MRC_Assignment10

January 16, 2018

1 Hochschule Bonn-Rhein-Sieg

2 Mathematics for Robotics and Control, WS17

3 Assignment 10

3.0.1 Team Members : Vajra Ganeshkumar, Jeeveswaran Kishaan

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        from scipy.misc import imread
```

3.1 Exercise 1: Naive Bayes Classifier [40 points]

The Naive Bayes classifier is a quite simple and popular classifier that is entirely based on a conditional independence assumption. Let's suppose that we have a set of n features $F_1, F_2, ..., F_n$ and a class label C. The Naive Bayes classifier assumes that the features are independent of each other given a known class label, which is another way of saying that

$$P(C, F_1, F_2, ..., F_n) = P(F_1, F_2, ..., F_n|C)P(C) = P(C)\prod_i P(F_i|C)$$

Now, the classification problem is to determine the class label given the values of the features $F_1, F_2, ..., F_n$. If we have m classes, we can find the class label by calculating $P(C_j|F_1, F_2, ..., F_n)$ for $1 \leq j \leq m$

$$P(C_j|F_1, F_2, ..., F_n) = \frac{P(C_j, F_1, F_2, ..., F_n)}{P(F_1, F_2, ..., F_n)} = \frac{P(C_j)\prod_i P(F_i|C_j)}{P(F_1, F_2, ..., F_n)} = \alpha P(C_j)\prod_i P(F_i|C_j)$$

and then choosing

$$k = \max_j P(C_j|F_1, F_2, ..., F_n)$$

as the class label. In the above equation, we notice that $P(F_1, F_2, ..., F_n)$ doesn't have to be calculated explicitly, i.e. instead of calculating $P(F_1, F_2, ..., F_n)$, we can calculate $P(C_j)\prod_i P(F_i|C_j)$ for $1 \leq j \leq m$ and then normalise the values using $\alpha = \sum_j P(C_j)\prod_i P(F_i|C_j)$.

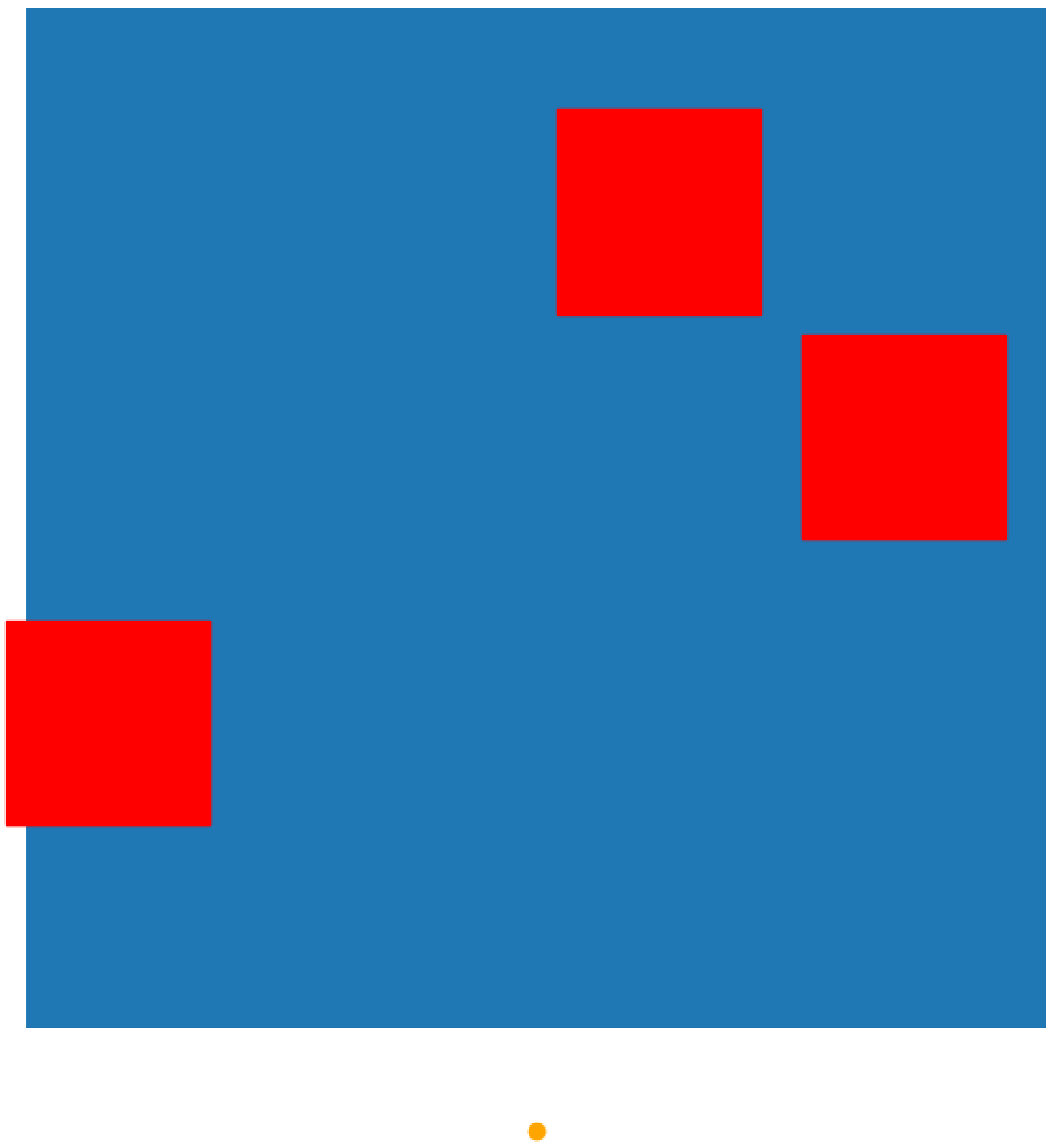
In this exercise, you will implement your very own Naive Bayes classifier that can be used for predicting the stability of object placements on a table. The scenario is one in which our robot

Jenny is putting objects on a table, such that we'll suppose that the robot chooses a random continuous table pose for placement and then tries to predict whether placing a *point object* there will be successful by describing the pose with a few features.

The scenario we are considering is visualised from a top view in the image below (note: the blue square is the table, the red squares are the objects on it, and the orange dot is Jenny).

