Homework 4

This is an individual assignment.

Due: Saturday, March 7 @ 11:59 PM

Grading Rubric

1. **Problem 1 (15 pts):** you've answered all 4 questions (4 points for problems 1,2 3).
2. **Problem 2 (15 pts):** you've answered all 2 questions (5 points first, 10 points se
3. **Problem 3 (20 pts):** you've answered all 4 questions (5 points each).
4. **Problem 4 (10 pts):** you've answered all 2 questions (5 points each).
5. **Problem 5 (15 pts):** you've answered all 3 questions (5 points each).
6. **Problem 6 (10 pts):** you've answered and justified your answer.
7. **Problem 7 (15 pts):** you've answered all 3 questions (5 points each).

Total: 100 pts

Supervised Classification

In this assignment, you will test your knowledge about all supervised classifiers we h

Neighbors (k-NNs), Decision Trees, Random Forests, Support Vector Machines (SV

particular, you will implement these models using `scikit-learn` and experiment diff

You are allowed to use `scikit-learn` modules.

Objectives

By completing this assignment you will practice and master the following skills:

- k-Nearest Neighbors
- Decision Trees
- Random Forests
- Support Vector Machines
- The Perceptron algorithm

Create your Repo

You can create the repo for this assignment by visiting the following link: https://class (https://classroom.github.com/a/Vj3ofz1F)

Problem 1 - k-Nearest Neighbors Classifie

Implement the k-NN algorithm in the Breast Cancer Data Set (https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html) (load
using Euclidean distance.

In order to successfully implement the k-NN algorithm, you should follow all the steps
problem.

Answer and justify the following questions:

1. What happens as you vary K from small to large? Why? Include a plot that shov
   as you vary k.
2. What is the optimal value for k given this data?
3. Compute the confusion matrix in the test set using the best value of k.
4. In a paragraph describe whether or not k-NN performs well in high dimensional

In [24]:

```
import numpy as np
from sklearn.datasets import load_breast_cancer

data = load_breast_cancer(return_X_y=False)

# The data set contains 569 samples and 30 features
X=data.data # Training samples
print(type(X), X.shape)

# Target labels for all 569 samples. Malignant class has labels 0, beni
y=data.target # Training labels
print(type(y), y.shape)

print("Number of benign (1):\t%d" % list(y).count(1))
print("Number of malignant (0):\t%d" % list(y).count(0))

print("\nX: ", X)
print("\ny: ", y)
```

```
<class 'numpy.ndarray'> (569, 30)
<class 'numpy.ndarray'> (569,)
Number of benign (1):   357
Number of malignant (0):        212

X:  [[1.799e+01 1.038e+01 1.228e+02 ... 2.654e-01 4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 ... 1.860e-01 2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 ... 2.430e-01 3.613e-01 8.758e-02]
 ...
 [1.660e+01 2.808e+01 1.083e+02 ... 1.418e-01 2.218e-01 7.820e-02]
 [2.060e+01 2.933e+01 1.401e+02 ... 2.650e-01 4.087e-01 1.240e-01]
 [7.760e+00 2.454e+01 4.792e+01 ... 0.000e+00 2.871e-01 7.039e-02]]

y:  [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1
 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0
 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 0 1 1
 1 0 1 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 0 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 0 1 1
 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 1 0 1 1 0 1 0 1 0 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 0 0 0 0 0 1]
```

In [2]:
```
# Uncomment the line below for a full data set description
print(data.DESCR)
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of the three
        largest values) of these features were computed for each image,
        resulting in 30 features.  For instance, field 3 is Mean Radius, field
        13 is Radius SE, field 23 is Worst Radius.

        - class:
                - WDBC-Malignant
                - WDBC-Benign

    :Summary Statistics:

    ===================================== ====== ======
                                           Min    Max
    ===================================== ====== ======
    radius (mean):                        6.981  28.11
    texture (mean):                       9.71   39.28
    perimeter (mean):                     43.79  188.5
    area (mean):                          143.5  2501.0
    smoothness (mean):                    0.053  0.163
    compactness (mean):                   0.019  0.345
    concavity (mean):                     0.0    0.427
    concave points (mean):                0.0    0.201
    symmetry (mean):                      0.106  0.304
    fractal dimension (mean):             0.05   0.097
    radius (standard error):              0.112  2.873
    texture (standard error):             0.36   4.885
    perimeter (standard error):           0.757  21.98
    area (standard error):                6.802  542.2
    smoothness (standard error):          0.002  0.031
```

```
        compactness (standard error):          0.002  0.135
        concavity (standard error):            0.0    0.396
        concave points (standard error):       0.0    0.053
        symmetry (standard error):             0.008  0.079
        fractal dimension (standard error):    0.001  0.03
        radius (worst):                        7.93   36.04
        texture (worst):                       12.02  49.54
        perimeter (worst):                     50.41  251.2
        area (worst):                          185.2  4254.0
        smoothness (worst):                    0.071  0.223
        compactness (worst):                   0.027  1.058
        concavity (worst):                     0.0    1.252
        concave points (worst):                0.0    0.291
        symmetry (worst):                      0.156  0.664
        fractal dimension (worst):             0.055  0.208
        ========================================= ====== ======

        :Missing Attribute Values: None

        :Class Distribution: 212 - Malignant, 357 - Benign

        :Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

        :Donor: Nick Street

        :Date: November, 1995

    This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
    https://goo.gl/U2Uwz2 (https://goo.gl/U2Uwz2)

    Features are computed from a digitized image of a fine needle
    aspirate (FNA) of a breast mass.  They describe
    characteristics of the cell nuclei present in the image.

    Separating plane described above was obtained using
    Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree
    Construction Via Linear Programming." Proceedings of the 4th
    Midwest Artificial Intelligence and Cognitive Science Society,
    pp. 97-101, 1992], a classification method which uses linear
    programming to construct a decision tree.  Relevant features
    were selected using an exhaustive search in the space of 1-4
    features and 1-3 separating planes.

    The actual linear program used to obtain the separating plane
    in the 3-dimensional space is that described in:
    [K. P. Bennett and O. L. Mangasarian: "Robust Linear
    Programming Discrimination of Two Linearly Inseparable Sets",
    Optimization Methods and Software 1, 1992, 23-34].

    This database is also available through the UW CS ftp server:

    ftp ftp.cs.wisc.edu
    cd math-prog/cpo-dataset/machine-learn/WDBC/

    .. topic:: References

       - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction
```

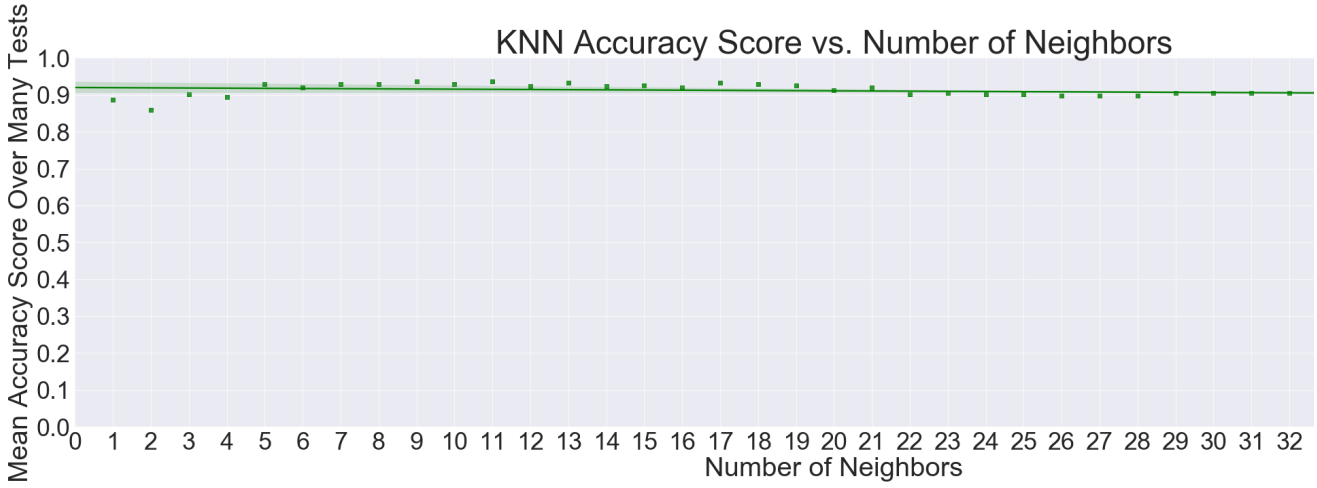
for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on Electronic Imaging: Science and Technology, volume 1905, pages 861-870, San Jose, CA, 1993.
- O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and prognosis via linear programming. Operations Research, 43(4), pages 570-577, July-August 1995.
- W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994) 163-171.

