# Name: Shubham Sanghavi

# Net ID: SAS190003

CS 6375- Problem Set 2

Problem 1: SPAM, SPAM, HAM (50 pts)

1. Primal SVMs

## Code

spam\_data= importdata('spam\_train.data',',');

X\_1 = spam\_data(:,1:end-1);

Y = spam\_data(:,end);

Y = (Y - 0.5) \* 2;

spam\_valid\_data = importdata('spam\_validation.data',',');

xv\_1 = spam\_valid\_data(:,1:end-1);

yv = spam\_valid\_data(:,end);

yv = (yv - 0.5) \* 2;

spam\_test\_data = importdata('spam\_test.data',',');

xt\_1 = spam\_test\_data(:,1:end-1);

yt = spam\_test\_data(:,end);

yt = (yt - 0.5) \* 2;

% second degree polynomial of X

X = X\_1;

N = size(X,1);

% X = X(1:15,:)

c = [1,10,10^2,10^3,10^4,10^5,10^6,10^7,10^8];

%c = [1,10,10^2];

train\_acc = zeros(size(c));

val\_acc = zeros(size(c));

test\_acc = zeros(size(c));

O = ones(size(X,1),1);

A = [O X];

A = -Y.\*A;

b = - O;

si\_constraints= -1 \*eye(size(X,1));

new\_A = [A si\_constraints];

constr\_2 = zeros(size(A));

constr\_2 = [constr\_2 si\_constraints];

new\_A = [new\_A; constr\_2];

new\_b = [b;zeros(size(b,1),1)];

f = zeros(size(A,2),1);

H = eye(size(A,2));

H(1,1) = 0;

si\_rows = zeros(size(X,1),size(H,2));

new\_H = [H ; si\_rows];

si\_cols = zeros(size(new\_H ,1),size(X,1));

new\_H = [new\_H si\_cols];

for i = 1:size(c,2)

f\_lambda = c(i) \* ones(size(X,1),1);

f\_new = [f;f\_lambda];

[w,fval,exitflag,output,lambda] = quadprog(new\_H,f\_new,new\_A,new\_b);

disp("Done with quadprog")

weights = w(2:size(X,2)+1);

bias = w(1);

pred = sign((X \* weights) + bias);

diff = abs(Y - pred)/2;

accuracy = 1 - sum(diff)/size(X,1);

train\_acc(i) = accuracy;

% finding the validation accuracy

pred\_v = sign((xv\_1 \* weights) + bias);

diff\_v = abs(yv - pred\_v)/2;

val\_accuracy = 1 - sum(diff\_v)/size(xv\_1,1);

val\_acc(i) = val\_accuracy;

% finding the test accuracy

pred\_t = sign((xt\_1 \* weights) + bias);

diff\_t = abs(yt - pred\_t)/2;

test\_accuracy = 1 - sum(diff\_t)/size(xt\_1,1);

test\_acc(i) = test\_accuracy;

end

## Accuracy on the training set

## Accuracy on the validation set

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C | 1 | 10 | 100 | 1000 | 10^4 | 10^5 | 10^6 | 10^7 | 10^8 |
| Training Accuracy | 0.9446 | 0.9473 | 0.9486 | 0.9483 | 0.9483 | 0.9486 | 0.9480 | 0.9480 | 0.9480 |
| Validation Accuracy | 0.9350 | 0.9387 | 0.9387 | 0.9375 | 0.9375 | 0.9375 | 0.9362 | 0.9362 | 0.9362 |

## Accuracy for the best C on test set

|  |  |  |  |
| --- | --- | --- | --- |
| C | 1 | 10 | 100 |
| Testing Accuracy | 0.7078 | 0.6317 | 0.6217 |

# 2. Dual

A.Code

spam\_data= importdata('spam\_train.data',',');

X\_1 = spam\_data(:,1:end-1);

Y = spam\_data(:,end);

Y = (Y - 0.5) \* 2;

spam\_data= importdata('spam\_validation.data',',');

X\_V = spam\_data(:,1:end-1);

Y\_V = spam\_data(:,end);

Y\_V = (Y\_V - 0.5) \* 2;

% second degree polynomial of X

X = X\_1;

c = 1;

N = size(X,1);

lambda\_H = ones(N,N);

sigma\_all = [10,100,1000];

c\_all = [1,10];

store\_results = []

for sig\_id = 1: size(sigma\_all,2)

for c\_id = 1: size(c\_all,2)

sigma = sigma\_all(sig\_id);

c = c\_all(c\_id);

% find guassian for the given input and sigma

trans\_X = [];

for i = 1:N

gaus\_col = gaussian\_ss(X,X(i,:),sigma);

trans\_X = [trans\_X gaus\_col];

end

% replace this with the guassian kernel

Y\_H = Y \* Y.';

