Machine Learning – HW2

Li-Hen Chen

Department of Computer Science & Engineering, Texas A&M University

Question 1: Decision Tree

(a) We have to split the tree according to the lowest entropy in order to gain the best decision tree. The process is to first choose which feature will have the lowest amount of entropy from our calculation. The equation for entropy is the following:

$$\_H(x) = -\sum p(x)\log_2(p(x))$$

By choosing the feature with the least amount of entropy, we can then say that group with have the maxi- mum amount of discrimination for our values. By hand performing my calculations, I discovered that the first group that needs to be split the data in half should be the " Sky Condition" feature. The calculations are below for the "Cloudy" and "Clear" branches respectively:

$$H_{cloudy} = -\left(\frac{25}{40}\log_2\left(\frac{25}{40}\right) + \frac{15}{40}\log_2(\frac{15}{40}\right) = 0.9544$$

$$H_{clear} = -\left(\frac{11}{40}\log_2\left(\frac{11}{40}\right) + \frac{29}{40}\log_2(\frac{29}{40}\right) = 0.8485$$

$$H(Rainy|SkyCondition) = \left(\frac{40}{80} * 0.9544\right) + \left(\frac{40}{80} * 0.8485\right) = 0.9014$$

This was the Lowest entropy that I found when computing the values for all three features. The Entropy for the "Humid" class was: 0.9457 and for the "Hot class was 0.9457.

Based on this first split, now I have halved the values and I retry each entropy equation now using the features based on "Humid" and "Hot". For the "Cloudy" branch, I discovered the "Humid" class had a lower entropy than the "Hot" class. The calculation is below:

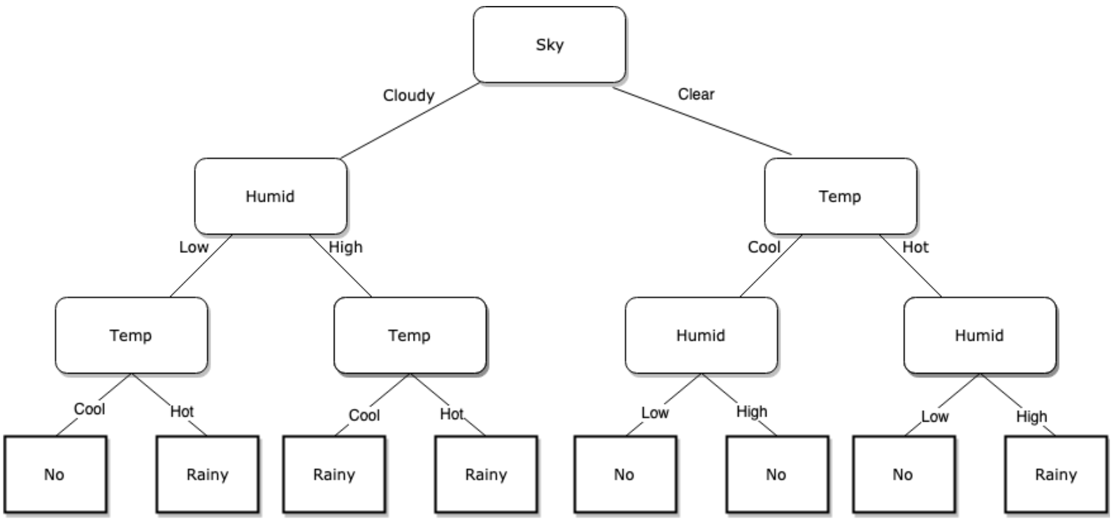
$$H_{High} = -\left(\frac{16}{20}\log_2\left(\frac{16}{20}\right) + \frac{4}{20}\log_2(\frac{4}{20}\right) = 0.7219$$

$$H_{Low} = -\left(\frac{9}{20}\log_2\left(\frac{9}{20}\right) + \frac{11}{20}\log_2(\frac{11}{20}\right) = 0.9927$$

$$H(Rainy|Humid) = \left(\frac{20}{40} * 0.7219\right) + \left(\frac{20}{40} * 0.9927\right) = 0..855$$

At this point, we have reached the final and last class for the branch of Cloudy → High → Temp. Since there are not more features to split on, we take the highest number of outcomes and if it is higher than the average from the group, we will classify the ending node as either the decision boundary whether to run or not.

By using this method and calculating the different partial entropy for each branch, finally I obtain the following figure:



(b.i) The number of Benign is 444 and the number of Malignant is 239. Benign is nearly twice more than Malignant in which Benign represents 65% of data and Malignant represents only 35 % of data. Then I separate them into a train (2/3 of the data) and a test (1/3 of the data) set. The result can be shown as the following format and both classes are represented with the same proportion in both sets.

