Homework II

10/25/2019

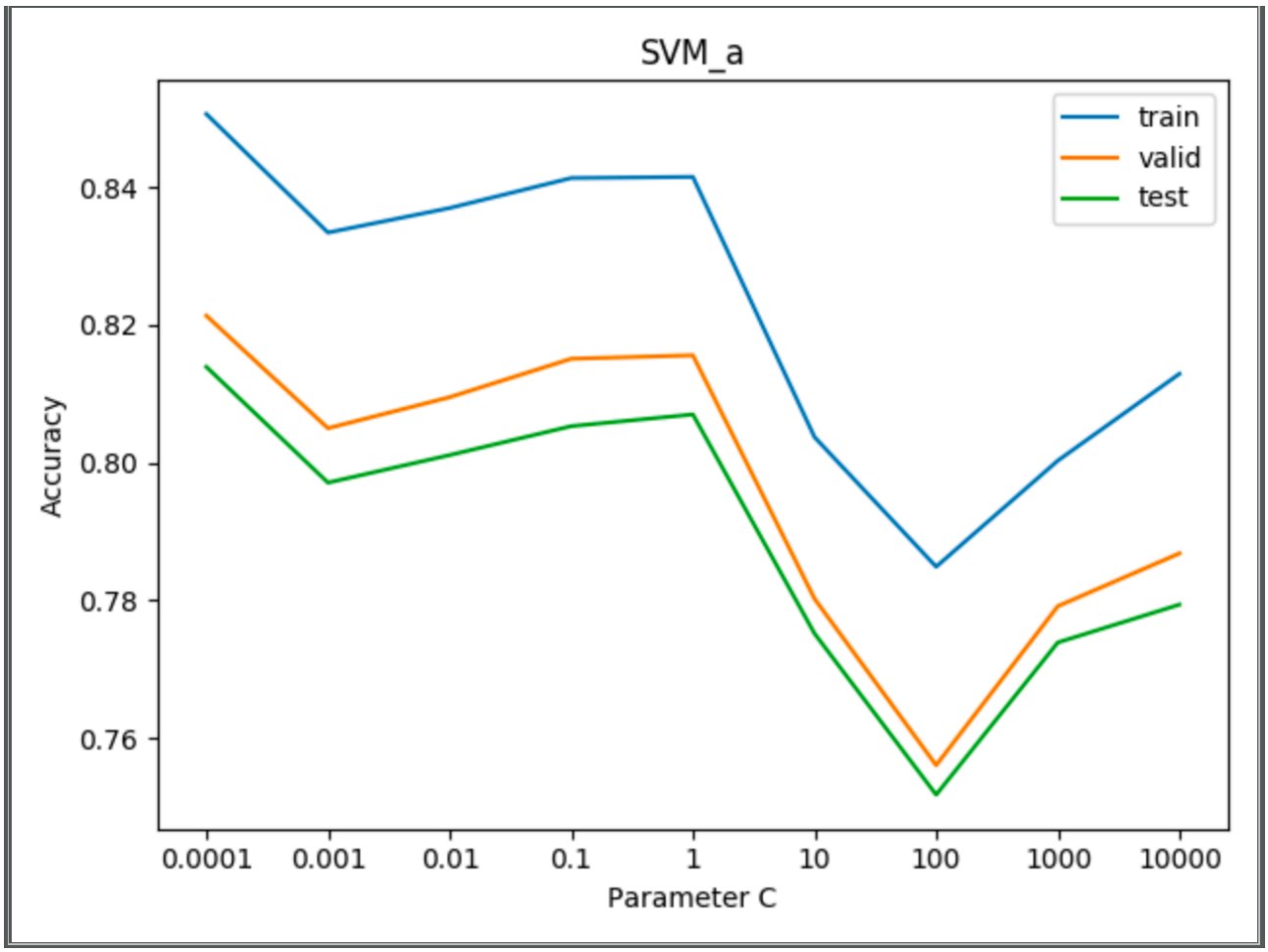
Instructor: Janardhan Rao (Jana) Doppa

Student: Yu-Chieh Wang 11641327

1. Programming Part

1. SVM

a. Compute the accuracy with C parameter: $10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}, 10^{4}$.



