TCSS 455 Machine Learning

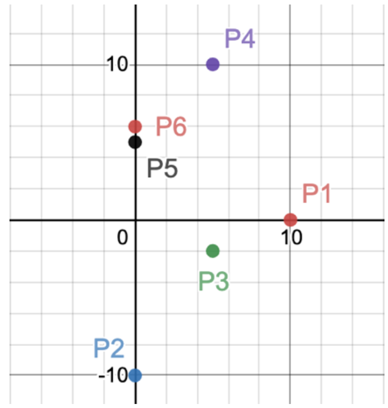
Homework #2

Tianyi Li
Student ID: 1827924

1. **Nearest Neighbour:**

Plot the given 6 training examples and show the decision boundary resulting from 1-NN.

	X_1	X_2	Y
Point 1 (P_1) →	*10*	*0*	+
Point 2 (P_2) →	*0*	*-10*	+
Point 3 (P_3) →	*5*	*-2*	+
Point 4 (P_4) →	*5*	*10*	-
Point 5 (P_5) →	*0*	*5*	-
Point 6 (P_6) →	*5*	*5*	-



From the given points, there are 3 positive examples and 3 negatives. We need calculate the Euclidean distances between all the positive points versus all the negative points:

$$P_1 - P_4 = \sqrt{\left(x_{1,P_1} - x_{1,P_4}\right)^2 + \left(x_{2,P_1} - x_{2,P_4}\right)^2} = \sqrt{(10-5)^2 + (0-10)^2} = \sqrt{5^2+10^2} = 11.180$$

$P_1 - P_5 = \sqrt{(10-0)^2+(0-5)^2} = \sqrt{10^2+5^2} = 11.180$

$P_1 - P_6 = \sqrt{(10-5)^2+(0-5)^2} = \sqrt{5^2+5^2} = 7.071$

$P_2 - P_4 = \sqrt{(0-5)^2+(-10-10)^2} = \sqrt{5^2+20^2} = 20.616$

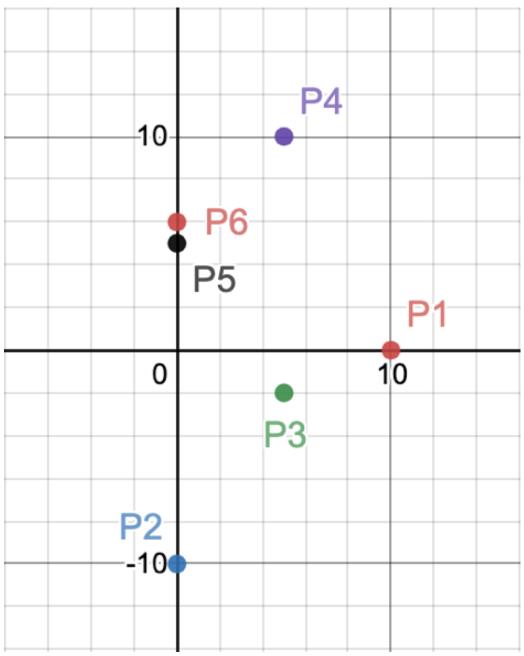
$P_2 - P_5 = \sqrt{(0-0)^2+(-10-5)^2} = \sqrt{0^2+15^2} = 15$

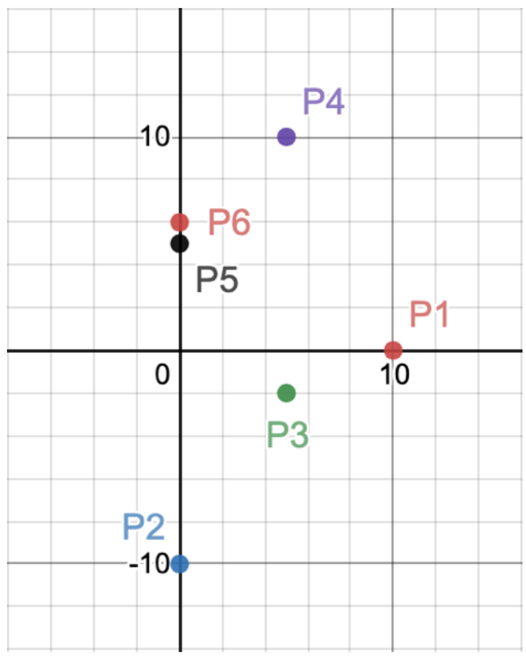
$P_2 - P_6 = \sqrt{(0-5)^2+(-10-5)^2} = \sqrt{5^2+15^2} = 15.811$

