Assignment 2: Naive Bayes and SVMs

COL 774 MACHINE LEARNING

Name: Aniket Chattopadhyay
Entry No: 2022CS11599

1 TEXT CLASSIFICATION USING NAIVE BAYES

1.1 TRAINING ON DESCRIPTION (BASIC TOKENIZATION)

In this part, I have done a basic pre-processing by removing numbers, converting the words to lowercase (which is a general practice and was also hinted to in a Piazza comment) and using the .split() command to split the text into a list of tokens.

Part (a): Train and test accuracy of the naive bayes model

Train accuracy = 91.68%
Test accuracy = 88.85%

Part (b): Constructing word clouds for all the classes

Figure 1: Word Cloud for class 1 i.e. World

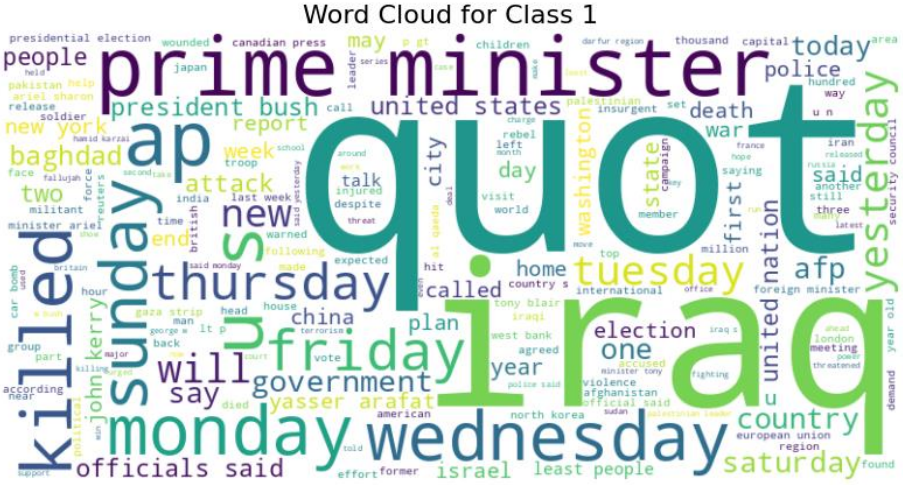


Figure 2: Word Cloud for class 2 i.e. Sports

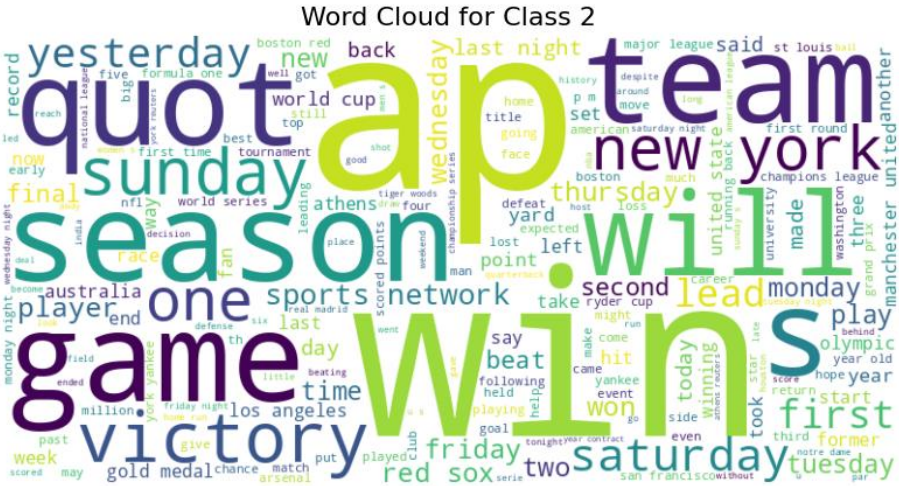


Figure 3: Word Cloud for class 3 i.e. Business

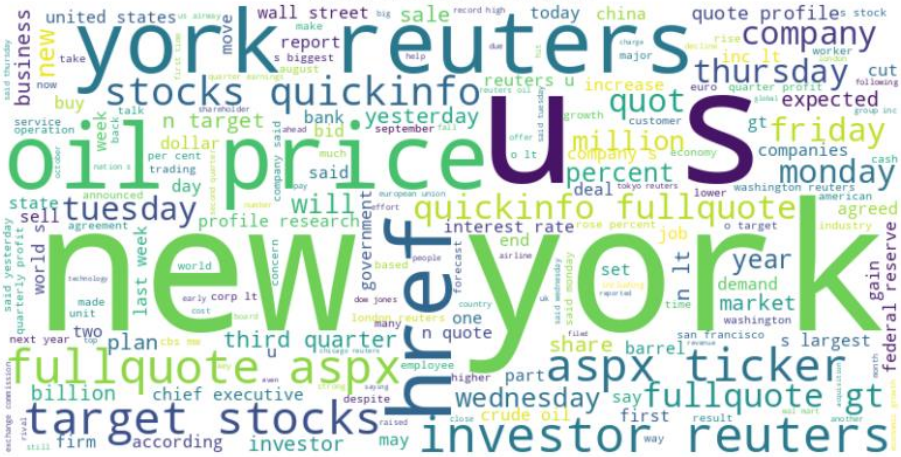
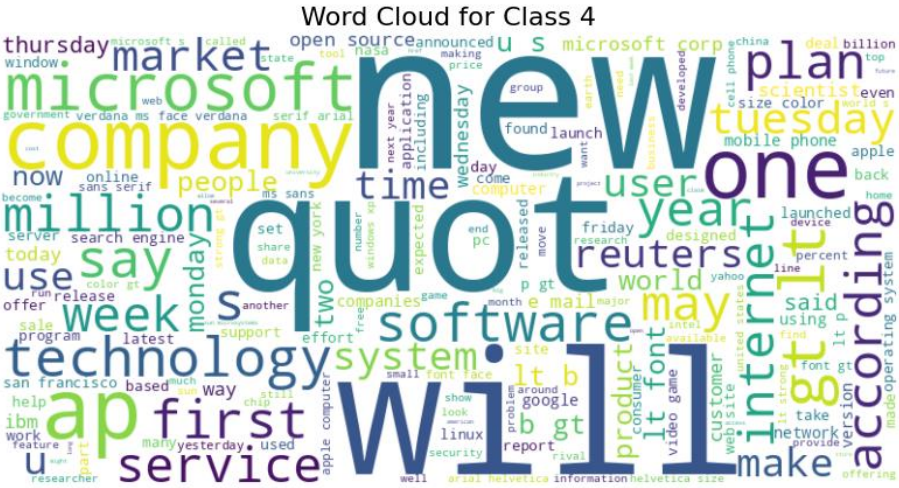


Figure 4: Word Cloud for class 4 i.e. Science / Technology



1.2 TRAINING ON DESCRIPTION (BASIC TOKENIZATION + STOPWORD REMOVAL AND STEMMING USING NLTK)

Part (a): Stemming and stop-word removal

In this part, apart from whatever was done earlier, I deleted stop words by deleting the tokens which were present in the nltk stopwords for English and used the PorterStemmer() from nltk library for stemming purposes.