In [45]:

```
# 1.1
# What happens as you vary  K  from small to large? Why?
# Include a plot that shows the accuracy performance as you vary  k .
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
import numpy as np
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
%matplotlib inline

score_arr = [] #accuracy score based on number of nearest neighbors k
knn_classifier = KNeighborsClassifier()

#n is the number of neighbors used
start=1
stop=40
step=1
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.
print(X_train)
for n in range(start, stop+1, step):
    knn_classifier=KNeighborsClassifier(n_neighbors=n, algorithm='auto'
    knn_classifier.fit(X=X_train, y=y_train)
    score_arr.append(knn_classifier.score(X=X_test, y=y_test))

print("score_arr:\n{}".format(score_arr))

#plotting accuracy score vs. number of neighbors
score_arr=np.asarray(a=score_arr)
fig=plt.figure(figsize=(40, 10))
sns.set_style(style='darkgrid')
sns.regplot(x=np.arange(start, stop+1, step), y=score_arr, marker='s',
plt.title(label="KNN Accuracy Score vs. Number of Neighbors", fontsize=
plt.xlabel(xlabel="Number of Neighbors", fontsize=40)
plt.ylabel(ylabel="Mean Accuracy Score Over Many Tests", fontsize=40)
plt.xlim(0, stop)
plt.ylim(0, 1.0, 0.1)
plt.xticks(ticks=np.arange(0, len(score_arr) + 1), fontsize=36)
plt.yticks(ticks=np.arange(0, 1.1, 0.1), fontsize=36)
plt.grid(b=True, which='major', axis='both')
```

```
plt.show()
```

```
[[1.799e+01 1.038e+01 1.228e+02 ... 2.654e-01 4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 ... 1.860e-01 2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 ... 2.430e-01 3.613e-01 8.758e-02]
 ...
 [1.174e+01 1.402e+01 7.424e+01 ... 8.290e-02 3.101e-01 6.688e-02]
 [1.940e+01 1.818e+01 1.272e+02 ... 2.252e-01 3.590e-01 7.787e-02]
 [1.624e+01 1.877e+01 1.088e+02 ... 1.732e-01 2.770e-01 1.063e-01]]
score_arr:
[0.887719298245614, 0.8596491228070176, 0.9017543859649123, 0.8947368421052632, 0.9298245614035(
45614035088, 0.9298245614035088, 0.9368421052631579, 0.9298245614035088, 0.9368421052631579, 0.9
3333, 0.9228070175438596, 0.9263157894736842, 0.9192982456140351, 0.9333333333333333, 0.92982456
0.9122807017543859, 0.9192982456140351, 0.9017543859649123, 0.9052631578947369, 0.90175438596491
56140350877, 0.8982456140350877, 0.8982456140350877, 0.9052631578947369, 0.9052631578947369, 0.9
7369, 0.9052631578947369, 0.8982456140350877, 0.9052631578947369, 0.9017543859649123, 0.90175438
0.8982456140350877, 0.8947368421052632]
```



Small values of K make the prediction accuracy subpar due to insensitivity to outliers outliers eventually outnumber outliers by a huge factor. However, if K becomes too la ratio of classes in the original dataset could cause misclassification. Suppose that K i the original dataset. The K nearest neighbors might include the entire cluster of point another class with a way higher population might be included as well due to the large graph for KNN classifier looks like a U or curved V. The plot above measures accura Since accuracy is the inverse of error, the graph above looks like a stretched-out inve K would be at the peak of the graph, in this case 9.

In [48]:
```
# 1.2
# What is the optimal value for  k  given this data?

optimal_k=np.argmax(score_arr) #indices of k with maximum accuracy scor
max_accuracies=score_arr[optimal_k] #corresponding accuracies of optima

print("optimal k:\t{}".format(optimal_k + 1))
print("accuracy at optimal k:\t{}".format(max_accuracies))
```

```
optimal k:      9
accuracy at optimal k:  0.9368421052631579
```

In [49]:
```
# 1.3
# Compute the confusion matrix in the test set using the best value of
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix

optimal_knn_classifier=KNeighborsClassifier(n_neighbors=optimal_k, algo
optimal_knn_classifier.fit(X=X, y=y)
y_pred=optimal_knn_classifier.predict(X=X)
cm1=confusion_matrix(y_true=y, y_pred=y_pred) #confusion matrix for pro
print("Confusion Matrix\n", cm1)
```

```
Confusion Matrix
 [[192  20]
 [ 15 342]]
```

In [42]:
```
# 1.4
# In a paragraph describe whether or not  k -NN performs well in high c
```

k-NN should be used with extreme caution when used in a high dimensional feature s
robust; when searching for the K nearest neighbors, the algorithm cannot tell outliers
more different classes there are to label a test point, the higher the chance a mislabe
having more dimensions mean that the cluster of K nearest neighbors often becomes
4 classes present, but expanding K might introduce new classes and shrinking K mig
either scenario, changing K can also change the class that is most popular. Therefore
choosing an optimal K very difficult. In short, K-NN suffers from the curse of dimensio

Problem 2 - Decision Tree

In order to reduce email load, let's build a decision tree algorithm to decide whether c simply file it away instead. To train the model, consider the data set of binary-valued including whether the author is known, whether the email is long or short, whether it l with any final decision about whether to read it ($y = 1$ for "read", $y = -1$ for "discar

x_1: know author?	x_2: is long?	x_3: has research?	x_4: has grade?	x_5: has lottery?	y: read?
0	0	1	1	0	-1
1	1	0	1	0	-1
0	1	1	1	1	-1
1	1	1	1	0	-1
0	1	0	0	0	-1
1	0	1	1	1	1
0	0	1	0	0	1
1	0	0	0	0	1
1	0	1	1	0	1
1	1	1	1	1	-1

In the case of any ties, we will prefer to predict class $y = 1$. For the next steps consi

1. Compute the information gain (entropy function) for each feature x_i. Which featu
2. Draw the complete decision tree that will be learned from this data. Justify every gain at every node.