	Benign	Malignant	Total
Train	297 (65%)	169 (35%)	457 (66%)
Test	147 (65%)	79 (35%)	226 (33%)
Total	444	239	

(b.ii)

$$Entropy = -\sum_j p_j \log_2 p_j$$

$$Gini = 1 - \sum_j p_j^2$$

At this part, I implement 2 decision trees with criteria “Entropy” and “Gini Index” separately. We can tell from the table and the figures that when the number of nodes is small the model would under fitting no matter criteria “Entropy” or criteria “Gini Index”. However, if there are too many nodes, the model would overfitting and lead to decreasing in accuracy. So again, since the number of nodes is hyperparameter, we still can use cross-validation to choose the most suitable number. Another to find best decision tree model is pruning including pre-pruning and post-pruning. Pre-pruning is to use a min entropy parameter to determine whether to stop growing the tree earlier. Post-pruning is to grow the tree full until no training error, then trim the nodes of the decision tree in a bottom-up fashion. If generalization error improves after trimming, replace sub-tree by a leaf node.

Furthermore, I do a little experiment that compare self-implement decision tree to the decision tree in ski-learn. The result is completely same. It really excited me proving that I implement the correct approach.

Table.

Number of Nodes	*Entropy Accuracy*	*Gini Accuracy*
1	90.3%	90.3%

2	90.3%	92.0%
3	94.7%	93.8%
4	93.8%	93.4%
5	94.2%	95.1%
6	94.2%	93.8%
7	94.2%	94.2%
8	94.7%	95.1%
9	94.7%	95.1%
10	92.9%	94.7%

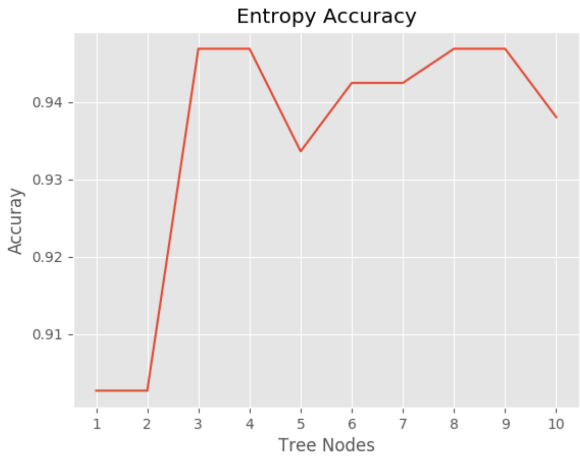


Fig.

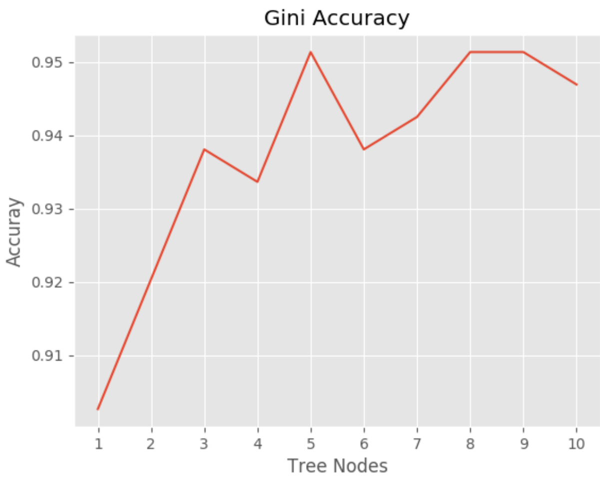


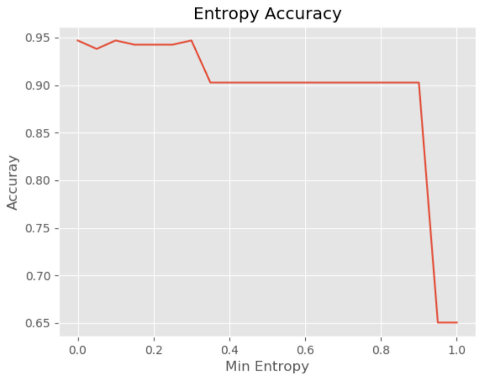
Fig.

(b.ii) Implement pre-pruning using a lower threshold on the values of the splitting criterion for each branch.

In this part of the question, we implement pre-pruning using a lower threshold on the values of splitting criterion for each branch. Threshold is for early stopping in tree growth. A node will split if its impurity is above the threshold, otherwise it is a leaf. We use the threshold range from 0 to 1 in order to see the impact of the pre-pruning. We can tell from the table and the figure that indeed a proper threshold can increase the accuracy of decision tree.