Part (b): Constructing word clouds for all the classes

Figure 1: Word Cloud for class 1 i.e. World

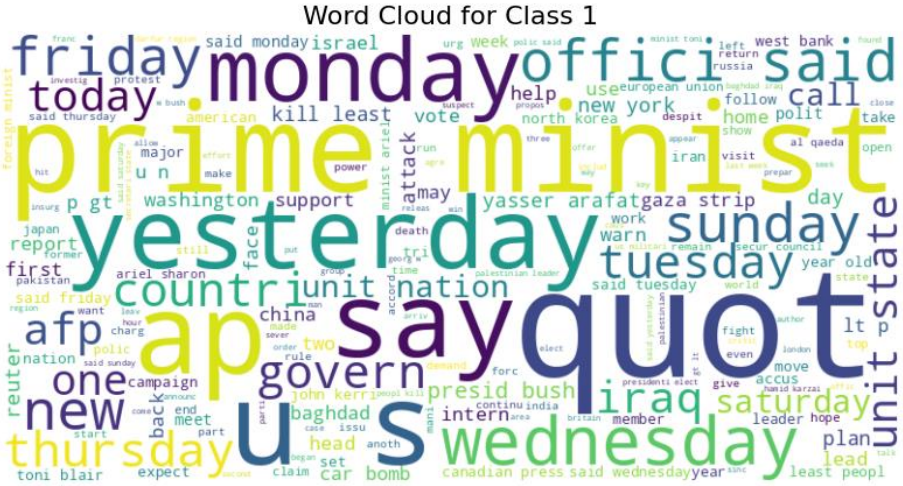


Figure 2: Word Cloud for class 2 i.e. Sports

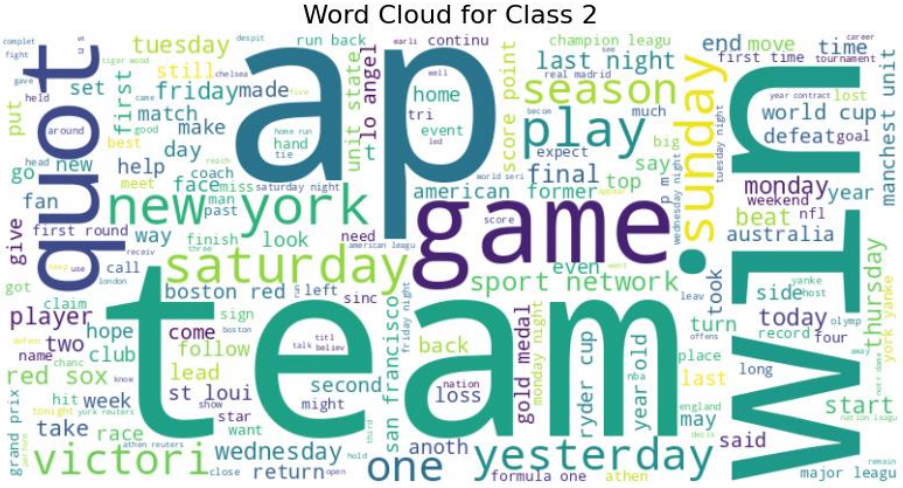


Figure 3: Word Cloud for class 3 i.e. Business

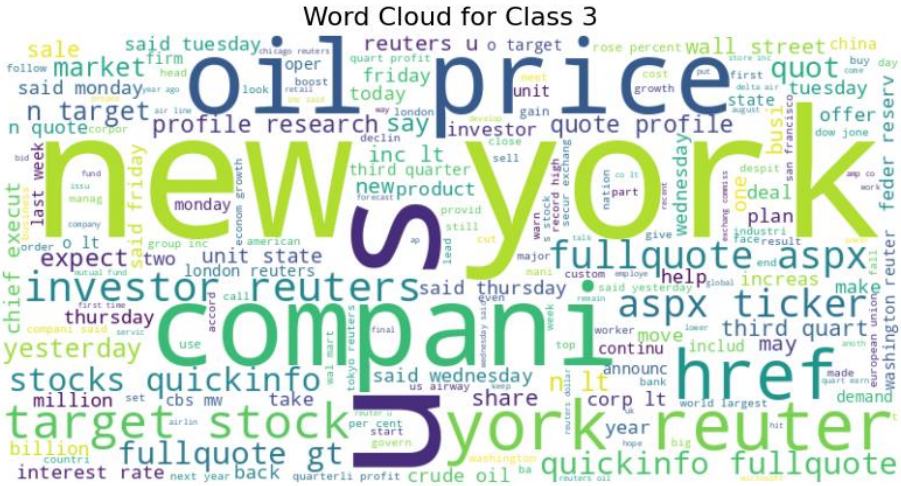
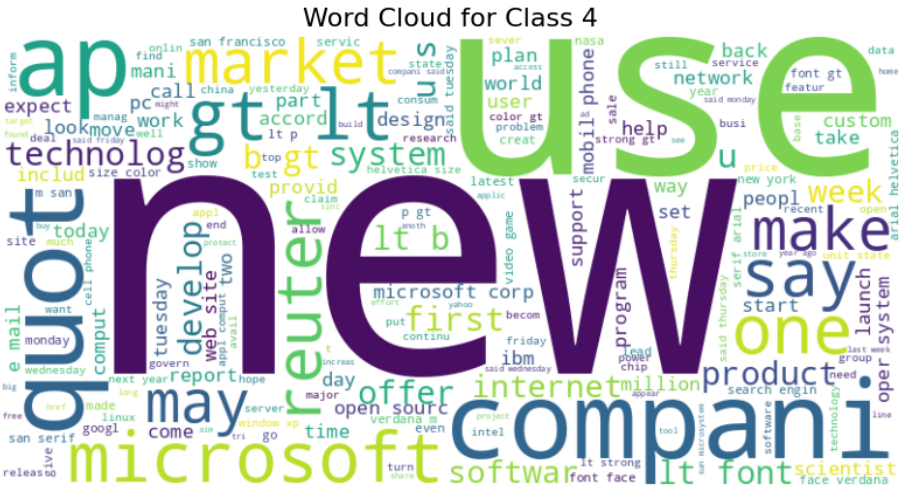


Figure 4: Word Cloud for class 4 i.e. Science / Technology



Part (c): Training the Naive Bayes model and reporting test (validation) and train accuracy

Train accuracy = 91.72%
Test (Validation) accuracy = 89.08%

Part (d): Changing of accuracy over the validation set

Change: There is an increase in the validation accuracy (from 88.85 to 89.08 %)
Reasoning: Removing **stopwords** eliminates non-informative words, reducing noise, while **stemming** consolidates word variants, improving feature consistency. This helps the model to focus on important terms, leading to better generalization and improved accuracy

1.3 TRAINING ON DESCRIPTION (BASIC TOKENIZATION + STOPWORD REMOVAL AND STEMMING USING NLTK + BIGRAMS)

Part (a): Inclusion of bigrams

I am now utilizing the bigram from nltk.utils library, which creates bigrams from a given text string. The 'Tokenized Description' column now contains the set of unigram tokens and the set of bigram tokens. Note that the order of unigram/bigram tokens doesn't matter due to the Naive Bayes assumption (i.e. distribution is independent of the position)