```
In [2]: import IPython
        IPython.core.display.Image("images/configuration.png", embed=True)
```

```
Out[2]:
```



Let's suppose that a pose is described using the following features, all of which are discrete:

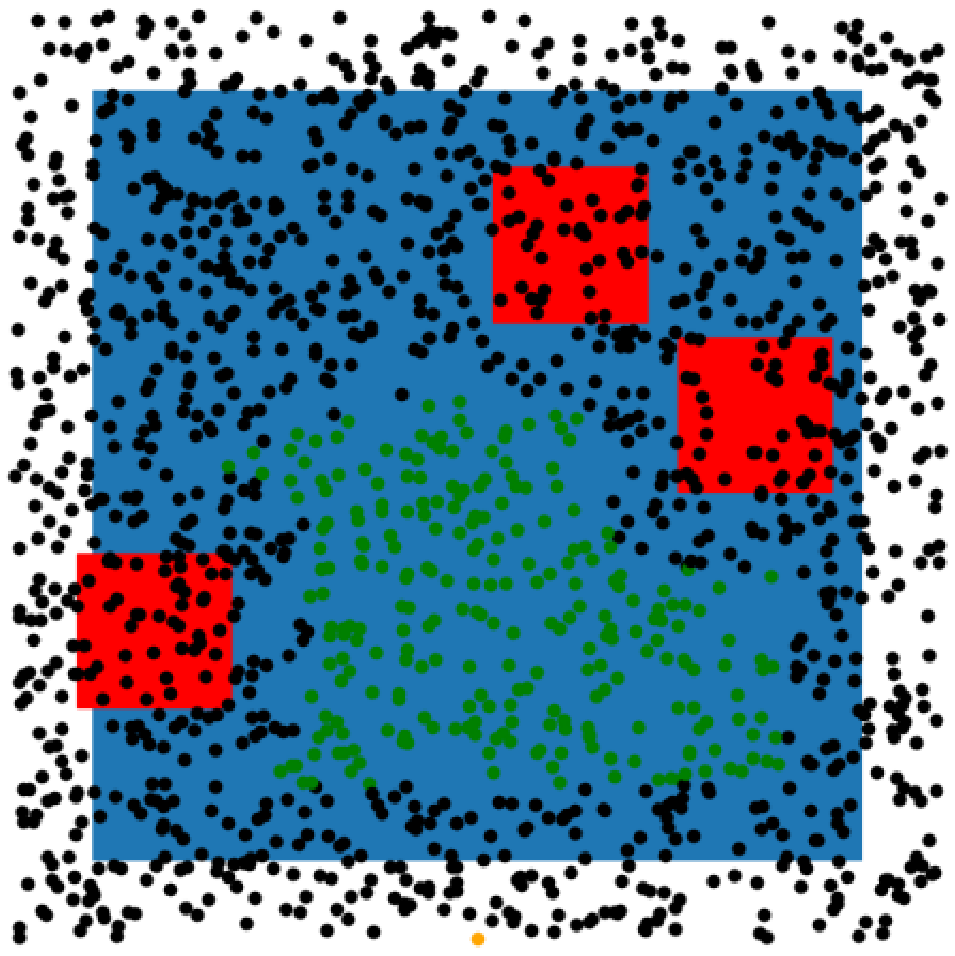
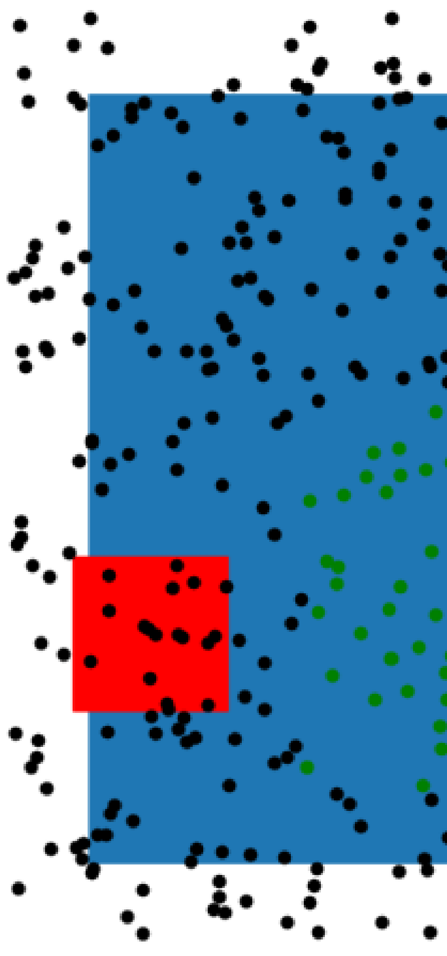
- *Inside table*: Takes the values 0 and 1, corresponding to whether a pose is outside or inside the table respectively.
- *Distance to the robot*: Takes the values 0, 1, and 2, corresponding to *very close*, *reachable*, and *far*.
- *Minimum distance to the other objects*: Takes the values 0, 1, and 2, corresponding to *very close*, *close*, and *far*.
- *Distance to the closest surface edge*: Also takes the values 0, 1, and 2, corresponding to *very close*, *close*, and *far*.

Each pose either leads to a successful execution or not, so we have two classes, namely 0 and 1, corresponding to the outcomes *failure* and *success* respectively.

Your task now consists of the following steps:

- You are given a data set (*data/train.txt*) of features describing 1500 poses and the class labels of these. Use the data in this data set for learning the prior probabilities $P(C_j)$ and the conditional probabilities $P(F_i|C_j)$, $i \in \{1,2,3,4\}$, $j \in \{1,2\}$. Note that *learning* in this context means calculating the values of the probabilities as relative frequencies.
- Use the test data set (*data/test.txt*) for testing your classifier (i.e. predict the class labels of the 500 test points using the given features and compare the predicted labels with the actual labels).

The labelled training and test data are visualised below (the green and black points correspond to stable and unstable placements respectively):

Training data	Test data
	

Implement your classifier below, plot your predicted labels (use the coordinates of the test points in *data/test_points.txt* for that purpose), and report the confusion matrix of your results on the test set.

