COL 774 - Machine Learning - Assignment 5

Due Date:
7:00 pm on Wednesday, 5th November, 2024 - Part 1 (SVM)
7:00 pm on Wednesday, 5th November, 2024 - Part 2 (Learning Oblique Decision Trees)
Max Marks : 100

Notes:

-
- All the announcements related to the assignment will be made on Piazza. The access code is wbd8avkblhb. You are strongly advised to have a look at the relevant Piazza post regularly.
- This assignment has two parts: 5.1 – SVM and 5.2 – Decision Trees.
- You are advised to use vector operations using numpy or scipy (wherever possible) for best performance.
- You should use Python 3 only for all your programming solutions.
- Your assignments will be auto-graded on our servers, make sure you test your programs with the provided evaluation scripts before submitting. We will use your code to train the model on a different training data set and predict on a different test set as well.
- Input/output format, submission format and other details are included. Your programs should be modular enough to accept specified parameters.
- You may submit the assignment **individually or in groups of 2**. No additional resource or library is allowed except for the ones specifically mentioned in the assignment statement. If you still wish to use some other library, please consult on piazza. If you choose to use any external resource or copy any part of this assignment, you will be awarded D or F grade or **DISCO**.
- *Graceful degradation:* For each late day from the deadline, there will be a penalty of 7%. So, if you submit, say, 4 days after the deadline, you will be graded out of 72% of the weightage of the assignment. Any submission made after 7 days would see a fixed penalty of 50%.
- For doubts, use Piazza.

1 Support vector machines (25 points)

Release date: Oct. 23, 2024, Due date: Nov. 5, 2024

Assignment Statement

In this part, we will try to classify a binary (2-class) dataset using Support Vector Machines. You will have to implement the objective equation and constraints and finally use the LP solver available in the **cvxpy** library to find the decision boundary.

We will use a different objective function here as compared to the original.In the vanilla SVM, the L2 norm of weights is used, however we will modify it slightly to use the **L1 norm of weights** instead.
So the modified objective function and constraints are :

$$\min_{w,b,\xi} \quad \frac{1}{2}\|w\|_1 + C\sum_{i=1}^{n}\xi_i \tag{1}$$

$$\text{subject to:} \quad y_i(w \cdot x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i \tag{2}$$

Your task is to predict whether a given dataset is linearly seperable, and to find the support vectors in the case where the dataset is linearly seperable. We will always use $C = 1$ in the objective function and a tolerance value of $1e^{-4}$ to find the support vectors that lie closest to the margin.

What have you been given?

- You are given with 2 datasets - **train_ls.csv** - a linearly seperable dataset and **train_nls.csv** - a non linearly seperable dataset. The last column is the class label for both the datasets.
- **weight_ls.json** and **weight_nls.json** containing the SVM weights and bias for the respective datasets
- **sv_ls.json** and **sv_nls.json** containing whether the dataset is linearly seperable and the support vectors (if the dataset is linearly seperable)

Format Information

- The *weight.json* files should contain 2 fields - *weights* and *bias.*
  A sample *weight.json* file should look like this -

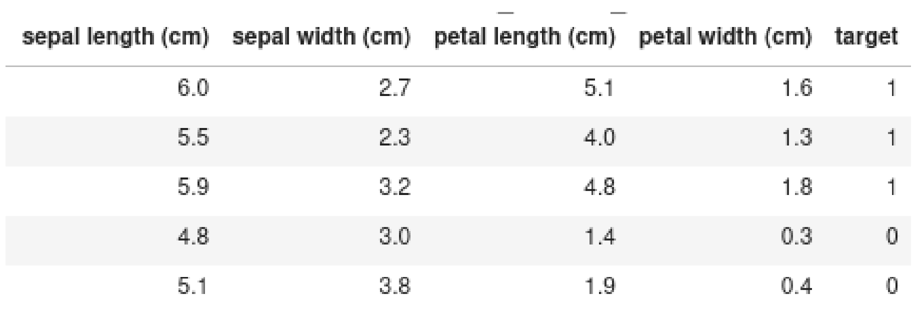
```
{
    "weights": [0.5, -1.2, 3.4, 2.0],
    "bias": 0.75
}
```

- The *sv.json* files should contain 2 fields - *seperable* and *support_vectors.*
  A sample *sv.json* file should look like this -

```
{
    "seperable": 0,
    "support_vectors": [1, 3, 5, 7]
}
```

  1. seperable should be 0 for non seperable and 1 for seperable dataset.
  2. support_vectors should be a list containing the index (starting from 0) of the items in the train dataset which are support vectors. Incase of a non seperable dataset it should be an empty list ([]).