H = trans\_X .\* Y\_H .\* lambda\_H;

% f is just the simple lambda sum

f = -1 \* ones(N,1);

% the constraints:

A = [];

b = [];

Aeq = Y.';

Beq = 0;

lb = zeros(N,1);

ub = c \* ones(N,1);

[w,fval,exitflag,output,lambda] = quadprog(H,f,A,b,Aeq,Beq,lb,ub);

b\_s = [];

for j = 1:size(w,1)

if w(j) > 0.01 && w(j) < c-0.001

gaus\_res = gaussian\_ss(X,X(j,:),sigma);

eq = gaus\_res .\* Y .\* w;

wtx = Y(j)-sum(eq);

b\_s = [b\_s wtx];

end

end

b = mean(b\_s);

% training predictions

train\_predictions = zeros(size(X,1),1);

for i = 1:size(X,1)

gaus\_col = gaussian\_ss(X,X(i,:),sigma);

p\_eq = gaus\_col .\* Y .\* w;

val = sum(p\_eq)+b;

train\_predictions(i) = sign(val);

end

diff = abs(Y - train\_predictions)/2;

accuracy = 1 - sum(diff)/size(X,1);

diff = abs(Y - train\_predictions)/2;

accuracy = 1 - sum(diff)/size(X,1);

% validation predictions

val\_predictions = zeros(size(X\_V,1),1);

for k = 1:size(X\_V,1)

gaus\_col = gaussian\_ss(X,X\_V(k,:),sigma);

p\_eq = gaus\_col .\* Y .\* w;

val = sum(p\_eq)+b;

val\_predictions(k) = sign(val);

end

diff = abs(Y\_V - val\_predictions)/2;

val\_accuracy = 1 - sum(diff)/size(X\_V,1);

store\_results = [store\_results; c sigma accuracy val\_accuracy];

end

end

function rbf = gaussian\_ss(x\_i,x\_j,sigma)

if size(x\_j,1) ~= 1

disp("Size of x\_j must be 1")

end

vec = x\_i - x\_j;

vec = vec .^ 2;

mod\_x = sum(vec,2);

rbf\_in = mod\_x / (2\* sigma);

rbf = exp(- rbf\_in);

end

## B. Results for training set

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Sigma^2**  **(down)** | **C->**  **(right)** | **1** | **10** | **100** | **1000** | **10^4** | **10^5** | **10^6** | **10^7** |
| **0.1** | | 0.9997 | 0.9997 | 0.9997 | 0.9997 | 0.9997 | 0.9997 | 0.9996 | 0.9997 |
| **1** | | 0.9993 | 0.9997 | 0.9997 | 0.9997 | 0.9997 | 0.9997 | 0.9996 | 0.9997 |
| **10** | | 0.9803 | 0.9977 | 0.9997 | 0.9997 | 0.9997 | 0.9997 | 0.9996 | 0.9997 |
| **100** | | 0.8967 | 0.9770 | 0.9952 | 0.9980 | 0.9987 | 0.9997 | 0.9996 | 0.9997 |
| **1000** | | 0.8357 | 0.9267 | 0.9693 | 0.9853 | 0.9940 | 0.9963 | 0.9983 | 0.9993 |

## C. Results for the validation set

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Sigma^2**  **(down)** | **C->**  **(right)** | **1** | **10** | **100** | **1000** | **10^4** | **10^5** | **10^6** | **10^7** |
| **0.1** | | 0.1963 | 0.1975 | 0.1975 | 0.1975 | 0.1975 | 0.1975 | 0.1975 | 0.1975 |
| **1** | | 0.2525 | 0.2625 | 0.2625 | 0.2625 | 0.2625 | 0.2625 | 0.2625 | 0.2625 |
| **10** | | 0.6775 | 0.6963 | 0.6912 | 0.6875 | 0.6863 | 0.6863 | 0.6862 | 0.6863 |
| **100** | | 0.7763 | 0.8375 | 0.8450 | 0.8488 | 0.8488 | 0.8375 | 0.8375 | 0.8375 |
| **1000** | | 0.7600 | 0.8500 | 0.9013 | 0.9038 | 0.8975 | 0.8950 | 0.8962 | 0.8938 |

## D. Report Test set results for the best sigma and C

Sigma^2 = 1000

C = 1000

**Test Accuracy = 0.7154**

# 3. KNN

Code:

import pandas as pd

import numpy as np

def knn\_classifier(X\_test,k,X\_train,Y\_train,distance\_func):

# apply to each row

dist = np.apply\_along\_axis(distance\_func, 1, X\_train, X\_test)

# pick the k-closest

idx = np.argpartition(dist, k)[:k]

# will count the number of True examples in the idx selected by the selection

pos = np.sum(Y\_train[idx])