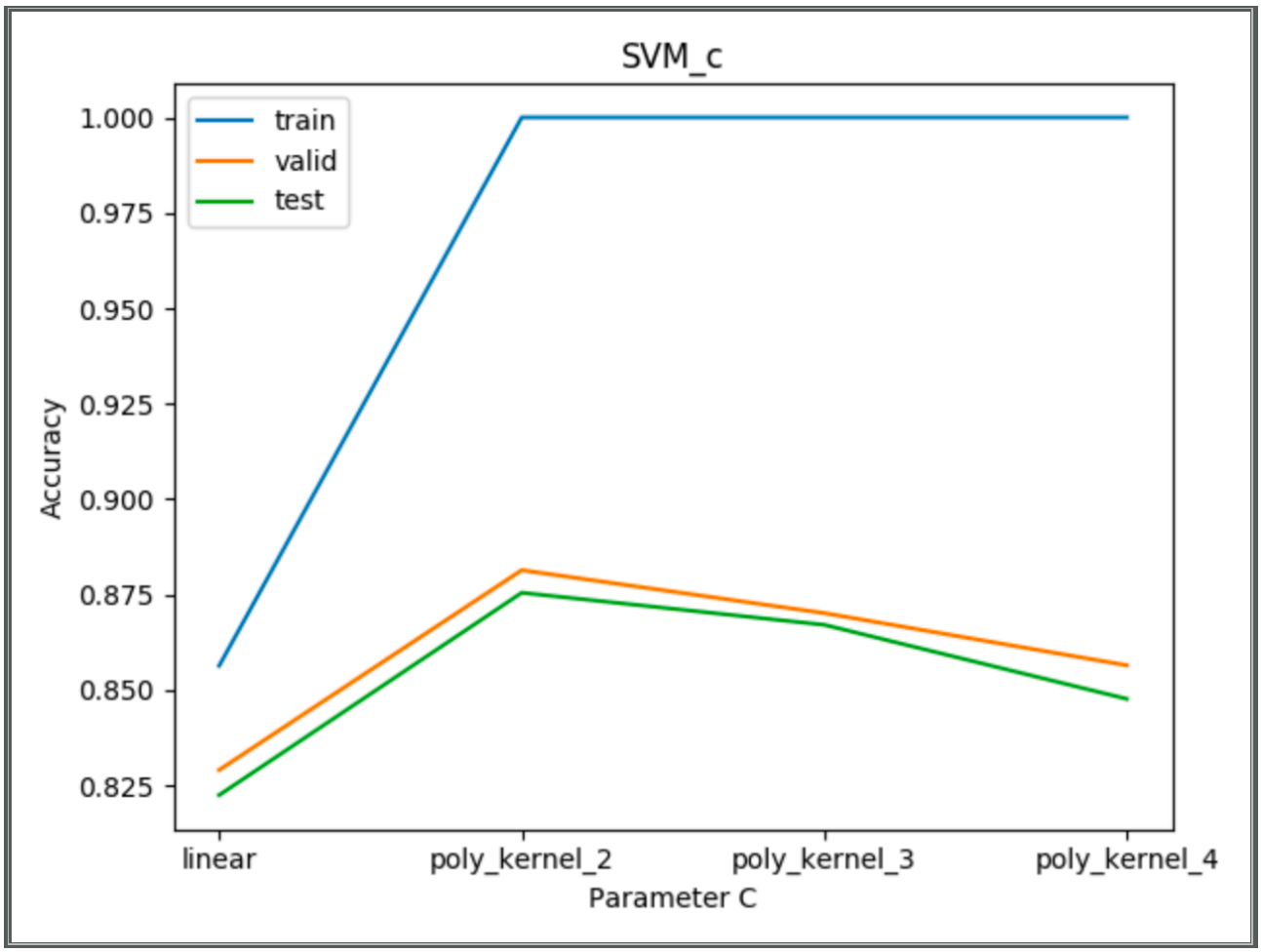
-According to the plot, we can get the highest accuracy when parameter $C = 0.0001$, and the lowest accuracy when $C = 10^2$. Also, the difference between training, validation and testing data with each parameter is less than 0.05.

b. Compute the testing accuracy and the corresponding confusion matrix(10 x 10).

-I try to modify the size in order to show the picture completely. However, no matter how I change the size of plot or figure, both of them don't work. Although I cannot find out the reason of it on the Internet, I will keep try to solve it.

-According to the picture, we can see how many examples have correct predictions, and how many examples has other results.

c. Compare the training, validation, testing accuracies in different kernels.



-In this plot, we can observe that the highest accuracy is based on the polynomial kernel with degree 2.

