```python
import pandas as pd
import numpy as np
from pprint import pprint
import sys

# Reads the data from CSV files, each attribute column can be obtained via its name, e.g., y = data['y']
def getDataframe(filePath):
    data = pd.read_csv(filePath)
    return data

# predicted_y and y are the predicted and actual y values respectively as numpy arrays
# function prints the accuracy
def compute_accuracy(predicted_y, y):
    acc = 100.0
    acc = np.sum(predicted_y == y)/predicted_y.shape[0]
    return acc

#Compute entropy according to y distribution
def compute_entropy(y):
    entropy = 0.0
    elements,counts = np.unique(y, return_counts = True)
    n = y.shape[0]

    for i in range(len(elements)):
        prob = counts[i]/n
        if prob!= 0:
            entropy -= prob * np.log2(prob)
    return entropy

#att_name: attribute name; y_name: the target attribute name for classification
def compute_info_gain(data, att_name, y_name):
    info_gain = 0.0

    #Calculate the values and the corresponding counts for the select attribute
    vals, counts = np.unique(data[att_name], return_counts=True)
    total_counts = np.sum(counts)
    #Calculate the conditional entropy
    #=======================#
    # STRART YOUR CODE HERE  #
    #=======================#
    total_info = compute_entropy(data[y_name])
    info_A = 0.0
    for i in range(len(vals)):
        info_A += (counts[i]/total_counts) * compute_entropy(data.loc[data[att_name] == vals[i]][y_name])
    #=======================#
    #   END YOUR CODE HERE   #
    #=======================#
    info_gain = total_info - info_A
    return info_gain


def comput_gain_ratio(data, att_name, y_name):
    gain_ratio = 0.0
    #Calculate the values and the corresponding counts for the select
```

```
55      vals, counts = np.unique(data[att_name], return_counts=True)
56      total_counts = np.sum(counts)
57
58      #Calculate the information for the selected attribute
59      att_info = 0.0
60      #========================#
61      # STRART YOUR CODE HERE  #
62      #========================#
63      for i in range(len(vals)):
64          p = counts[i] / total_counts
65          att_info -= p * np.log2(p)
66      #========================#
67      #   END YOUR CODE HERE   #
68      #========================#
69      gain_ratio = 0.0 if np.abs(att_info) < 1e-9 else min(1,
compute_info_gain(data, att_name, y_name) / att_info)
70      return gain_ratio
71
72  # Class of the decision tree model based on the ID3 algorithm
73  class DecisionTree(object):
74      def __init__(self):
75          self.train_data = pd.DataFrame()
76          self.test_data = pd.DataFrame()
77
78      def load_data(self, train_file, test_file):
79          self.train_data = getDataframe(train_file)
80          self.test_data = getDataframe(test_file)
81
82      def train(self, y_name, measure, parent_node_class = None):
83          self.y_name = y_name
84          self.measure = measure
85          self.tree = self.make_tree(self.train_data, parent_node_class)
86
87      def make_tree(self, train_data, parent_node_class = None):
88          data = train_data
89          features = data.drop(self.y_name, axis = 1).columns.values
90          measure =  self.measure
91          #Stopping condition 1: If all target_values have the same value,
return this value
92          if len(np.unique(data[self.y_name])) <= 1:
93              leaf_value = -1
94              #========================#
95              # STRART YOUR CODE HERE  #
96              #========================#
97              leaf_node = np.unique(data[self.y_name])[0]
98              #========================#
99              #   END YOUR CODE HERE   #
100             #========================#
101             return leaf_node
102
103         #Stopping condition 2: If the dataset is empty, return the
parent_node_class
104         elif len(data)== 0:
105             return parent_node_class
106
107         #Stopping condition 3: If the feature space is empty, return the
majority class
108         elif len(features) == 0:
109             return np.unique(data[self.y_name])
```

```
110
111         # Not a leaf node, create an internal node
112         else:
113             #Set the default value for this node --> The mode target feature
    value of the current node
114             parent_node_class = np.unique(data[self.y_name])
    [np.argmax(np.unique(data[self.y_name],return_counts=True)[1])]
115
116             #Select the feature which best splits the dataset
117             if measure == 'info_gain':
118                 item_values = [compute_info_gain(data, feature, self.y_name)
    for feature in features] #Return the information gain values for the features