# check if more than half of the k examples selected are positive

if pos > (k/2):

Y\_test = True

else:

Y\_test = False

return Y\_test

def euclidean(vec1,vec2):

vec = (vec1 - vec2) \*\* 2

ans = np.sum(vec)

return np.sqrt(ans)

def get\_mean\_std(in\_array):

return (in\_array.mean(),in\_array.std())

def normalize\_arr(in\_array,mean,std):

return (in\_array-mean)/std

if \_\_name\_\_ == '\_\_main\_\_':

df\_train = pd.read\_csv('spam\_train.data', sep=',',header=None)

df\_dev = pd.read\_csv('spam\_validation.data', sep=',',header=None)

df\_test = pd.read\_csv('spam\_test.data', sep=',',header=None)

# prepare the numpy arrays for faster predictions

X\_train = df\_train.iloc[:,:-1].values

Y\_train = df\_train.iloc[:,-1].values

X\_dev = df\_dev.iloc[:,:-1].values

Y\_dev = df\_dev.iloc[:,-1].values

X\_test = df\_test.iloc[:,:-1].to\_numpy()

Y\_test = df\_test.iloc[:,-1].to\_numpy()

mean\_std = np.apply\_along\_axis(get\_mean\_std,axis=0,arr=X\_train)

X\_train\_norm = np.empty\_like(X\_train)

for col in range(np.shape(X\_train)[1]):

X\_train\_norm[:,col] = normalize\_arr(X\_train[:,col],mean\_std[0,col],mean\_std[1,col])

X\_dev\_norm = np.empty\_like(X\_dev)

for col in range(np.shape(X\_dev)[1]):

X\_dev\_norm[:,col] = normalize\_arr(X\_dev[:,col],mean\_std[0,col],mean\_std[1,col])

X\_test\_norm = np.empty\_like(X\_test)

for col in range(np.shape(X\_test)[1]):

X\_test\_norm[:,col] = normalize\_arr(X\_test[:,col],mean\_std[0,col],mean\_std[1,col])

results = []

k\_values = [1,5,11,15,21]

for k in k\_values:

# apply to each row in our X\_dev

train\_pred = np.apply\_along\_axis(knn\_classifier, 1, X\_train\_norm,k,X\_train\_norm,Y\_train,euclidean)

train\_acc = np.sum(train\_pred==Y\_train)/np.shape(Y\_train)

val\_pred = np.apply\_along\_axis(knn\_classifier, 1, X\_dev\_norm,k,X\_train\_norm,Y\_train,euclidean)

val\_acc = np.sum(val\_pred == Y\_dev)/np.shape(Y\_dev)

test\_pred = np.apply\_along\_axis(knn\_classifier, 1, X\_test\_norm,k,X\_train\_norm,Y\_train,euclidean)

test\_acc = np.sum(test\_pred == Y\_test)/np.shape(Y\_test)