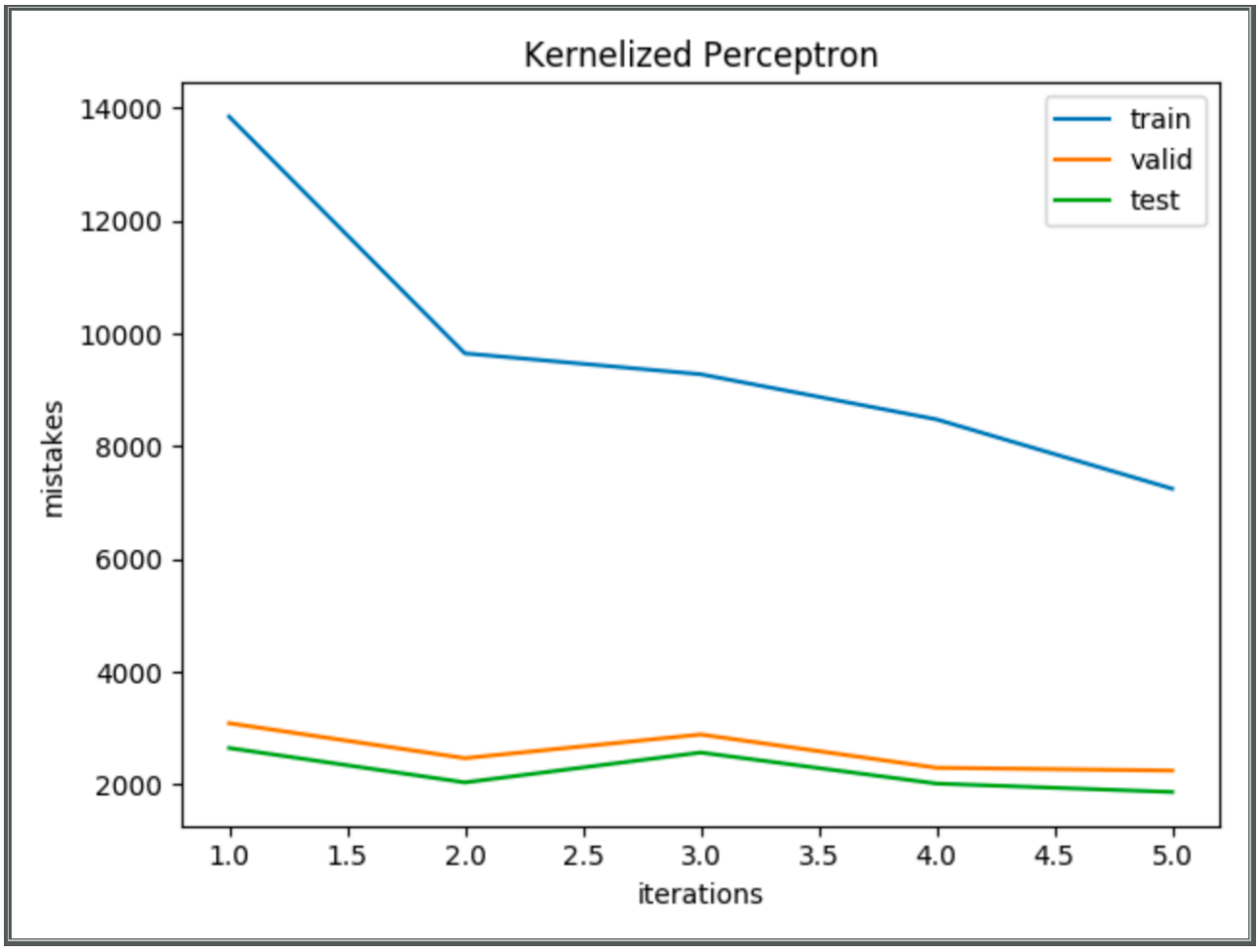
2. Multi-class Kernelized Perceptron

a. Train the classifier for 5 iterations with polynomial kernel of degree 2.

-In this part of questions, I meet a lot of problems during coding. At the beginning, the accuracy result I get is 0.1 which equal to the probability of guessing. Therefor, I use normalized function to fix training, validation, and testing data, which make the accuracy up to 0.63.

-The next problem is that I try to use a matrix to store the result of $K(x_i, x_n)$ because it will be quicker when runing 5 iterations. However, after the program have run 6 hours and almost complete the matrix, the computer warns that the program is going

to run out of the memory. So, in other to solve the problem, I comment the code for building matrixes, and run the computing every time when I need it. Nevertheless, I find it take too much time to finish and get the result. According to my rough estimate, it will take at least 20 hours for running. To solve this problem, I have to reduce all examples(training, validation, and testing) and cancel the standard normalized function. As the result, I run 6 hours to finish the program and get training and testing mistakes as following:



Iterations	1	2	3	4	5
Train	0.7116	0.7991	0.8068	0.8235	0.8491
Validation	0.7431	0.7948	0.7598	0.8090	0.8132
Test	0.7357	0.7968	0.7434	0.7998	0.8138

-The result shows an important thing: As the number of training iterations, the accuracy of each one also increases.

