Report

for

COMP 3314 Assignment 1

Prepared by Shunhua Guo, 3035447635

Logistic Regression & Random Forest

Discuss their implementations, obtained results, advantages and limitations of the implemented methods in their reports.

March 26, 2021

1. Implementations

The algorithms are implemented in Jupyter Notebook.
Required Libraries are specified in each notebook.

1.1. Logistic Regression

As the sigmoid function applies to binary classification cases while the Softmax function applies to multi-class classification problems, the Softmax function is implemented below.

```
def softmax(x):
    """
    input: x as a vector
    output: a vector
    """
    x_exp = [np.exp(i) for i in x]
    x_exp_sum = sum(x_exp)
    return [i / x_exp_sum for i in x_exp]
```

The structure of the implementation is as follows. The main function is 'fit' for training the regression using data input, which will run gradient descend for 'n_iter' times.

```
class LogisticRegressionMulti(object):
    def __init__(self, learning_rate=0.05, n_iter=1000,
                 regularization_param=0.01, regularization=2, validation=None):

    def fit(self, X, y):

        for i in range(self.n_iter):

            #gredient decend

        return self

    def net_input(self, X):

    def activation(self, z):

    # output the gradient for weight vector
    def gradient(self, output, X, y):

    #calculate gradient for single data point as flattened
    def compute_gradients(self, out, x, y):

    def loss(self, output, y):

    def predict(self, X):

    def accuracy(self, X, y):
```

The loss function is negative-log-probability, and gradient is the derivative of it.

$$L(\theta, D) = -logP(Y = y_i|X = x_i, W, b)$$

$$L(\theta, D) = -log\frac{e^{W_i x+b_i}}{\sum_j e^{W_j x+b_j}}$$

$$L(\theta, D) = -loge^{W_i x+b_i} - log\sum_j e^{W_j x+b_j}$$

$$L(\theta, D) = -W_i x + b_i + log\frac{1}{\sum_j e^{W_j x+b_j}}$$

1.2. Decision Tree

As stated in the lecture, the following impurity measures or splitting criteria are commonly used in decision trees:
1. Gini impurity (gini),
2. Entropy (entropy),
3. Classification error (classificationError).

And to record Decision Tree training result, I initialize a tree node structure.

```
class DTnode(object):
    def __init__(self, level=0):
        self.level = level

        self.x_index = None
        self.val = None
        self.left = None
        self.right = None
        self.c = None

    def update_info(self, x_index, val, left, right):

    def add_class(self, c):

    def get_info(self):
```

For the Decision Tree model, I implement splitting group in a recursively manner. The structure of the model class is as follows.

```
class DecisionTree(object):
    def __init__(self, criterion='gini', max_depth=4, min_size=1):

    def fit(self, X, y):

        return self

    #recursively split the group
    def split_node(self, node, group, max_depth):

    # record class classification at the node
    def assign_class(self, node, group):

    # select split
    def select_split(self, group):

    # compute information gain at a split point
    def info_gain(self, group, x_index, val):

    #define the impurity/error function called loss
    def loss(self, group):

    # ------------------- Evalution ------------------- #
    # predict single x input
    def predict(self, node, x):

    #compute model accuracy
    def accuracy(self, X, y):
```

1.3. Random Forest

The basic logic of random forest is implementing multiple decision trees, with sample bagging strategy.

```
class RandomForest(object):
    def __init__(self, criterion='gini', max_depth=4, min_size=1,
                 k_estimators=25, random_seed=42):
    # ------------------- Main ------------------- #

    def fit(self, X, y, n_sample_size=None, d_features=None):

        #initialize DT list
        self.trees = []
        for i in range(self.k_estimators):
            # set random seed, sample bagging strategy
            # selecting random 'row's in the dataset
            # masking non-seleted columns (set to all zeros)
            self.trees.append(self.return_DT(X_sampled, y_sampled))
        return self

    def return_DT(self, X_sampled, y_sampled):
        # call Decision Tree Model
    # ------------------- Evalution ------------------- #
    #predict single x sample
    def predict(self, x):

    # calculate accuracy of the model
    def accuracy(self, X, y):
```