```
In [3]: ### write your code here ###
        data = np.loadtxt("data/train.txt")
        # training features
        train_x = data[:,:-1]
        # train labels
        train_y = data[:,-1]

        number_samples = np.shape(train_x)[0]

        class_0 = np.sum(train_y == 0) # 1245
        class_1 = np.sum(train_y == 1) # 255

        # Probability of class 0
        P_c0 = class_0 / float(len(data))
```

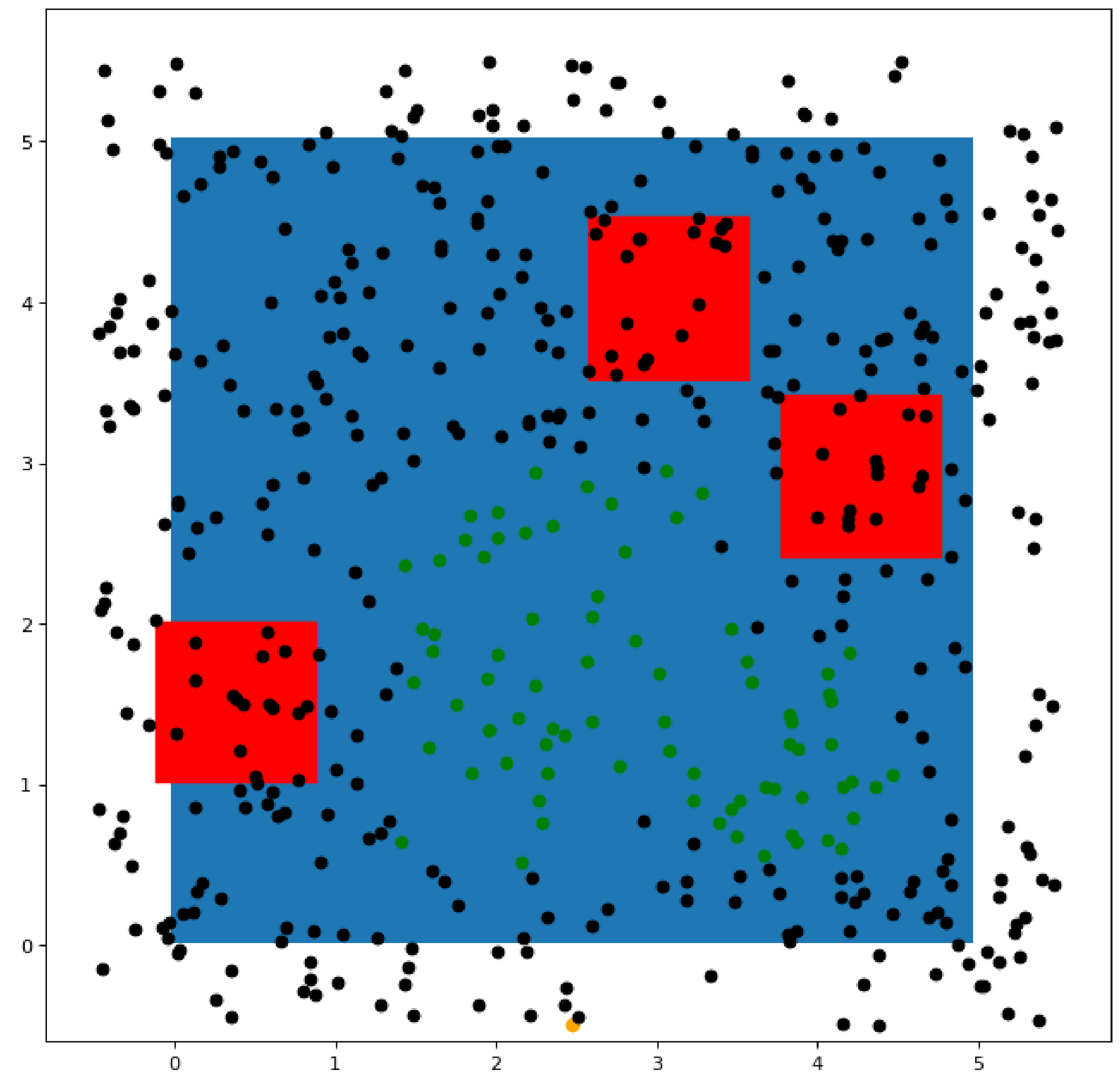
```
# Probability of class 1
P_c1 = class_1 / float(len(data))