Table.

Threshold	Entropy Acc	Gini Acc
0	93.81%	94.69%
0.05	93.36%	92.92%
0.1	94.25%	93.81%
0.15	94.69%	90.27%
0.2	94.69%	90.27%
0.25	95.58%	90.27%
0.3	94.69%	90.27%
0.35	90.27%	90.27%
0.4	90.27%	90.27%
0.45	90.27%	90.27%
0.5	90.27%	65.04%
0.55	90.27%	65.04%
0.6	90.27%	65.04%
0.65	90.27%	65.04%
0.7	90.27%	65.04%
0.75	90.27%	65.04%
0.8	90.27%	65.04%
0.85	90.27%	65.04%
0.9	90.27%	65.04%
0.95	65.04%	65.04%
1	65.04%	65.04%



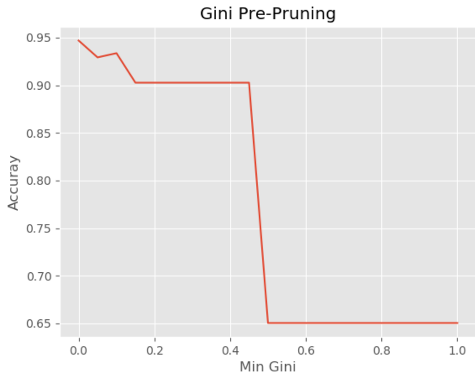


Fig. Pre-pruning using Entropy

Fig. Pre-pruning using Gini

Question 2: Support Vector Machine

(a) Data pre-processing

Import the Phishing Websites Data Set and process properly. All the features in the datasets are categorical. In the given dataset, the features 2, 7, 8, 14, 15, 16, 26, 29 are the features we need to transform. They all have three values {-1,0,1}. I transform 1 into [0,0,1], 0 into [0,1,0] and 1 into[1,0,0]. As follow figure, I take the first 5 rows to show that the result after transforming. We can utilize the following result to expand each list columns into 3 columns.

	0	1	2	3	4	5	...	25	26	27	28	29	30
0	-1	[0, 0, 1]	1	1	-1	-1	...	[1, 0, 0]	-1	1	[0, 0, 1]	-1	-1
1	1	[0, 0, 1]	1	1	1	-1	...	[0, 1, 0]	-1	1	[0, 0, 1]	1	-1
2	1	[0, 1, 0]	1	1	1	-1	...	[0, 0, 1]	-1	1	[0, 1, 0]	-1	-1
3	1	[0, 1, 0]	1	1	1	-1	...	[0, 0, 1]	-1	1	[1, 0, 0]	1	-1
4	1	[0, 1, 0]	-1	1	1	-1	...	[0, 1, 0]	-1	1	[0, 0, 1]	1	1

Fig.

(b) Use linear SVM in LIBSVM

Use the LibSVM library to run SVM to the data after pre-processing. At the SVM library we have plenty of parameters to choose such as the following.

-s svm_type : set type of SVM (default 0)

-t kernel_type : set type of kernel function (default 2)

0 -- linear: u'*v

1 -- polynomial: (gamma*u'*v + coef0)^degree

2 -- radial basis function: exp(-gamma*|u-v|^2)

3 -- sigmoid: tanh(gamma*u'*v + coef0)

4 -- precomputed kernel (kernel values in training_set_file)

-c cost : set the parameter C of C-SVC, epsilon-SVR, and nu-SVR (default 1)

-v n: n-fold cross validation mode

At this question, we have to set -t to 0 which means linear kernel, -v to 3 which means 3-fold cross validation and use cross validation to choose best parameter of cost also record the running time of every cross-validation on choosing cost parameter. Here we choose c from 1 to 100 which has the highest accuracy. The following is the figure of the relationship between cost value and the average time and the relationship between cost value and accuracy at each cross-validation. The C parameter tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points. For very tiny values of C, you should get misclassified examples, often even if your training data is linearly separable. As a result, I set cost value from $\mathbf{2^{-5}}$ $\boldsymbol{to}$ $\mathbf{2^{8}}$. Obviously, the cost value and consuming time are in direct proportion. But the cost value has little impact on the accuracy in general. In this case the best cost value is **64.** Although it has slightly accuracy different compared to other cost values.

Besides, use the model we train the fit the test data we can get the accuracy. I think the accuracy is high enough.

The test Accuracy = **94.6835%** (3455/3649) (classification)

Fig.