2. Obtained Results

2.1. Logistic Regression on Iris Data

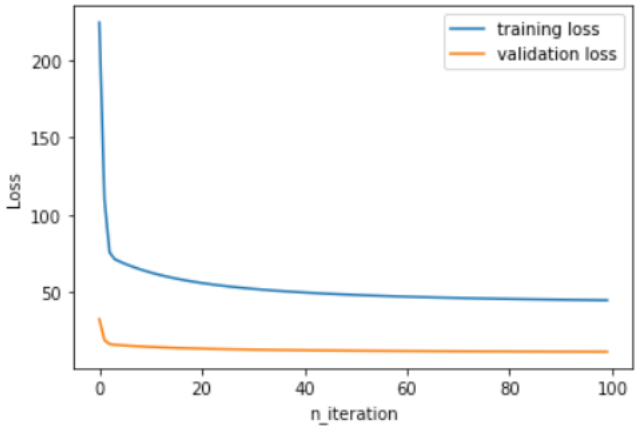
The following is an example training Logistic Regression on Iris Dataset.

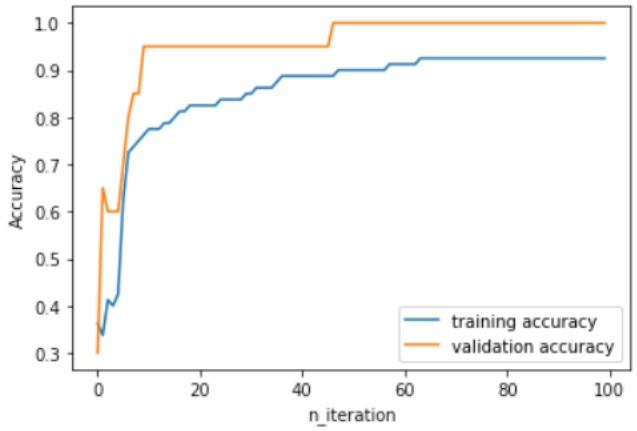
```
# Apply the model, and fit the training data
model = LogisticRegressionMulti(learning_rate=0.05, n_iter=100, regularization_param=0.01,
                regularization=2, validation=80)

model.fit(X_train, y_train)

#test accuracy result
model.accuracy(X_test, y_test) #0.98
```

And the below graphs are Loss and Accuracy plot for training and validation set.





2.2. Logistic Regression on Car Data

The following is an example training Logistic Regression on Car Dataset.

```
# Apply the model, and fit the training data
model = LogisticRegressionMulti(learning_rate=0.05, n_iter=50, regularization_param=0.01,
                regularization=2, validation=1000)

model.fit(X_train, y_train)

#test accuracy result
model.accuracy(X_test, y_test) # 0.7148362235067437
```

2.3. Random Forest on Iris Data

The following is an example training Decision Tree and Random Forest on Iris Dataset.

Single Decision Tree training result

```
tree = DecisionTree()
tree.fit(X_train, y_train)

print("training accuracy: ", tree.accuracy(X_train, y_train))
print("test accuracy: ", tree.accuracy(X_test, y_test))
```

```
training accuracy:  0.98
test accuracy:  0.98
```

Random Forest training result

```
forest = RandomForest(k_estimators=100)
forest.fit(X_train, y_train, d_features=3)

print("training accuracy: ", forest.accuracy(X_train, y_train))
print("test accuracy: ", forest.accuracy(X_test, y_test))
```

```
Reminder: n_sample_size is not specified here, assume to be total/5
training accuracy:  0.94
test accuracy:  1.0
```

2.4. Random Forest on Car Data

The following is an example training Decision Tree and Random Forest on Car Dataset.

