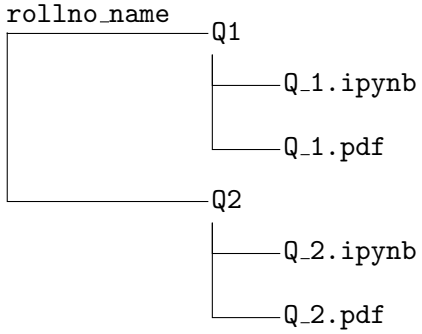
Programming Assignment: Support Vector Machines (SVM) with Regularization, Kernel Tricks, Overfitting & Underfitting, Probability, Naive Bayes

Submission deadline: 06-Nov-2023, 11:55 PM

Aryan Singh, Shrinivas Khiste, Biplab Roy, Yashkumar Shaileshbhai Paneliya

Instructions

1. The submission deadline is hard. There may be unforeseen glitches during submission. So, for safety, submit your files well ahead.
2. All submissions should be on microsoft teams only. No email submission will be accepted. Special consideration may be made for medical emergencies.
3. Directory structure to be submitted:
   - Question 1: SVM Implementation
   - Question 2: Naive Bayes



4. Please submit a Jupyter Notebook file (.ipynb) and a PDF file with cell outputs, Organize your notebook with clear sections and subsections corresponding to different parts of the question. Include your code and explanations within these sections.

   Ensure that you add discussion notes wherever required, especially for interpreting results, discussing observations, and explaining your approach.

5. After implementing the required machine learning models, you must submit only the <rollno>_<name> directory. No need to submit the dataset directory. The name of the zip file should be your roll number, followed by an underscore, followed by your name. For example, if your roll number is 22CS0100 and your name is John Doe, then the zip file should be named 22CS0100_JohnDoe.zip. Failing this, your assignment will not be evaluated.

6. You should not use any code available on the Web. Submissions found to be plagiarised will be awarded zero marks for all the students concerned.

7. All the theoretical/mathematical questions need to be answered in the report. You may also solve the problems in the notebook and share the snap in the report. Question 1 and Question 2 should be distinguishable in the report.

Objective

In this programming assignment, you will implement and experiment with Support Vector Machines (SVM) to classify a dataset. You will explore the concepts of regularization, kernel tricks, overfitting, and underfitting, and their impact on the performance of SVMs. By the end of this assignment, you should have a better understanding of how to fine-tune SVM models for different datasets. In the second part of the assignment, we will be using Naive Bayes. We will understand about importance of data preprocessing from Naive Bayes's Point of View.

Dataset

You will work on the task of Email Spam Classification. In this dataset, an email has been represented by various features like frequency of occurrences of certain keywords, length of capitalized words etc. The dataset contains 4601 instances, each with 57 attributes and a spam label. You can read more about the dataset here.

Here is an example code to load the dataset. First install *ucimlrepo* using 'pip install ucimlrepo'.

```
from ucimlrepo import fetch_ucirepo
import pandas as pd

# fetch dataset
spambase = fetch_ucirepo(id=94)

# data (as pandas dataframes)
X = spambase.data.features
y = spambase.data.targets

# metadata
print(spambase.metadata)

# variable information
print(spambase.variables)

# loading as dataframe
x = spambase.data.features
y = spambase.data.targets
```

Libraries

Please use NumPy, Pandas, Matplotlib/Seaborn, and Scikit-Learn libraries for data manipulation, visualization, and SVM implementation in your Python Notebook. Ensure these libraries are properly imported for efficient code execution and clear visualization. Notice that Scikit-Learn is still restricted for Naive Bayes Algorithm Implementation.
You can use the scikit-learn's SVC class for SVM implementation as demostrated below:-

```
# Import Scikit learn
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

# Load a Dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=42)

# Train SVM Model
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)

# Get predictions
y_pred = svm_model.predict(X_test)
```

Tasks

Question 1: SVM Implementation

Part A: SVM Implementation (40 Marks)

Implement a basic SVM classifier using a linear kernel. The workflow for the implementation can be as follows: (Note that you can use built-in libraries for the implementation)

1. **Data loading**: Load the dataset and split it into training and testing sets (80% for training, 20% for testing).
2. **Training the SVM**: Vectorise the data into X and y vectors for the features and the label respectively. Fit the SVM model on the training data and report the accuracy.
3. **Prediction and Evaluation**: Implement functions to generate predictions on the test set and calculate accuracy, precision, recall, and F1-score for the SVM model.
4. **Regularisation**: Regularisation is necessary to overcome overfitting. Vary the regularisation parameter of the SVM and tabularise and plot the accuracy. You can use the following values: [0.001, 0.1, 1, 10, 100]

(Optional: You can also observe the changes in the parameter values (by observing the mean, maximum, etc.) as the value of the regularisation parameter is changed.)