Part (b): Training the Naive Bayes model and reporting test (validation) and train accuracy

Train accuracy = 95.95% (sharp increase from the previous case)
Test (Validation) accuracy = 89.97 %

Part (c): Changing of accuracy over the validation set (over part 1.2)

Change: Increase in test accuracy (89.08 to 89.97 %)
Reasoning: Tokenizing into **bigrams** captures contextual relationships between words, improving feature representation in a **Naïve Bayes** text classifier. The sharp increase in **training accuracy** suggests overfitting, as the model memorizes frequent bigram patterns but may struggle to generalize to unseen data

1.4 COMPARISON OF MODEL PERFORMANCE METRICS OVER VARIOUS DATA-PREPROCESSING POSSIBILITIES (DESCRIPTION)

Part (a): Performance metrics used for comparison

Accuracy Score: Accuracy is the ratio (or percentage) of correctly predicted instances to the total number of instances in the test data. It is useful in balanced datasets and can be misleading in imbalanced datasets. Since in our test dataset, we have 1900 test instances for each class, this is a good metric to compare model performance

$$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Precision: Precision for a specific class is the proportion of correctly predicted instances of that class out of all instances predicted as belonging to that class. Precision can be averaged over all the classes as a micro-average, macro-average, weighted-average or averaging on samples. High precision means fewer false positives for each class. False positives are costly here, as a person who is reading only, say business news, might miss out on some news (and he may lose his job 😳, for e.g.: people in compliance division)

$$Precision_i = \frac{TP_i}{TP_i + FP_i}$$

Where:

- TP_i = True Positives for class i
- FP_i = False Positives for class i

Averaging Methods:

- **Macro Precision**:
  The average precision across all classes.

$$Macro\ Precision = \frac{1}{N}\sum_{i=1}^{N} Precision_i$$

- **Micro Precision**:
  Computes precision globally by summing up all TP and FP across classes before applying the formula.

$$Micro\ Precision = \frac{\sum TP}{\sum (TP + FP)}$$

These averaging methods will be similar for recall as well as f1 score (our other performance metrics). However, note that we have a balanced dataset, so all these averaging methods are equivalent, and you can use any of these.

Recall: Recall for a class is the proportion of actual instances of that class that were correctly predicted. High recall means fewer false negatives for each class. It is useful when false negatives are costly. A similar example as in the case of precision may be given here. Hence, this is also an important metric for us.

$$Recall_i = \frac{TP_i}{TP_i + FN_i}$$

Where:

- TP_i = True Positives for class i
- FN_i = False Negatives for class i

F1 Score: It is the harmonic mean of precision and recall, ensuring a balance between the two. It is useful when you need to balance the false positives and false negatives. In our case, this is also an important metric as we would not either of false negatives or false positives to be very high.

$$F1_i = 2 \times \frac{Precision_i \times Recall_i}{Precision_i + Recall_i}$$

Part (b): Performance comparison based on these metrics

Case	Remove Stop Words	Apply Stemming	Use Bigrams	Train Accuracy	Test Accuracy	Test Precision	Test Recall	Test F1 Score
0	❌ No	❌ No	❌ No	0.9168	0.8886	0.8882	0.8886	0.8882
1	❌ No	❌ No	✅ Yes	0.9503	0.8958	0.8955	0.8958	0.8955
2	❌ No	✅ Yes	❌ No	0.9132	0.8886	0.8883	0.8886	0.8883
3	❌ No	✅ Yes	✅ Yes	0.9480	0.8949	0.8946	0.8949	0.8947
4	✅ Yes	❌ No	❌ No	0.9208	0.8921	0.8917	0.8921	0.8918
5	✅ Yes	❌ No	✅ Yes	0.9617	0.9018	0.9015	0.9018	0.9016
6	✅ Yes	✅ Yes	❌ No	0.9172	0.8908	0.8904	0.8908	0.8905
7	✅ Yes	✅ Yes	✅ Yes	0.9596	0.8997	0.8995	0.8997	0.8995

Part (c): Choosing the best token pre-processing over description text

- Based on test accuracy, test precision, test recall and test f1 score, we can clearly see that **case 5 i.e. stop word removal and bigram tokens but no stemming on description text, gives us the best result** (Note that everything is built upon the basic tokenizer in 1.1)
- Also, the smaller gap between train and test accuracy in case 5 than in case 7 is an indicator that this data pre-processing is less clone to overfitting, than the other.