# conditional probabilities of each and every feature given class 0
cond_pro_0 = {}
# conditional probabilities of each and every feature given class 1
cond_pro_1 = {}

#=========== Calculations given class 0 =================

# features for class 0
data_c0 = train_x[np.where(train_y == 0)]

# feature 1
P_10_c0 = np.sum(data_c0[:,0] == 0.) / float(class_0)
P_11_c0 = np.sum(data_c0[:,0] == 1.) / float(class_0)
cond_pro_0[1] = [P_10_c0 , P_11_c0]

# feature 2
P_20_c0 = np.sum(data_c0[:,1] == 0.) / float(class_0)
P_21_c0 = np.sum(data_c0[:,1] == 1.) / float(class_0)
P_22_c0 = np.sum(data_c0[:,1] == 2.) / float(class_0)
cond_pro_0[2] = [P_20_c0 , P_21_c0 , P_22_c0]

# feature 3
P_30_c0 = np.sum(data_c0[:,2] == 0.) / float(class_0)
P_31_c0 = np.sum(data_c0[:,2] == 1.) / float(class_0)
P_32_c0 = np.sum(data_c0[:,2] == 2.) / float(class_0)
cond_pro_0[3] = [P_30_c0 , P_31_c0 , P_32_c0]

# feature 4
P_40_c0 = np.sum(data_c0[:,3] == 0.) / float(class_0)
P_41_c0 = np.sum(data_c0[:,3] == 1.) / float(class_0)
P_42_c0 = np.sum(data_c0[:,3] == 2.) / float(class_0)
cond_pro_0[4] = [P_40_c0 , P_41_c0 , P_42_c0]

# ========== Calculations given class 1 ================

# features for class 1
data_c1 = train_x[np.where(train_y == 1)]

# feature 1
P_10_c1 = np.sum(data_c1[:,0] == 0.) / float(class_1)
P_11_c1 = np.sum(data_c1[:,0] == 1.) / float(class_1)
cond_pro_1[1] = [P_10_c1 , P_11_c1]

# feature 2
```

```
P_20_c1 = np.sum(data_c1[:,1] == 0.) / float(class_1)
P_21_c1 = np.sum(data_c1[:,1] == 1.) / float(class_1)
P_22_c1 = np.sum(data_c1[:,1] == 2.) / float(class_1)
cond_pro_1[2] = [P_20_c1 , P_21_c1, P_22_c1]

# feature 3
P_30_c1 = np.sum(data_c1[:,2] == 0.) / float(class_1)
P_31_c1 = np.sum(data_c1[:,2] == 1.) / float(class_1)
P_32_c1 = np.sum(data_c1[:,2] == 2.) / float(class_1)
cond_pro_1[3] = [P_30_c1 , P_31_c1, P_32_c1]

# feature 4
P_40_c1 = np.sum(data_c1[:,3] == 0.) / float(class_1)
P_41_c1 = np.sum(data_c1[:,3] == 1.) / float(class_1)
P_42_c1 = np.sum(data_c1[:,3] == 2.) / float(class_1)
cond_pro_1[4] = [P_40_c1 , P_41_c1, P_42_c1]


# ====== calculating the alpha value =====
cond_0 = 1
cond_1 = 1

# probability of features given class 0
for k,v in cond_pro_0.iteritems():
    for i in v:
        cond_0 *= i
cond_0 *= cond_0 * P_c0

# probability of features given class 1
for k,v in cond_pro_1.iteritems():
    for i in v:
        cond_1 *= i
cond_1 *= cond_1 * P_c1

alpha = cond_0 + cond_1

# ================ testing phase ===================
test_data = np.loadtxt("data/test.txt")
data_xy = np.loadtxt("data/test_points.txt")

# array to store the predictions
predictions = []
# array to store the stable points
stable_points_x = []
stable_points_y = []
# array to store the unstable points
unstable_points_x = []
unstable_points_y = []
```