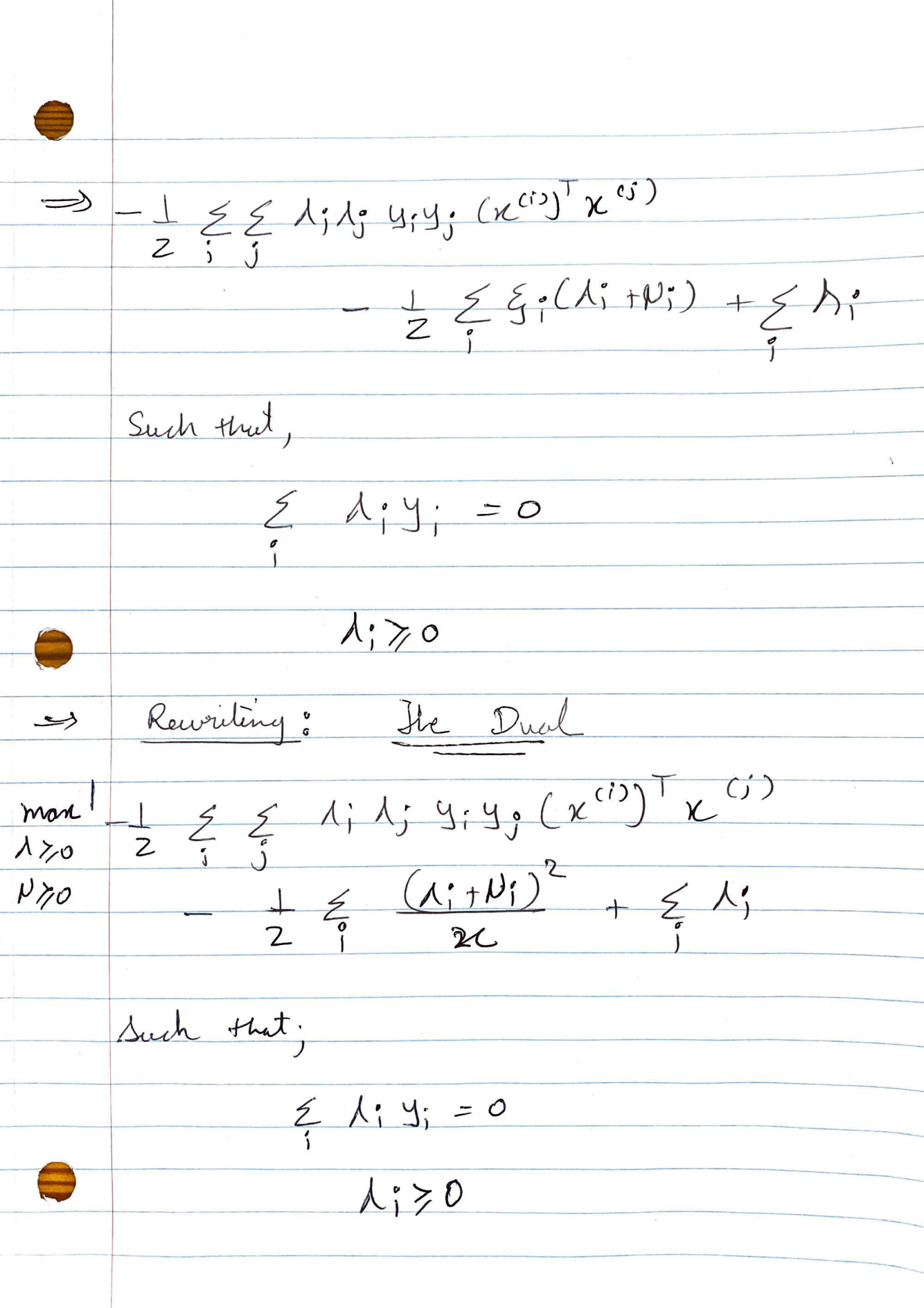
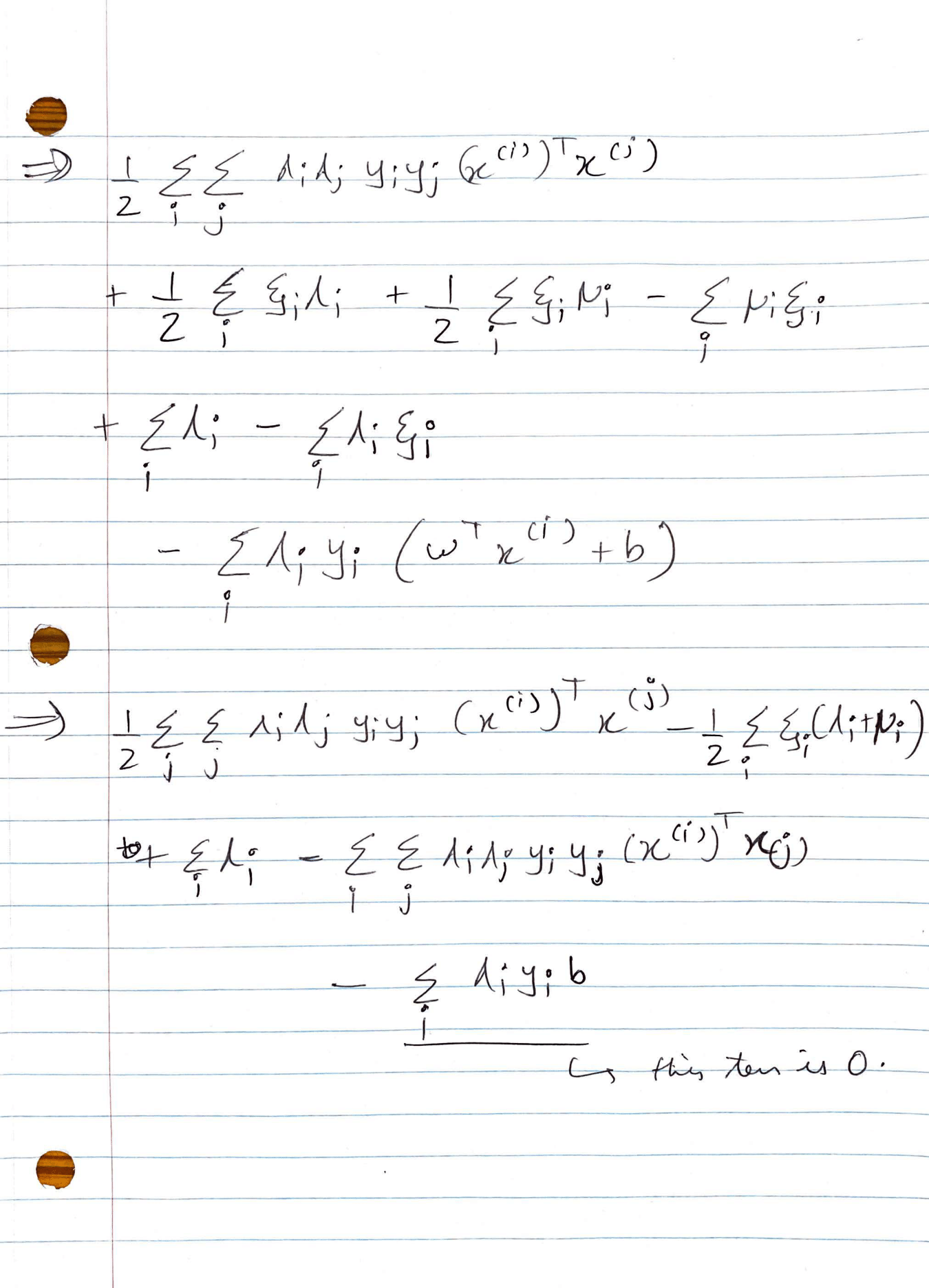