1.5 TRAINING ON TITLE DATA

1.5.1 TRAINING ON TITLE (BASIC TOKENIZATION)

Part (a): Train and test accuracy of the naive bayes model

Train accuracy = 89.68%
Test accuracy = 86.21%

Part (b): Constructing word clouds for all the classes

Figure 1: Word Cloud for class 1 i.e. World

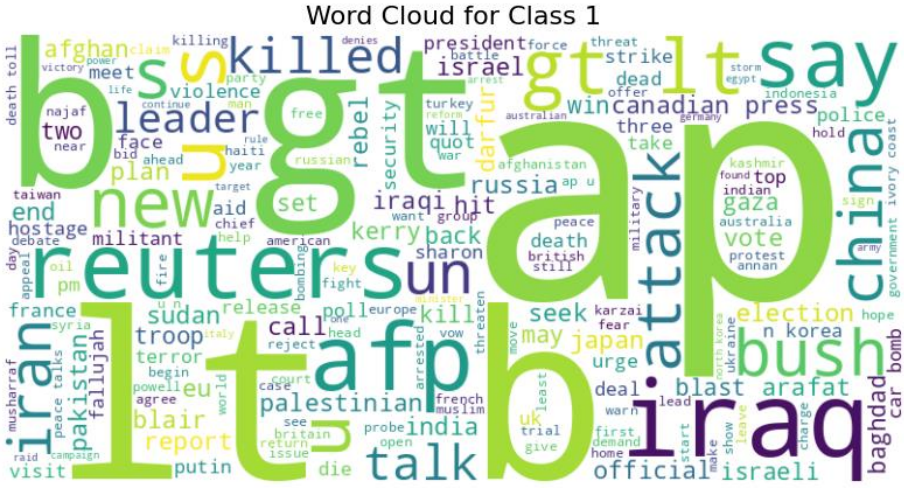


Figure 2: Word Cloud for class 2 i.e. Sports

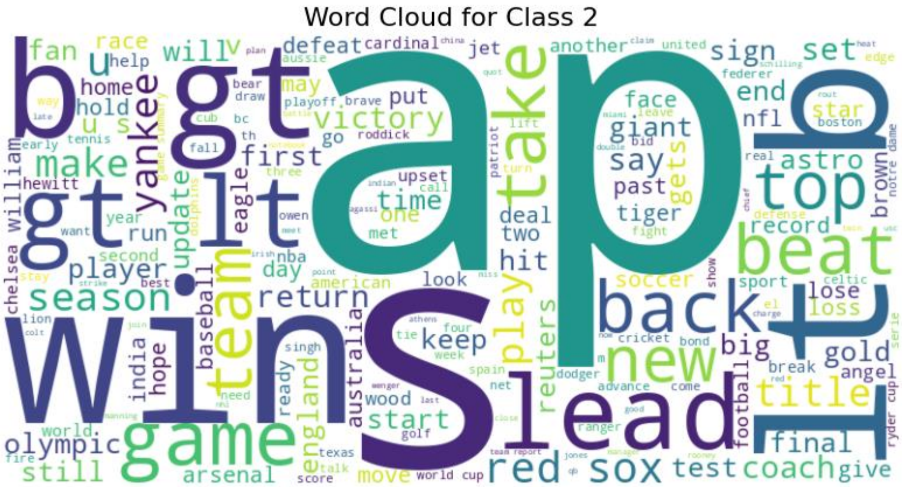


Figure 3: Word Cloud for class 3 i.e. Business

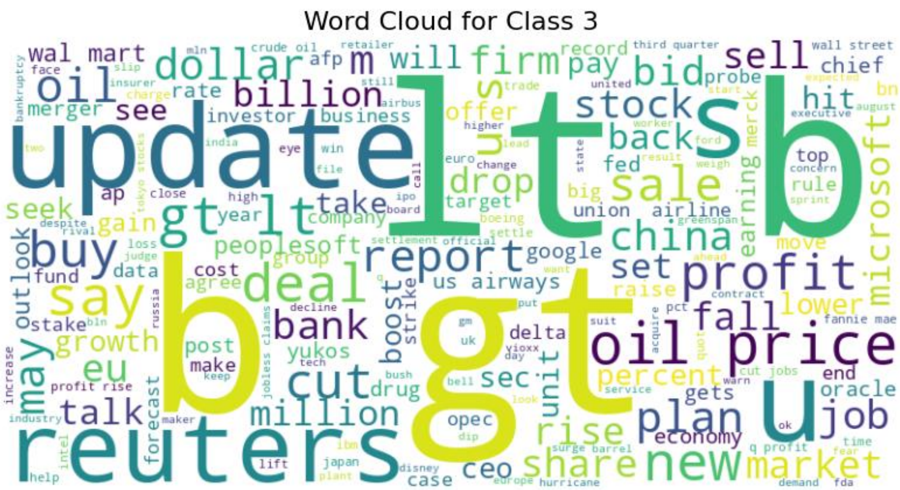
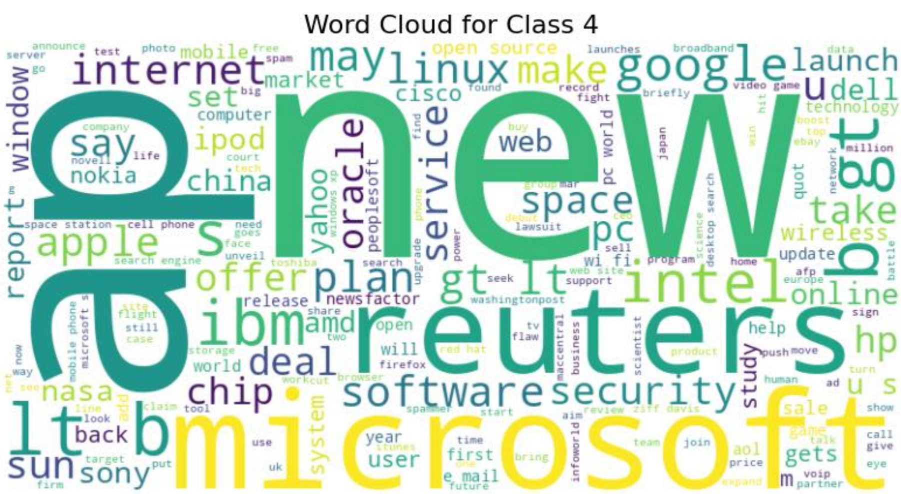


Figure 4: Word Cloud for class 4 i.e. Science / Technology



1.5.2 TRAINING ON TITLE (BASIC TOKENIZATION + STOPWORD REMOVAL AND STEMMING USING NLTK)

Part (a): Stemming and stop-word removal

In this part, apart from whatever was done earlier, I deleted stop words by deleting the tokens which were present in the nltk stopwords for English and used the PorterStemmer() from nltk library for stemming purposes.

Part (b): Constructing word clouds for all the classes

Figure 1: Word Cloud for class 1 i.e. World

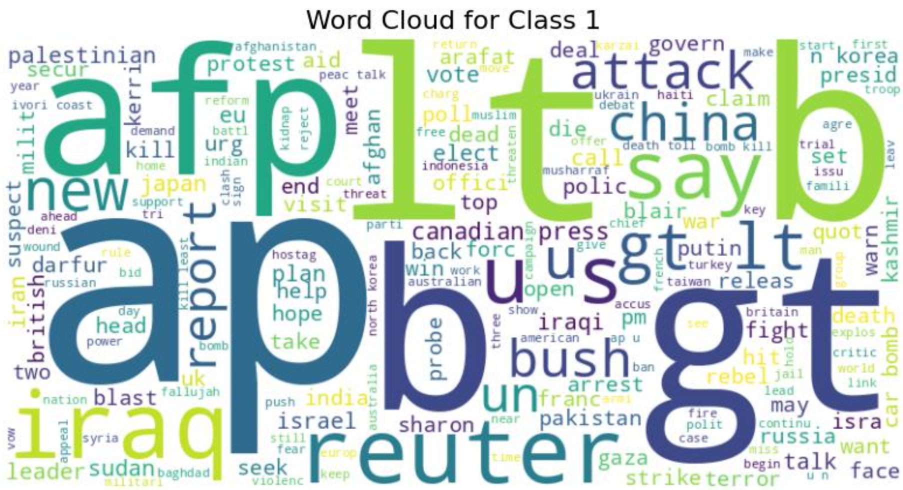
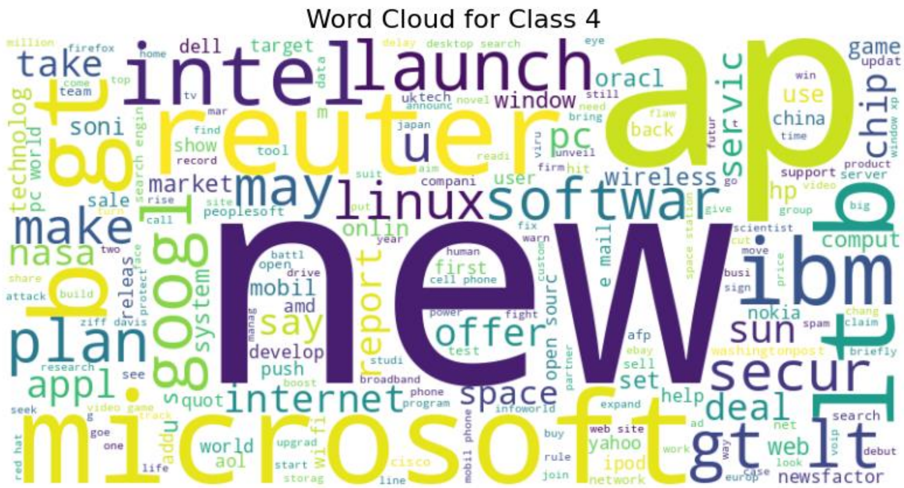