    in the dataset
119             elif measure == 'gain_ratio':
120                 item_values = [comput_gain_ratio(data, feature, self.y_name)
    for feature in features] #Return the gain_ratio for the features in the
    dataset
121             else:
122                 raise ValueError("kernel not recognized")
123
124             best_feature_index = np.argmax(item_values)
125             best_feature = features[best_feature_index]
126             print('best_feature is: ', best_feature)
127
128             #Create the tree structure. The root gets the name of the feature
    (best_feature)
129             tree = {best_feature:{}}
130
131
132         #Grow a branch under the root node for each possible value of the
    root node feature
133
134         for value in np.unique(data[best_feature]):
135             #Split the dataset along the value of the feature with the
    largest information gain and therwith create sub_datasets
136             sub_data = data.where(data[best_feature] == value).dropna()
137
138             #Remove the selected feature from the feature space
139             sub_data = sub_data.drop(best_feature, axis = 1)
140
141             #Call the ID3 algorithm for each of those sub_datasets with the
    new parameters --> Here the recursion comes in!
142             subtree = self.make_tree(sub_data, parent_node_class)
143
144             #Add the sub tree, grown from the sub_dataset to the tree under
    the root node
145             tree[best_feature][value] = subtree
146
147         return tree
148
149
150     def test(self, y_name):
151         accuracy = self.classify(self.test_data, y_name)
152         return accuracy
153
154     def classify(self, test_data, y_name):
155         #Create new query instances by simply removing the target feature
    column from the test dataset and
156         #convert it to a dictionary
```

```
158         test_y = test_data[y_name]
159 
160         n = test_data.shape[0]
161         predicted_y = np.zeros(n)
162 
163         #Calculate the prediction accuracy
164         for i in range(n):
165             predicted_y[i] = DecisionTree.predict(self.tree, test_x.iloc[i])
166 
167         output = np.zeros((n,2))
168         output[:,0] = test_y
169         output[:,1] = predicted_y
170         accuracy = compute_accuracy(predicted_y, test_y.values)
171         return accuracy
172 
173     def predict(tree, query):
174         # find the root attribute
175         default = -1
176         for root_name in list(tree.keys()):
177             try:
178                 subtree = tree[root_name][query[root_name]]
179             except:
180                 return default ## root_name does not appear in query
    attribute list (it is an error!)
181 
182             ##if subtree is still a dictionary, recursively test next
    attribute
183             if isinstance(subtree,dict):
184                 return DecisionTree.predict(subtree, query)
185             else:
186                 leaf = subtree
187                 return leaf
188 
189 
```

```python
import numpy as np
from numpy import linalg
import cvxopt
import cvxopt.solvers
import sys
import pandas as pd
cvxopt.solvers.options['show_progress'] = False

# Reads the data from CSV files, converts it into Dataframe and returns x and y dataframes
def getDataframe(filePath):
    dataframe = pd.read_csv(filePath)
    y = dataframe['y']
    x = dataframe.drop('y', axis=1)
    y = y*2 -1.0
    return x.to_numpy(), y.to_numpy()

def compute_accuracy(predicted_y, y):
    acc = 100.0
    acc = np.sum(predicted_y == y)/predicted_y.shape[0]
    return acc

def gaussian_kernel_point(x, y, sigma=5.0):
    return np.exp(-linalg.norm(x-y)**2 / (2 * (sigma ** 2)))

def linear_kernel(X, Y=None):
    Y = X if Y is None else Y
    m = X.shape[0]
    n = Y.shape[0]
    assert X.shape[1] == Y.shape[1]
    kernel_matrix = np.zeros((m, n))
    #=======================#
    # STRART YOUR CODE HERE  #
    #=======================#
    for i in range(m):
        for j in range(n):
            kernel_matrix[i,j] = np.dot(X[i], Y[j])
    #=======================#
    #   END YOUR CODE HERE   #
    #=======================#
    return kernel_matrix

def polynomial_kernel(X, Y=None, degree=3):
    Y = X if Y is None else Y
    m = X.shape[0]
    n = Y.shape[0]
    assert X.shape[1] == Y.shape[1]
    kernel_matrix = np.zeros((m, n))
    #=======================#
    # STRART YOUR CODE HERE  #
    #=======================#
    for i in range(m):
        for j in range(n):
            kernel_matrix[i,j] = (np.dot(X[i], Y[j]) + 1) ** degree
    #=======================#
    #   END YOUR CODE HERE   #
    #=======================#
    return kernel_matrix