```
for i,row in enumerate(test_data):
    sample = row[:-1]
    pro_c0 = alpha * P_c0 * cond_pro_0[1][int(sample[0])] * cond_pro_0[2][int(sample[1])
                  * cond_pro_0[3][int(sample[2])] * cond_pro_0[4][int(sample[3])]

    pro_c1 = alpha * P_c1 * cond_pro_1[1][int(sample[0])] * cond_pro_1[2][int(sample[1])
                  * cond_pro_1[3][int(sample[2])] * cond_pro_1[4][int(sample[3])]

    if pro_c0 > pro_c1:
        predictions.append(0)
        unstable_points_x.append(data_xy[i][0])
        unstable_points_y.append(data_xy[i][1])
    else:
        predictions.append(1)
        stable_points_x.append(data_xy[i][0])
        stable_points_y.append(data_xy[i][1])

# number of correct predictions
correct_predictions =  np.sum(predictions == test_data[:,-1])

prediction_acc = correct_predictions / float(len(test_data))
print "prediction accuracy: ", prediction_acc
print "correct predictions: ", correct_predictions
print "wrong predictions: ", len(test_data) - correct_predictions


# Count_prediction_actual
count_00 = 0
count_01 = 0
count_10 = 0
count_11 = 0

for p,a in zip(predictions, test_data[:,-1]):
    if p == 0 and a == 0:
        count_00 += 1
    elif p == 0 and a == 1:
        count_01 += 1
    elif p == 1 and a == 0:
        count_10 += 1
    else:
        count_11 += 1

print "\nPredicted 0 and Actual value 0: ",count_00
print "Predicted 0 and Actual value 1: ",count_01
print "Predicted 1 and Actual value 0: ",count_10
print "Predicted 1 and Actual value 1: ",count_11
```

```
img = imread('images/configuration.png')
plt.figure(num=None, figsize=(10, 10), dpi=80, facecolor='w', edgecolor='k')
plt.imshow(img, extent = [-0.2, 5.15,-0.6,5.1])
plt.scatter(stable_points_x, stable_points_y, c='g')
plt.scatter(unstable_points_x, unstable_points_y, c='k')
plt.show()
```