Figure 2: Word Cloud for class 2 i.e. Sports

Figure 3: Word Cloud for class 3 i.e. Business

Figure 4: Word Cloud for class 4 i.e. Science / Technology



Part (c): Training the Naive Bayes model and reporting test (validation) and train accuracy

Train accuracy = 88.94%
Test (Validation) accuracy = 85.89%

Part (d): Changing of accuracy over the validation set

Change: There is a slight decrease in the validation accuracy (from 86.21 to 85.89 %)
Reasoning: Removing stopwords eliminates non-informative words, but it may also remove contextually relevant words that contribute to classification. Stemming reduces word variants, but it can also lead to loss of meaning (e.g., "running" → "run"), which may negatively impact model performance. These factors can cause a slight decrease in validation accuracy

1.5.3 TRAINING ON TITLE (BASIC TOKENIZATION + STOPWORD REMOVAL AND STEMMING USING NLTK + BIGRAMS)

Part (a): Inclusion of bigrams – same explanation as in case of description

Part (b): Training the Naive Bayes model and reporting test (validation) and train accuracy

Train accuracy = 94.57% (sharp increase from the previous case)
Test (Validation) accuracy = 87.21 %

Part (c): Changing of accuracy over the validation set (over part 1.2)

Change: Increase in test accuracy (85.89 to 87.21 %)
Reasoning: Tokenizing into **bigrams** captures contextual relationships between words, improving feature representation in a **Naïve Bayes** text classifier. The sharp increase in **training accuracy** suggests overfitting, as the model memorizes frequent bigram patterns but may struggle to generalize to unseen data

1.5.4 COMPARISON OF MODEL PERFORMANCE METRICS OVER VARIOUS DATA-PREPROCESSING POSSIBILITIES (TITLE)

Part (a): Performance metrics used for comparison – we'll use the same metrics

Part (b): Performance comparison based on these metrics

Case	Remove Stop Words	Apply Stemming	Use Bigrams	Train Accuracy	Test Accuracy	Test Precision	Test Recall	Test F1 Score
0	❌ No	❌ No	❌ No	0.8968	0.8621	0.8616	0.8621	0.8617
1	❌ No	❌ No	✅ Yes	0.9464	0.8695	0.8690	0.8695	0.8691
2	❌ No	✅ Yes	❌ No	0.8879	0.8621	0.8616	0.8621	0.8617
3	❌ No	✅ Yes	✅ Yes	0.9420	0.8682	0.8677	0.8682	0.8678
4	✅ Yes	❌ No	❌ No	0.8982	0.8613	0.8609	0.8613	0.8610
5	✅ Yes	❌ No	✅ Yes	0.9506	0.8718	0.8714	0.8718	0.8716
6	✅ Yes	✅ Yes	❌ No	0.8894	0.8589	0.8586	0.8589	0.8586
7	✅ Yes	✅ Yes	✅ Yes	0.9457	0.8721	0.8718	0.8721	0.8718

Part (c): Choosing the best token pre-processing over description text

- The above data has minute differences by changing the stop word removal/ stemming, however, we can see a major change on the change of addition of bigrams. So, we'll go with the case where there is a highest (positive) change, which is from case 6 to case 7
- Also, the other parameters are most-favored for case 7. Also, a lower train-test difference among the other cases where bigrams is true indicates that it will lead to the least overfitting, so we have all the boxes ticked for this case (quite literally 😊)

1.6 TRAINING ON BOTH TITLE AND DESCRIPTION FEATURES

Part (a): Learning the same set of parameters for both title and description features (note that they work over the same vocabulary)

Train Accuracy: 96.35%
Test Accuracy: 91.10%

In comparison to the previously trained model using a single set of features (title/description), this model has both a high train accuracy as well as validation (or test) accuracy. Using both title and description provides a richer feature set, capturing more contextual and semantic information. This reduces information loss and helps the model generalize better, leading to higher test accuracy. Also, the spike in training accuracy is due to the use of bigrams (as seen in earlier single-set models)

For comparison, you can see the table in 1.4 and 1.5.4 (the above reasoning works for all those comparisons, as we have anyways considered the best performing text preprocessing from each of those tables, which leads to the performance of the Naive Bayes Classifier)

Note that this simply works well over the models with a single set of features because now more features are being added to the model, and since the accuracies and other performance metrics of the best performing single-set models are nearby (i.e. 87.21% and 90.15%), their addition is constructive and leads to a better performing model

Part (b): Learning different sets of parameters for both title and description features

Here, I'd like to note that this can be done in two ways:
(1) Maintaining the same vocabulary for title and description tokens
(2) Maintaining different vocabularies for title and description tokens

However, for this given dataset, I've empirically got the same result, so I'll write the results for (2) only (and based on Piazza clarifications this was also what we needed to do)

Train Accuracy: 96.44%
Test Accuracy: 91.03%

In comparison to the previously trained model using a single set of features (title/description), this model has both a high train accuracy as well as validation (or test) accuracy. Using both title and description provides a richer feature set, capturing more contextual and semantic information. This reduces information loss and helps the model generalize better, leading to higher test accuracy. Also, the spike in training accuracy is due to the use of bigrams (as seen in earlier single-set models).

For comparison, you can see the table in 1.4 and 1.5.4 (the above reasoning works for all those comparisons, as we have anyways considered the best performing text preprocessing from each of those tables, which leads to the performance of the Naive Bayes Classifier)

In a **Naïve Bayes classifier**, we estimate the probability of a class **C** given features **X** using Bayes' theorem:

$$P(C \mid X) = \frac{P(X \mid C)P(C)}{P(X)}$$

(1) Single-Feature Models (Title or Description Only)

If we train the model using only the **title** or only the **description**, our likelihood estimation is:

$$P(C \mid X_{\text{title}}) = \frac{P(X_{\text{title}} \mid C)P(C)}{P(X_{\text{title}})}$$

or

$$P(C \mid X_{\text{desc}}) = \frac{P(X_{\text{desc}} \mid C)P(C)}{P(X_{\text{desc}})}$$

This means we only use one set of features, potentially losing valuable contextual information.

(2) Learning Separate Parameters for Title and Description (Model (b))

In (b), we treat title and description as independent given the class, so the probability becomes:

$$P(C \mid X_{\text{title}}, X_{\text{desc}}) = \frac{P(X_{\text{title}} \mid C)P(X_{\text{desc}} \mid C)P(C)}{P(X_{\text{title}}, X_{\text{desc}})}$$

This is better than using a single feature because:

- Title and description contribute separately to the likelihood calculation, meaning the model captures more information.
- The assumption of conditional independence reduces variance and improves generalization.