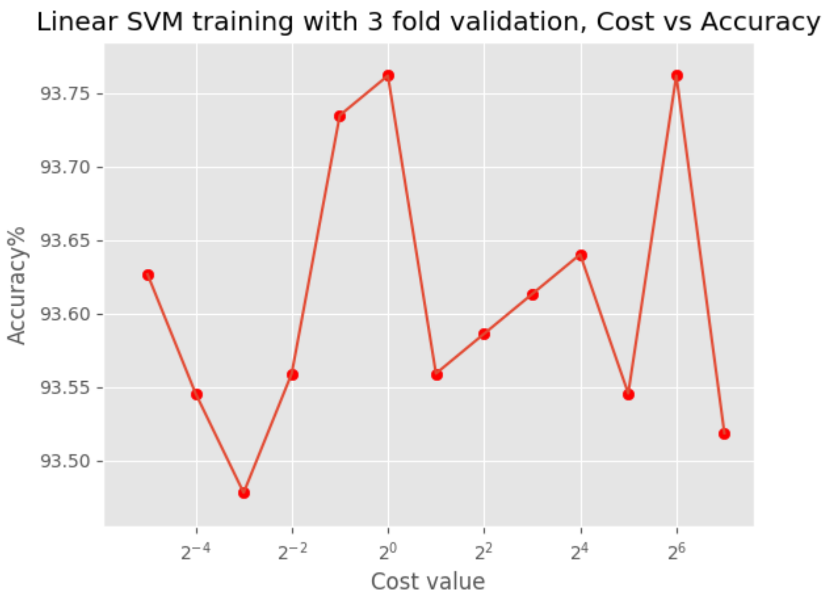


Fig.

(c) Use kernel SVM in LIBSVM

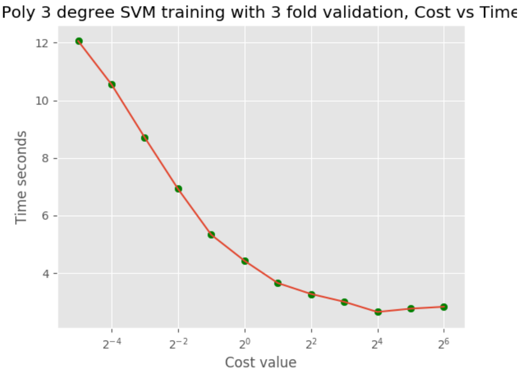
I try 3 different kernels (degree 2 poly, degree 3 poly and RBF) with cross-validation to choose cost value from 2^{-5} *to* 2^{7}**. The results are as following show.**

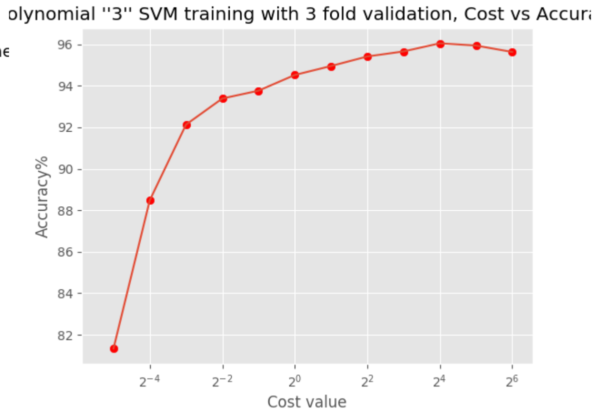
Table. Cross-Validation Accuracy of different Kernels

Cost Value	**Degree 2 Polynomial**	**Degree3 Polynomial**	**RBF**
2^{-5}	88.73	81.34	92.24
2^{-4}	91.47	88.48	92.40
2^{-3}	92.53	92.13	92.98
2^{-2}	93.34	93.38	93.65
2^{-1}	93.78	93.76	94.26
2^{0}	94.19	94.52	94.56
2^{1}	94.64	94.95	95.03
2^{2}	94.87	95.41	95.31
2^{3}	95.13	95.65	95.64
2^{4}	95.45	96.04	95.98
2^{5}	95.27	95.94	95.67
2^{6}	95.54	95.63	95.61