```

```
    Y = X if Y is None else Y
    m = X.shape[0]
    n = Y.shape[0]
    assert X.shape[1] == Y.shape[1]
    kernel_matrix = np.zeros((m, n))
    #=======================#
    # STRART YOUR CODE HERE  #
    #=======================#
    for i in range(m):
        for j in range(n):
            kernel_matrix[i,j] = gaussian_kernel_point(X[i], Y[j])
    #=======================#
    #   END YOUR CODE HERE   #
    #=======================#
    return kernel_matrix


# Bonus question: vectorized implementation of Gaussian kernel
# If you decide to do the bonus question, comment the gaussian_kernel function above,
# then implement and uncomment this one.
# def gaussian_kernel(X, Y=None, sigma=5.0):
#     return

class SVM(object):
    def __init__(self):
        self.train_x = pd.DataFrame()
        self.train_y = pd.DataFrame()
        self.test_x = pd.DataFrame()
        self.test_y = pd.DataFrame()
        self.kernel_name = None
        self.kernel = None

    def load_data(self, train_file, test_file):
        self.train_x, self.train_y = getDataframe(train_file)
        self.test_x, self.test_y = getDataframe(test_file)


    def train(self, kernel_name='linear_kernel', C=None):
        self.kernel_name = kernel_name
        if(kernel_name == 'linear_kernel'):
            self.kernel = linear_kernel
        elif(kernel_name == 'polynomial_kernel'):
            self.kernel = polynomial_kernel
        elif(kernel_name == 'gaussian_kernel'):
            self.kernel = gaussian_kernel
        else:
            raise ValueError("kernel not recognized")

        self.C = C
        if self.C is not None:
            self.C = float(self.C)

        self.fit(self.train_x, self.train_y)

    # predict labels for test dataset
    def predict(self, X):
        if self.w is not None: ## linear case
            n = X.shape[0]
```

```
            #========================#
            # STRART YOUR CODE HERE  #
            #========================#
            predicted_y = np.dot(X, self.w) + self.b
            #========================#
            #   END YOUR CODE HERE   #
            #========================#
            return predicted_y

        else: ## non-linear case
            n = X.shape[0]
            predicted_y = np.zeros(n)
            #========================#
            # STRART YOUR CODE HERE  #
            #========================#
            kernel_matrix = self.kernel(X, self.sv)
            predicted_y = np.dot(kernel_matrix * self.sv_y, self.a)  + self.b

            #========================#
            #   END YOUR CODE HERE   #
            #========================#
            return predicted_y

    #================================================#
    # Please DON'T change any code below this line!  #
    #================================================#
    def fit(self, X, y):
        n_samples, n_features = X.shape
        # Kernel matrix
        K = self.kernel(X)

        # dealing with dual form quadratic optimization
        P = cvxopt.matrix(np.outer(y,y) * K)
        q = cvxopt.matrix(np.ones(n_samples) * -1)
        A = cvxopt.matrix(y, (1,n_samples),'d')
        b = cvxopt.matrix(0.0)

        if self.C is None:
            G = cvxopt.matrix(np.diag(np.ones(n_samples) * -1))
            h = cvxopt.matrix(np.zeros(n_samples))
        else:
            tmp1 = np.diag(np.ones(n_samples) * -1)
            tmp2 = np.identity(n_samples)
            G = cvxopt.matrix(np.vstack((tmp1, tmp2)))
            tmp1 = np.zeros(n_samples)
            tmp2 = np.ones(n_samples) * self.C
            h = cvxopt.matrix(np.hstack((tmp1, tmp2)))

        # solve QP problem
        solution = cvxopt.solvers.qp(P, q, G, h, A, b)
        # Lagrange multipliers
        a = np.ravel(solution['x'])

        # Support vectors have non zero lagrange multipliers
        sv = a > 1e-5
        ind = np.arange(len(a))[sv]
        self.a = a[sv]
        self.sv = X[sv]
        self.sv_y = y[sv]
```

```
178         print("%d support vectors out of %d points" % (len(self.a),
    n_samples))
179 
180         # Intercept via average calculating b over support vectors
181         self.b = 0
182         for n in range(len(self.a)):
183             self.b += self.sv_y[n]
184             self.b -= np.sum(self.a * self.sv_y * K[ind[n],sv])
185         self.b /= len(self.a)
186 
187         # Weight vector
188         if self.kernel_name == 'linear_kernel':
189             self.w = np.zeros(n_features)
190             for n in range(len(self.a)):
191                 self.w += self.a[n] * self.sv_y[n] * self.sv[n]
192         else:
193             self.w = None
194 
195 
196     def test(self):
197         accuracy = self.classify(self.test_x, self.test_y)
198         return accuracy
199 
200     def classify(self, X, y):
201         predicted_y = np.sign(self.predict(X))
202         accuracy = compute_accuracy(predicted_y, y)
203         return accuracy
```

