Homework 7
Due: Thursday, April 4, 2019 at 12:00pm (Noon)

Programming Assignment

Introduction

In this assignment, you will be implementing both Support Vector Machines (SVMs) and Decision Trees to solve binary classification problems.

Support Vector Machines

While some modern solvers use gradient-based methods to train the SVM, you will be using a Python quadratic programming library, quadprog as an optimizer. The book sections relevant to this assignment are in chapter 15 on page 202, and chapter 16 on page 215.

Decision Trees

By the end of this assignment, you will have also implemented a generic Decision Trees class that you will use to predict the result of chess matches and classify spam emails.

Stencil Code & Data

You can find the stencil code for this assignment in the course filesystem (more directions on copying this over below). We have provided the following files:

- main.py is the entry point of your program, which will read in the data, run the classifiers and print the results. There are separate main.py files in the decision_trees and svm directories for the respective modules
- svm/svm.py contains the SVM model that you will be implementing the kernels for.
- svm/qp.py contains a quadprog wrapper function, solve_QP, which can efficiently solve quadratic programs.
- decision_trees/decision_tree.py contains the DecisionTree class. We have left most of the methods in this class for you to implement.
- decision_trees/get_data.py contains the data loading and processing. You do not need to change the contents of this file.

You should not modify any code in svm/main.py, decision_trees/get_data.py, or svm/qp.py. All the functions you need to fill in reside in svm/svm.py, decision_trees/main.py, and decision_trees/decision_tree.py, marked by TODOs. To run the SVM program, run python main.py <path_to_dataset> in a terminal from the svm directory. To run the decision trees program, you just need to run python main.py in a terminal from the decision_trees directory

Fake Datasets

We've provided two fake datasets, fake-data1.csv and fake-data2.csv, for you to train your SVM classifier with. Both datasets contain only two dimensional data so that you can easily plot the data if you like. The first dataset is linearly separable while the second is not. You should use the fake datasets for debugging purposes.
Spambase Dataset

You will also be testing your SVM classifier and Decision Trees on a real world dataset, the Spambase dataset. You can find more details on the dataset [here](#). We will only be using a subset of the full dataset. Our version is available along with the other two datasets on department machine at `/course/cs1420/data/hw7/spambase.csv`.

Chess Dataset

Each row of the `chess.csv` dataset contains 36 features, which represent the current state of the chess board. Given this representation, the task is to use the Decision Trees to classify whether or not it is possible for white to win the game. For more information on the dataset, see [here](#).

All of the data comes prepackaged with the stencil files. You can access these by copying the source code from the course directory, which can be done using the following command:

```bash
cp /course/cs1420/homeworks/hw7/stencil/* <DEST DIRECTORY>
```

where `<DEST DIRECTORY>` is the directory where you would like to copy the stencil to. If you are working locally, you will need to use the `scp` command to copy the files to your computer over `ssh`:

```bash
scp <login>@ssh.cs.brown.edu:/course/cs1420/homeworks/hw7/stencil/* <DEST DIRECTORY>
```

All datasets are available as CSV files and are located in their respective `data` folder. We have taken care of all the data preprocessing required so that you can focus on implementing the machine-learning algorithms!
The Assignment

SVMs as Quadratic Programs

Recall that the hard margin SVMs can be trained by solving a quadratic program. In lecture, we showed that we can formulate the problem in the following way.

$$\max_{\alpha \in \mathbb{R}^n} \left( \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} (2y_i - 1)(2y_j - 1)\alpha_i \alpha_j \langle x_i, x_j \rangle \right).$$

subject to $\alpha \geq 0$

However, in the assignment, you will be dealing with data that isn’t necessarily linearly separable, so we must solve a similar, yet different, quadratic program. It is the dual of the soft-margin SVM classifier. The corresponding quadratic program has the same objective function but also has the following additional constraints.

subject to $\sum_{i=1}^{n} \alpha_i (2y_i - 1) = 0$ and $0 \leq \alpha_i \leq \frac{1}{2n\lambda}, \forall i.$