In [118]:

```
# 2.1
# Compute the information gain (entropy function) for each feature xi.
# Which feature should be the root node?

import numpy as np
import pandas as pd

def entropy(feature_array, feature_value, targets):
    """compute the entropy of an attribute/feature"""
    #all target values that matches given feature_value under specifiec
    node=targets[np.argwhere(feature_array==feature_value)] #all target
    unique_targets=np.unique(ar=node) #unique target values caught in g
    targets_proportions=np.asarray(a=[list(node).count(u)/float(node.si
#     print("targets_proportions", targets_proportions)

    #computing entropy
    ent=-np.dot(a=targets_proportions, b=np.log2(targets_proportions))
    return ent



def entropy_weighted(feature_array, feature_value, targets):
    """
        compute weighted entropy for a specified attribute/feature and
        this is equivalent to (proportion of feature_value in feature)
        node/cell containing target values aligned with the same featur
    """
    feature_proportion=list(feature_array).count(feature_value) / float
    ent_weighted=feature_proportion * entropy(feature_array=feature_arr
                                              feature_value=feature_val
                                             targets=targets)
    return ent_weighted

def entropy_feature(feature_array, targets):
    """
        compute entropy for entire attribute/feature; this is the same
        entropies fo each unique value that the feature can have
    """
    unique_features=np.unique(ar=feature_array)
    weighted_entropies=np.asarray(a=[entropy_weighted(
```

```
            feature_array=feature_array,
            feature_value=fv,
            targets=targets)
            for fv in unique_features])
    ent_feature=np.sum(a=weighted_entropies)
    return ent_feature

def entropy_parent(targets):
    """
        compute entropy for target values, treating all targets as if i
    """
    unique_targets=np.unique(ar=targets)
    targets_proportions=np.asarray(a=[list(targets).count(u)/float(tarc

    #computing entropy
    ent=-np.dot(a=targets_proportions, b=np.log2(targets_proportions))
    return ent

def information_gain(feature_array, targets):
    """
        compute information gain for a specific attribute/feature
    """
    parent_entropy=entropy_parent(targets=targets)
    info_gain=parent_entropy - entropy_feature(feature_array=feature_ar
    return info_gain

#preparing data into arrays and dataframes
#each attribute/feature
x1=np.array([0, 1, 0, 1, 0, 1, 0, 1, 1, 1])
x2=np.array([0, 1, 1, 1, 1, 0, 0, 0, 0, 1])
x3=np.array([1, 0, 1, 1, 0, 1, 1, 0, 1, 1])
x4=np.array([1, 1, 1, 1, 0, 1, 0, 0, 1, 1])
x5=np.array([0, 0, 1, 0, 0, 1, 0, 0, 0, 1])

#targets
y=np.array([-1, -1, -1, -1, -1, 1, 1, 1, 1, -1])

#shapes of each attribute/feature vector and targets
print(x1.shape, x2.shape, x3.shape, x4.shape, x5.shape, y.shape)
```

```
# calculating information gain for each attribute/feature
print("Information Gain Per Attribute/Feature:\n")
email_features=['x1', 'x2', 'x3', 'x4', 'x5']
information_gain_dict={}
index=0
for feature in [x1, x2, x3, x4, x5]:
    info_gain=information_gain(feature_array=feature, targets=y)
    information_gain_dict.update({str(email_features[index]) : info_gai
    index += 1

print(information_gain_dict)
```

```
(10,) (10,) (10,) (10,) (10,) (10,)
Information Gain Per Attribute/Feature:

{'x1': 0.0464393446710154, 'x2': 0.6099865470109874, 'x3': 0.0058021490143456145, 'x4': 0.091277
43456145}
```

In [30]:

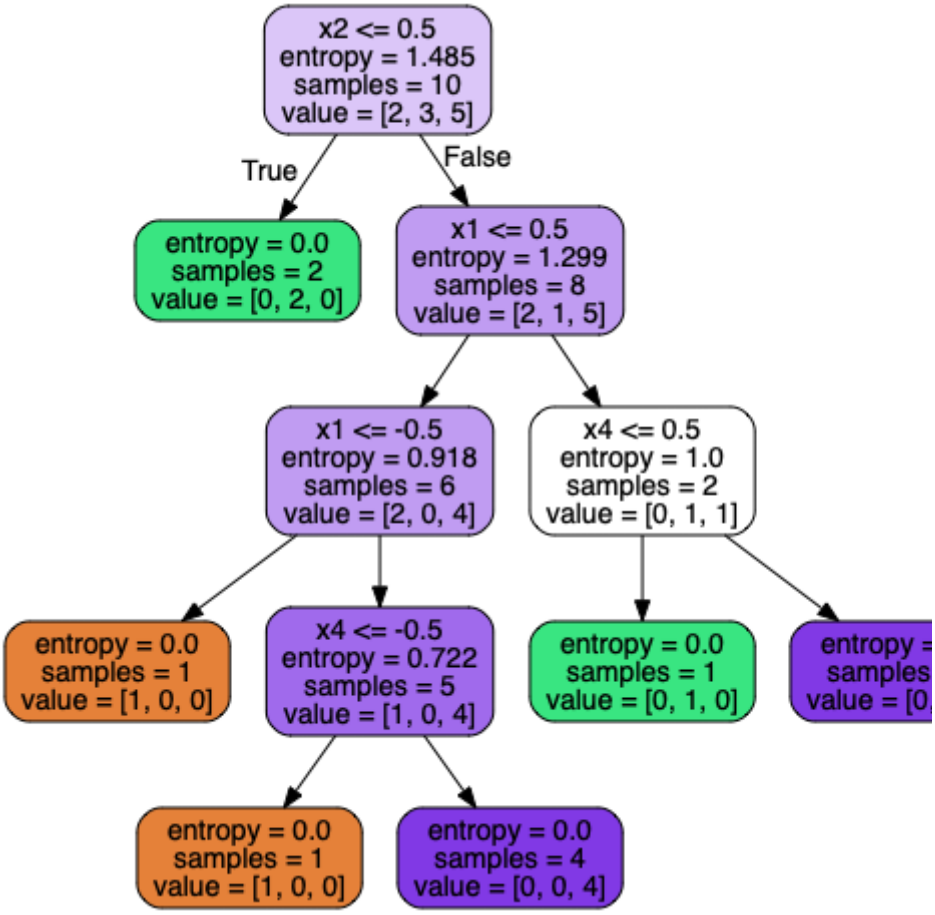
```
%pip install graphviz
```

```
Collecting graphviz
  Downloading graphviz-0.13.2-py2.py3-none-any.whl (17 kB)
Installing collected packages: graphviz
Successfully installed graphviz-0.13.2
Note: you may need to restart the kernel to use updated packages.
```

In [113]:

```
# 2.2
# Draw the complete decision tree that will be learned from this data.
# Justify every split based on information gain at every node.

from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz
from graphviz import Source

email_dt=DecisionTreeClassifier(criterion='entropy', splitter='best')
email_dt_fitted=email_dt.fit(X=X_train_email, y=y_train_email)

email_export=export_graphviz(
    decision_tree=email_dt,
    out_file='email_decision_tree.dot',
    feature_names=email_features,
    filled=True,
    rounded=True
)

Source(email_export);
```

Initially, the attribute/feature contributing the most information gain is x2 (length of em
as the root and determines the first split in the decision tree. At each further split, the
justified by having contributing the most information gain considering the subset of th
into its most recent split. Looking at the decision tree visual below, each additional le
entropy until reaching a leaf node with entropy = 0.



Problem 3 - Decision Tree

In this problem you will work with the Immunotherapy Dataset Data Set (https://archive.ics.uci.edu/ml/datasets/Immunotherapy+Dataset). This data set conta treatment results of 90 patients using immunotherapy. The data set contains 7 featur warts, type, area and induration diameter). The class label is the "result of treatment" successful and 1 means treatment was successful.

Answer the following questions:

1. Split the training data into training and test using a 80/20 split.
2. Learn a decision tree classifier on the data. Visualize the resulting tree using `plo` module `sklearn.tree` (there is an example in Lecture 12).
3. Now, try varying the maximum depth parameter (`max_depth`), which forces the t many levels. Test values $1, 2, 3, 4, 5, 6$ and compare their performance (both tra full depth. Is complexity increasing or decreasing with the depth cutoff? Identify v begins overfitting, and if so, when. If you use this parameter for complexity contr choose at best?
4. Now, using high maximum depth ($d = 6$), use `min_samples_leaf` to control cor Is complexity increasing or decreasing as `min_samples_leaf` grows? Identify wh to overfit, and what value you would use for this type of complexity control.

In [56]:

```
import pandas as pd
df=pd.read_excel('Immunotherapy.xlsx')

X=df.iloc[:,:6] # Training samples
y=df.iloc[:,7] # Training labels
features=df.columns.to_numpy()[:X.shape[1]+1]
targets=np.asarray(a=[df.columns.to_numpy()[X.shape[1]+1]])
print("fearures and target names:")
print(features, targets)
print("X and y types and shapes:")
print(type(X), type(y))
print(X.shape, y.shape)
print("Head of dataframe:")
print(df.head())
```

```
fearures and target names:
['sex' 'age' 'Time' 'Number_of_Warts' 'Type' 'Area' 'induration_diameter'] ['Result_of_Treatment
X and y types and shapes:
<class 'pandas.core.frame.DataFrame'> <class 'pandas.core.series.Series'>
(90, 6) (90,)
Head of dataframe:
   sex  age   Time  Number_of_Warts  Type  Area  induration_diameter  \
0    1   22   2.25               14     3    51                   50
1    1   15   3.00                2     3   900                   70
2    1   16  10.50                2     1   100                   25
3    1   27   4.50                9     3    80                   30
4    1   20   8.00                6     1    45                    8

   Result_of_Treatment
0                    1
1                    1
2                    1
3                    1
4                    1
```