CS145 Howework 2

Important Note: HW2 is due on **11:59 PM PT, Oct 30 (Friday, Week 4)**. Please submit through GradeScope.

Print Out Your Name and UID

**Name: Ali Mirabzadeh, UID: 305179067 **

Before You Start

You need to first create HW2 conda environment by the given `cs145hw2.yml` file, which provides the name and necessary packages for this tasks. If you have `conda` properly installed, you may create, activate or deactivate by the following commands:

```
conda env create -f cs145hw2.yml
conda activate hw1
conda deactivate
```

OR

```
conda env create --name NAMEOFYOURCHOICE -f cs145hw2.yml
conda activate NAMEOFYOURCHOICE
conda deactivate
```

To view the list of your environments, use the following command:

```
conda env list
```

More useful information about managing environments can be found here (https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html).

You may also quickly review the usage of basic Python and Numpy package, if needed in coding for matrix operations.

In this notebook, you must not delete any code cells in this notebook. If you change any code outside the blocks (such as some important hyperparameters) that you are allowed to edit (between STRART/END YOUR CODE HERE), you need to highlight these changes. You may add some additional cells to help explain your results and observations.

In [10]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import sys
import random as rd
import matplotlib.pyplot as plt
%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

If you can successfully run the code above, there will be no problem for environment setting.

1. Decision trees

This workbook will walk you through a decision tree.

1.1 Attribute selection measures

For classification models, misclassification rate is usually used as the final performance measurement. However, for classification trees, when selecting which attribute to split, measurements people often use includes information gain, gain ratio, and Gini index. Let's investigate these different measurements through the following problem.

Note: below shows how to calculate the misclassification rate of a classification tree with N total data points, K classes of the value we want to predict, and M leaf nodes.

In a node m, $m = 1, \ldots, M$, let's denote the number of data points using N_m, and the number of data points in class k as N_{mk}, so the class prediction under majority vote is $j = argmax_k N_{mk}$. The misclassification rate of this node m is $R_m = 1 - \frac{N_{mj}}{N_m}$. The total misclassification rate of the tree will be $R = \frac{\sum_{m=1}^{M} R_m * N_m}{N}$

Questions

Note: this question is a pure "question answer" problem. You don't need to do any coding.

Suppose our dataset includes a total of 800 people with 400 males and 400 females, and our goal is to do gender classification. Consider two different possible attributes we can split on in a decision tree model. Split on the first attribute results in a node11 with 300 male and 100 female, and a node12 with 100 male and 300 female. Split on the second attribute results in in a node21 with 400 male and 200 female, and a node22 with 200 female only.

1. Which split do you prefer when the measurement is misclassifcation rate and why?
2. What is the entropy in each of these four node?
3. What is the information gain of each of the two splits?

4. Which split do you prefer if the measurement is information gain. Do you see why it is an uncertainty or impurity measurement?
5. What is the gain ratio (normalized information gain) of each of the two splits? Which split do you prefer under this measurement. Do you get the same conclusion as information gain?

Your answer here:

Note: you can use several code cells to help you compute the results and answer the questions. Again you don't need to do any coding.