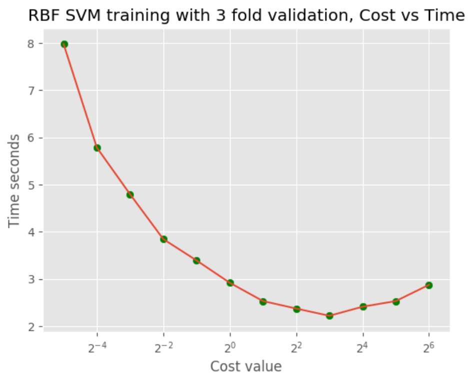
- Degree 2 Polynomial

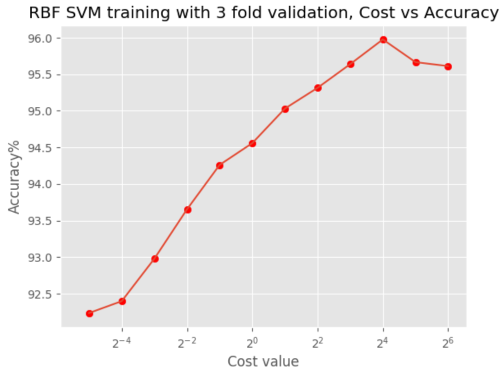
- Degree 3 Polynomial





- RBF





- **Discussion**

According to the above results, I would choose RBF as the SVM kernel. First of all, although their accuracy are nearly the same which highest values are around 96%, the RBF kernel is more stable. It is not strictly influenced by the cost value. Second RBF kernel has fast performance on fitting the mode. To sum up, in this case I choose RBF as my kernel function. Besides, usually linear and polynomial kernels are less time consuming and provides less accuracy than the RBF kernel.

(d) Implement Linear SVM

SVM model

$$f(\mathbf{x_n}) = \begin{cases} 1, & \text{if } \mathbf{w}^T\mathbf{x_n} + w_0 \geq 1 \\ -1, & \text{if } \mathbf{w}^T\mathbf{x_n} + w_0 \leq -1 \end{cases}$$

For all xi in training Data:

$$W^T x_n + w_0 \geq 1$$

For all support vectors (SV) (data points which decides margin)

$$W^T x_n + w_0 = 1$$

Our Objective is to maximize width w or we can say minimize |w|. Once we have found optimized w and b using algorithm. $W^T x_n + w_0 = 0$ is decision boundary It is not necessary that support vector lines always pass through support vectors It is a Convex Optimization problem and will always lead to a global minimum. This is Linear SVM means kernel is linear.

As the result we can implement the SVM algorithm in following procedure.

I. Start with random big value of w say(w0,w0) we will decrease it later
II. Select step size as w0*0.1
III. A small value of b, we will increase it later
   - b will range from (-b0 < b < +b0, step = step*b_multiple)
   - This is also computational expensive. So select b0 wisely
IV. We will check for points xi in dataset:
   - Check will for all transformation of w.
   - if not $w^T n_n + w_0 \geq 1$for all points then break
   - Else find |w| and put it in dictionary as key and (w, w_0) as values
   - If w<=0 then current step have been completed and go to step 6
   - Else decrease w as $(w_0 - step, w_0 - step)$ and continue with step 3
V. Do this step until step becomes $w_0 * 0.001$ because further it will be point of expense
   - $step = step * 0.1$
   - go to step 3
VI. Select (w,b) which has min |w| form the dictionary

HW2 Li-Hen Chen 928003907

Ref : https://www.youtube.com/watch?v=mA5nwGoRAOo&feature=youtu.be

Code for linear SVM:

```
def SVM(self, X,y):
        #train with data
    self.data = X + y
    opt_dict = {}
    transforms = [[1, 1], [-1, 1], [-1, -1], [1, -1]]
    all_data = np.array([])
    for yi in self.data:
    all_data = np.append(all_data, self.data[yi])
    self.max_feature_value = max(all_data)
    self.min_feature_value = min(all_data)
    all_data = None
    #with smaller steps our margins and db will be more precise
    step_sizes = [self.max_feature_value * 0.1,
                  self.max_feature_value * 0.01,
                  #point of expense
                  self.max_feature_value * 0.001, ]

    #extremly expensise
    b_range_multiple = 5
    #we dont need to take as small step as w
    b_multiple = 5

    latest_optimum = self.max_feature_value*10

    """
    objective is to satisfy yi(x.w)+b>=1 for all training dataset such that ||w|| is minimum
    for this we will start with random w, and try to satisfy it with making b bigger and bigger
    """
    #making step smaller and smaller to get precise value
    for step in step_sizes:
    w = np.array([latest_optimum, latest_optimum])

    #we can do this because convex
```

```
        optimized = False
        while not optimized:
            for b in np.arange(-1*self.max_feature_value*b_range_multiple,
                               self.max_feature_value*b_range_multiple,
                               step*b_multiple):
                for transformation in transforms:
                    w_t = w*transformation
                    found_option = True

                    #weakest link in SVM fundamentally
                    #SMO attempts to fix this a bit
                    # ti(xi.w+b) >=1
                    for i in self.data:
                        for xi in self.data[i]:
                            yi = i
                            if not yi*(np.dot(w_t, xi)+b) >= 1:
                                found_option = False
                    if found_option:
                        """
                        all points in dataset satisfy y(w.x)+b>=1 for this cuurent w_t, b
                        then put w,b in dict with ||w|| as key
                        """
                        opt_dict[np.linalg.norm(w_t)] = [w_t, b]
            if w[0] < 0:
                optimized = True
                print("optimized a step")
            else:
                w = w-step
        norms = sorted([n for n in opt_dict])
        opt_choice = opt_dict[norms[0]]

        self.w = opt_choice[0]
        self.b = opt_choice[1]
        latest_optimum = opt_choice[0][0]+step*2
```

