Name: Shubham Sanghavi
Net ID: SAS190003

# CS 6375- Problem Set 2

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

1. Primal SVMs

### 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_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
```

---

## B. Accuracy on the training set

## C. 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 |

## D. 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

→ SVM with quadratic penalty for slack

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_i \xi_i^2$$

such that, $y_i(w^T x^{(i)}+b) \geq 1-\xi_i$ , for all $i$

$\xi_i \geq 0$ , for all $i$

⇒ Therefore,

$L(w,b,\xi,\lambda,\nu) =$

$$\Rightarrow \frac{1}{2}w^T w + C\sum_i \xi_i^2 + \sum_i \lambda_i(1-\xi_i - y_i(w^T x^{(i)}+b)) + \sum_i -\mu_i \xi_i$$

convex in $w, b, \xi$.

Taking derivatives

$$\frac{\partial L}{\partial w_K} = w_K + \sum_i -\lambda_i y_i x_K^{(i)} = 0$$

$$\therefore w_K = \sum_i \lambda_i y_i x_K^{(i)} \quad \text{--- (1)}$$

$$\frac{\partial L}{\partial b} = \sum_i -\lambda_i y_i = 0$$

$$\therefore \quad \sum_i \lambda_i y_i = 0 \qquad \text{——} \quad (2)$$

$$\frac{\partial L}{\partial \xi_K} = 2c\xi_K - \lambda_K - \nu_K = 0$$

$$2c\xi_K = \lambda_K + \nu_K \qquad \text{——} \quad (3)$$

Substituting (1), (2) and (3) in the
$L(\omega, b, \xi, \lambda, \nu) =$

$$\Rightarrow \frac{1}{2}\left[\left(\sum_i \lambda_i y_i x^{(i)}\right)^T \left(\sum_j \lambda_j y_j x^{(j)}\right)\right]$$

$$+ \frac{1}{2}\sum_i \xi_i (\lambda_i + \nu_i) \quad - \sum_i \nu_i \xi_i$$