```
prediction accuracy:  0.996
correct predictions:  498
wrong predictions:  2

Predicted 0 and Actual value 0:  419
Predicted 0 and Actual value 1:  2
Predicted 1 and Actual value 0:  0
Predicted 1 and Actual value 1:  79
```



3.1.1 Confusion Matrix

		Actual	Values
		0	1
Predicted	0	419	2
Values	1	0	79

3.1.2 Reference:

https://en.wikipedia.org/wiki/Confusion_matrix

3.2 Exercise 2: Total Probability and Bayes' Rule [40 points]

Let's suppose that there is a box of tools in the RoboCup@Work lab which has 20 wrenches, 50 screwdrivers, and 30 pairs of pliers, such that we let one of our youBots pick up a single tool from the box at random. From historical data, we know that there is a 0.2 probability that the robot will drop a wrench, a 0.1 probability that it will drop a screwdriver, and a 0.3 probability that it will drop a pair of pliers; in other words,

$P(d|w) = 0.2 \qquad P(d|s) = 0.1 \qquad P(d|p) = 0.3$

where d is the event in which the youBot drops a tool, while w, s, and p are the events in which the robot picks up a wrench, a screwdriver, and a pair of pliers respectively. If the youBot drops a tool, a human operator puts the tool back into the gripper and the robot continues its operation.

1. What is the probability that the robot drops a tool?
2. If we know that the robot has dropped a tool, what is the probability that it had picked up a pair of pliers?
3. Show that the probability that the youBot had picked up a screwdriver given two observed drops is equal to

$$P(s|d_1, d_2) = \frac{P(d_2|s)P(s|d_1)}{P(d_2|s)P(s|d_1) + P(d_2|w)P(w|d_1) + P(d_2|p)P(p|d_1)}$$

   which means that, if we want to calculate the probability that the robot had picked up a screwdriver given two drops, we first need to calculate the posterior probabilities that the robot had picked up any of the tools after dropping the tool for the first time (which can in fact be seen as updating the prior probabilities in a recursive manner).
4. Generalise the above result to the case in which we observe n drops.

3.2.1 write your answers here

3.2.2 Answers

1. What is the probability that the robot drops a tool? We use total probability theorem to compute this,so

$Probability\ of\ dropping\ a\ tool\ = (Probability\ of\ dropping\ wrenchs\ \times\ Probability\ of\ picking\ wrenchs)$

$$+ (probability\ of\ dropping\ screwdrivers\ \times\ Probability\ of\ picking\ screwdrivers)$$
$$+ (probability\ of\ dropping\ pairs\ of\ pliers \times\ Probability\ of\ picking\ pairs\ of\ pliers)$$

$$Probability\ of\ dropping\ a\ tool\ = P(d \mid w) \times P(w) + P(d \mid s) \times P(s) + P(d \mid p) \times P(p)$$
$$Probability\ of\ dropping\ a\ tool\ = 0.2 \times 0.2 + 0.1 \times 0.5 + 0.3 \times 0.3$$