HW2 Li-Hen Chen 928003907

```
import numpy as np

import matplotlib.pyplot as plt

import math

# Will split the given data for a specific feature (The index)

import pandas as pd

from sklearn.model_selection import train_test_split,
StratifiedShuffleSplit

import numpy as np

from collections import Counter

from sklearn.metrics import accuracy_score

from sklearn.tree import DecisionTreeClassifier

def load_data():
    data = pd.read_csv('hw2_question1.csv', names=[
                        'Clump', 'CellSize', 'CellShape',
'Adhesion',
                        'Epithelial', 'Nuclei',
'Chromatin', 'Nucleoli',
                        'Mitoses', 'Class'])
    return data

def split_d(X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33,

random_state=42, stratify=y)

    return X_train, y_train, X_test, y_test

def split_data(index, data):
```

```
    groups = [[] for _ in range(10)]

    for s in data:
        groups[s[index]-1].append(s)

    return groups

def compute_entropy(group, data):

    e = 0

    data_in_group = 0
    for g in group:
        data_in_group += len(g)

    # For each group we need to compute the entropy and
    # and then sum it all up.
    for g in group:

        b, m = 0, 0    # Keep track of
benign/malignant frequency

        for d in g:
            if d[9] == 2:
                b += 1
            else:
                m += 1

        if b == 0 or m == 0:
            continue
        else:
            p1 = float(b)/(m+b) *
math.log(float(b)/(m+b), 2)
```

```
                p2 = float(m)/(m+b) *
math.log(float(m)/(m+b), 2)

            e += (-(p1+p2))*(float(m+b)/data_in_group)

    return e

# I utilized the same method from the following website:
# https://machinelearningmastery.com/implement-decision-
tree-algorithm-scratch-python/


def compute_gini_index(group, data):
    n_instances = float(sum([len(g) for g in group]))
    classes = [2, 4]

    gini = 0.0
    for g in group:
        size = float(len(g))
        # avoid divide by zero
        if size == 0:
            continue
        score = 0.0
        # score the group based on the score for each
class
        for c in classes:
            p = [row[-1] for row in g].count(c) /
size
            score += p * p
        # weight the group score by its relative size
        gini += (1-score)*score * (size / n_instances)

    return gini
```

```
def determine_split(data, e):
    best_index, best_score, best_group = 1000000,
100000, None

    # For each feature in our data, we will test the split
    # and choose the one with the best entropy
    for f in range(9):
        # Split the group for feature f
        group = split_data(f, data)
        # Compute entropy for the given group
        if e:
            entropy = compute_entropy(group,
data)
        else:
            entropy = compute_gini_index(group,
data)

        if entropy <= best_score:
            best_index, best_score, best_group =
f, entropy, group

    return {'index': best_index, 'score': best_score,
'group': best_group}

# This function was influenced by:
# https://machinelearningmastery.com/implement-decision-
tree-algorithm-scratch-python/


def terminal(g):
    o = [row[9] for row in g]
    return max(set(o), key=o.count)


def split(node, max_depth, depth, e, test):
```

```
    for g in range(10):
        # If a group is empty, we can just take the
majority
        # class for all the nodes on the level.
        if not node[g]:
            for i in range(10):
                if node[i] == 2:
                    node[g] = 2
                elif node[i] == 4:
                    node[g] = 4
                else:
                    node[g] = 4
            continue

        if depth >= max_depth:
            node[g] = terminal(node[g])
            continue

        node[g] = determine_split(node[g], e)
        # for i in node[g]['group']:
        #     print len(i)
        split(node[g]['group'], max_depth, depth+1, e,
test)


def entropy_tree(data, test):

    root = determine_split(data, True)
    split(root['group'], 5, 1, True, test)
    return root


def gini_tree(data, test):
    root = determine_split(data, False)
    split(root['group'], 5, 1, False, test)
    return root

# Iterate through tree for sample s


def test_tree(node, s):
    if isinstance(node, dict):
        idx = node['index']
        return test_tree(node['group'][s[idx]-1], s)
    else:
        return node


def compute_accuracy(test, cls):

    num_correct = 0

    for i in range(len(cls)):
        if cls[i] == test[i][-1]:
            num_correct += 1

    return (float(num_correct)/len(test))


def print_tree(node, depth=0):
    if isinstance(node, dict):
        print('%s[X%d]' % ((depth*' ',
(node['index']+1))))
        for g in node['group']:
            print_tree(g, depth+1)
    else:
        print('%s[%s]' % ((depth*' ', node)))
```