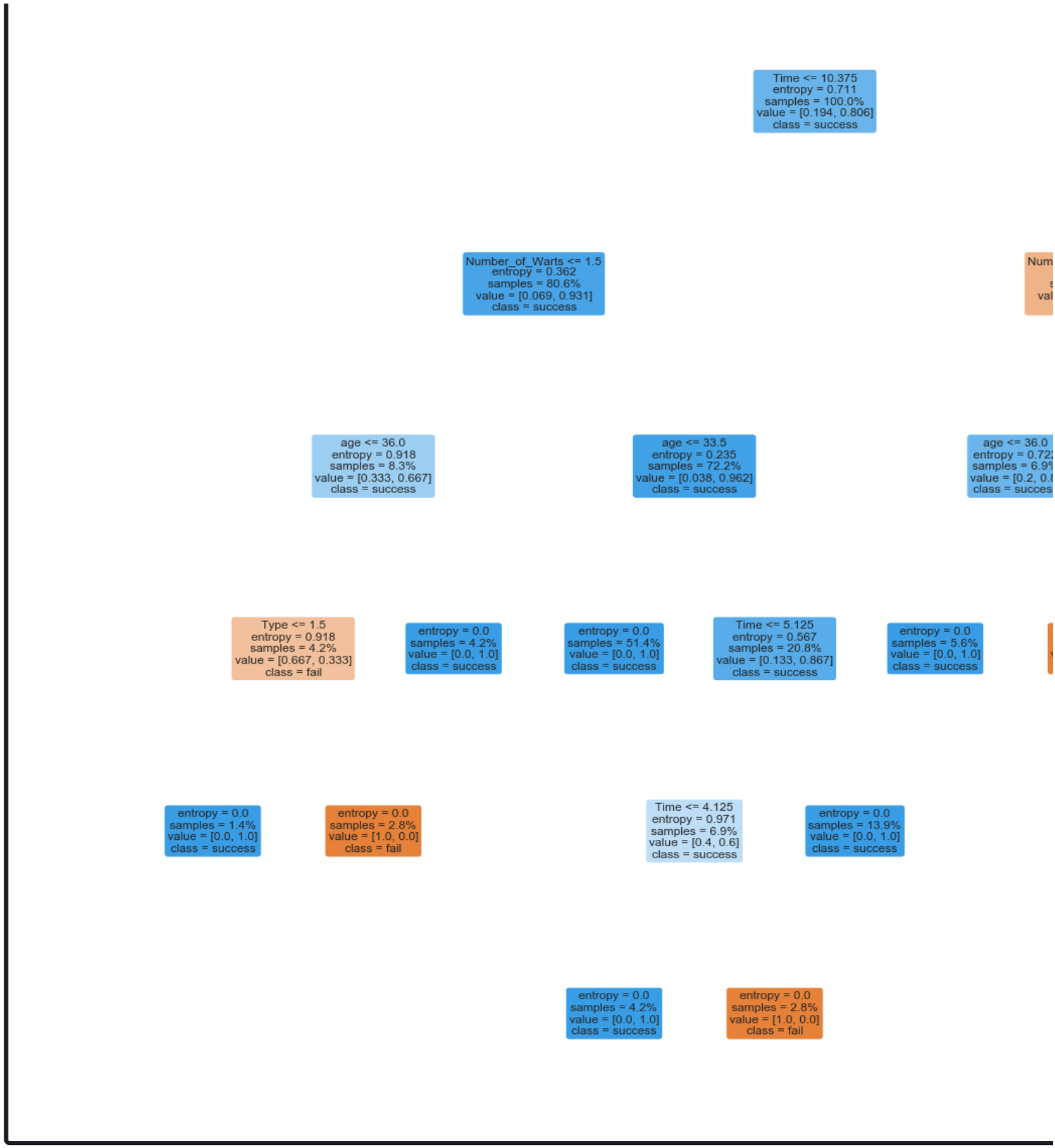
In [67]:

```
# 3.1
# Split the training data into training and test using a 80/20 split.
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
X_train_immuno, X_test_immuno, y_train_immuno, y_test_immuno=train_test_s

#converting to numpy arrays
X_train_immuno=X_train_immuno.to_numpy()
X_test_immuno=X_test_immuno.to_numpy()
y_train_immuno=y_train_immuno.to_numpy().reshape(y_train_immuno.size, 1)
y_test_immuno=y_test_immuno.to_numpy().reshape(y_test_immuno.size, 1)
print("Training and testing set shapes:")
print(X_train_immuno.shape, X_test_immuno.shape, y_train_immuno.shape, y_
print("X_train_immuno:\n{}".format(X_train_immuno))
print("X_test_immuno:\n{}".format(X_test_immuno))
print("y_train_immuno:\n{}".format(y_train_immuno))
print("y_test_immuno:\n{}".format(y_test_immuno))
```

```
[1]
[0]
[0]
[1]
[1]
[0]
[1]
[1]
[0]
[0]
[1]
[1]

[1]
[1]
[1]
[0]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
[1]
```

In [68]:

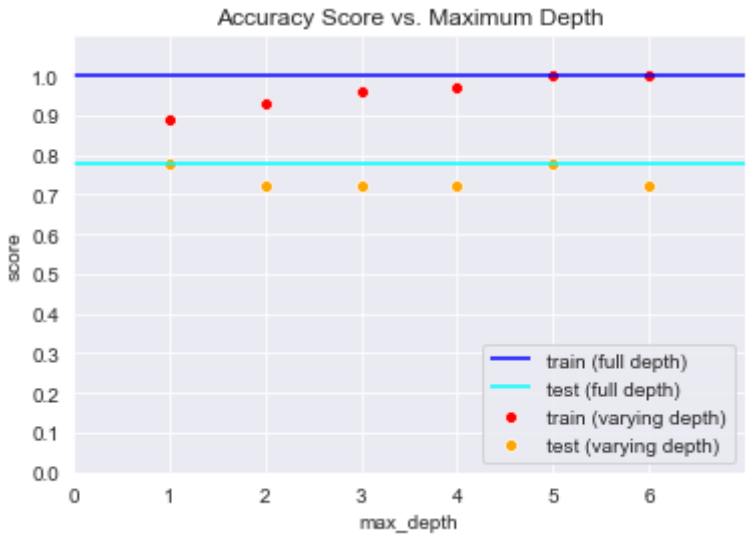
```
# 3.2
# Learn a decision tree classifier on the data.
# Visualize the resulting tree using plot_tree function from the module
# (there is an example in Lecture 12).
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
from sklearn.tree import export_graphviz
%matplotlib inline

dtc_immuno=DecisionTreeClassifier(criterion='entropy', splitter='best')
dtc_immuno_fitted=dtc_immuno.fit(X=X_train_immuno, y=y_train_immuno)

fig_immuno=plt.figure(figsize=(20, 20))
ax_immuno=plt.axes()
plot_tree(
        decision_tree=dtc_immuno_fitted,
        feature_names=features,
        class_names=['fail', 'success'],
        filled=True,
        impurity=True,
        proportion=True,
        rounded=True,
        fontsize=12,
        ax=ax_immuno
        )
plt.show()
```


Time <= 10.375
entropy = 0.711
samples = 100.0%
value = [0.194, 0.806]
class = success
Number_of_Warts <= 1.5
entropy = 0.362
samples = 80.6%
value = [0.069, 0.931]
class = success
Num
age <= 36.0
entropy = 0.918
samples = 8.3%
value = [0.333, 0.667]
class = success
age <= 33.5
entropy = 0.235
samples = 72.2%
value = [0.038, 0.962]
class = success
age <= 36.0
entropy = 0.72
samples = 6.9%
value = [0.2, 0.8
class = succes
Type <= 1.5
entropy = 0.918
samples = 4.2%
value = [0.667, 0.333]
class = fail
entropy = 0.0
samples = 4.2%
value = [0.0, 1.0]
class = success
entropy = 0.0
samples = 51.4%
value = [0.0, 1.0]
class = success
Time <= 5.125
entropy = 0.567
samples = 20.8%
value = [0.133, 0.867]
class = success
entropy = 0.0
samples = 5.6%
value = [0.0, 1.0]
class = success
entropy = 0.0
samples = 1.4%
value = [0.0, 1.0]
class = success
entropy = 0.0
samples = 2.8%
value = [1.0, 0.0]
class = fail
Time <= 4.125
entropy = 0.971
samples = 6.9%
value = [0.4, 0.6]
class = success
entropy = 0.0
samples = 13.9%
value = [0.0, 1.0]
class = success
entropy = 0.0
samples = 4.2%
value = [0.0, 1.0]
class = success
entropy = 0.0
samples = 2.8%
value = [1.0, 0.0]
class = fail