Also, note that this performs better than single-set feature classifiers for a similar reason explained above for part (a). However, note that the training accuracy in this part is slightly higher than the one in part (a), and the test accuracy is slightly lower. This is a clear indication of some **overfitting** happening, which can be explained by **virtue of the higher number of parameters** to be learned. This overshadows the advantage of (b) over (a) for **independent parameter learning** and **better feature representation**

1.7 ANALYSIS AGAINST BASELINE RESULTS

For this problem, let us consider the prediction problem as a game, with say n instances to predict in the test data. A correct prediction from the model will award it a (+1), whereas a wrong prediction does not award anything (or awards a 0)

Part (a): Random Classifier

For a random classifier, expected number of points in a single problem is 1/k, where k is the number of classes to be predicted (in our case k = 4)

So, expected number of points in the game = 1/k * n = n/k
(due to summation of expectations, where event i is the prediction of instance i of test data)

Thus, accuracy = n/k / n = 1/k = 0.25 or 25 %

Part (b): Single-class output Classifier

Since the test data has an equal number of samples for each class, it does not matter which class is the output of this classifier.

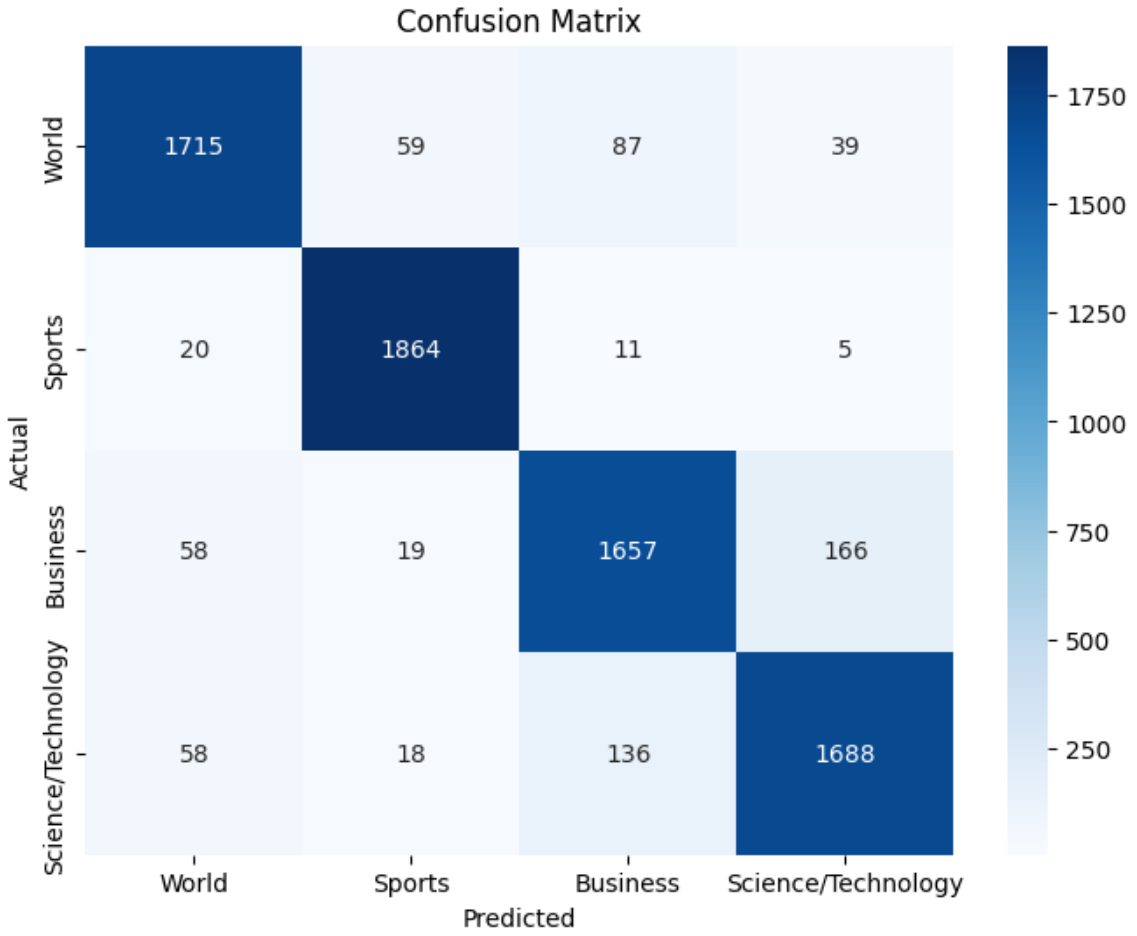
Accuracy will always be 0.25 or 25 % (since, n/4 of the predictions are correct as they belong to the same class as the one predicted)

Part (c): Comparision with our best model

Our algorithm gives about 65% improvement over the above baseline models

1.8 CONFUSION MATRIX FOR THE BEST PERFORMING MODEL

Part (a): The Confusion Matrix



A **confusion matrix** is a table used to evaluate the performance of a classification model. For a **multi-class** classification problem, it looks like this:

Actual \ Predicted	Class 1	Class 2	Class 3	Class 4
Class 1	**TP_1**	FP_{12}	FP_{13}	FP_{14}
Class 2	FP_{21}	**TP_2**	FP_{23}	FP_{24}
Class 3	FP_{31}	FP_{32}	**TP_3**	FP_{34}
Class 4	FP_{41}	FP_{42}	FP_{43}	**TP_4**

Part (b): Analysis of the Confusion Matrix

The highest value of the diagonal is for the ‘Sports’ class. This means that the model is the best classifier for news related to sports category. You can also see that the off-diagonal entries for Sports are also quite small, implying that some tokens are exclusive to this category, the idea of which we had got in our class maps in 1.1, 1.2, 1.5.1 and 1.5.2.

The highest off-diagonal values for Science/Technology and Business indicates that there are a lot of overlapping tokens/phrases between these two, and the model is thus not able to sometimes classify them properly.

1.9 ADDITIONAL FEATURE ENGINEERING

I have tried a lot of methods, like including trigrams, lexical analysis etc. I was only able to get slight changes in the test accuracy, though the training accuracy varied greatly (especially while using higher n-grams, for instance, with trigrams for description, I got a training accuracy of 98%, but a test of 90-odd % only, showing high overfitting)
However, this trigram logic for title, gave me a slightly better accuracy of 91.137%, over a different tokenization and custom stopwords.

2 IMAGE CLASSIFICATION USING SVM

2.1 BINARY CLASSIFIER (SVM WITH LINEAR KERNEL)

Support Vector Machine (SVM) Optimization with Linear Kernel

For a **linearly separable** classification problem with **soft margin** (to handle noise), the SVM optimization problem aims to maximize the margin while allowing some misclassification through slack variables ξ_i. The primal form of the optimization problem is:

$$\min_{w,b,\xi} \quad \frac{1}{2}w^T w + C\sum_{i=1}^{m}\xi_i$$

subject to:

$$y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i = 1, ..., m$$

where:

- w is the weight vector,
- b is the bias term,
- ξ_i are slack variables to allow misclassification,
- C is a regularization parameter controlling the trade-off between maximizing margin and minimizing misclassification.

Dual Formulation in Quadratic Programming (QP) Form

To solve this problem efficiently, we convert it into its **Lagrange dual form**, introducing Lagrange multipliers α_i:

$$\max_{\alpha} \quad \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j K(x_i, x_j)$$

subject to:

$$\sum_{i=1}^{m}\alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C, \quad \forall i$$

where $K(x_i, x_j) = x_i^T x_j$ is the **linear kernel function**.