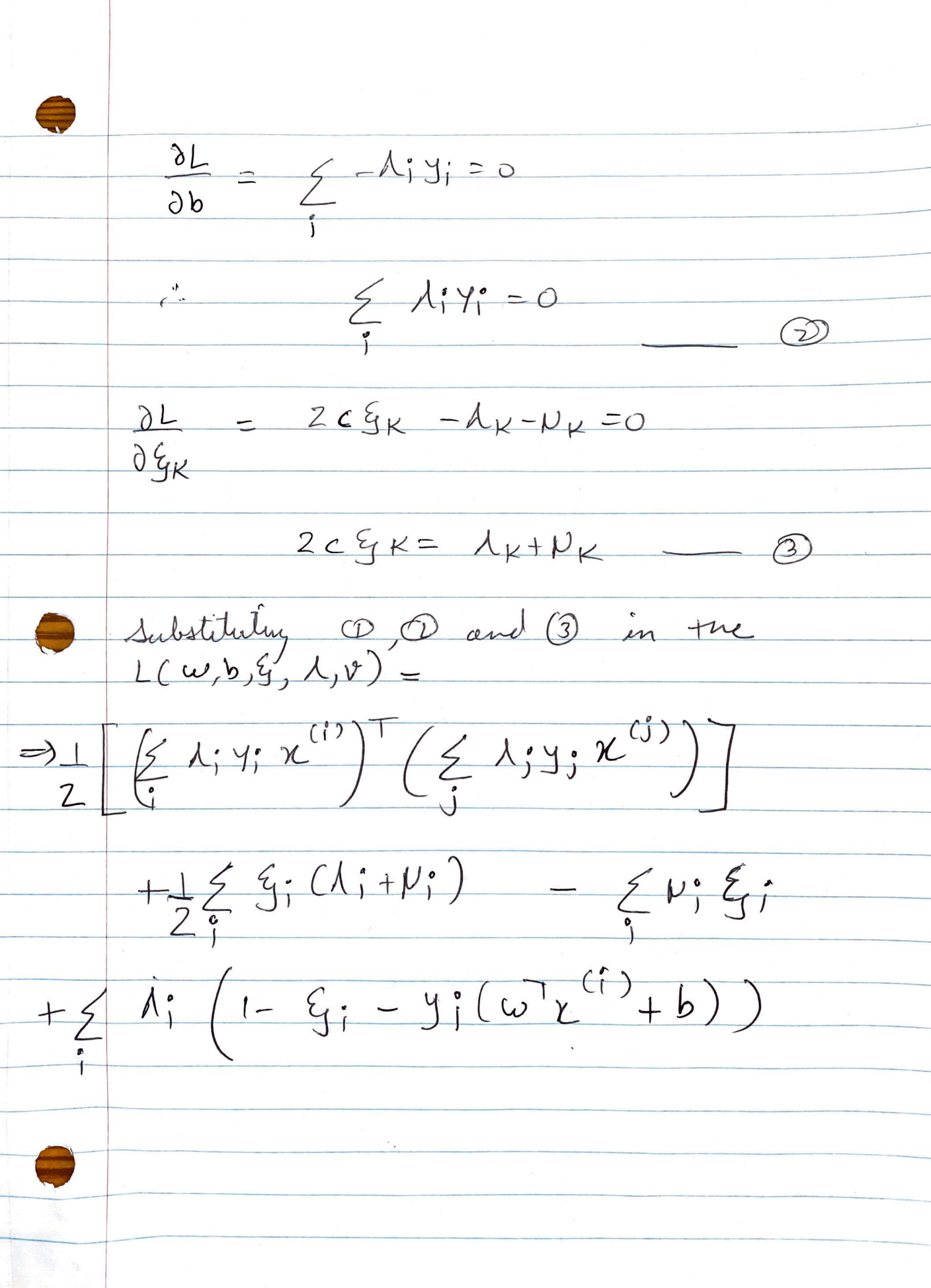
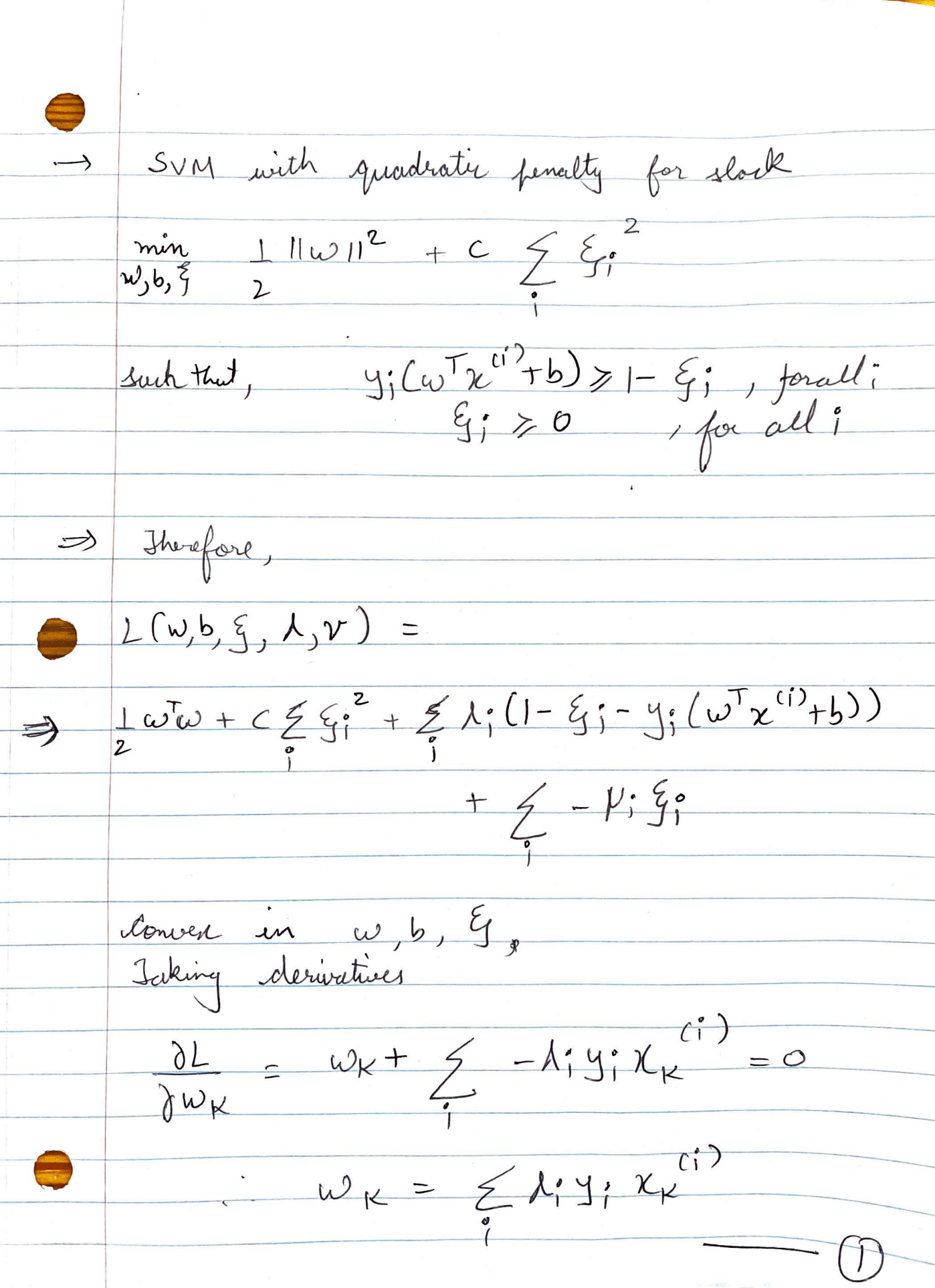
results.append([k,train\_acc[0],val\_acc[0],test\_acc[0]])