- The *train.csv* file contains supervised training data. The target column contains the class.



sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
6.0	2.7	5.1	1.6	1
5.5	2.3	4.0	1.3	1
5.9	3.2	4.8	1.8	1
4.8	3.0	1.4	0.3	0
5.1	3.8	1.9	0.4	0

Figure 1: Head of the training dataframe

Submission Instructions

You need to submit a **svm.py** file. We will run the following command :

```
python svm.py train_anystring.csv
```

which should result in the making of **weight_anystring.json** and **sv_anystring.json** file (pay attention to the file names - here anystring can be any arbitrary string that we use to label the dataset) according to the format information given above.

---

2 Learning Oblique Decision Trees - (75 points)

Release date: Oct. 23, 2024, Due date: Nov. 5, 2024

Assignment Statement

In this part, we will try to implement **Oblique decision trees**. The vanilla decision trees that split based on a single feature are known as axis-aligned decision trees. Incase of oblique decision trees, we split based on a linear combination of features - To determine this linear combination we will use logistic regression. So at every node, you will run a logistic regression model on all the examples present at that node, to find the best combination of features that minimizes the cross entropy loss. Finally you will use the **Gini impurity** to find the best value of threshold to split the examples at that node. You should use sklearn for logistic regression. You should use the following :

```
model = LogisticRegression(solver='lbfgs')
```

This assignment is divided into 4 parts :

Part(a) In this part you will implement the oblique decision trees (as described in the assignment statement) that fully trains on the training data, to give a training accuracy of 100%. You should find out the minimum depth required to achieve this on your own. You should use the training dataset provided in **train_sample.csv** for this part. The file **weights_sample_unpruned.csv** has been provided so that you are able to verify the weights and thresholds you obtain on the provided dataset.

The decision tree class that you implement should have the following definition :

```
class ObliqueDecisionTree:
    def __init__(self, max_depth, min_samples_split=2):
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.tree = None
```

Part(b) Now, based on the validation set given, you need to prune the tree in a bottom up manner i.e starting from the leaves all the way to the root, based on whether it is better to remove the complete subtree under a node in order to improve the classification accuracy on the validation set. The training dataset is provided in **train_sample.csv** and the validation dataset is provided in **val_sample.csv** for this part. The file **weights_sample_pruned.csv** has been provided so that you are able to verify the weights and thresholds you obtain on the provided dataset.

Part(c) In this part you are provided with a real world dataset, in the files **train_real.csv**, **val_real.csv** and **test_real.csv** which contain the train, validation and test data respectively.

(i) You need to train the oblique decision tree completely (100% training accuracy with minimum possible depth) and submit the weights and thresholds in *weights_real_unpruned.csv* file.

(ii) You need to prune the tree obtained in part(i) using the validation data and submit the weights and thresholds of the pruned tree in *weights_real_pruned.csv* file.

(iii) You need to submit the predictions on the test dataset made using the pruned tree obtained in part(ii) in *prediction_real_pruned.csv* file.

Part(d) This is the **competitive** part of the assignment. You will work with the same dataset provided in part(iii). You are free to modify the objective function (logistic regression loss function as well as threshold deciding function) as well the pruning algorithm here, so that the decision tree does not overfit. You are also allowed to modify the decision tree in any way, adding more arguments in this part. So, it is advised that you make a separate decision tree class for this part. You are however not allowed to change the input features of the decision tree, in other words, feature engineering is not allowed. In this part you will submit 2 files -

1. *prediction_real_competitive.csv* - The file containing the predictions on the test dataset provided in part(iii).
2. *weights_real_competitive.csv* - The file containing the weights and thresholds of the final tree used to predict on the test dataset.

What have you been given?

