COL 774: Assignment 2 (Semester I, 2025-26)

Due Date: October 8, 2025. 11:59 pm, Total Points: 48 + 35.
Updated: September 26, 2025

Notes:

- This assignment has two main parts - text classification using Naïve Bayes (Part 1), and Classification using SVMs (Part II).
- You should submit all your code (including any pre-processing scripts written by you) and any graphs that you might plot.
- Do not submit the datasets.
- Include a single write-up (pdf) file which includes a brief description for each question explaining what you did. Include any observations and/or plots required by the question in this single write-up file.
- You should use Python for all your programming solutions.
- Your code should have appropriate documentation for readability.
- You will be graded based on what you have submitted as well as your ability to explain your code.
- Refer to the course website for assignment submission instructions.
- This assignment is supposed to be done individually. You should carry out all the implementation by yourself.
- We plan to run Moss on the submissions. We will also include submissions from previous years since some of the questions may be repeated. Any cheating will result in a zero on the assignment, a penalty of -10 points and possibly much stricter penalties (including a fail grade and/or a DISCO).
- For the implementation of Naive Bayes and SVM, we except you to build it from scratch using only libraries numpy and pandas (along with cvxopt for SVM). If there are any confusion on whether a library is allowed, feel free to ask on Piazza. We may penalize submission which uses a library outside the above permitted libraries and those approved on piazza for implementation. Note that you are free to use other libraries for analysis or pre-processing (like nltk and opencv).

1 (48 points) Text Classification using Naïve Bayes

In this problem, we will use the Naïve Bayes algorithm for text classification. You have been provided with the dataset, with the training split containing 1,40,000 samples (10k for each category), and 35,000 sized test (2.5k for each category). We would like to learn a Naïve bayes model using the train set. Then, given an useen article from the test, the task is to predict the category of each article using the learned model. Data is available at this link.

In the dataset, each article is associated with a title and a content. We will begin by exploring various standard techniques used in text classification to enhance model performance. Since this dataset has two set of features (list of words in **title** and list of words in **content**), these methods will first be applied individually to a single set of feature (either the title or content). After identifying the best-performing models for each, we will combine their strengths to develop an optimal model that effectively utilizes both the title and content together.

1. (10 points) Implement the Naïve Bayes Multiclass classification algorithm to classify each sample into one of the given 14 categories. You should implement the model where for each word position in a document, we generate the word using a single (fixed across word positions) Multinoulli distribution.

   (a) Train the implemented Naïve Bayes Classifier using only the **content** text. Report the accuracy over the training as well as the test set.

   (b) Read about word cloud. Construct a word cloud representing the most frequent words for each class.

   Notes:

   - Make sure to use the Laplace smoothing for Naïve Bayes (as discussed in class) to avoid any zero probabilities.
   - You should implement your algorithm using logarithms to avoid underflow issues.
   - You should implement Naïve Bayes from the first principles and not use any existing Python modules.
   - We have provided a starter code class with a `fit()` method that takes in a pandas Dataframe containing a column with each entry containing a list of tokens. This class will be autograded.
   - It may be useful to write a "tokenizer" functions (e.g., a simple tokenizer is `.split()`) for various parts below, and simply call the above implemented class to fit models on the newly tokenized column.

2. (4 points) The dataset provided to you is in the raw format i.e., it has all the words appearing in the original set of articles. This includes words such as 'of', 'the', 'and' etc. (called stopwords). Presumably, these words may not be relevant for classification. In fact, their presence can sometimes hurt the performance of the classifier by introducing noise in the data. Similarly, the raw data treats different forms of the same word separately, e.g., 'eating' and 'eat' would be treated as separate words. Merging such variations into a single word is called stemming. Read about stopword removal and stemming (for text classification) online. **As earlier, you should perform this analysis on content text features.**

   (a) Perform stemming and remove the stop-words in the training as well as the validation data.

(b) Construct word clouds for both classes on the transformed data.

(c) Learn a new model on the transformed data. Report the validation set accuracy.

(d) How does your accuracy change over the validation set? Comment on your observations.

3. (4 points) Feature engineering is an essential component of Machine Learning. It refers to the process of manipulating existing features/constructing new features in order to help improve the overall accuracy of the prediction task. In this part, we will use word based bi-grams as features.

   Bigrams are word pairs created by combining two consecutive words in a sentence. For example, the phrase "Pizza is awfully good" would be tokenized into the following bigrams: ["Pizza is", "is awfully", "awfully good"]. Bigrams help capture contextual meaning, such as how the word "awfully" may have a negative connotation on its own but contributes to a positive sentiment in the phrase "awfully good."

   Train a model that utilizes both unigrams (individual words) and bigrams as features, ensuring that preprocessing from part (2) is applied beforehand. After training, compare the model's performance in terms of training and test accuracy against the previous model to assess any improvements. **As earlier, you should perform this analysis on content text features.**

4. (2 points) Analyze the performance of different models to identify which one works best for classifying based on the content text (e.g., a unigram vs bigram model, model with/without stemming and/or stopword removal, etc.). Justify your selection using relevant performance metrics such as accuracy, precision, recall, F1-score, or any other evaluation criteria.