Please type your answer here!

answer 1: I prefer the 2nd split as has a lower $R_m = 0.33$ whereas the first one has $R_m = 0.5$

answer 2 :
node11: $entropy(3/4, 1/4) = -3/4log2(3/4) - 1/4log2(1/4) = 0.811$
node12: $entropy(1/4, 3/4) = -1/4log2(1/4) - 3/4log2(3/4) = 0.811$
node21: $entropy(2/3, 1/3) = -2/3log2(2/3) - 1/3log2(1/3) = 0.918$
node22: $entropy(0, 1) = -0log2(0) - 1log2(1) = 0$

$info(beforesplit1) = entropy(1/2, 1/2) = 1$
$info(beforesplit2) = entropy(1/2, 1/2) = 1$

answer 3:
info gain(split1) =
$info([400, 400]) - info([300, 100], [100, 300]) = 1 - ((1/2)0.811) + (1/2) * 0.811) = 0.189$
info gain(split2) =
$info([600, 200]) - info([400, 200], [0, 200]) = 1 - ((3/4)0.918) + (1/4) * 0) = 0.312$

answer 4: Split1 is better as it has a better information gain. Yes, I can see that information gain is biased towards attributes with larger values

answer 5:
Gain Ratio (split1) = $0.189/1 = 0.189$
Gain Ratio (split2) = $0.312/0.811 = 0.384$
I got different results as you can see! I think split two would be better as at least in one branch we're getting all females

1.2 Coding decision trees

In this section, we are going to use the decision tree model to predict the the animal `type` class of the `zoo` dataset. The dataset has been preprocessed and splited into `decision-tree-train.csv` and `decision-tree-test.csv` for you.

In [7]:
```
from hw2code.decision_tree import DecisionTree
mytree = DecisionTree()
mytree.load_data('./data/decision-tree-train.csv','./data/decision-tree-tes
# As a sanity check, we print out the size of the training data (80, 17) an
print('Training data shape: ', mytree.train_data.shape)
print('Testing data shape:', mytree.test_data.shape)
```

```
Training data shape:  (80, 17)
Testing data shape: (21, 17)
```

1.2.1 Infomation gain

Complete the `make_tree` and `compute_info_gain` function in `decision_tree.py` .

Train you model using `info_gain` measure to classify `type` and print the test accuracy.

In [8]:
```
mytree = DecisionTree()
mytree.load_data('./data/decision-tree-train.csv','./data/decision-tree-tes
test_acc= 0
#========================#
# STRART YOUR CODE HERE  #
#========================#
mytree.train('type', 'info_gain')
test_acc = mytree.test('type')
#========================#
#   END YOUR CODE HERE   #
#========================#
print('Test accuracy is: ', test_acc)
```

```
best_feature is:  legs
best_feature is:  fins
best_feature is:  toothed
best_feature is:  eggs
best_feature is:  hair
best_feature is:  hair
best_feature is:  toothed
best_feature is:  aquatic
Test accuracy is:  0.8571428571428571
```

1.2.2 Gain ratio

Complete the `compute_gain_ratio` function in `decision_tree.py` .

Train you model using `gain_ratio` measure to classify `type` and print the test accuracy.

In [9]:
```
mytree = DecisionTree()
mytree.load_data('./data/decision-tree-train.csv','./data/decision-tree-tes
test_acc = 0
#=========================#
# STRART YOUR CODE HERE  #
#=========================#
mytree.train('type', 'gain_ratio')
test_acc = mytree.test('type')
#=========================#
#   END YOUR CODE HERE   #
#=========================#
print('Test accuracy is: ', test_acc)
```

```
best_feature is:  feathers
best_feature is:  backbone
best_feature is:  airborne
best_feature is:  predator
best_feature is:  milk
best_feature is:  fins
best_feature is:  legs
Test accuracy is:  0.8095238095238095
```

Question

Which measure do you like the most and why?

Your answer here:

I would chose information gain as it has a better accuracy. Here the lower accuracy for Gain Ratio might be becasue it reduced the bias towards multi-valued attributes.

2. SVM

This workbook will walk you through a SVM.

2.1 Support vectors and decision boundary

Note: for this question you can work entirely in the Jupyter Notebook, no need to edit any .py files.