$$+ \sum_i \lambda_i \left(1 - \xi_i - y_i(\omega^T x^{(i)} + b)\right)$$

$$\Rightarrow \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j (x^{(i)})^T x^{(j)}$$

$$+ \frac{1}{2} \sum_i \xi_i \lambda_i + \frac{1}{2} \sum_i \xi_i \mu_i - \sum_i \mu_i \xi_i$$

$$+ \sum_i \lambda_i - \sum_i \lambda_i \xi_i$$

$$- \sum_i \lambda_i y_i \left( w^T x^{(i)} + b \right)$$

$$\Rightarrow \frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j (x^{(i)})^T x^{(j)} - \frac{1}{2} \sum_i \xi_i (\lambda_i + \mu_i)$$

$$+ \sum_i \lambda_i - \sum_i \sum_j \lambda_i \lambda_j y_i y_j (x^{(i)})^T x^{(j)}$$

$$- \underline{\sum_i \lambda_i y_i b}$$

↳ this term is 0.

$$\Rightarrow -\frac{1}{2}\sum_i\sum_j \lambda_i\lambda_j y_i y_j (x^{(i)})^T x^{(j)}$$

$$- \frac{1}{2}\sum_i \xi_i(\lambda_i + \nu_i) + \sum_i \lambda_i$$

Such that,

$$\sum_i \lambda_i y_i = 0$$

$$\lambda_i \geq 0$$

$\rightarrow$ Rewriting: The Dual

$$\max_{\substack{\lambda \geq 0 \\ \nu \geq 0}} -\frac{1}{2}\sum_i\sum_j \lambda_i\lambda_j y_i y_j (x^{(i)})^T x^{(j)} - \frac{1}{2}\sum_i \frac{(\lambda_i + \nu_i)^2}{2C} + \sum_i \lambda_i$$

Such that;

$$\sum_i \lambda_i y_i = 0$$

$$\lambda_i \geq 0$$

## 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

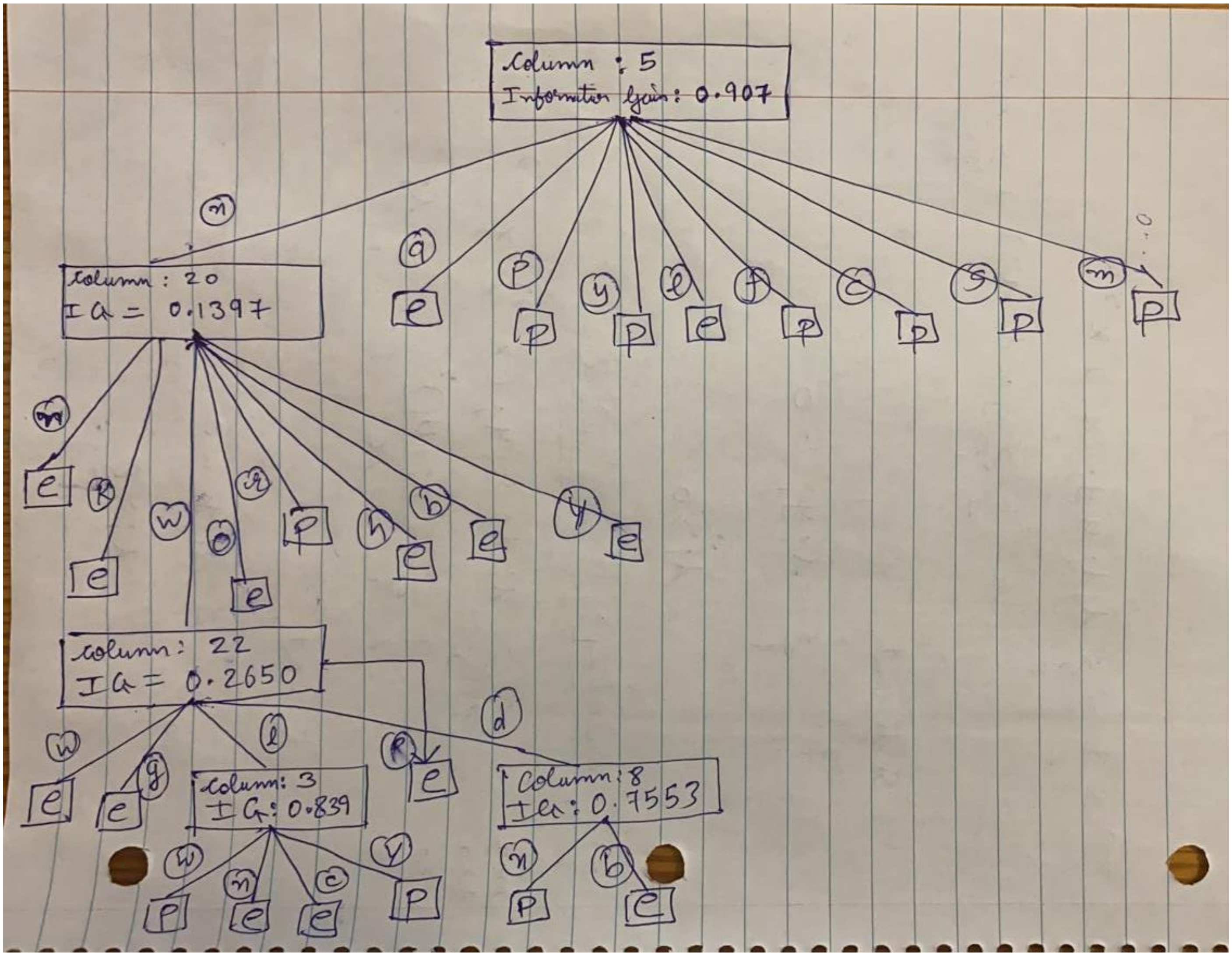
```
import numpy as np
import pandas as pd
import math

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




## 2. Accuracy on the test data

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

## 3. 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.