 47%. SVM's strengths are being good for many dimensions and preventing overfitting but the limitation was that it took a long time to run. The tradeof was in best accuracy vs slowest runtime, and would be best for smaller datasets. KNN's strengths are that it is faster than SVM and has decent accuracy, but was slower than naive bayes. The trade off is that it did not perform the best in any category. Naive Bayes's strength is that it is the fastest running model and had a very small difference in runtime between a sample and the entire dataset, but had the lowest accuracy. The tradeoff is between being the fastest but also least accurate, and would be better for datasets over 5gb.

Gantt Table

TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION
Project Team Composition	All	8/31/24	9/14/24	14
Project Proposal				
Introduction & Background	Ayush	9/26/24	10/4/24	8
Problem Definition	Ayush	9/26/24	10/4/24	8
Methods	Krishi & Jiya	9/26/24	10/4/24	8
Potential Dataset	Krishi & Jiya	9/26/24	10/4/24	8
Potential Results & Discussion	Divya & Ananya	9/26/24	10/4/24	8
Video Creation & Recording	All	9/26/24	10/4/24	8
GitHub Page	Krishi	9/26/24	10/4/24	8
Midterm Report				
Model 1 (M1) Design & Selection	All	10/7/24	10/25/24	18
M1 Data Cleaning	Krishi	10/7/24	10/25/24	18
M1 Data Visualization	Ananya	10/10/24	10/25/24	15
M1 Feature Reduction	Divya	10/10/24	10/25/24	15
M1 Implementation & Coding	Ayush	10/7/24	10/25/24	18
M1 Results Evaluation + Streamlit + README	Jiya	10/14/24	10/25/24	11
Model 2 (M2) Design & Selection	All	10/28/24	11/15/24	10
M2 Data Cleaning	Jiya	10/28/24	11/15/24	17
M2 Data Visualization	Ayush	10/30/24	11/15/24	15
M2 Feature Reduction	Ayush	10/30/24	11/15/24	15
M2 Coding & Implementation	Divya	10/30/24	11/15/24	15
M2 Results Evaluation + Streamlit + README	Krishi	11/4/24	11/15/24	11
Midterm Report	All	10/28/24	11/8/24	10
Final Report				
Model 3 (M3) Design & Selection	All	11/11/24	11/22/24	11
M3 Data Cleaning	Divya	11/11/24	11/22/24	11
M3 Data Visualization	Ananya	11/14/24	11/22/24	11
M3 Feature Reduction	Krishi	11/14/24	11/22/24	8
M3 Implementation & Coding	Jiya	11/14/24	11/22/24	8
M3 Results Evaluation + Streamlit + README	Ananya	11/18/24	11/22/24	4
M1-M3 Comparison	All	11/25/24	12/3/24	8
Video Creation & Recording	All	11/25/24	12/3/24	8
Final Report	All	11/25/24	12/3/24	8

Preview of Gantt Table

You can see the full Gantt Table here link

Contribution Table

Name	Contribution
Ayush	Model 1 Implementation & Coding, incorporated confusion matrix into new Naive Bayes Model, updated Streamlit, Model 2 Data Visualization and Feature Reduction, Worked on Google Slides Report
Jiya	Model 1 Results Evaluation, Updated Streamlit for Midterm Report, Model 2 Data Cleaning, Model 3 Coding & Implementation, worked on Streamlit and README
Krishi	Model 1 Data Cleaning, worked on README, Model 2 Result Discussion & Evaluation, Model 3 Feature Reduction, Worked on Google Slides, worked on README
Ananya	Model 1 Data Visualization, set up meetings, Updated Streamlit, Model 3 Data Visualization, Model 3 Discussion & Evaluation, Worked on Google Slides, worked on README
Divya	Model 1 Feature Reduction, updated Naive Bayes python file for new calculation, Model 2 Coding & Implementation, Model 3 Data Cleaning, worked on Streamlit and Google Slides

Contribution Table

Citations/References

M. Medhat, A. Hassan, and H. Korashy, “Sentiment Analysis Algorithms and Applications: A Survey,” IEEE Access, vol. 5, pp. 847–870, 2017. Available: https://ieeexplore.ieee.org/document/8376299.

S. K. Rana, S. Singh, and N. Kumar, “Predicting Product Rating using Text Mining over Review Data,” in Proceedings of the 2nd International Conference on Computer, Communication and Computational Sciences (IC4S 2017), Singapore: Springer, 2018, pp. 487–495. Available: https://link.springer.com/chapter/10.1007/978-981-10-7512-4_40.

Amazon Product Review Dataset 2023, Amazon Review Dataset Documentation, 2023. Available: https://amazon-reviews-2023.github.io/.

S. Wassan, T. A. Alzahrani, and S. A. Omer, “Amazon Product Sentiment Analysis using Machine Learning Techniques,” International Journal of Computer Applications, vol. 174, no. 22, pp. 1–6, 2021. Available: https://www.researchgate.net/profile/Sobia-Wassan-2/publication/349772322_Amazon_Product_Sentiment_Analysis_using_Machine_Learning_Techniques/links/60411e09a6fdcc9c78121992/Amazon-Product-Sentiment-Analysis-using-Machine-Learning-Techniques.pdf.