Consider classifying the following 20 data points in the 2-d plane with class label y

In [7]:
```
ds = pd.read_csv('data/svm-2d-data.csv')
ds
# This command above will print out the first five data points
# in the dataset with column names as "x1", "x2" and "y"
# You may use command "ds" to show the entire dataset, which contains 20 da
```

Out[7]:

	x1	x2	y
0	0.52	-1.00	1
1	0.91	0.32	1
2	-1.48	1.23	1
3	0.01	1.44	1
4	-0.46	-0.37	1
5	0.41	2.04	1
6	0.53	0.77	1
7	-1.21	-1.10	1
8	-0.39	0.96	1
9	-0.96	0.08	1
10	2.46	2.59	-1
11	3.05	2.87	-1
12	2.20	3.04	-1
13	1.89	2.64	-1
14	4.51	-0.52	-1
15	3.06	1.30	-1
16	3.16	-0.56	-1
17	2.05	1.54	-1
18	2.34	0.72	-1
19	2.94	0.13	-1

Suppose by solving the dual form of the quadratic programming of svm, we can derive the α_i's for each data point as follows: Among $j = 0, 1, \cdots, 19$ (note that the index starts from 0), α_1 = 0.5084, α_5 = 0.4625, α_{17} = 0.9709, and α_j = 0 for all other j.

Questions

1. Which vectors in the training points are support vectors?
2. What is the normal vector of the hyperplane w?
3. What is the bias b?
4. With the parameters w and b, we can now use our SVM to do predictions. What is predicted label of $x_{new} = (2, -0.5)$? Write out your $f(x_{new})$.

5. A plot of the data points has been generated for you. Please change the `support_vec` variable such that only the support vectors are indicated by red circles. Please also fill in the code to draw the decision boundary. Does your prediction of part 4 seems right visually on the plot?

Your answer here

Note: you can use several code cells to help you compute the results and answer the questions. Again you don't need to edit any .py files.

Please type your answer here!

answer 1: The support vectors are those with a non-zero alpha: data points 1, 5, and 17(indexed 0) : <0.91, 0.32>, <0.41, 2.04>, <2.05, 1.54>

answer 2: $w = \sum_i(\alpha_i * y_i * x_i)$ => $w =< -1.34, 0.39 >$

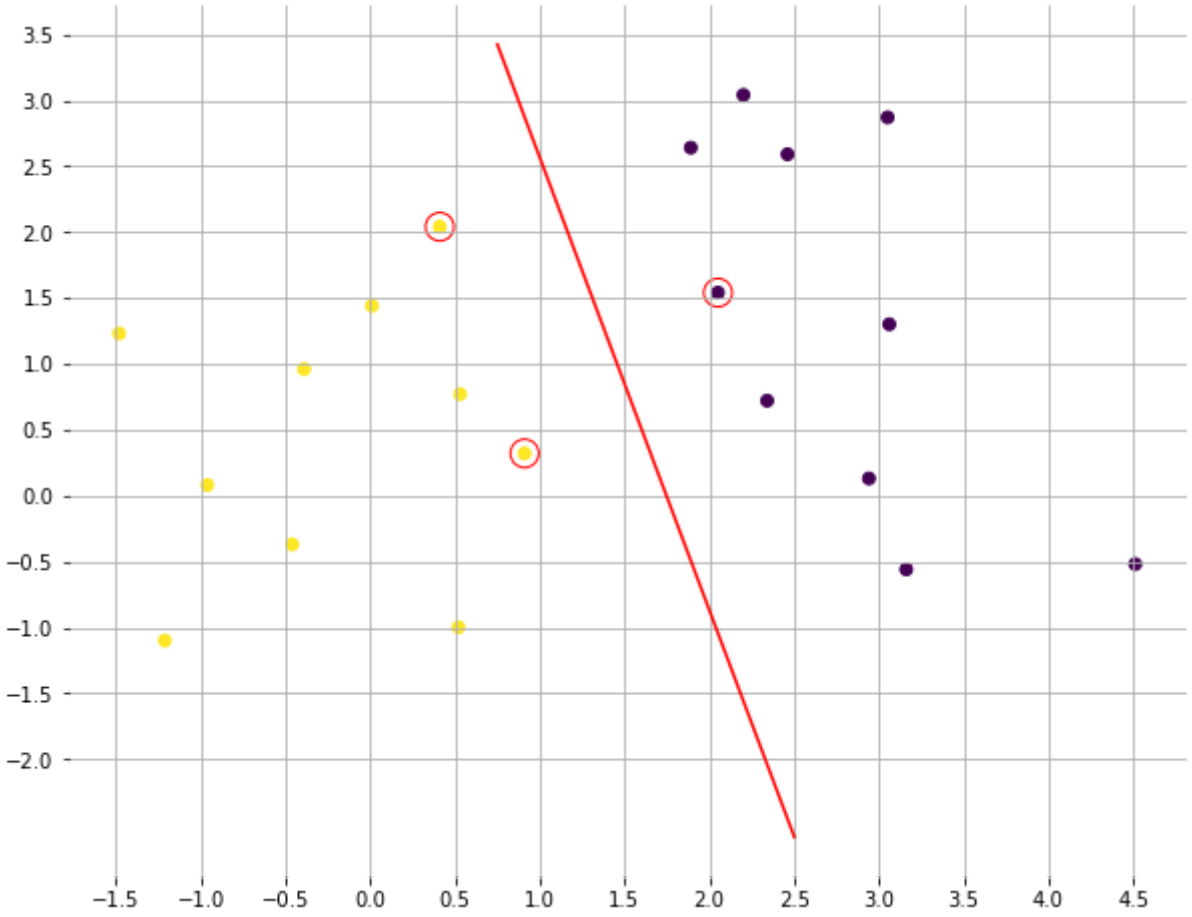
answer 3: $b = \frac{\sum y_k - w^T x_k}{N_k}$ => b = 2.34

