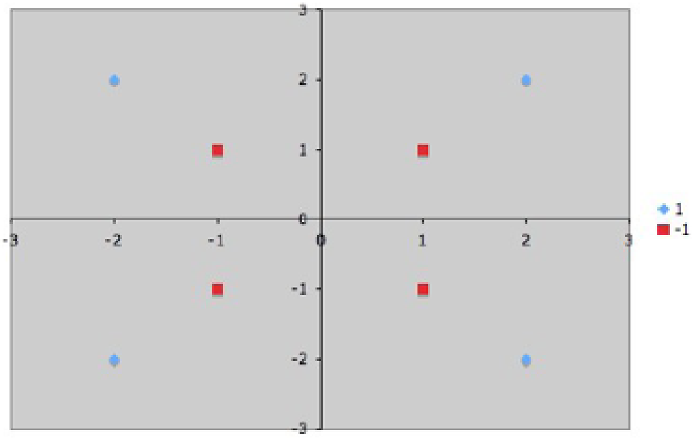
Name: ______Anthony_Sharp________ **45 points possible**

CS 5402 – Intro to Data Mining
Fall 2019
HW #6

Submit as a single pdf file via Canvas by 11:59 p.m. on Nov. 6, 2020

1. Write a **Python** program to create a **Support Vector Machine** for the dataset shown below. Note that a linear kernel is likely not the best to use since the data points are not linearly separable. You must determine what is the **best kernel** to specify when you call the Python sklearn SVC function; **JUSTIFY YOUR SELECTION!** Include a **listing** of your Python source code **AND** show what your SVM predicts for each of the following points: **(4, 5), (2, 2), (1, 1), (0, -0.5). (5 pts.)**

x	y	z
2	2	1
2	-2	1
-2	-2	1
-2	2	1
1	1	-1
1	-1	-1
-1	-1	-1
-1	1	-1



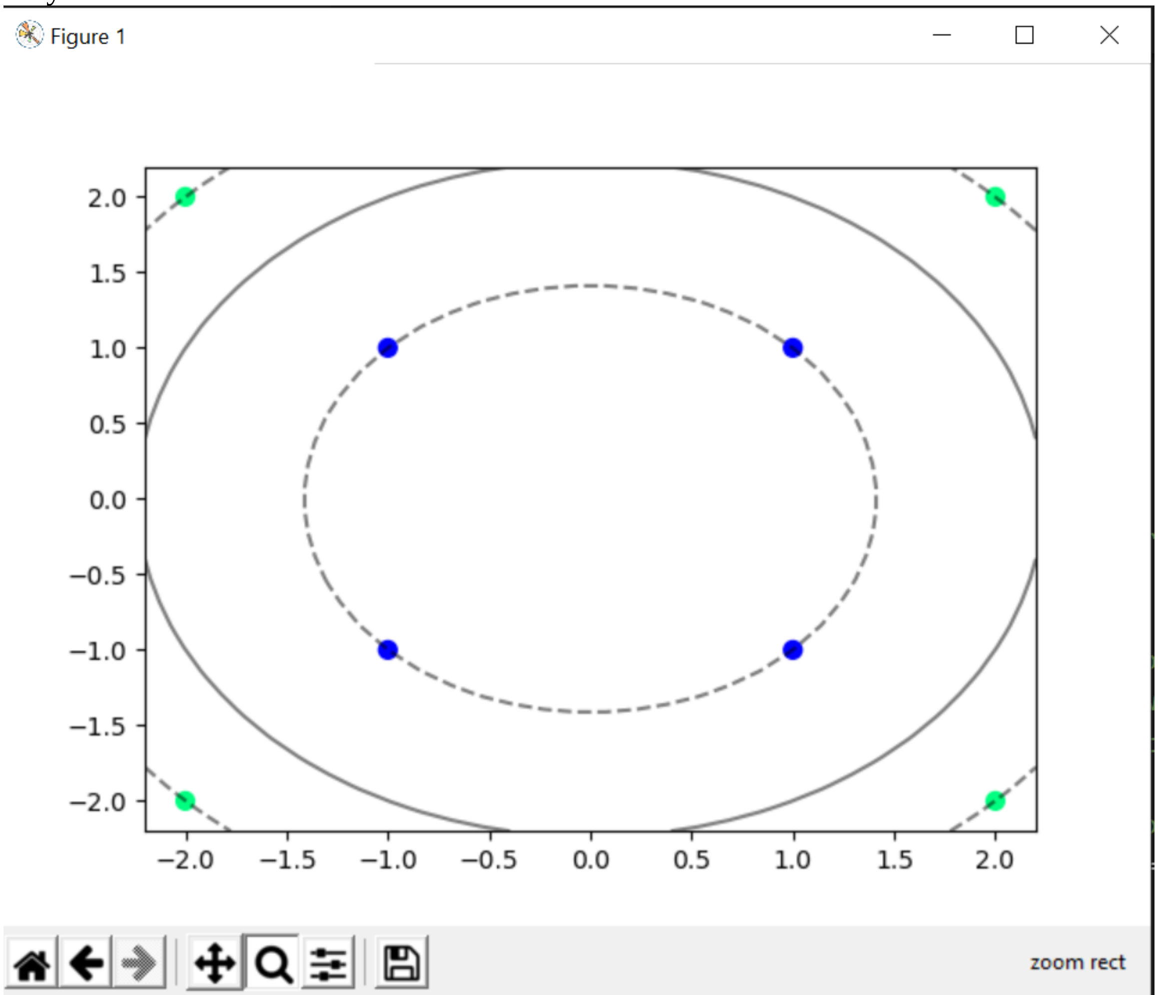
Name: ______Anthony_Sharp________ **45 points possible**

