Support Vector Machines and Ensembling
ELL409 Assignment 3

Deadline - 11:50 PM, 8th November, 2024

1 Preliminaries

Please take note of the following:

1. This is a comprehensive assignment, so it is advisable to start early as the deadline **will not** be extended under any circumstances.
2. **The assignment will be auto-graded**, so ensure that your code follows the function signatures provided in the starter code and that the output format for the prediction methods is consistent.
3. We have provided an auto-grading script to help you verify that your code will run correctly on our servers.
4. A 'requirements.txt' file has been provided. During evaluation, we will use the library versions specified in this file. If you need to use any other non-standard library, please raise a query on Piazza.

2 Problem Statement

In this assignment, you will be exploring Support Vector Machines (SVM) and ensemble learning techniques. All implementations are to be done from scratch, and starter code has been provided to assist you.

3 Dataset Description

The dataset provided consists of multiple classes. The last digit of your entry number will determine the positive examples for your training data. You must consider the (last_digit + 1)%10, (last_digit + 2)%10, and (last_digit − 1)%10 classes as your negative examples. For each label, we have provided training and validation data. The dataset directory is structured as follows:

```
Dataset
├─ Train
│  ├─ 0
│  │  ├─ 0.png
│  │  ├─ 1.png
│  │  └─ ...
│  ├─ 1
│  ├─ ...
│  └─ 9
├─ Val
└─ Test
```

The test data contains dummy data to help you test the autograder script.

For example, if the last digit of your entry number is 6, you will use images in the folder named ’6’ (in each of the ‘train‘, ‘val‘, and ‘test‘ directories) as your positive examples. The folders named ’5’, ’7’, and ’8’ will contain your negative samples.

Another example: If the last digit is 9, the positive samples will be in the folder named ’9’, while the negative samples will be in the folders named ’8’, ’0’, and ’1’.

The dataset can be found here

4 Tasks to Implement

Part 1 - Support Vector Machines

In the first part of the assignment, you will implement Support Vector Machines (SVMs) and conduct an analysis on their performance. You are required to implement SVMs from scratch. To solve the SVM optimization problem, use the general-purpose convex optimization package CVXOPT.

To express the SVM dual problem in a form that CVXOPT can handle, you need to cast it as a quadratic programming problem:

$$\text{minimize: } \frac{1}{2}\alpha^T P \alpha + q^T \alpha$$

subject to the constraints:

$$G\alpha \leq h$$

$$A\alpha = b$$

For an SVM with a kernel (linear or RBF) and noise handling (i.e., a soft-margin SVM), the dual problem can be reformulated as:

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

subject to:

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

where:

- α is the vector of dual variables
- y_i is the label of the i-th training example
- x_i is the i-th training example
- $C = 1.0$ is the regularization parameter

Your implementation should support both linear and RBF kernels (refer to the starter code for more details).

Analysis

Hyperparameter Tuning
You are not allowed to use ‘scikit-learn‘ for hyperparameter tuning. Implement the following strategies yourself:

1. Grid Search
2. Randomized Search

Use the validation dataset for hyperparameter tuning.

Run your implementation for both soft and hard margin SVMs (set C close to 0 for hard margin). For soft-margin SVMs, tune the regularization parameter C.

Run both soft and hard margin SVMs with linear and RBF kernels, performing hyperparameter tuning specific to each case. Add all hyperparameters to ‘config.py‘ under the relevant sections.

Report

Include the following in your report:

- Number of support vectors for each case.
- Transform the top 6 support vectors in each case back to their original image dimensions and plot them in 2D.
- Training accuracy, F1-score, and number of misclassified instances, along with images of any 4 misclassified instances.

- Best validation F1-score for each case (must also be added to ‘config.py‘, or it will receive a 0 score).
- For the best validation F1-score, report accuracy and images of any 4 misclassified instances.
- Hyperparameters used to achieve the best F1-score (include them in ‘config.py‘).

Discrepancies between the reported and actual validation scores will result in severe penalties.

Part 2 - Ensembling

In this part, you will explore ensemble learning methods. Implement Bagging and AdaBoost from scratch, as taught in class. You may reuse your decision tree implementation from Assignment 2.

Your implementation must be completed in ‘ensemble.py‘. Any hyperparameters used must be tuned, reported, and added to ‘config.py‘.

Analysis

Hyperparameter Tuning
As with SVMs, do not use ‘scikit-learn‘ for hyperparameter tuning. Implement your own versions of:

1. Grid Search
2. Randomized Search

Use the validation dataset for hyperparameter tuning.

Optimize parameters like the number of trees, maximum depth, and minimum splits to improve performance.

Report

Include the following in your report:

- Training accuracy, F1-score, and number of misclassified instances, along with images of any 4 misclassified instances.
- Best validation F1-score for each case (must also be added to ‘config.py‘, or it will receive a 0 score).
- For the best validation F1-score, report accuracy and images of any 4 misclassified instances.
- Hyperparameters used to achieve the best F1-score (include them in ‘config.py‘).

Discrepancies between the reported and actual validation scores will result in severe penalties.

5 Grading Criteria

Scoring

1. Report: 30 Points (6 * 5)
2. Each of the six experiments will be graded based on results from the test set. If the maximum score for a part is A, your score will be $A \times \min(1, (\frac{\text{F1-score}}{\text{Exp-F1}})) \times \text{val-penalty}$. The value of expected-F1(Exp-F1) will be hidden and will be released along with assignment grades.
3. The val-penalty is calculated as: $\min(1, \exp(-5 \times |\text{reported} - \text{calculated}| - 0.1))$.
4. The maximum scores for test data evaluation are:
   (a) Hard-Margin-Linear: 4
   (b) Hard-Margin-RBF: 6
   (c) Soft-Margin-Linear: 6
   (d) Soft-Margin-RBF: 9
   (e) Bagging: 10
   (f) Boosting: 10
5. Implementation and quality of code : 25
6. Total Score: 100

6 Time Constraints

The grading script should not take more than 15 minutes to train and evaluate all six methods. You may use multiprocessing, but the number of processes must not exceed 20.

Please try to optimise the optimisation for SVM as convex optimisation can take a lot of time. Kernel functions should be given a lot of attention as at times they could end up being the bottleneck.

7 Submission Instructions

: The following files need to be submitted

1. Report.pdf: implementation details, analysis details, hyperparamtere tuning details, plots etc.
2. svm.py : Containing your implementation for Support Vector Machines with both kernels.
3. ensembling.py: Containing your implementation for Bagging Decision Trees and Boosting Decision Trees.

4. config.py: configuration with best hyperparamters for each experiment and the validation score

5. utils.py: Containing code for pre-processing and other utility functions you require.

6. analysis.py or analysis.ipynb: Code for analysis over the models you have implemented.

All the files need to be put inside a zip file named $\langle ENTRY\_NUMBER \rangle$.zip and need to be submitted on moodle.