5. (10 points) Evaluating the Best Model for Title Features

   - Follow the same procedures outlined in parts 1, 2, and 3 & 4 above but this time only using the set of features (words) in **title**.
   - How does the accuracy obtained using best combination of title features compare with the accuracy obtained using (best combination of) content features? Comment on your observations.

6. (7 points) Now, we will explore how to develop models that incorporate both the title and content. Based on the best-performing model identified in the previous analysis, apply the same tokenization approach to these features. For example, if a bigram model without preprocessing from part (2) performed best for the content, tokenize the content using unigrams and bigrams from the raw text. Similarly, ensure that the title is tokenized according to the optimal approach determined earlier.

   (a) (3 points) To begin, we will ensure that our model learns the same set of parameters θ for both the title and content. This approach is equivalent to "concatenating" the two set of features into a single text representation and training our classifier on the merged text. After training, report the accuracies and compare with previous models using single set (title/content) of features.

   (b) (4 points) Now, we will allow the model to learn different parameters for title and content, $\theta^{(title)}$ and $\theta^{(content)}$. Mathematically compute the best fit expression for these parameters using the maximum likelihood estimation (remember to include Laplace smoothing). Report the accuracies and compare with previous models using single set (title/content) of features. Also, how does the accuracy compare with a model using joint set of features, but using a simple concatenation (as in the part above)?

7. (3 points) Analyze the performance of your current best model compared to very simple baselines by performing the following steps:
   (a) What is the validation set accuracy that you would obtain by randomly guessing one of the categories as the target class for each of the articles (random prediction)?
   (b) What accuracy would you obtain if you simply predicted each sample as positive?
   (c) How much improvement does your algorithm give over the random/positive baseline?
8. (3 points) Read about the confusion matrix. Explore the confusion matrix for the best model obtained so far:
   (a) Draw the confusion matrix for your best performing model (using both sets of features).
   (b) For each confusion matrix, which category has the highest value of the diagonal entry? What does that mean?
9. (5 points) As part of the feature engineering process, identify and create at least one additional set of features that could enhance your model's performance. Retrain the best-performing model obtained so far by including the newly engineered feature(s). Evaluate whether the inclusion of this feature leads to an improvement in accuracy. Compare the updated results with the previous model's performance and provide insights on the impact of the new feature.

2 (35 points) Image Classification using SVM

In this problem, we will use Support Vector Machines (SVMs) to build a binary image classifier. We will be solving the SVM optimization problem using a general purpose convex optimization package CVXOPT as well as using a scikit-learn library function based on a customized solver known as LIBSVM.

The dataset consists of $20,000$ images across 10 different classes. We have split the data into test and train, which can be accessed from the link: Q2_data. Each class has its own folder containing images. Before training the SVM, some **pre-processing** steps must be applied. Each image must then be resized to 32×32 pixels. Since SVMs require numerical feature vectors, each RGB image of size $32 \times 32 \times 3$ should be flattened into a one-dimensional vector of length 3072. As is the general practice for images, perform a min-max scaling for normalization (scale range $[0, 255]$ into $[0, 1]$ by dividing by 255). Once pre-processing is complete, the SVM classifier will be trained using both approaches, and their performance will be compared to evaluate the effectiveness of each method.

(18 points) Binary Classification: Let d be the last 2 digits of your entry number. Take the subset of images for the classes d and $(d+1)$ mod 10) from the train/validation data provided to you (arranged alphabetically, i.e., `airplane` is 0) and perform the following experiments in the context of binary classification.

1. (8 points) Download and install the CVXOPT package. Formulate the SVM dual optimization problem with a linear kernel in a form that can be solved using the CVXOPT package. The objective function should be expressed in the standard quadratic programming form $\alpha^T P \alpha + q^T \alpha + c$ matrix where P is an $m \times m$ matrix (m being the number of training examples), q is an m-sized column vector and c is a constant. For your optimization problem, remember to use the constraints on α_i 's in the dual. Use the SVM formulation which can handle noise and use $C = 1.0$ (i.e. C in the expression $\frac{1}{2}w^T w + C * \sum_i \xi_i$). You can refer this link to get a working overview of cvxopt module and it's formulation.

(a) How many support vectors do you get in this case? What percentage of training samples constitute the support vectors?

(b) Calculate the weight vector w and the intercept term b and classify each of the examples in the test file into one of the two labels. Report the test set accuracy.You will need to carefully think about how to represent w and b in this case.

(c) Reshape the support vectors corresponding to the top-5 coefficients to get images of $32 \times 32 \times 3$ and plot these (as images). Similarly, reshape and plot the weight vector w.

2. (5 points) Again use the CVXOPT package to solve the dual SVM problem using a Gaussian kernel. Think about how the P matrix will be represented. Use $C = 1.0$ and $\gamma = 0.001$ (here, γ in $K(x, z) = \exp^{-\gamma * \|x-z\|^2}$) for this part.

   (a) How many support vectors do you get in this case as compared to the linear case above? How many support vectors obtained here match with the linear case above?