In [73]:

```
# 3.3
# Now, try varying the maximum depth parameter (max_depth), which force
# Test values 1,2,3,4,5,6 and compare their performance (both training
# Is complexity increasing or decreasing with the depth cutoff?
# Identify whether you think the model begins overfitting, and if so, w
# If you use this parameter for complexity control, what depth would yo

import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier

dtc=DecisionTreeClassifier(criterion='entropy', splitter='best')
dtc.fit(X=X_train_immuno, y=y_train_immuno)

#scores using the training set (no depth constraint)
dtc_full_depth_train_immuno_scores=dtc.score(X=X_train_immuno, y=y_trai

#scores using the testing set (no depth constraint)
dtc_full_depth_test_immuno_scores=dtc.score(X=X_test_immuno, y=y_test_i
dtc_train_immuno_scores=[] #scores using the training set (varying dept
dtc_test_immuno_scores=[] #scores using the testing set (varying depths
for depth in np.arange(1, 7):
    dtc=DecisionTreeClassifier(criterion='entropy', splitter='best', ma
    dtc.fit(X=X_train_immuno, y=y_train_immuno)
    dtc_train_immuno_scores.append(dtc.score(X=X_train_immuno, y=y_trai
    dtc_test_immuno_scores.append(dtc.score(X=X_test_immuno, y=y_test_i

sns.set_style(style='darkgrid')
plt.hlines(y=dtc_full_depth_train_immuno_scores, xmin=0, xmax=7, color=
plt.hlines(y=dtc_full_depth_test_immuno_scores, xmin=0, xmax=7, color='
sns.scatterplot(x=np.arange(1, 7), y=dtc_train_immuno_scores, color='re
sns.scatterplot(x=np.arange(1, 7), y=dtc_test_immuno_scores, color='ora
plt.title(label='Accuracy Score vs. Maximum Depth')
plt.xlabel(xlabel='max_depth')
plt.ylabel(ylabel='score')
plt.grid(b=True, which='major', axis='both')
plt.legend(labels=['train (full depth)', 'test (full depth)', 'train (v
           loc='lower right')
plt.xlim(0, 7)
```

```
plt.ylim(0, 1.1)
plt.xticks(ticks=np.arange(0, 7))
plt.yticks(ticks=np.arange(0, 1.1, 0.1))
plt.show()
```

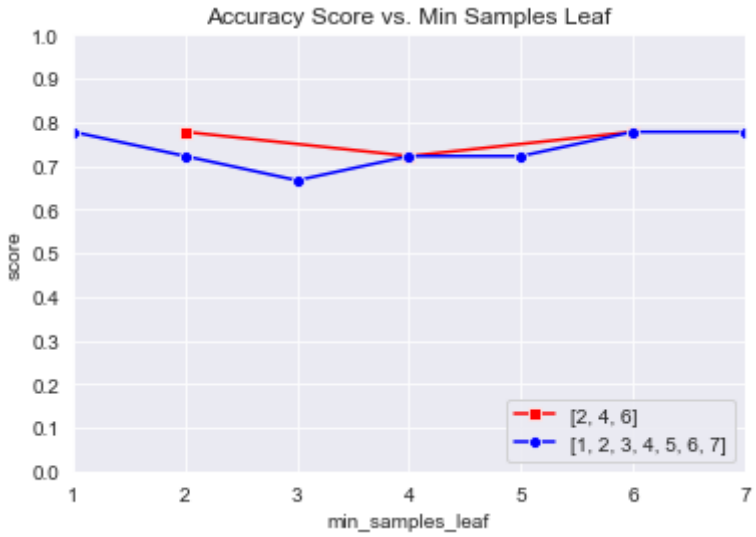


Firstly, note that the decision tree classifier is fitted on the X and y training sets befor
This makes a fair comparison by reserving the max_depth parameter to be the only v
values. Since the training set is used to fit the data, it is unsurprising that the score us
constant 1.0 for full depth and tapers closely to 1.0 across varying depths. The best e
occurs would depend on analyzing the scores for the test set across changing depths
a max_depth value of 1, the complexity starts of very simple as most of the targets ar
original data. This means that purely high chance of success rather than deduction fr
systematically classify a test point. What if a test point has y=0 actually? Therefore, a
deeper than just 1. As the max_depth increases to 4, the complexity increases and th
visually the same as the accuracy decreasing and then increasing. As more attributes
the decision tree, more complexity is added. However, better deduction by considerin
place simultaneously. At max_depth=4, the trade off between complexity and deducti
is paying off in favor of using more features. Afterwards, the score dips down again a
great. At 5 and after, overfitting starts to take effect. Note that both the training set an
score as each corresponding decision tree without a constraint on depth. That means
more, the classifier is overly familiarized with the original data used to fit the classifier

seems optimal and just right. It is not too small such that the imbalanced ratio of 1's t
successes correctly by sheer probability and not too large to induce overfitting either.
slightly when re-running the cell since the score() function calls the predict() function

In [75]:

```
# 3.4
# Now, using high maximum depth (d=6), use min_samples_leaf to control
# Try values {2,4,6}. Is complexity increasing or decreasing as min_sam
# Identify when (if) the model is starting to overfit, and what value y

import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
%matplotlib inline

dtc_immuno_6=DecisionTreeClassifier(max_depth=6)
dtc_immuno_6_scores=[]
for msl in [2, 4, 6]:
    dtc_immuno_6=DecisionTreeClassifier(max_depth=6, min_samples_leaf=m
    dtc_immuno_6.fit(X=X_train_immuno, y=y_train_immuno)
    dtc_immuno_6_scores.append(dtc_immuno_6.score(X=X_test_immuno, y=y_

dtc_immuno_6_all_scores=[]
for msl in np.arange(1, 8):
    dtc_immuno_6=DecisionTreeClassifier(max_depth=6, min_samples_leaf=m
    dtc_immuno_6.fit(X=X_train_immuno, y=y_train_immuno)
    dtc_immuno_6_all_scores.append(dtc_immuno_6.score(X=X_test_immuno,

#plotting accuracy scores vs. min_samples_leaf
sns.set_style(style='darkgrid')
sns.lineplot(x=[2, 4, 6], y=dtc_immuno_6_scores, color='red', marker='s
sns.lineplot(x=np.arange(1, 8), y=dtc_immuno_6_all_scores, color='blue'
plt.title(label='Accuracy Score vs. Min Samples Leaf')
plt.xlabel(xlabel='min_samples_leaf')
plt.ylabel(ylabel='score')
plt.grid(b=True, which='major', axis='both')
plt.legend(labels=['[2, 4, 6]', '[1, 2, 3, 4, 5, 6, 7]'], loc='lower ri
plt.xlim(2, 6)
plt.ylim(0, 1)
plt.xticks(ticks=np.arange(1, 8))
plt.yticks(ticks=np.arange(0, 1.1, 0.1))
plt.show()
```



Comparing only 2, 4, and 6 for min_samples_leaf, the score decreases when
increases from 2 to 4 before rising again. What most likely happened is t
naturally less than 4 data points (rows) ending up in some leaf nodes if
set to None. Setting the parameter to 4 forces some leaf nodes to accept
meet the size requirement to be 4 or more. This can explain the dip in sc
overfitting starts to take effect. Considering that there were way more s
(1's in targets) than failed treatments (0's in targets), having a min_sa
high would accidentally classify many test points as successful by chance
and test sets are derived from the same data set, which has a disproporti
1's to 0's, both training and test sets would have a similar ratio of 1's
overfitting would correctly classify test points that are 1 by chance rat
it is extremely common to encounter a 1 in both sets. The data suggests t
min_samples_leaf value. However, the scores based on min_samples_leaf set
also plotted. From the more complete plot, min_samples_leaf may have an o
but not equal to 2. Note that the graph changes slightly when re-running
score() function calls the predict() functions in possibly changed order.

Problem 4 - Random Forests

In this problem you will also work with the Immunotherapy Dataset Data Set (https://archive.ics.uci.edu/ml/datasets/Immunotherapy+Dataset).