- **train_sample.csv, val_sample.csv** - Sample dataset to check the correctness of the implementation at your end
- **train_real.csv, val_real.csv, test_real.csv** - Real world dataset used by us to check the correctness of your implementation and to be used as dataset in the competitive part
- **weights_sample_unpruned.csv** - File containing the decision tree weights for the unpruned decision tree on the sample dataset
- **weights_sample_pruned.csv** - File containing the decision tree weights for the pruned decision tree on the sample dataset

Format Information

- The *weights.csv* files that you submit must contain on each line, the node number/id, the node weights/-coefficients for the different features (in the same order as in the original dataset), the threshold, seperated by commas. Only the non leaf nodes should be included and written in the file.

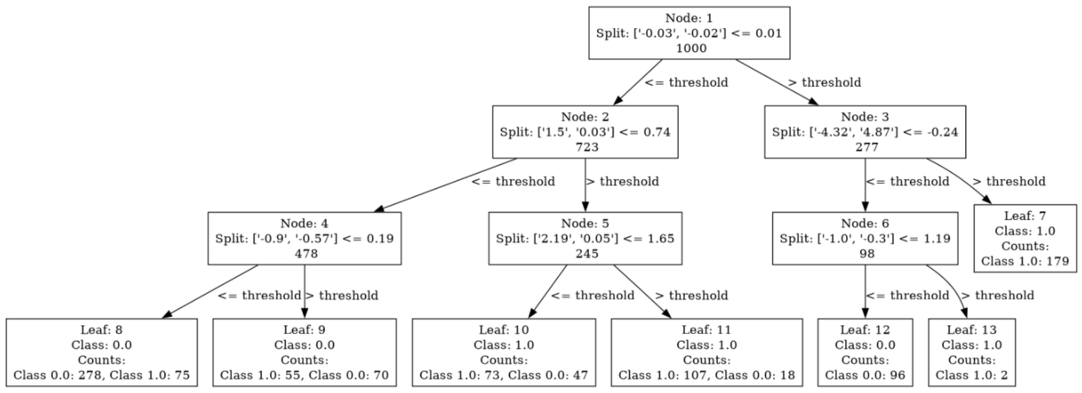


Figure 2: Sample decision tree

The above image shows a sample oblique decision tree and how the nodes should be numbered. The root node should be numbered 1. The children of any node with node id x are labeled $2x$ (left child) and $2x + 1$ (right child).

For example, the first line in the weights.csv file for the above tree should be -

```
    1, -0.03, -0.02, 0.01
```

- The *prediction.csv* files should contain the class labels predicted for the test data instances. Do ensure that you dont shuffle the test data when making predictions and write the labels in the *prediction.csv* file in the same order in which the instances were provided in the input file.

Submission Instructions

Your final submission directory for this part should be as follows :

```
\EntryNo1_EntryNo2
    weights_real_unpruned.csv
    weights_real_pruned.csv
    prediction_real_pruned.csv
    weights_real_competitive.csv
    prediction_real_competitive.csv
    decision_tree.py
    comp_decision_tree.py
```

We will run the following commands:

```
python decision_tree.py train unpruned train.csv 8 weight.csv
python decision_tree.py train pruned train.csv val.csv 8 weight.csv
python decision_tree.py test train.csv val.csv test.csv 8 prediction.csv
python comp_decision_tree.py test train.csv val.csv test.csv prediction.csv weight.csv
```

- The first command should train the unpruned tree using train.csv and should save the weights and thresholds of the resulting tree in the weight.csv file
- The second command should train the pruned tree using train.csv and val.csv and should save the weights and thresholds of the resulting tree in the weight.csv file

- The third command should train the pruned tree using train.csv and val.csv and should save the predictions made on the test.csv file in the prediction.csv file
- The fourth command should train the modified decision tree that you implemented for the competitive part of the assignment using train.csv and val.csv and should save the predictions made on the test.csv file in the prediction.csv file, and the weights of the final tree in the weights.csv file.

References

1. Murthy, S. K., Kasif, S., Salzberg, S. (1994). A system for induction of oblique decision trees. Journal of artificial intelligence research, 2, 1-32.
2. Carreira-Perpinán, M. A., Tavallali, P. (2018). Alternating optimization of decision trees, with application to learning sparse oblique trees. Advances in neural information processing systems, 31.