Kernels

We can use the kernel trick to implicitly transform the data to higher dimensions. Two popular kernels for SVMs are the polynomial kernel and the radial basis function kernel. The kernel function replaces the inner product, so you can replace $\langle x_i, x_j \rangle$ with $K(x_j, x_k)$, which takes a different form depending on the kernel used:

- **Linear Kernel** (the same as the dot product):
  $$K(x_j, x_k) = x_j^T x_k.$$

- **Polynomial Kernel**:
  $$K(x_j, x_k) = (x_j^T x_k + c)^d.$$

- **Radial Basis Function Kernel**:
  $$K(x_j, x_k) = \exp\left(-\gamma \|x_j - x_k\|^2\right).$$

Note that $c$, $d$ and $\gamma$ are all hyperparameters here. That is, the kernels will behave differently depending on the values you use for these parameters. For example, as $d$ increases, the polynomial kernel can represent more flexible decision boundaries. We provide you with hyperparameter values that perform well, and do not expect you to change them; however, feel free to experiment yourselves.

Training and Classification

After solving for the various values of $\alpha$ via the quadratic program detailed above, you can calculate the bias $b$ by finding an index $i$ such that $0 < \alpha_i < \frac{1}{2n\lambda}$ (notice that these are strict inequalities, unlike the constraints) and then computing

$$b = w \cdot x_i - (2y_i - 1) = \left( \sum_{j=1}^{n} \alpha_j (2y_j - 1) K(x_i, x_j) \right) - (2y_i - 1)$$

Important Note: When finding an $\alpha_i$ that satisfies this strict inequality, we recommend using `np.isclose` with a absolute tolerance of `1e-3` as opposed to `<` or `>` due to floating point imprecision.
Then, to classify a data point \( z \), calculate the following expression:

\[
c = w \cdot z - b = \left( \sum_{i=1}^{n} \alpha_i (2y_i - 1)K(x_i, z) \right) - b
\]

If \( c > 0 \), the example should be classified as 1 and 0 otherwise.

**Quadprog**

We will be using the Python library `quadprog` to solve quadratic programs. You do not need to call any of the methods provided by `quadprog` since we have written a wrapper function `solve_qp`. However, you will still need to make sure that `quadprog` is installed on the machine where you are running your code.

If you are on a department machine, we have installed `quadprog` in our virtualenv. If you are running locally, you should run `pip3 install quadprog`. You can test that `quadprog` is successfully installed by running the example quadratic program located in `qp.py`.

Recall that quadratic programs follow the formulation

\[
\text{minimize} \quad \frac{1}{2} x^T Q x + c^T x \\
\text{subject to} \quad A x \leq b \text{ and } E x = d.
\]

Note: The \( b \) we use here is different from the \( b \) that denotes the bias of the SVM above.

We’ve written a wrapper around all of the `quadprog` functions, so you just need to worry about the kernels! Your writeup will deal with exploring how these different kernels perform.
Decision Trees

Part I: Generic Decision Trees in Python

In this part, you will be implementing a generic decision tree for binary classification given binary features. Your decision tree will take training data $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$—where $x_i \in \{0, 1\}^d$ represents the binary feature space and $y \in \{0, 1\}$ are the class labels—and attempt to find a tree that minimizes training error. Recall that training error for a hypothesis $h$ is defined as

$$L_S(h) = \sum_{(x, y) \in S} (y \neq h(x)).$$

The primary methods of the `DecisionTree` class are as follows:

- `DecisionTree(data, max_depth, validation_data=None, gain_function=train_error)` creates a `DecisionTree` that greedily minimizes training error on the given dataset. The depth of the tree should not be greater than `max_depth`. If `validation_data` is passed as an argument, the validation data should be used to prune the tree after it has been constructed.

- `predict(features)` predicts a label $y \in \{0, 1\}$ given features $\in \{0, 1\}^d$. Note that in our implementation features are represented as Python `bool` types (`True`, `False`) and class labels are Python `ints` (0, 1).