answer 4: $f(x_{new}) = -1.34x_1 - 0.39x_2 + 2.34$ => $f(x_{new})$ = -0.14

In [9]:
```
# answer 5
x1_range = np.arange(-2, 5, 0.5)
x2_range = np.arange(-2, 4., 0.5)

fig, ax = plt.subplots(figsize=(10, 8))
ax = fig.gca()
ax.set_xticks(x1_range)
ax.set_yticks(x2_range)
ax.grid()
ax.scatter(ds['x1'], ds['x2'], c=ds['y'])

support_vec = ds
#=========================#
# STRART YOUR CODE HERE  #
#=========================#
support_vec =  ds.loc[(ds['x1'] == 0.91) | (ds['x1'] == 0.41) | (ds['x1'] =
w = -1.34/0.39
x = np.linspace(0.75, 2.5)
y = w * x - 2.34 / -0.39
plt.plot(x, y, 'r-')
#=========================#
#   END YOUR CODE HERE   #
#=========================#
ax.scatter(support_vec['x1'], support_vec['x2'], marker='o', facecolor='non
sns.despine(ax=ax, left=True, bottom=True, offset=0)
plt.show()
```



2.2 Coding SVM

In this section, we are going to use SVM for classifying the `y` value of 4-dimensional data points. The dataset has been preprocessed and splited into `svm-train.csv` and `svm-test.csv` for you.

For this question we are going to use the `cvxopt` package to help us solve the optimization problem of SVM. You will see it in the .py files, but you don't need to any coding with it. For this question, you only need to implement the right kernel function, and your kernel matrix `K` in `svm.py` line 135 will be pluged in the cvxopt optimization problem solver.

For more information about `cvxopt` please refer to http://cvxopt.org/ (http://cvxopt.org/)

In [2]:
```
from hw2code.svm import SVM
svm = SVM()
svm.load_data('./data/svm-train.csv', './data/svm-test.csv')
# As a sanity check, we print out the size of the training data (1098, 4) a
print('Training data shape: ', svm.train_x.shape, svm.train_y.shape)
print('Testing data shape:', svm.test_x.shape, svm.test_y.shape)
svm.train_x
```

```
Training data shape:  (1098, 4) (1098,)
Testing data shape: (274, 4) (274,)
```

Out[2]:
```
array([[  3.6216 ,   8.6661 ,  -2.8073 ,  -0.44699],
       [  4.5459 ,   8.1674 ,  -2.4586 ,  -1.4621 ],
       [  3.866  ,  -2.6383 ,   1.9242 ,   0.10645],
       ...,
       [ -4.4775 , -13.0303 ,  17.0834 ,  -3.0345 ],
       [ -4.1958 ,  -8.1819 ,  12.1291 ,  -1.6017 ],
       [ -3.38   ,  -0.7077 ,   2.5325 ,   0.71808]])
```

2.2.1 Linear kernel

Complete the `SVM.predict` and `linear_kernel` function in `svm.py` . Train a hard margin SVM and a soft margin SVM with linear kernel. Print the test accuracy for both cases.