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.svm import SVC

df = pd.read_csv('HW6-1_Data.csv')

# Just for graphing purposes, change non-decision attr's
# to have numeric values
df= df.replace({'z': r'positive'}, {'z':1}, regex=True)
df=df.replace({'z': r'negative'}, {'z':-1}, regex=True)
X = df.iloc[:, 0:2].values
y = df.iloc[:, 2].values
r,_ = df.shape

# Display original data points
plt.scatter(X[:, 0], X[:, 1], c=y, s=r, cmap='winter')
plt.show()

'''
Link to the source of scikit-learn page
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
'''

# Make a linear SVM
#model = SVC(kernel='linear')

# Make a Polynomial SVC
'''This seems to work the best'''
model = SVC(kernel = 'poly', degree = 2)

# Make a sigmoid SVC
#model = SVC(kernel = 'sigmoid', degree = 10)

# Make a precomputed SVC
#model = SVC(kernel = 'precomputed', degree = 2)

model.fit(X, y)
print(model.support_vectors_)   # The support vectors
#print(model.coef_)                # The weights (x and y)
print(model.intercept_)          # Theintercept

# Function to plot SVM boundary lines-cool!
def plot_svc_decision_function(model, ax=None, plot_support=True):
```

```
    if ax is None:
        ax = plt.gca()

    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # Create grid to evaluate model
    x = np.linspace(xlim[0], xlim[1], 30)
    y = np.linspace(ylim[0], ylim[1], 30)
    Y, X = np.meshgrid(y, x)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = model.decision_function(xy).reshape(X.shape)

    # Plot decision boundary and margins
    ax.contour(X, Y, P, colors='k',
            levels=[-1, 0, 1], alpha=0.5,
            linestyles=['--', '-', '--'])

    # Plot support vectors
    if plot_support:
        ax.scatter(model.support_vectors_[:, 0],
            model.support_vectors_[:, 1],
            s=300, linewidth=1, facecolors='none')
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)

# Call the function to show points and SV boundaries
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='winter')
plot_svc_decision_function(model)
plt.show()

# Make predictions (will be array([-1]) or array([1]))

print("Test prediction for [2,2] is: ",model.predict([[2,2]]))
print("Test prediction for [-2,2] is: ",model.predict([[-2,2]]))
print("Test prediction for [-1,-1] is: ",model.predict([[-1,-1]]))

# Make predictions (will be array([-1]) or array([1]))
print("SVM prediction for [4,5] is: ",model.predict([[4,5]]))
print("SVM prediction for [2,2] is: ",model.predict([[2,2]]))
print("SVM prediction for [1,1] is: ",model.predict([[1,1]]))
print("SVM prediction for [0,-0.5] is: ",model.predict([[0,-0.5]]))
```

Name: ______Anthony_Sharp________ 45 points possible

Polynormal in the SVM kernal



2. Write a **Python** function which, given a dataframe, constructs (and returns) a **Naïve Bayesian network**.

   You can assume that all of the attributes have <u>nominal</u> values and that the decision attribute is the <u>last</u> attribute in the dataframe.

   Apply **Laplace smoothing** to the conditional probabilities of the attributes (as explained in class) using a value of **λ = 1**.

   Output the conditional probability table for <u>each</u> node in the Bayesian network so that we can check your work!

   Test your function by running it on **contact-lenses.csv <u>AND</u> hypothyroid.csv** (both of which are posted on Canvas in **Files -> Files for Weka Examples**). Note that you can check your work by running **Classify -> weka -> Classifiers -> bayes -> NaiveBayesSimple** in **Weka**.

   <u>ALSO</u> demonstrate that you have successfully created the Bayesian network by executing code that predicts an instance of your choice for each test file. <u>Clearly</u>

specify **exactly what** you are making a prediction for and what the **result** is. **(40 pts.)**

Note: You will NOT get full credit for your solution if you hard-code your code to work just for the specified test datasets!