- `loss(data)` computes the average loss, $L_{\text{data}}(h)/\text{len(data)}$.

- `accuracy(data)` computes accuracy, defined as $1 - \text{loss(self, data)}$.

- `print_tree()` prints the tree to the command line. We have provided a working implementation, which you are free to improve. The current tree visualization works best for very shallow trees.

- `loss_plot_vec(data)` returns a vector of loss values where the $i$-th element corresponds to the loss of the tree with $i$ nodes. The result can be plotted with `matplotlib.pyplot` to visualize the loss as your tree expands.

Most of the algorithmic work will actually take place in the helper functions, each beginning with an underscore: `_predict_recurs`, `_prune_recurs`, `_is_terminal`, `_split_recurs`, `_calc_gain`. You should implement these functions without changing the function signatures.

Your task is to ensure that the `DecisionTree` class is fully implemented. If you are unsure where to begin, we have provided `TODO` comments in the stencil code to help get you started!

We recommend testing your code incrementally. It would be easiest to program `_is_terminal`, `_calc_gain` and one of the gain functions first as they are all needed in `_split_recurs`. You should start working on pruning at the last step when you are sure that other functions work. You are free to write your own tests for any of the provided functions to ensure that they are working correctly.

Part II: Measures of Gain

As mentioned in lecture, there are multiple measures of gain that an algorithm can use when determining on which feature to split the current node. In this assignment, you will be implementing and comparing the results of three measures of gain: decrease in training error, information gain (entropy) and Gini index. We recommend reviewing the lecture slides or textbook if these terms sound unfamiliar.

The `DecisionTree` class takes an optional `gain_function` parameter. This function will be one of the three functions left for you to implement: `train_error`, `entropy` and `gini_index`. 

---

5
Part III: Chess Predictions & Spam Classification

Once you have implemented the `DecisionTree` class, you are ready to explore the chess and spam datasets! You should now write code in `main.py` that will print the following loss values:

- For each dataset (`chess.csv`, `spam.csv`)
  - For each gain function (Training error, Entropy, Gini)
    * Print training loss without pruning
    * Print test loss without pruning
    * Print training loss with pruning
    * Print test loss with pruning

Your final program should print **exactly** 24 lines of output. Each line may contain text, but should end with the loss values defined above.
Project Report

SVM: Guiding Questions

- Comment on the testing and training accuracy of your SVM classifier on the Spam dataset. Discuss how kernels (and their hyperparameters) affect the classifier’s accuracy.

- (Extra Credit) Plot the differences in training and testing error as you change the hyperparameters of the polynomial and RBF kernel.

- (Extra Credit) Produce a visualization of the decision boundaries on the both of the 2D datasets (fake-data1.csv, fake-data2.csv). Your plot should contain the training data and decision boundaries. In addition, the label of each training point should be included. This should all be contained within a single figure for each dataset.

Decision Trees: Guiding Questions

- Comment on the results of your final program. Discuss the differences in training and test error of pruned and non-pruned trees. Which measure of gain most effectively reduced training error? Was pruning effective? Could you use graphs or numbers to explain what you have found?

- Using the spam.csv dataset, plot the loss of your decision tree on the training set for trees with maximum depth set to each value between 1 to 15. Discuss your findings in a paragraph.

- (Extra credit) In this assignment, you used a greedy algorithm in an attempt to minimize the training error of your decision tree. Think of another algorithm that could optimize the construction of your decision tree, implement it and compare the results. Include at least one graph.

Grading Breakdown

We will not be providing accuracy targets for this assignment. You should use the fake datasets that we have provided to ensure that your algorithm is working correctly.

<table>
<thead>
<tr>
<th>Component</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM Classifier</td>
<td>25%</td>
</tr>
<tr>
<td>Decision Tree Classifier</td>
<td>50%</td>
</tr>
<tr>
<td>Report</td>
<td>25%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>100%</strong></td>
</tr>
</tbody>
</table>