$$Probability\ of\ dropping\ a\ tool\ ,\ P(d) = 0.18$$

2. If we know that the robot has dropped a tool, what is the probability that it had picked up a pair of pliers?

$$P(p \mid d) = \frac{P(d \mid p)\ \times\ P(p)}{P(d)}$$

We know that from the above equation, $P(d) = 0.18$ $P(p) = 0.3$ $P(d \mid p) = 0.3$ Substituting the above results we have,

$$P(p \mid d) = \frac{0.3\ \times\ 0.3}{0.18}$$

probability that it had picked up a pair of pliers given it has dropped something,

$$P(p \mid d) = 0.5$$

3. Show that the probability that the youBot had picked up a screwdriver given two observed drops is equal to

$$P(s|d_1, d_2) = \frac{P(d_2|s)P(s|d_1)}{P(d_2|s)P(s|d_1) + P(d_2|w)P(w|d_1) + P(d_2|p)P(p|d_1)}$$

Probability of robot dropping the tool for the first time is given by,

$$P(d_1) = P(d_1 \mid w) \times P(w) + P(d_1 \mid s) \times P(s) + P(d_1 \mid p) \times P(p)$$

Now, given the Probability of the tool being dropped by the Youbot for the first time, the prior probabilities $P(s)$, $P(w)$, and $P(p)$ will become the posterior probabilities as shown below

$$P(s) => P(s \mid d_1) = \frac{P(d_1 \mid s) \times P(s)}{P(d_1)} \quad ---------------(1)$$

$$P(w) => P(w \mid d_1) = \frac{P(d_1 \mid w) \times P(w)}{P(d_1)} \quad ---------------(2)$$

$$P(s) => P(p \mid d_1) = \frac{P(d_1 \mid p) \times P(p)}{P(d_1)} \quad ---------------(3)$$

Now, Probability of robot dropping the tool for the second time is given by,

$$P(d_2) = P(d_2 \mid s) \times P(s \mid d_1) + P(d_2 \mid w) \times P(w \mid d_1) + P(d_2 \mid p) \times P(p \mid d_1) \quad -----------(4)$$

So, since we know the probability of the second drop, we can compute the probability that the Youbot had picked up a screwdriver,

$$P(s \mid d_2) = \frac{P(d_2 \mid s) \times P(s \mid d_1)}{P(d_2)} \quad ---------------(5)$$

Substituting equation (4) in equation (5) we get,

$$P(s|d_1,d_2) = \frac{P(d_2|s)P(s|d_1)}{P(d_2|s)P(s|d_1) + P(d_2|w)P(w|d_1) + P(d_2|p)P(p|d_1)}$$

4. Generalise the above result to the case in which we observe ***n*** **drops** We know that,probability of picking up a tool(screw driver) given the two drops i.e, first and second is given by,

$$P(s|d_1,d_2) = \frac{P(d_2|s)P(s|d_1)}{P(d_2|s)P(s|d_1) + P(d_2|w)P(w|d_1) + P(d_2|p)P(p|d_1)}$$

And also,the probability of picking up a tool(screw driver) given the two drops (second and third) is given by,

$$P(s|d_2,d_3) = \frac{P(d_3|s)P(s|d_2)}{P(d_3|s)P(s|d_2) + P(d_3|w)P(w|d_2) + P(d_3|p)P(p|d_2)}$$

To generalise the result,

$$P(s|d_{i-1},d_i) = \frac{P(d_i|s)P(s|d_{i-1})}{P(d_i|s)P(s|d_{i-1}) + P(d_i|w)P(w|d_{i-1}) + P(d_i|p)P(p|d_{i-1})}$$

3.2.3 Reference

Lecture slides: **Bayesian Filtering** Slide number - 51 to 56