```
instance:  ['primary_hypothyroid', 'M', None, None, None, None, None, None, None, 'SVI']
likelihood_yes 0.0016949152542372874
likelihood_no 0.061475988700564976
2.683063163778646 97.31693683622134
PS C:\Users\ashar\OneDrive\Documents\cs5402\hw6>
```

```python
import numpy as np, pandas as pd
from sklearn import preprocessing
import yaml
from pomegranate import *

#df = pd.read_csv('contact-lenses.csv')
df = pd.read_csv('hypothyroid.csv')

# Making the Column index
col_list = []
for i in df:
    col_list.append(i)

#print(col_list)

# Lambda is always equal to 1
_lambda = 1

def myMap(df, col_list):
    atr = {}
    for i in df[col_list]:
        atr[i] = i
    return list(atr)

# Laplace Smoothing
# P(x1 = aj | y = Ck) = (sum(I*(x1 = aj, y = Ck) + Lambda)/()
#
def ConditionalProbabilityT(x1, A1, y1, A2, k):
    Top = df.loc[(df[x1] == A1) & (df[y1] == A2),:].index.size
    Bot = df.loc[(df[x1] == A1),:].index.size

    return (Top + _lambda)/(Bot+ k * _lambda)


def Bayesian_Network(dataframe, title):
    num_size = dataframe.index.size
```

```
    P = []
    model = BayesianNetwork(title)

    decision_str = col_list[len(col_list) - 1]

    #print("decision", decision_str)
    dictionary = {}
    #k = len(myMap(dataframe, decision_str))
    for i in myMap(dataframe, decision_str):
        k = len(myMap(dataframe, decision_str))
        dictionary[i] = ((dataframe.loc[dataframe[decision_str] == i,:].in
dex.size)+(_lambda))/(num_size+k*_lambda)

    # Table for discrete Distrubtion(isn't conditional on any other nodes)
    P.append(DiscreteDistribution(dictionary))
    #print(P)
    for i in range(0,len(col_list)-1):
        con = []
        L = len(myMap(dataframe, col_list[i]))
        for j in myMap(dataframe, decision_str):
            for k in myMap(dataframe, col_list[i]):
                con.append([j,k,
                ConditionalProbabilityT(decision_str, k, col_list[i], k, L
)])

        C = ConditionalProbabilityTable(con, [P[0]])

        P.append(C)
        #print(P)

    # Create nodes with their probability tables
    s1 = Node(P[0], name=decision_str)

    #model.add_states(s1, s2, s3, s4, s5) # add the nodes
    model.add_state(s1)

    # add the nodes and edges
    for i in range(1, len(P)):
        s2 = Node(P[i], name = col_list[i-1])
        model.add_state(s2)
        model.add_edge(s1,s2)

    model.bake()
    return model
```

Name: ______Anthony_Sharp________ **45 points possible**

```
in_model = Bayesian_Network(df,"Bayesian Network")
#print(in_model)
# Hardcode Prediction
# sex,on_thyroxine,on_antithyroid_medication,thyroid_surgery,lithium,goitr
e,hypopituitary,psych,referral_source,Class
'''testing'''
#instance = ['negative','M','f','f','f','f','f','f','t','SVHC']
instance = ['primary_hypothyroid',"M",None,None,None,None,None,None,None,'
SVI']

# From Leopold's notes below
likelihood_yes = in_model.probability([instance])
print("instance: " , instance)
print("likelihood_yes" , likelihood_yes)
instance[0] = 'negative'
likelihood_no = in_model.probability([instance])
print("likelihood_no" , likelihood_no)
prob_yes = (likelihood_yes / (likelihood_yes + likelihood_no))* 100
prob_no = (likelihood_no / (likelihood_yes + likelihood_no))* 100
print(prob_yes, prob_no)
```