KNN:

|  |  |  |  |
| --- | --- | --- | --- |
| **K** | **Training Accuracy** | **Validation Accuracy** | **Testing Accuracy** |
| 1 | 0.999 | 0.885 | 0.725 |
| 5 | 0.938 | 0.896 | 0.707 |
| 11 | 0.922 | 0.885 | 0.721 |
| 15 | 0.913 | 0.864 | 0.715 |
| 21 | 0.905 | 0.865 | 0.709 |

### 4. Which approach should be preferred for the classification task?

For this task we should use SVM with gaussian kernel. When we use validation set to avoid overfitting on the training data, we will get good results with the gaussian kernel as it can fit more complex functions.

Problem 2: Method of Lagrange Multipliers

## Can this be kernelized?

Yes, this can also be kernelized as we have inner product of input data in our dual optimization function

Problem 3: Poisonous Mushrooms

1. Code

import numpy as np

import pandas as pd

import math

def getEntropy(entropy\_df):

# assuming zeroth column has the variable

if entropy\_df.shape[1] == 0:

#print("ERROR: Entropy function")

exit()

counts = entropy\_df.iloc[:,0].value\_counts()

probs = counts/sum(counts)

#print(probs)

h = 0

for p in probs:

h += p \* math.log(p,2)

return -h

# question == column for the sake of this dataset

def getEntropy\_question(q\_entropy\_df,ques):

total\_h = 0

total\_samples = q\_entropy\_df.shape[0]

weights = q\_entropy\_df[ques].value\_counts()/total\_samples

for split\_label in q\_entropy\_df[ques].unique():

# split data for this label

df\_split = q\_entropy\_df[q\_entropy\_df[ques] == split\_label]

total\_h += weights[split\_label] \* getEntropy(df\_split)

#print(total\_h)

return total\_h

class treeNode:

def \_\_init\_\_(self):

# intialize a node here

self.ntype = 'internal'

self.result = None

self.question = None

self.infoGain = None

self.children = {}

def build\_tree(data,entropy,parent\_maxvote,level):

node = treeNode()

if data.shape[0] == 0:

# that means we dont have any samples

node.ntype = 'leaf'

node.result = parent\_maxvote

elif len(train[0].unique()) == 1:

# we just have one label

node.ntype = 'leaf'

node.result = data[0].mode()

elif data.shape[1] == 1:

# we have no more columns to split on

node.ntype = 'leaf'

node.result = data[0].mode()

else:

min\_entropy = entropy

new\_question = None

for column in data:

if column == 0:

continue

label\_entropy = getEntropy\_question(data,column)

#print("Col : ",column,"\tentropy :",label\_entropy)

# this will take care of ties. We are going from left to right

if label\_entropy < min\_entropy:

min\_entropy = label\_entropy

new\_question = column

if new\_question is not None:

print("level: ",level," split > ",new\_question)

node.question = new\_question

node.infoGain = entropy - min\_entropy

# node.children = [None for \_ in data[new\_question].unique()]

node.result = data[0].mode()

new\_data = data.drop(new\_question,1)

print("Counts for this label:",new\_question)

print(data[new\_question].value\_counts())

for split\_label in data[new\_question].unique():

print("----trying: ",new\_question,',',split\_label)

# split data for this label

df\_split = new\_data[data[new\_question]== split\_label]

parent\_entropy = getEntropy(df\_split)

child = build\_tree(df\_split,parent\_entropy ,node.result,level+1)

if child.ntype == 'leaf':

print("-----Leaf - ",split\_label)

node.children[split\_label]= child

else:

node.ntype = 'leaf'

node.result = data[0].mode()

return node

def print\_tree(node,level):

if not node:

return

print("\t"\*level,node.ntype)

if node.ntype == 'leaf':

print("\t"\*level,"Node.label:",node.result[0])

else:

print("\t"\*level,"Node.I:",node.infoGain)

print("\t"\*level,"Node.question,",node.question)

for child in node.children:

print("\t"\*(level+1),str(child))

print\_tree(node.children[child],level+1)

def make\_predictions(data,node):

if node.ntype == 'leaf':

return node.result

col = node.question

# see what label do we have

if data[col] in node.children:

child = node.children[data[col]]

return make\_predictions(data,child)

else:

return node.result

def check(X):

if X.iloc[0]==X.iloc[1]:

return 1

else:

return 0

# Driver Code

if \_\_name\_\_ == '\_\_main\_\_':

# read training data

train = pd.read\_csv("mush\_train.data",header = None)

\_train = train.copy()

train.head()

unq\_labels = [train[column].unique() for column in train]

lmap = []

# for each column

# transform for ease of use

for column in train:

col\_map = {}

idx = 0

for label in train[column].unique():

col\_map[label] = idx

idx = idx + 1

lmap.append(col\_map)

train[column] = train[column].map(col\_map)

max\_entropy = getEntropy(train)

max\_votes = train[0].mode()

tree = build\_tree(train,max\_entropy,max\_votes,0)

print\_tree(tree,0)

predictions = train.apply(make\_predictions,axis=1,args=(tree,))

train = pd.concat([predictions,train],axis=1)

results = train.apply(check,axis=1)

train\_accuracy = sum(results)/len(results)

test = pd.read\_csv("mush\_test.data",header = None)

for column in test:

test[column] = test[column].map(lmap[column])