Part B: Kernel Tricks (30 Marks)

The "kernel trick" in Support Vector Machines (SVMs) is a clever mathematical technique that allows SVMs to operate in a higher-dimensional space without explicitly calculating the transformations. This not only saves computational resources but also opens up a variety of usage scenarios by enabling SVMs to handle different types of data distributions and complexities. In this part of the assignment, we will observe the impact of various kernels on the performance of the model.

Report the accuracy, precision, recall and F1 score on the test set using the following kernels:

1. Polynomial with degree 2
2. Polynomial with degree 3
3. Sigmoid
4. Radial Basis Function (RBF)

Part C: Overfitting & Underfitting Analysis (30 Marks)

In this part, we will explore the impact of kernel complexity and regularization in SVMs on model performance, and the presence of underfitting or overfitting.

1. Reuse training and testing datasets from Part A.
2. Train SVM models with different degrees of the polynomial kernel and different 'C' values as mentioned in Table 1, Plot and tabulate the test and train accuracy to showcase overfitting and underfitting.

Experiment	Polynomial Degree	Regularization Parameter 'C'
1	1	0.01
2	1	100
3	3	0.01
4	3	100

Table 1: Suggested Experiments with Polynomial Degree and 'C' Values

Question 2: Naive Bayes

Part A: Probability

There is a biased k-faced die, numbered 1 to k. The probability that the upward face is i from a random roll is $\frac{1}{2^{i-1}}$ for $2 \leq i \leq k$ and probability that upward face is 1 is $\frac{1}{2^{k-1}}$. So, $P(1) = P(k) = \frac{1}{2^{k-1}}, P(2) = \frac{1}{2}, P(3) = \frac{1}{4}$ ans so on.

1. Consider $k = 4$ and randomly roll the die 4 times and calculate the sum of the upward face value. Repeat this task 1000 times and plot a frequency distribution histogram. Print the five-number summary of the distribution. Show that the theoretical Expected sum of the event is close to the actual sum you got in the Python program simulation.
   If X is a random variable and x_i are the outcomes of the variable. Then Expected value of the random variable:
   $$E[X] = \sum_{x_i} x \cdot P(X = x_i)$$

2. Consider $k = 4$ and randomly roll the die 8 times and calculate the sum of the upward face value. Repeat this task 1000 times and plot a frequency distribution histogram. Print the five-number summary of the distribution. Show that the theoretical Expected sum of the event is close to the actual sum you got in the Python program simulation.

3. Repeat (a) for k = 16 in python simulation. (Do not need to show mathematical calculation)

Part B: Implementation of Naive Bayes (From Scratch)

1. **Dataset:** We will be using the same dataset of Email Spam Classification that we used in SVM. In this dataset, an email has been represented by various features like frequency of occurrences of certain keywords, length of capitalized words etc. The dataset contains 4601 instances, each with 57 attributes and a spam label. You can read more about the same dataset here.

2. **Loading Dataset:** Load the data with a 70:15:15 split for train, validation, and testing (You may use sklearn for this part).

3. **Plot Distribution:** Choose some 5 columns from the dataset and plot the probability distribution. You may choose any bins and boundaries to make the plotting understandable.

4. **Priors:** Calculate and print the priors of classes.

5. **Train Model:** Implement the Naive Bayes algorithm from scratch (preferably object-oriented implementation with fit and predict function, this will make your later questions easier to handle). Also, mention the total number of parameters needed to be stored for the model.

6. **Prediction and Evaluation:** Implement functions to generate predictions on the test set and calculate accuracy, precision, recall, and F1-score for the Naive Bayes model.

7. **Log Transformation:** Apply log transformation to all the columns of the dataset. Then again train the Naive Bayes Classifier and do the evaluations the same as earlier. (Note: Train/Test splits remain the same)

8. **Discuss:** Explain the changes you noticed in the results before and after modifying the dataset.

Part C: Implemention of Naive Bayes (sklearn)

1. **Train the model:** Import **GaussianNB** from sklearn.naive_bayes. Train the model with the actually loaded dataset and again after log transformation. For now, keep both models and their validation data and test data.

2. **High precision:** Draw a ROC curve for two models you got previously. As we understand the importance of emails, we don't want not spam mail classified as spam (However very little error is acceptable). Choose one best model from the ROC curve and continue with this for the next questions. (Note: Do not use any function to plot ROC directly, you have to plot it using plt.plot function) You may take a look at the required function from **here**.

3. **Compare Accuracy:** Compare and discuss the accuracy of Naive Bayes and SVM.