This can be rewritten in the **standard quadratic programming (QP) form**:

$$\min_{\alpha} \quad \frac{1}{2}\alpha^T P\alpha + q^T\alpha + c$$

where:

- $P_{ij} = y_i y_j K(x_i, x_j)$ is an $m \times m$ matrix,
- q is an m-dimensional vector with all elements **-1**,
- c is a constant (typically 0),
- α is the vector of Lagrange multipliers.

Part (a):

Number of support vectors = 381
Percentage of training samples = 30.80%

Part (b):

Test set accuracy (Linear Kernel): 85.53%

Weight vector (w): [0.00944532 -0.00047554 0.0017714 ... -0.01874945 -0.03258711

-0.02074077]

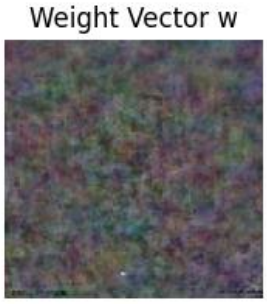
Intercept term (b): -0.0747

A better way to represent the weight vector will be in terms of a plot, which will be shown in the next part. This plot clearly explains the distribution of weights among different channels, enabling us to understand which parts of an image are useful for classification.

Here, the weight vector is an array of length 3X100X100

Part (c):

These are the top 5 unique support vectors (there are duplicates due to image duplicates, which have been ignored). Note that all these images are of the class 'fogsmog'



This is the weight vector. As can be seen from the images in the dataset, there is no specific portion of the images which clearly can classify them

2.2 BINARY CLASSIFIER (SVM WITH GAUSSIAN KERNEL)

The approach is the same as in the case above, however, now the Kernel function changes, and we now have a Gaussian kernel:

$$K(x, x') = \exp\left(-\gamma \|x - x'\|^2\right)$$

Part (a):

Number of support vectors: 306
Percentage of training samples that are support vectors: 24.74%

These are less than the support vectors in case of a linear kernel (we had 381 there, around 30%)

Number of common support vectors between linear and gaussian: 222

Part (b):

Test set accuracy (Gaussian Kernel): 95.50%
Intercept term (b): -1.2425

Part (c):

These are the top 5 unique support vectors (there are duplicates due to image duplicates, which have been ignored). Note that all these images are of the class 'dew'

2.3 COMPARISON BETWEEN LIBSVM AND CVXOPT SVMs

LIBSVM and CVXOPT are the packages that are used to solve convex optimization problems. We have used CVXOPT in our implementation whereas SVC of scikit-learn uses LIBSVM. For a comparison, we compare different metrics in our and SVC implementations.

Part (a):

Kernel Type	CVXOPT (nSV)	Sklearn (nSV)	Common SVs
Linear	381	372	372
Gaussian	306	298	298
Linear vs Gaussian (CVXOPT)	-	-	217
Linear vs Gaussian (Sklearn)	-	-	216

LIBSVM uses heuristics; shrinking and tolerance adjustments, leading to slightly fewer support vectors than CVXOPT. The Gaussian kernel results in fewer support vectors than the linear kernel because it creates more flexible decision boundaries, requiring fewer critical points.

Part (b):

Parameter	CVXOPT (Linear)	Sklearn (Linear)
$\|w\|$ (norm)	1.5342	1.5086
Bias (b)	-0.07469	-0.06606

The weight norm is slightly higher in CVXOPT (1.5342 vs. 1.5086) because of differences in optimization techniques and stopping criteria. The bias differs slightly due to variations in support vector selection and numerical precision in solving the dual problem.

Part (c):

Kernel Type	CVXOPT Accuracy (%)	Sklearn Accuracy (%)
Linear	85.53	85.21
Gaussian	95.50	95.50

The accuracies are almost same, which confirms consistency of the solvers

Part (d):

Kernel Type	CVXOPT Time (s)	Sklearn Time (s)
Linear	6.33	11.23
Gaussian	5.67	10.91

CVXOPT is faster because it directly solves the quadratic programming problem without additional heuristics. LIBSVM takes longer due to extra strategies like shrinking and caching. The Gaussian kernel trains slightly faster than the linear kernel as it requires fewer support vectors for decision boundary definition.

2.4 COMPARISON BETWEEN SGD AND LIBLINEAR

Although SGD finds only an approximate solution to the convex optimization problem, the convex formulation itself is already an approximation of the true underlying data distribution. This means that an exact solution to the convex problem (as obtained by LIBLINEAR) is not necessarily the best for generalization. In some cases, SGD's iterative updates introduce implicit regularization, helping it generalize better by avoiding overfitting to the training data. As a result, despite being an approximation, SGD can outperform LIBLINEAR, especially when the dataset is noisy; a well-tuned learning rate helps it converge to a more robust decision boundary.

Solver	Training Time (s)	Test Accuracy (%)
LIBLINEAR (Sklearn SVM)	11.23	85.21
SGD SVM	2.58	88.42

SGD is significantly faster than LIBLINEAR, reducing training time from **11.23s to 2.58s** by using an incremental optimization approach. Despite being an approximation, SGD achieves **better test accuracy (88.42%)**, likely due to implicit regularization effects that prevent overfitting and allow better generalization, especially on noisy data.

2.5 ONE-VS-ONE MULTICLASS IMAGE CLASSIFIER

Test set accuracy: 64.92%
The scoring has been done based on the number of votes, and if a tie, is decided by comparing the scores of the classes among all the classifiers

2.6 COMPARISON OF CVXOPT OPTIMISED ONE-VS-ONE MULTICLASS IMAGE CLASSIFIER VS SVC

Part (a):

Test Set Accuracy: 64.78%

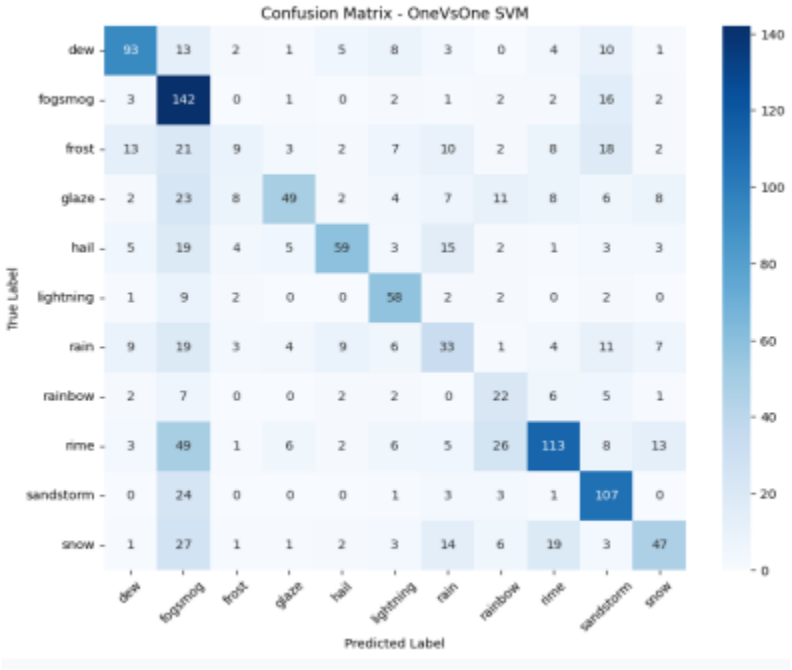
Part (b):