3. Decision Tree

a. Implement ID3.

```
data = np.genfromtxt('data/breast-cancer-wisconsin.data.txt', delimiter=',')
X_train = data[:490]
X_valid = data[490:560]
X_test = data[560:]
def countResult(rows): #count the number of each result(output)
    results = {}
    for row in rows:
        r = row[len(row)-1]
        if r not in results:
            results[r] = 0
        results[r] += 1
    return results #return a dic ex.{2:295, 4:195}


def entropy(rows):
    results = countResult(rows)
    #Get entropy H(x)
    ent = 0.0
    for r in results.keys(): #2 and 4
        p = float(results[r])/len(rows) #the probability of the result=2 or 4
        ent = ent - p*(log(p)/log(2)) #count H(x)
    return ent

class treeNode:
    def __init__(self, column=-1, value=None, results=None, trueBranch=None, falseBranch=None):
        self.column = column
        self.value = value
        self.results = results
        self.trueBranch = trueBranch
        self.falseBranch = falseBranch


def divideset(rows, column, value): #seperate the rows by the results
    set1 = [row for row in rows if row[column] >= value]
    set2 = [row for row in rows if row[column] < value]
    return(set1, set2)
```

-First of all, separate the data to two set by the label(2, 4) and then, start to build a tree: Check the entropy to find the best way to separate rows to two groups in each column. Finally, test the tree to get an accuracy.

```
def buildtree(rows):
    if len(rows) == 0:
        return treeNode() #rootNode
    best_gain = 0.0
    best_criteria = None
    best_sets = None


    for column in range(1, len(rows[0])-1):
        column_values = {}
        for row in rows:
            column_values[row[column]] = 1  # store every value of a column

        # find the highest gain of a column to separate data to two sets
        for value in column_values.keys():
            (set1, set2) = divideset(rows, column, value)
            #gain information
            p = float(len(set1)) / len(rows)
            gain = entropy(rows) - p * entropy(set1) - (1 - p) * entropy(set2)
            if gain > best_gain and len(set1) > 0 and len(set2) > 0:
                best_gain = gain
                best_criteria = (column, value)
                best_sets = (set1, set2)
    if best_gain > 0: # build branches
        return treeNode(column=best_criteria[0], value=best_criteria[1],
                        trueBranch=buildtree(best_sets[0]), falseBranch=buildtree(best_sets[1]))
    else:
        return treeNode(results=countResult(rows))

def classify(input, tree):

    if tree.results!= None:
        return tree.results
    else:
        if input[tree.column] >= tree.value:
            branch = tree.trueBranch
        else:
            branch = tree.falseBranch

        return classify(input, branch)
```

b. Run training examples to get the accuracy of validation and testing examples.

-The result is as following:

```
Accuracy of valid: 0.9571428571428572
Accuracy of test: 0.9496402877697842
```

-We can observe the accuracies of both validation and testing data are high!

c. Implement Decision Tree pruning algorithm.

```
def pruning(tree, value):
    # If the branch isn't a node, cut the branch
    if tree.trueBranch.results == None:
        pruning(tree.trueBranch, value)
    if tree.falseBranch.results == None:
        pruning(tree.falseBranch, value)
    # Check if branches can be combined
    if tree.trueBranch.results != None and tree.falseBranch.results != None:
        trueBranch, falsebranch = [], []
        for i, j in tree.trueBranch.results.items():
            trueBranch += [[i]]*j
        for i, j in tree.falseBranch.results.items():
            falsebranch += [[i]]*j
        delta = entropy(trueBranch + falsebranch) - (entropy(trueBranch) + entropy(falsebranch)/2)
        if delta < value: # If the difference is less than...combine two branches
            tree.trueBranch, tree.falseBranch = None,None
            tree.results = countResult(trueBranch + falsebranch)
```

-To cut branches which are useless and combine branches with the same result.

d. Run the pruning algorithm to get the accuracy of validation and testing examples, and compare the result with/ without the pruning algorithm.

```
New/old entropy difference is less than 0.5
Accuracy of valid: 0.9571428571428572
Accuracy of test: 0.9640287769784173
```

```
New/old entropy difference is less than 0.8
Accuracy of valid: 1.0
Accuracy of test: 0.9712230215827338
```

-According to the result, we can see that when we input a higher difference which we can accept, the accuracy will become higher. That means the pruning function works!