   (b) Note that you may not be able to explicitly store the weight vector (w) or the intercept term (b) in this case. Use your learned model to classify the test examples and report the test accuracy.

   (c) Reshape the support vectors corresponding to the top- 5 coefficients to get images of $32 \times 32 \times 3$ and plot these.

   (d) Compare the test accuracy obtained here with part (a).

3. (5 points) Repeat parts -(a) & (b) with the scikit-learn SVM function, which is based on the LIBSVM package.

   (a) Compare the nSV (Number of Support Vectors) obtained here with the first part for the linear kernel and the second part for the Gaussian kernel. How many of the support vectors obtained here match with the support vectors obtained in both these cases?

   (b) Compare weight (w), bias (b) obtained here with the first part for linear kernel.

   (c) Report the test accuracy for both linear and Gaussian kernel.

   (d) Compare the computational cost (training time) of the CVXOPT with the sklearn implementation in both the linear and Gaussian case.

(17 points) Multi-Class Image Classification: In this section, we will use the full subset of data provided in Question 2 to tackle a multi-class classification problem using Support Vector Machines (SVMs). For this task, we will utilize the Gaussian kernel to capture complex decision boundaries and improve classification performance.

5. (4 points) In class, we described the SVM formulation for a binary classification problem. In order to extend this to the multi-class setting, we train a model on each pair of classes to get $\binom{k}{2}$ classifiers, k being the number of classes (here, k = 10). During prediction time, we output the class which has the maximum number of votes from all the $\binom{k}{2}$ classifiers. You can read more about one-vs-one classifier setting at the following link. Using your CVXOPT solver from previous section, implement one-vs-one multi-class SVM. Use a Gaussian Kernel with $C = 1.0$ and $\gamma = 0.001$ (as earlier, γ in $K(x, z) = \exp^{-\gamma * \|x-z\|^2}$).

(a) Classify the test examples and report test set accuracy. In case of ties, choose the label with the highest score.

6. (3 points) Now train a multi-class SVM on this dataset using the scikit-learn SVM function, which is based on the LIBSVM package. Repeat part (a) using a Gaussian kernel with $\gamma = 0.001$. Use $C = 1.0$ as earlier.

   (a) Classify the test examples and report test set accuracy.

   (b) How do the test set accuracy and the training time obtained here compare with part (a) above?

7. (4 points) Draw the confusion matrix for both of the above parts CVXOPT and LIBSVM. What do you observe? Which classes are miss-classified into which ones most often? Visualize (and report) 10 examples of misclassified objects. Do the results make sense? Comment.

8. (6 points) The validation set is typically used to estimate the optimal value of model hyper-parameters, such as C in our SVM with a Gaussian kernel. This is done by randomly selecting a small subset of the training data as the validation set, training the model on the remaining training data, and then evaluating its performance on the validation set. For a more detailed introduction, you can refer to this video. You can check the correctness of your intuition by trying this test.

   A more systematic approach to hyper-parameter tuning is K-fold cross-validation, which is commonly used in practice. In this technique, the training data is divided into K equal parts (folds). Each fold is used as a validation set once, while the remaining K-1 folds are used for training. This process is repeated for a range of hyper-parameter values, and the hyper-parameters yielding the highest K-fold cross-validation accuracy are selected as the best. You can read more about cross-validation here [1] (see Section 1) for more details.

   For this problem, we will perform **5-fold cross-validation** to determine the optimal C **value for the Gaussian kernel SVM**. The **test data should remain untouched** throughout this process. We will use the **scikit-learn SVM** function to implement this approach.

   (a) Fix γ as 0.001 and vary the value of C in the set $\{10^{-5}, 10^{-3}, 1, 5, 10\}$ and compute the 5 -fold cross validation accuracy and the test accuracy for each value of C.

   (b) Now, plot both the 5-fold cross validation accuracy as well as the test set accuracy on a graph as you vary the value of C on x-axis (you may use log scale on x-axis). What do you observe? Which value of C gives the best 5 -fold cross validation accuracy?

   (c) Train an SVM classifier using the above found hyperparameter C (on the entire train set) and report the accuracy. Does this value of C improve accuracy from the previous modelaccuracy of the test set? Comment on your observations.

Submission Instructions

You can find the starter code along with the data (downloaded from kaggle for the purpose of this assignment) at Assignment2_starter_code. The directory structure is given below. Your implementations will be autograded, and thus we request you to follow the following instructions while submitting:

[1] These are from Andrew Ng notes posted on the course website, and the link is available only from the internal IIT Delhi network. Feel free to read additional material online about this topic.

- You need to implement the functions mentioned in the starter code files (e.g. in `naive_baiyes.py`). You may add any other functions or methods, but do not change the signature (name and arguments) of the given functions.
- You may add new files for the analysis (for plotting etc.) for each question in their respective folders. You may use a jupyter notebook or python file - whichever you prefer. Note that any and all code used by you (for plotting etc.) needs to submitted.
- While submitting, make sure you follow the same directory structure. You may add other helper files in the Q1 and Q2 directories.

```
Assignment2/
├── report.pdf
├── Q1/
│   └── naive_baiyes.py
└── Q2/
    └── svm.py
```