HW2 Li-Hen Chen 928003907

```
def main():
    df = load_data()
    X, y = df.values[:,:-1] ,df['Class'].values
    X_train, y_train, X_test, y_test = split_d(X, y)
    train = np.hstack((X_train, y_train[:,None]))
    test = np.hstack((X_test, y_test[:, None]))
    e_tree = entropy_tree(train, test)
    g_tree = gini_tree(train, test)

    e_classes = []
    g_classes = []
    for s in test:
        p = test_tree(e_tree, s)
        e_classes.append(p)
        p = test_tree(g_tree, s)
        g_classes.append(p)

    a = compute_accuracy(test, e_classes)
    print('Entropy Accuracy: %s ' % a)
    a = compute_accuracy(test, g_classes)
    print('Gini Index Accuracy: %s' % a)



if __name__ == '__main__':
    main()

import pandas as pd
from sklearn.model_selection import train_test_split,
StratifiedShuffleSplit
import numpy as np
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
plt.style.use('ggplot')
```

```
def load_data():
    data = pd.read_csv('hw2_question1.csv', names=[
                                'Clump', 'CellSize', 'CellShape',
'Adhesion',
                                'Epithelial', 'Nuclei',
'Chromatin', 'Nucleoli',
                                'Mitoses', 'Class'])
    return data


def split_data(X,y):
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33,
                              random_state=0, stratify=y)

    return X_train,y_train,X_test,y_test

# Function to perform training with giniIndex.


def train_using_gini(X_train, y_train,maxdepth=10):


    # Creating the classifier object
    clf_gini = DecisionTreeClassifier(criterion="gini",

max_depth=maxdepth)

    # Performing training
    clf_gini.fit(X_train, y_train)
    return clf_gini


# Function to perform training with entropy.



def tarin_using_entropy(X_train, y_train, maxdepth=10):
```

```
    # Decision tree with entropy
    clf_entropy = DecisionTreeClassifier(
        criterion="entropy",
        max_depth=maxdepth)

    # Performing training
    clf_entropy.fit(X_train, y_train)
    return clf_entropy



def main():
    df = load_data()
    X, y = df.values[:,:-1] ,df['Class'].values
    X_train, y_train, X_test, y_test = split_data(X, y)
    #print(y_train)
    #print(Counter(y_test))
    """
    entropy
    """
    res = []
    for i in range(1,11):
        clf_entropy =
tarin_using_entropy(X_train,y_train,i)
        y_pred_entropy = clf_entropy.predict(X_test)
        acc_entro = accuracy_score(y_test,
y_pred_entropy)
        print("Entropy Accuracy: ", acc_entro)
        res.append(acc_entro)

    plt.title('Entropy Accuracy')
    plt.xlabel('Tree Nodes')
    plt.ylabel('Accuray')
    plt.xticks(np.arange(1, 11, 1))
    plt.plot(np.arange(1,11,1),res)
    plt.show()


    """
    Gini
    """
    res = []
    for i in range(1,11):
        clf_gini = train_using_gini(X_train,y_train,i)
        y_pred_gini = clf_gini.predict(X_test)
        acc_gini = accuracy_score(y_test, y_pred_gini)
        print("Gini Accuracy: ", acc_gini)
        res.append(acc_gini)


    plt.title('Gini Accuracy')
    plt.xlabel('Tree Nodes')
    plt.ylabel('Accuray')
    plt.xticks(np.arange(1, 11, 1))
    plt.plot(np.arange(1, 11, 1), res)


    plt.show()
   #print(df['Class'].value_counts())


if __name__ == "__main__":
    main()


import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import pandas as pd
from libsvm.python.svmutil import *
import time
import scipy
plt.style.use('ggplot')
```