Random Forests are bagged collections of decision trees, which select their decision samples. You can implement this easily in `BaggingClassifier` using option `n_estim` number of learners.

Using your validation split from the previous problem, learn a bagged ensemble of de data and evaluate both training and validation performance.

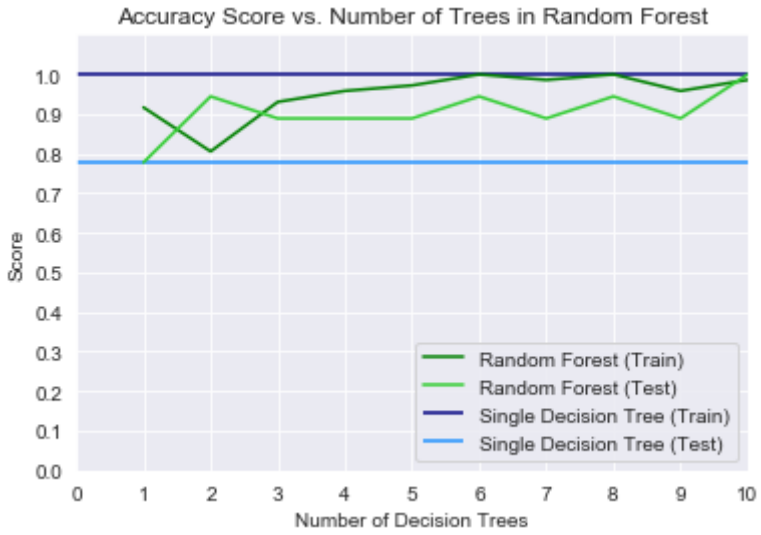
For your individual learners, use high complexity control (depth cutoff 6, `min_samples`
bagging will be used to control overfitting.

- How many decision trees would you use in your ensemble?
- Compare the performance of a **single** decision tree (with overfitting parameters `min_samples_leaf`) and a random forests composed on overfitting decision tree

In [83]:

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
%matplotlib inline

bc_immuno=BaggingClassifier()

bc_train_immuno_scores=[] #scores relating to number of trees used (prec
bc_test_immuno_scores=[] #scores relating to number of trees used (predi
max_n=10 #maximum number of decision trees in a single random forest to
for n in np.arange(1, max_n + 1):
    bc_immuno=BaggingClassifier(
        base_estimator=DecisionTreeClassifier(criterion='entropy', split
        n_estimators=n,
        bootstrap=True,
        bootstrap_features=True,
        n_jobs=1
    )
    bc_immuno.fit(X=X_train_immuno, y=y_train_immuno.flatten())
    bc_train_immuno_scores.append(bc_immuno.score(X=X_train_immuno, y=y_
    bc_immuno.fit(X=X_test_immuno, y=y_test_immuno.flatten())
    bc_test_immuno_scores.append(bc_immuno.score(X=X_test_immuno, y=y_te

# computing the score of a single decision tree with no restrictions on
# min_samples_leaf; prediction based on training set
dtc_overfit_immuno=DecisionTreeClassifier(criterion='entropy', splitter=
dtc_overfit_immuno.fit(X=X_train_immuno, y=y_train_immuno)
dtc_overfit_immuno_train_score=dtc_overfit_immuno.score(X=X_train_immunc

# computing the score of a single decision tree with no restrictions on
# min_samples_leaf; prediction based on test set
dtc_overfit_immuno_test_score=dtc_overfit_immuno.score(X=X_test_immuno,

#plotting scores
sns.set_style(style='darkgrid')
plt.hlines(y=dtc_overfit_immuno_train_score, xmin=0, xmax=max_n, color='
plt.hlines(y=dtc_overfit_immuno_test_score, xmin=0, xmax=max_n, color='d
sns.lineplot(x=np.arange(1, 11), y=np.asarray(a=bc_train_immuno_scores),
```

```
sns.lineplot(x=np.arange(1, 11), y=np.asarray(a=bc_test_immuno_scores),
plt.title(label='Accuracy Score vs. Number of Trees in Random Forest')
plt.xlabel(xlabel='Number of Decision Trees')
plt.ylabel(ylabel='Score')
plt.legend(labels=['Random Forest (Train)', 'Random Forest (Test)', 'Sin
           loc='lower right')
plt.grid(b=True, which='major', axis='both')
plt.xlim(0, max_n)
plt.ylim(0, 1.1)
plt.xticks(ticks=np.arange(max_n + 1))
plt.yticks(ticks=np.arange(0, 1.1, 0.1))
plt.show()
```



As expected, the prediction accuracy for the overfitted decision tree built on the traini
DecisionTreeClassifier also uses the training set for fitting. However, this helps us kn
As seen in the graph above, the random forest score for the training set slams into th
score for the training set at n_estimators = 6. This means that for all values 6 and aft
so familiarized with the training set used in fit() that the accuracy is nearly always 1 a
the overfitted decision tree. Therefore, the optimal value for n_estimators is right befc
optimal number of trees in the random forest as it is highly accurate without overfittin
forest is more accurate in predictions than a single decision tree. For the testing set,
forest (light green) and decision tree (cyan); the random forest always have a higher
tree score for the training set should be avoided for comparison since the fitting of th
on the training set.

Problem 5 - Support Vector Machines

In this problem you will be working with the Digits Data Set (https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html).

Build a pipeline structure using `make_pipeline` that (1) standardizes the data, (2) ex performs classification using SVM with an RBF kernel.

1. Split the data into training and test sets with an 80/20 split.
2. Using `GridSearchCV` find the best set of parameters for: the parameter `n_compo` number of components to project the data onto) in `PCA` and the parameters `C` (v hardness) and `gamma` (which controls the size of the radial basis function kernel)
3. Use the selected model to evaluate the results in both training and test set. Use assess the full classification metric report.

In [35]:

```
from sklearn.datasets import load_digits

# Training samples (X) and training labels (y)
X_digits, y_digits=load_digits(return_X_y=True)
y_digits=y_digits.reshape(y_digits.size, 1)

print(X_digits.shape, y_digits.shape)
print(type(X_digits), type(y_digits))
print("\nX_digits:\n{}".format(X_digits))
print("\ny_digits:\n{}".format(y_digits))
```

```
(1797, 64) (1797, 1)
<class 'numpy.ndarray'> <class 'numpy.ndarray'>

X_digits:
[[ 0.  0.  5. ...  0.  0.  0.]
 [ 0.  0.  0. ... 10.  0.  0.]
 [ 0.  0.  0. ... 16.  9.  0.]
 ...
 [ 0.  0.  1. ...  6.  0.  0.]
 [ 0.  0.  2. ... 12.  0.  0.]
 [ 0.  0. 10. ... 12.  1.  0.]]

y_digits:
[[0]
 [1]
 [2]
 ...
 [8]
 [9]
 [8]]


/Applications/anaconda3/lib/python3.7/importlib/_bootstrap.py:219: RuntimeWarning: numpy.ufunc s
y incompatibility. Expected 192 from C header, got 216 from PyObject
  return f(*args, **kwds)
```

In [37]:

```
# 5.1
# Split the data into training and test sets with an 80/20 split.
from sklearn.model_selection import train_test_split

X_train_digits, X_test_digits, y_train_digits, y_test_digits=train_test
print("Training and testing shapes:\n")
print(X_train_digits.shape, X_test_digits.shape, y_train_digits.shape,
print("\nX_train_digits:\n{}".format(X_train_digits))
print("\nX_test_digits:\n{}".format(X_test_digits))
print("\ny_train_digits:\n{}".format(y_train_digits))
print("\ny_test_digits:\n{}".format(y_test_digits))
```

```
[[ 0.  1. 14. ...  4.  0.  0.]
 [ 0.  0.  4. ...  0.  0.  0.]
 [ 0.  2. 15. ...  9.  1.  0.]
 ...
 [ 0.  0.  1. ... 11.  2.  0.]
 [ 0.  0.  5. ... 15.  3.  0.]
 [ 0.  0.  8. ...  5.  0.  0.]]