In [3]:
```
svm_hard = SVM()
svm_hard.load_data('./data/svm-train.csv', './data/svm-test.csv')
hard_test_acc = 0
#=========================#
# STRART YOUR CODE HERE  #
#=========================#
svm_hard.train('linear_kernel')
hard_pred = svm_hard.predict(svm_hard.train_x)
hard_test_acc = svm_hard.test()
#=========================#
#   END YOUR CODE HERE   #
#=========================#

svm_soft = SVM()
svm_soft.load_data('./data/svm-train.csv', './data/svm-test.csv')
soft_test_acc = 0
#=========================#
# STRART YOUR CODE HERE  #
#=========================#
svm_soft.train('linear_kernel', 100)
soft_pred = svm_soft.predict(svm_soft.train_x)
soft_test_acc = svm_soft.test()
#=========================#
#   END YOUR CODE HERE   #
#=========================#
print('Hard margin test accuracy is: ', hard_test_acc)
print('Soft margin test accuracy is: ', soft_test_acc)
```

```
1098 support vectors out of 1098 points
30 support vectors out of 1098 points
Hard margin test accuracy is:  0.5547445255474452
Soft margin test accuracy is:  0.9890510948905109
```

Questions

Are these two results similar? Why or why not?

Your Answer

No, they are not similar! Hard margin has lower accuracy as there are no missclassification. Soft margin has a better accuracy as we allow missclassification

2.2.2 Polynomial kernel

Complete the `polynomial_kernel` function in `svm.py` . Train a soft margin SVM with degree 3 polynomial kernel and parameter `C = 100` for the regularization term. Print the test accuracy.

In [4]:
```
svm = SVM()
svm.load_data('./data/svm-train.csv', './data/svm-test.csv')
test_acc = 0
#========================#
# STRART YOUR CODE HERE  #
#========================#
svm.train('polynomial_kernel', 100)
pred = svm.predict(svm.train_x)
test_acc = svm.test()
#========================#
#   END YOUR CODE HERE   #
#========================#
print('Test accuracy is: ', test_acc)
```

```
19 support vectors out of 1098 points
Test accuracy is:  0.927007299270073
```

Questions

Is the result better than linear kernel? Why or why not?

Your Answer

No, Polynomial Kernel has a lower accuracy and I think that's because of our dataset as it's not that noisy and classifies better with linear_kernel

2.2.3 Gaussian kernel

Complete the `gaussian_kernel` function using the `gaussian_kernel_point` in `svm.py`. Train a soft margin SVM with Gaussian kernel and parameter `C = 100` for the regularization term. Print the test accuracy.

In [5]:
```
svm = SVM()
svm.load_data('./data/svm-train.csv', './data/svm-test.csv')
test_acc = 0
#========================#
# STRART YOUR CODE HERE  #
#========================#
svm.train('gaussian_kernel', 100)
pred = svm.predict(svm.train_x)
test_acc = svm.test()
#========================#
#   END YOUR CODE HERE   #
#========================#
print('Test accuracy is: ', test_acc)
```

```
35 support vectors out of 1098 points
Test accuracy is:  1.0
```

Questions

1. Is the result better than linear kernel and polynomial kernel? Why or why not?
2. Which one of these four models do you like the most and why?
3. (Bonus question, optional) Can you come up with a vectorized implementation of `gaussian_kernel` without calling `gaussian_kernel_point` ? Fill that in svm.py.

Your Answer

Please write down your answers and/or observations here

answer 1: Yes, it's better as the accuracy is 1. and that is because it becomes easier to overfit the data; however, the computation is slow

answer 2: For this dataset, I'd choose Linear Kernel as it has a really good accuracy and it's faster than gaussian_kernel

End of Homework 2 :)

After you've finished the homework, please print out the entire `ipynb` notebook and two `py` files into one PDF file. Make sure you include the output of code cells and answers for questions. Prepare submit it to GradeScope. Also this time remember assign the pages to the questions on GradeScope