HW2 Li-Hen Chen 928003907

```
# The feature numbers that need to be transformed in the
data
t_features = [1, 6, 7, 13, 14, 15, 25, 28]
def data_preprocess():
    df = pd.read_csv('hw2_question3.csv',header=None)
    for i in t_features:
        df[i] = df[i].apply(lambda x: [0,0,1] if x==1
else([0,1,0] if x==0 else [1,0,0]))
    #print(df.head(5))
    X = df[0].values[:,None]
    tmp = []

    for i in range(1,30):
        if i in t_features:
            tmp = []
            for t in df[i].values:
                tmp.append(t)
            tmp = np.asarray(tmp)
            X = np.hstack((X, tmp))
            #(X.shape, tmp.shape)
        else:
            #print(X.shape, df[i].values[:, None].shape)
            X = np.hstack((X,df[i].values[:,None]))

    print(X.shape)
    y = df.values[:,-1]

    return X,y



def split_data(X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33,

random_state=0, stratify=y)
    return X_train, y_train, X_test, y_test



def SVM(X,y,c):
    prob = svm_problem(y, X)
    par = '-t 0 -c {} -v 3'.format(c)
    param = svm_parameter(par)
    m = svm_train(prob, param)
    #lin_accuracy.append(m)
    return m



def SVM_all(X, y, type,cost, n=0.5, degree=3, v=3):
    prob = svm_problem(y, X)
    par = '-t {} -c {} -v {} -n {} -d {}'.format(type, cost, v,
n, degree)
    print(par)
    param = svm_parameter(par)
    m = svm_train(prob, param)
    #lin_accuracy.append(m)
    return m


def main():
    X,y = data_preprocess()
    #X, y = df.values[:,:-1], df.values[:,-1]
    X_train, y_train, X_test, y_test = split_data(X, y)
    # Cross validation of linear svm
    """
    c_values = []
    lin_accuracy = []
    lin_timed = []
    for c in range(-5,8,1):
        c0 = time.clock()
```

```
        m = SVM(X_train,y_train,2**c)
        diff = time.clock()-c0
        lin_timed.append(diff)
        lin_accuracy.append(m)
        c_values.append(2**c)

    print('Max Accuracy and its index : {},
{}'.format(max(lin_accuracy),lin_accuracy.index(max(lin_ac
curacy))))
    plt.scatter(c_values, lin_accuracy, color='r')
    plt.xscale('log', basex=2)
    plt.plot(c_values, lin_accuracy)
    plt.grid(True)
    plt.title("Linear SVM training with 3 fold validation,
Cost vs Accuracy")
    plt.xlabel("Cost value")
    plt.ylabel("Accuracy%")
    plt.show()


    plt.xscale('log', basex=2)
    plt.plot(c_values, lin_timed)
    plt.scatter(c_values, lin_timed, color='g')
    plt.grid(True)
    plt.title("Linear SVM training with 3 fold validation,
Cost vs Time")
    plt.xlabel("Cost value")
    plt.ylabel("Time seconds")
    plt.show()
    """
    """
    prob = svm_problem(y_train, X_train)
    m = svm_train(prob, '-t 0 -c 64 -n 3')
    svm_save_model('linear.model', m)
    m = svm_load_model('linear.model')
    p_label, p_acc, p_val = svm_predict(y_test, X_test, m)
    print(p_acc)
    """
    c_values = []
    lin_accuracy = []
    lin_timed = []
    for c in range(-5, 7, 1):
        c0 = time.clock()
        m = SVM_all(X_train,
y_train,type=1,cost=2**c,degree=3)
        diff = time.clock()-c0
        lin_timed.append(diff)
        lin_accuracy.append(m)
        c_values.append(2**c)

    print(lin_accuracy)
    print('Max Accuracy and its index : {}, {}'.format(
    max(lin_accuracy),
lin_accuracy.index(max(lin_accuracy))))
    plt.scatter(c_values, lin_accuracy, color='r')
    plt.xscale('log', basex=2)
    plt.plot(c_values, lin_accuracy)
    plt.grid(True)
    plt.title("Poly 3 degree SVM training with 3 fold
validation, Cost vs Accuracy")
    plt.xlabel("Cost value")
    plt.ylabel("Accuracy%")
    plt.show()

    plt.xscale('log', basex=2)
    plt.plot(c_values, lin_timed)
    plt.scatter(c_values, lin_timed, color='g')
    plt.grid(True)
    plt.title("Poly 3 degree SVM training with 3 fold
validation, Cost vs Time")
```

```
    plt.xlabel("Cost value")

    plt.ylabel("Time seconds")

    plt.show()




if __name__ == "__main__":


     main()
```