Single Decision Tree training result

```
tree = DecisionTree()
tree.fit(X_train, y_train)

print("training accuracy: ", tree.accuracy(X_train, y_train))
print("test accuracy: ", tree.accuracy(X_test, y_test))
```

```
training accuracy:  0.8709677419354839
test accuracy:  0.8420038535645472
```

Random Forest training result

```
forest = RandomForest(k_estimators=100)
forest.fit(X_train, y_train, d_features=5)

print("training accuracy: ", forest.accuracy(X_train, y_train))
print("test accuracy: ", forest.accuracy(X_test, y_test))
```

```
Reminder: n_sample_size is not specified here, assume to be total/5
training accuracy:  0.8635235732009926
test accuracy:  0.859344894026975
```

3. Advantages and Limitations of the implemented methods

3.1 Logistic Regression

Good at handling data with linear separatable or almost characteristics.

Not able to handle higher-order data, like quadratic relationships.

3.2 Decision Tree

Easily overfit the training set, though we can apply pruning or early termination to reduce overfitting problem.
High variance of the model.

Interpretable.

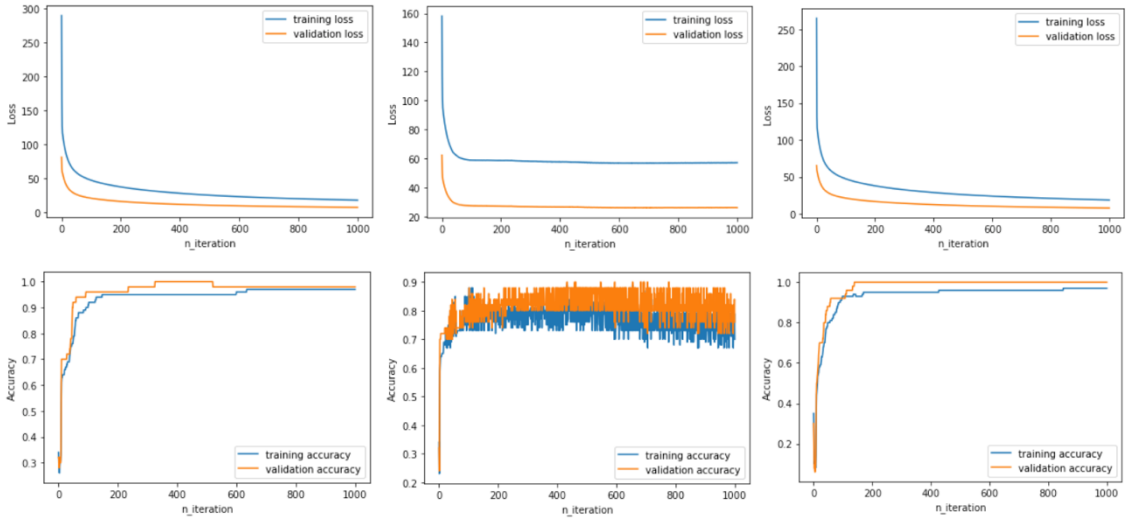
3.3 Random Forest

Can solve overfitting problem of Decision tree by average across multiple decision trees, and reduce the variance of the model.

4. Parameter analysis

4.1 Logistic Regression

A simple plotting of different **regularization term** result vs. number of iterations.

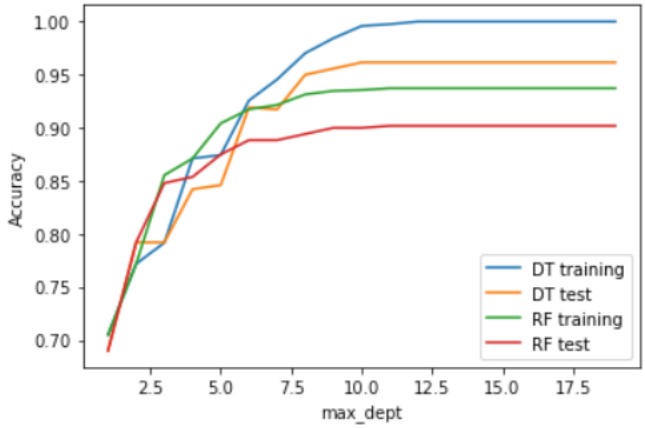


From the left to the right, they are No regularization, L1 regularization and L2 regularization cases.