Solver	Training Time (s)	Test Accuracy (%)
CVXOPT (Gaussian Kernel)	467.3	64.92
Sklearn SVC (LIBSVM)	602.8	64.78

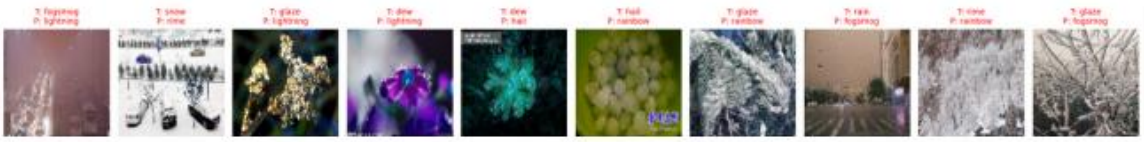
CVXOPT is faster because it directly solves the quadratic programming problem without additional heuristics. LIBSVM takes longer due to extra strategies like shrinking and caching. The **test accuracy is nearly identical** (64.92% vs. 64.78%), indicating that both solvers converge to a similar decision boundary despite different optimization techniques.

2.7 CONFUSION MATRIX ANALYSIS

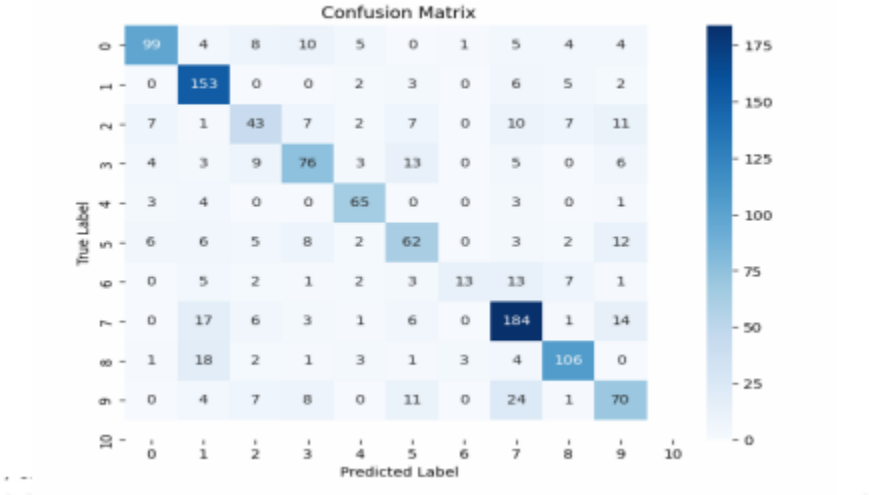
SVC Confusion matrix



Misclassified images for the SVC confusion matrix



CVXOPT Confusion matrix



Misclassified images for the CVXOPT confusion matrix

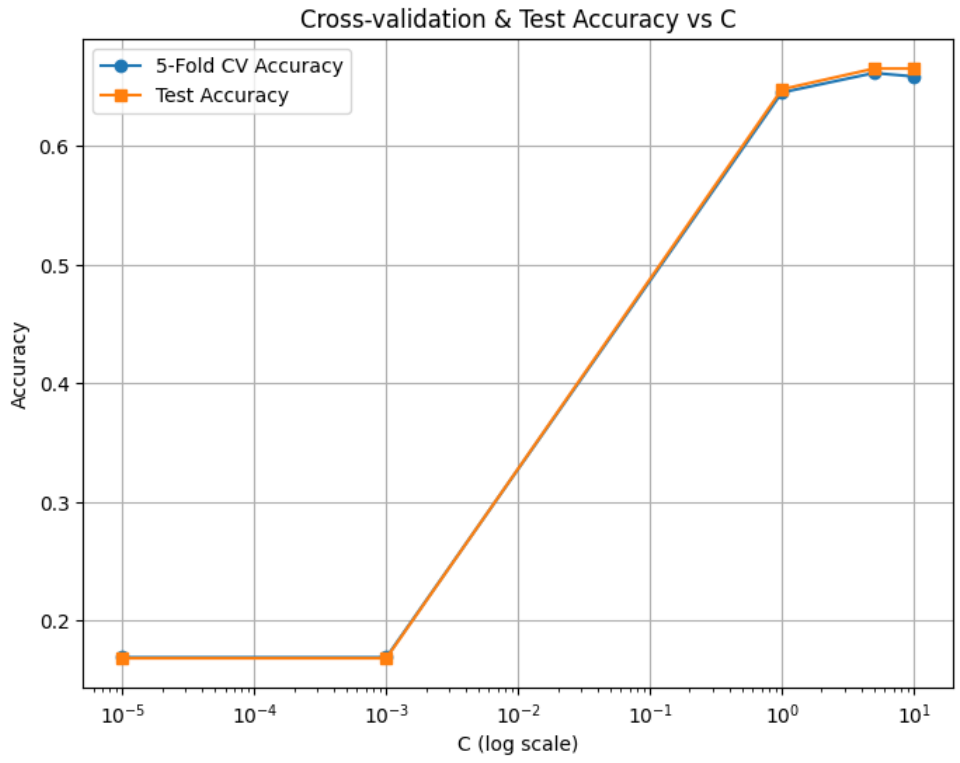
2.8 k-FOLD CROSS VALIDATION ANALYSIS

Part (a):

For very small values of C (1e-05, 0.001), the model underfits, leading to poor accuracy (~16.9%) as it prioritizes a large margin and ignores misclassifications. As C increases (1 to 5), the model finds a better balance between margin size and misclassification, improving accuracy significantly (~66.5%). However, at C = 10, accuracy does not improve further, suggesting diminishing returns or slight overfitting

C Value	CV Accuracy	Test Accuracy
1e-05	0.1693	0.1685
0.001	0.1693	0.1685
1	0.6451	0.6478
5	0.6615	0.6652
10	0.6586	0.6652

Part (b):



Best C from cross-validation accuracy: 5

Observations: When plotted, the graph of **C vs. Accuracy** shows a **sharp increase** in accuracy between **C = 0.001 and C = 1**, indicating that the model transitions from underfitting to a well-generalized state. As **C increases beyond 5**, accuracy plateaus, suggesting that larger values do not provide significant improvements. This implies that an **optimal C lies between 5 and 10**, where the model achieves its best performance without excessive complexity.

Part (c):

Final model test accuracy with C=5: 0.6652

Yes, the test accuracy with this value of C is an improvisation over the previous one.

With **C = 5**, the model achieves its highest test accuracy of **66.52%**, showing an improvement over lower C values. This suggests that the model effectively balances **margin maximization and misclassification tolerance**, reducing both underfitting and overfitting. Further increasing C to **10** does not provide additional gains, indicating that **C = 5** is a well-tuned choice for this dataset.