X_test_digits:
[[ 0.  0.  9. ...  8.  0.  0.]
 [ 0.  0.  6. ...  0.  0.  0.]
 [ 0.  0.  1. ... 16. 13.  1.]
 ...
 [ 0.  0. 12. ... 15.  1.  0.]
 [ 0.  0.  0. ... 12.  0.  0.]
 [ 0.  2.  9. ...  9.  1.  0.]]

y_train_digits:
[[5]
 [1]
 [2]
 ...
 [6]
 [9]
 [3]]
```

In [39]:

```
# 5.2
# Using GridSearchCV find the best set of parameters for:
# the parameter n_components (which control the number of components to
# and the parameters C (which controls the margin hardness)
# and gamma (which controls the size of the radial basis function kerne

import numpy as np
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

#best hyper-parameter values for PCA
gs_digits_pca=GridSearchCV(
                    estimator=PCA(),
                    param_grid={'n_components' : np.arange(1, 65)},
                    refit=True,
                    n_jobs=1
                    )
gs_digits_pca.fit(X=X_train_digits, y=y_train_digits.flatten())
best_n_components=gs_digits_pca.best_estimator_.n_components
print("Best n_components value:\t{}".format(best_n_components))

#best hyper-parameter values for svm.SVC
#will take a long time to run
gs_digits_svc=GridSearchCV(estimator=SVC(kernel='rbf'),
                           param_grid={'C' : np.arange(0.1, 2.0, 0.1),
                                       'gamma' : np.arange(0.001, 0.01, 0
                            refit=True,
                            n_jobs=1
                           )
gs_digits_svc.fit(X=X_train_digits, y=y_train_digits.flatten())
best_C=gs_digits_svc.best_estimator_.C
best_gamma=gs_digits_svc.best_estimator_.gamma
print("Best C value:\t{}".format(best_C))
print("Best gamma value:\t{}".format(best_gamma))
```

```
/Applications/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_search.py:788: Runt
ntered in subtract
```

```
            array_means[:, np.newaxis]) ** 2,


Best n_components value:        57
Best C value:   1.1
Best gamma value:       0.001
```

In [43]:

```
# 5.3
# Use the selected model to evaluate the results in both training and t
# Use classification_report to assess the full classification metric re

from sklearn.metrics import classification_report
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.svm import SVC

# building pipeline
pipeline_digits=make_pipeline(
    StandardScaler(),
    PCA(n_components=best_n_components),
    SVC(C=best_C, kernel='rbf', gamma=best_gamma),
    verbose=True
)

print("pipeline:\n{}".format(pipeline_digits))

y_pred_train_digits=pipeline_digits.fit(X=X_train_digits, y=y_train_dig
cr_train_digits=classification_report(y_true=y_train_digits, y_pred=y_p
print("\nClassification Report for Training Set:\n{}".format(cr_train_d

y_pred_test_digits=pipeline_digits.fit(X=X_train_digits, y=y_train_digi
cr_test_digits=classification_report(y_true=y_test_digits, y_pred=y_pre
print("\nClassification Report for Test Set:\n{}".format(cr_test_digits
```

```
pipeline:
Pipeline(memory=None,
         steps=[('standardscaler',
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
                ('pca',
                 PCA(copy=True, iterated_power='auto', n_components=57,
                     random_state=None, svd_solver='auto', tol=0.0,
                     whiten=False)),
                ('svc',
                 SVC(C=1.1, break_ties=False, cache_size=200, class_weight=None,
                     coef0=0.0, decision_function_shape='ovr', degree=3,
                     gamma=0.001, kernel='rbf', max_iter=-1, probability=False,
                     random_state=None, shrinking=True, tol=0.001,
                     verbose=False))],
         verbose=True)
```

```
[Pipeline] .... (step 1 of 3) Processing standardscaler, total=   0.0s
[Pipeline] ............... (step 2 of 3) Processing pca, total=   0.0s
[Pipeline] ............... (step 3 of 3) Processing svc, total=   0.1s


/Applications/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.py:760: DataConvers
as passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for examp
  y = column_or_1d(y, warn=True)



Classification Report for Training Set:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       142
           1       0.91      0.98      0.94       148
           2       0.99      0.97      0.98       148
           3       0.99      0.93      0.96       138
           4       0.99      0.97      0.98       145
           5       0.98      0.96      0.97       143
           6       0.99      0.99      0.99       151
           7       0.94      0.99      0.96       143
           8       0.91      0.91      0.91       135
           9       0.92      0.92      0.92       144

    accuracy                           0.96      1437
   macro avg       0.96      0.96      0.96      1437
weighted avg       0.96      0.96      0.96      1437

[Pipeline] .... (step 1 of 3) Processing standardscaler, total=   0.0s
[Pipeline] ............... (step 2 of 3) Processing pca, total=   0.0s
[Pipeline] ............... (step 3 of 3) Processing svc, total=   0.1s

Classification Report for Test Set:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        36
           1       0.92      1.00      0.96        34
           2       1.00      1.00      1.00        29
           3       1.00      0.89      0.94        45
           4       1.00      1.00      1.00        36
           5       0.95      1.00      0.97        39
           6       1.00      0.97      0.98        30
           7       0.92      1.00      0.96        36
           8       0.95      0.90      0.92        39
           9       0.94      0.94      0.94        36

    accuracy                           0.97       360
   macro avg       0.97      0.97      0.97       360
weighted avg       0.97      0.97      0.97       360



/Applications/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.py:760: DataConvers
as passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for examp
  y = column_or_1d(y, warn=True)
```

Problem 6 - The Perceptron

Is Rosenblatt's Perceptron a universal learner (universal function approximator)? If n
justify its implications.

No. To start with, part of the reason Rosenblatt created the Perceptron was to enhan
earlier McCulloch-Pits artificial neuron. Nonetheless, the Perceptron remained with a
convergence. A Perceptron simply sums up the weighted inputs, produces an approp
adjusts the weights using the Perceptron algorithm. There were no hidden layers inv
the Universal Approximation Theorem, says that any real function in any dimension c
neural network with at least a single hidden layer of finite amount of neurons. Unfortu
handles the inputs and computes the output directly with no layers at all. A simple ex
Perceptron cannot learn all real functions. Suppose we have a trigonometric function
regularly, such as f(x)=sin(x). There input dimension is 1, and the output dimension is
the bias is, training the Perceptron would lead to the weights being adjusted back an
cyclic nature of the function. From a general perspective, a single Perceptron is very
single heaviside function. A single Perceptron can model a constant function i.e. f(x)=
f(x)=sgn(x) effectively, but most curvy functions would need multiple Perceptrons to w
combine many heaviside (or step) functions together. That way, a step-like approxima
function linked to some neuron, is modeled by a whole neural network resembling a r
Perceptron cannot accomplish this demanding task.

Problem 7 - Multi-Layer Perceptron Decisi Surface

The problem that inspired MLPs and the learning rule is the Exclusive OR (XOR) pro

$$\text{XOR}(x_1, x_2) = x_1\bar{x}_2 + \bar{x}_1 x_2$$

x_1	x_2	t
0	0	0
0	1	1
1	0	1
1	1	0

Consider the activation function as heaviside function:

$$\phi(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

Suppose you had the noisy XOR data shown in the figure below. Answer the followin

1. Design an MLP that can correctly solve this classification problem. (Find the wei

2. What does the decision surface of this network look like graphically? Draw it out

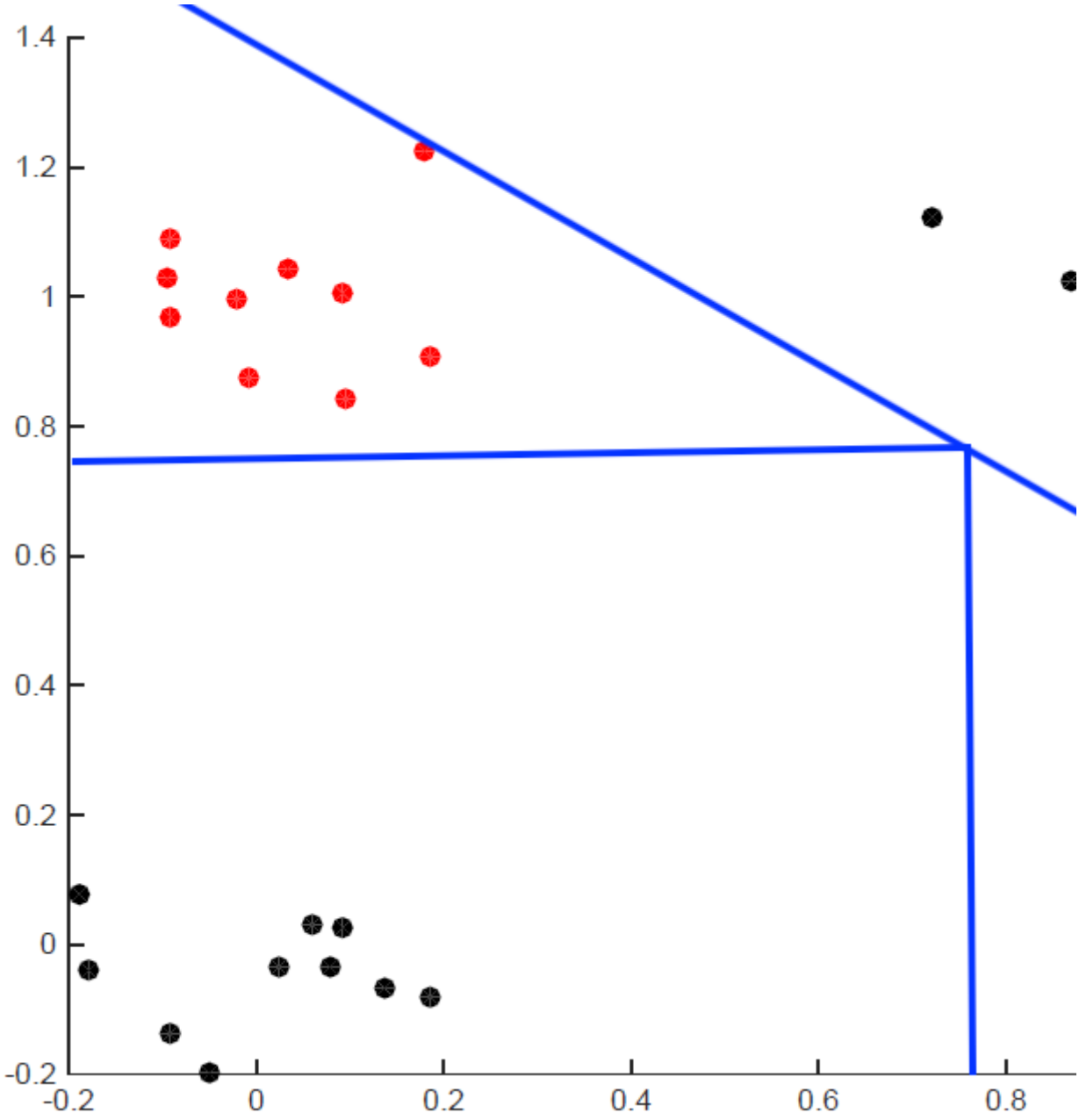
3. For the MLP you designed, what is the predicted label for the test point $[0.5, 0.5$

In [3]:

```
# 7.1
# Design an MLP that can correctly solve this classification problem. (
weights={'X1-M1':1, 'X1-M2':1, 'X2-M2':1, 'X2-M3':1, 'M1-t':1, 'M2-t':-
bias={'b-M1':-0.75, 'b-M2':-1.5, 'b-M3':-0.75}
print("weights:\t{}".format(weights))
print("bias:\t{}".format(bias))
```

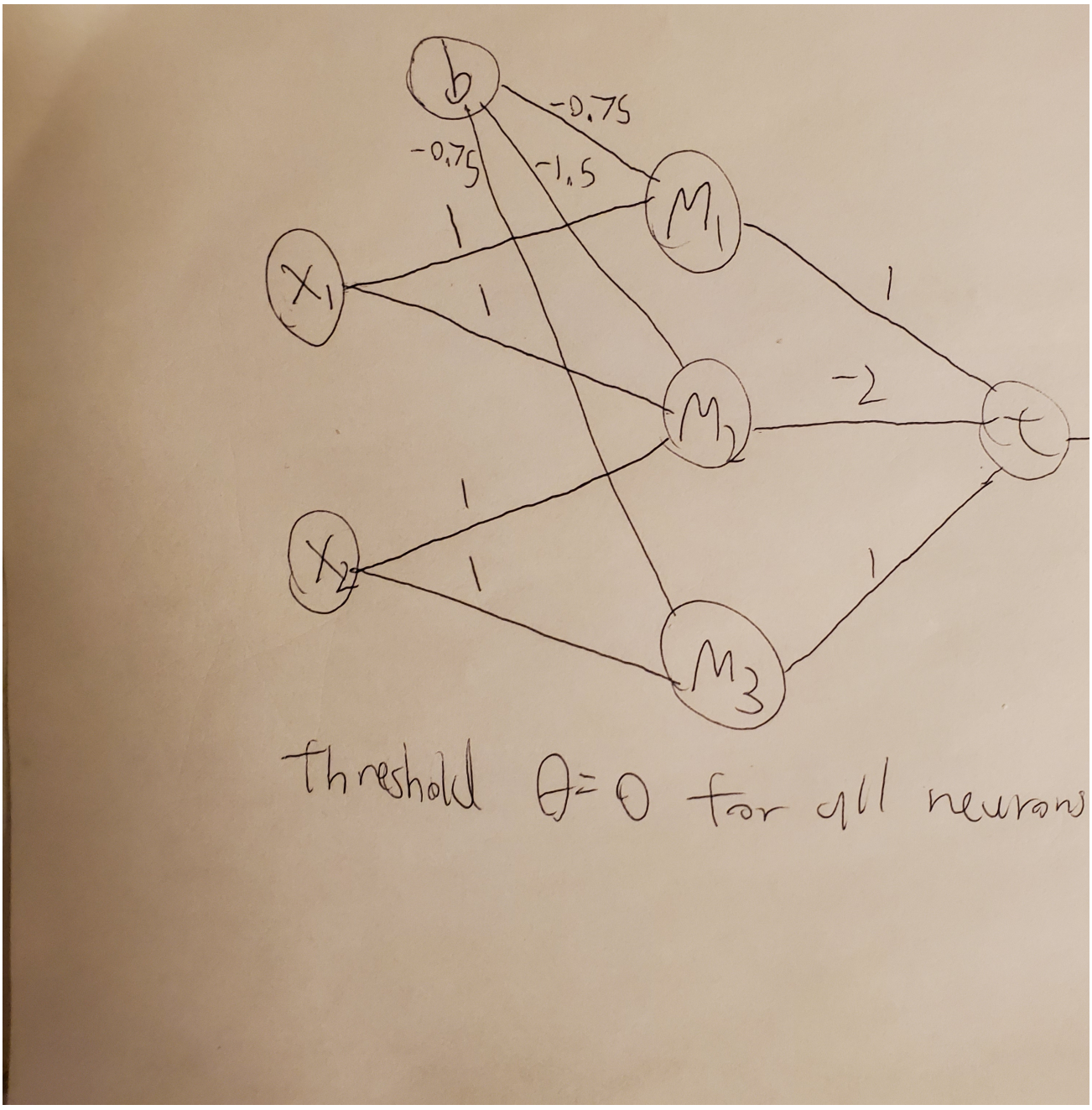
```
weights:        {'X1-M1': 1, 'X1-M2': 1, 'X2-M2': 1, 'X2-M3': 1, 'M1-t': 1, 'M2-t': -2, 'M3-t':
bias:   {'b-M1': -0.75, 'b-M2': -1.5, 'b-M3': -0.75}
```

There are 2 inputs {X1, X2}, a single hidden layer of 3 neurons {M1, M2, M3}, and a s
are all 1's except for the edge between M2 and t, where the weight is -2. The bias, b,
-0.75} respectively for the hidden layer. The weights and bias values are also given ir
the MLP draws 3 decision boundaries. A point (x1, x2) is only classified as true only i
greater than 0.75 and the sum of the inputs is less than 1.5. This MLP conveys impor
be close to (1, 0) or (0, 1) and be inside the existing red clusters. In other words, any
existing red clusters will more likely be classfied as false for XOR than true.



In [ ]:

```
# 7.2
# What does the decision surface of this network look like graphically?
```



In [ ]:

```
# 7.3
# For the MLP you designed, what is the predicted label for the test pc
```

Since the point (0.5, 0.5) fails the criteria requiring either parameter to be greater tha as false; it belongs to the black cluster near (0, 0).