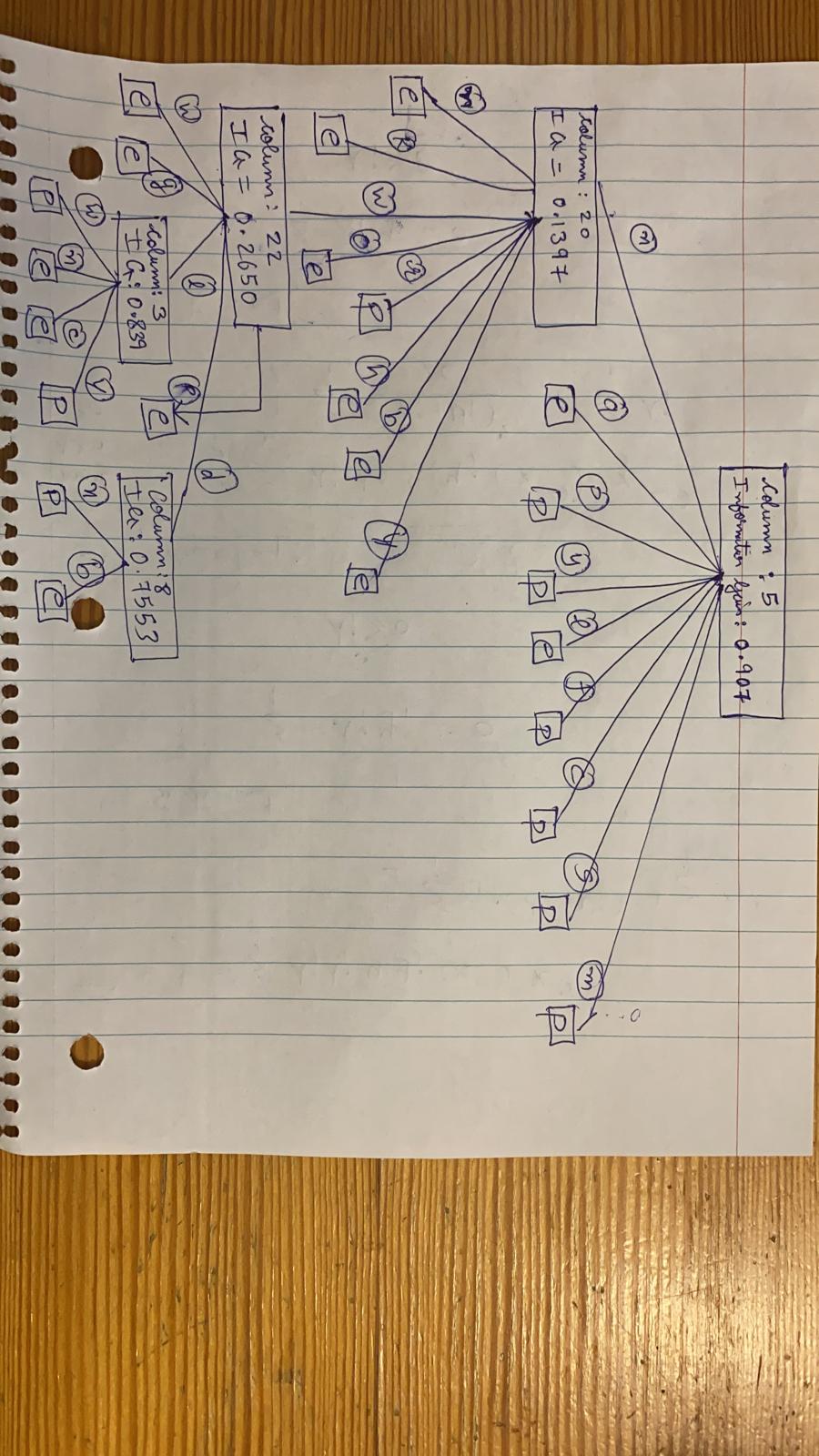
test\_pred = test.apply(make\_predictions,axis=1,args=(tree,))

test\_res = pd.concat([test\_pred,test],axis=1)

final\_results = test\_res.apply(check,axis=1)

test\_accuracy = sum(final\_results )/len(final\_results )

## Decision Tree



## Accuracy on the test data

We get 100% accuracy on the test data with this decision tree

## Does information gain gives best decision tree for 1-level?

Yes, for any arbitrary inut data if a 1-level decision tree is generated using information gain it will give the best training accuracy from all the possible 1-level decision trees.

The reason being that we would have selected the splits which puts similar labels together and thus when we do a majority vote, we will have maximum labels on the training set to be true.