$P_3 - P_4 = \sqrt{(5-5)^2+(-2-10)^2} = \sqrt{0^2+12^2} = 12$

$P_3 - P_5 = \sqrt{(5-0)^2+(-2-5)^2} = \sqrt{5^2+7^2} = 8.602$

$P_3 - P_6 = \sqrt{(5-5)^2+(-2-5)^2} = \sqrt{0^2+7^2} = 7$





2. Decision Trees:

```
import sys
import math
import pandas as pd

class DecisionNode:

    # A DecisionNode contains an attribute and a dictionary of children.
    # Either the attribute being split on, or the predicted label if the node has no children.
    def __init__(self, attribute):
        self.attribute = attribute
        self.children = {}

    # Visualizes the tree
    def display(self, level = 0):
        if self.children == {}: # reached leaf level
            print(": ", self.attribute, end="")
        else:
            for value in self.children.keys():
                prefix = "\n" + " " * level * 4
                print(prefix, self.attribute, "=", value, end="")
                self.children[value].display(level + 1)

    # Predicts the target label for instance x
    def predicts(self, x):
        if self.children == {}: # reached leaf level
            return self.attribute
        value = x[self.attribute]
        subtree = self.children[value]
        return subtree.predicts(x)

####################################################################

def entropyOneLiner(pos, neg, total):
    pos_ratio = pos / total
    neg_ratio = neg / total
    entropy = - ((0 if pos_ratio == 0 else pos_ratio * math.log(pos_ratio, 2)) +
                 (0 if neg_ratio == 0 else neg_ratio * math.log(neg_ratio, 2)))
    return entropy


def entropy(examples, target):
    target_value = examples[target].unique()
    pos = len(examples[examples[target] == target_value[0]])
    neg = len(examples[examples[target] == target_value[1]])
    total = pos + neg
    entropy = entropyOneLiner(pos, neg, total)
    return entropy

def calInfoGain(attributes, examples, entropy):
    attribute_list = examples[attributes].unique()
    target_value = examples[target].unique()
    num_examples = len(examples)
    gain = entropy
    for i in attribute_list:
        total = len(examples[examples[attributes] == i])
        pos = len(examples[(examples[target] == target_value[0]) & (examples[attributes] == i)])
        neg = len(examples[(examples[target] == target_value[1]) & (examples[attributes] == i)])
        gain = gain - total / num_examples * entropyOneLiner(pos, neg, total)
    return gain

def selectAttribute(examples, target, attributes):
    if len(attributes) == 1:
        return attributes[0]
    attribute = attributes[0]
    maxInfoGain = calInfoGain(attribute, examples, entropy(examples, target))
    for i in attributes:
        nextGain = calInfoGain(i, examples, entropy(examples, target))
        if maxInfoGain < nextGain:
            maxInfoGain = nextGain
            attribute = i
    return attribute
```

```
def id3(examples, target, attributes):
    target_value = examples[target].unique()

    # Basecases
    if len(target_value) == 1:
        return DecisionNode(target_value[0])
    if len(attributes) == 0:
        target_value = examples[target].value_counts()
        return DecisionNode(target_value.keys()[0])

    attribute = selectAttribute(examples, target, attributes)
    root = DecisionNode(attribute)
    for i in examples[attribute].unique():
        selected_examples = examples[examples[attribute] == i]
        if len(selected_examples) == 0:
            root.children[i] = DecisionNode(examples[target].value_counts().keys()[0])
        else:
           # print ("initial attributes is %s" % attributes)
            new_attributes = attributes.copy()
          #  print ("copied new_attributes is %s" % new_attributes)
            new_attributes.remove(attribute)
          #  print ("renewed new_attributes is %s" % new_attributes)
            root.children[i] = id3(selected_examples, target, new_attributes)
    return root


####################   MAIN PROGRAM #####################

# Reading input data
train = pd.read_csv(sys.argv[1])
test = pd.read_csv(sys.argv[2])
target = sys.argv[3]
attributes = train.columns.tolist()
attributes.remove(target)

# Learning and visualizing the tree
tree = id3(train,target,attributes)
tree.display()

# Evaluating the tree on the test data
correct = 0
for i in range(0,len(test)):
    if str(tree.predicts(test.loc[i])) == str(test.loc[i,target]):
        correct += 1
print("\nThe accuracy is: ", correct/len(test))
```