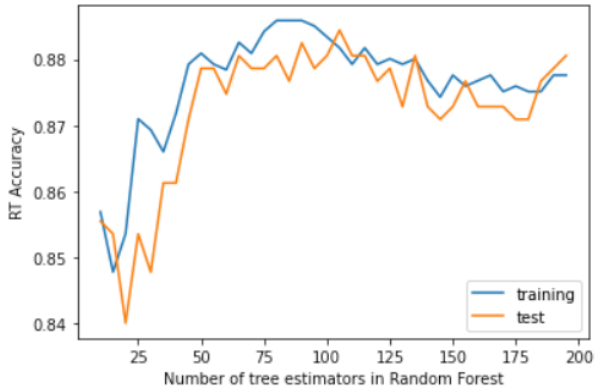
3.2 Decision Tree & Random Forest

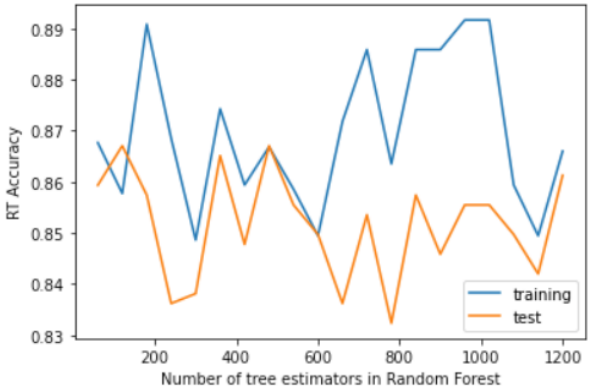
It is a heatmap plotting of the accuracy from each impurity or error function after training Car dataset.

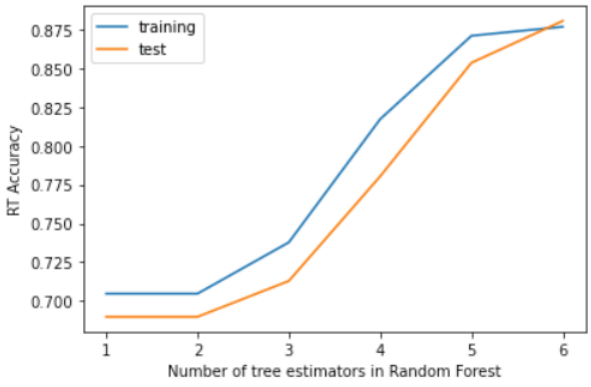
This is a graph studying Accuracy vs. max_depth of Decision Tree.



In the random forest model, I also studied the impact on accuracy of Number of tree estimators in Random Forest (k), Number of sample size in each sample bagging (n) and Number of featurses selected in each sample bagging (d).







5. References

4.1 Datasets

[1] Iris dataset
(url: https://archive.ics.uci.edu/ml/datasets/Iris)

The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. The 3 plants are Versicolour (class 0), Virginica (class 1) and Setosa (class 2) respectively.
There are 4 input attributes: sepal length, sepal width, petal length and petal width.
There are 100 training samples and 50 testing samples.

[2] Car evaluation dataset
(url: https://archive.ics.uci.edu/ml/datasets/Car+Evaluation)

This model evaluates cars according to their status and classifies them as unacceptable (class 0), acceptable (class 1), good (class 2) and very good (class 3).
The 6 input attributes are buying price, price of the maintenance, number of doors, number of persons to carry, size of luggage boot, safety of the car.
There are 1209 training samples and 519 testing samples.