INF 552 Homework 3

Task 1:

i)

$P(A_i = v_j / c_k) = (n_{ijk} + 1)/(n_k + s_i)$

where n_k is the number of samples in class c_k, n_{ijk} is the number of samples in class c_k where Attribute $A_i = v_j$, and s_i is the number of possible values for A_i

$P(c_k = +) = 4/8 = 1/2$

$P(c_k = -) = 4/8 = 1/2$

$P(A_1 = 1/c_k = +) = (3+1)/(4+2) = 4/6$
$P(A_1 = 0/c_k = +) = (1+1)/(4+2) = 2/6$
$P(A_1 = 1/c_k = -) = (2+1)/(4+2) = 3/6$
$P(A_1 = 0/c_k = -) = (2+1)/(4+2) = 3/6$

$P(A_2 = 1/c_k = +) = (3+1)/(4+2) = 4/6$
$P(A_2 = 0/c_k = +) = (1+1)/(4+2) = 2/6$
$P(A_2 = 1/c_k = -) = (1+1)/(4+2) = 2/6$
$P(A_2 = 0/c_k = -) = (3+1)/(4+2) = 4/6$

$P(A_3 = 1/c_k = +) = (2+1)/(4+2) = 3/6$
$P(A_3 = 0/c_k = +) = (2+1)/(4+2) = 3/6$
$P(A_3 = 1/c_k = -) = (2+1)/(4+2) = 3/6$
$P(A_3 = 0/c_k = -) = (2+1)/(4+2) = 3/6$

$P(A_4 = 1/c_k = +) = (3+1)/(4+2) = 4/6$
$P(A_4 = 0/c_k = +) = (1+1)/(4+2) = 2/6$
$P(A_4 = 1/c_k = -) = (0+1)/(4+2) = 1/6$
$P(A_4 = 0/c_k = -) = (4+1)/(4+2) = 5/6$

ii)

$P(c_k = +/A_1=1, A_2=1, A_3=0, A_4=0)$ = $P(A_1=1, A_2=1, A_3=0, A_4=0/ c_k = +) * P(c_k = +)$

$= P(A_1=1/c_k = +) * P(A_2 = 1/c_k = +) * P(A_3 = 0/c_k = +) * P(A_4=0/c_k = +) * P(c_k = +)$

$= (4/6)*(4/6)*(3/6)*(2/6)*(1/2)$

$=1/27$

$P(c_k = -/A_1=1, A_2=1, A_3=0, A_4=0)$ $= P(A_1=1, A_2=1, A_3=0, A_4=0/ c_k = -) * P(c_k = -)$

$= P(A_1=1/c_k = -) * P(A_2 = 1/c_k = -) * P(A_3 = 0/c_k = -) * P(A_4=0/c_k = -) * P(c_k = -)$

$= (3/6)*(2/6)*(3/6)*(5/6) *(1/2)$

$= 5/144$

$P(c_k = +/A_1=1, A_2=1, A_3=0, A_4=0) > P(c_k = -/A_1=1, A_2=1, A_3=0, A_4=0)$

Naïve Bayes Classifier would classify the example as belonging to class positive (+)

Task 2:

i) 10-fold cross validation:

Divided the training data into 10 disjoint subsets of equal size containing equal number of positive and negative examples. There were 12500 examples in the training data containing 6250 positive examples and 6250 negative examples. Hence, I used the split command and divided the training data into 10 disjoint subsets, each containing 625 positive examples and 625 negative examples.

Command used: split -l <line_number> <training_file> <output_filename>

Example: split -l 625 DogsVsCats.train trainingSet

Validation Accuracy for Linear Kernel: 60.136%

Validation Accuracy for Polynomial Kernel: 61.912%

ii)

Training Accuracy for Linear Kernel: 60.46% (7558 correct, 4942 incorrect, 12500 total)

Training Accuracy for Polynomial Kernel: 62.90% (7862 correct, 4638 incorrect, 12500 total)

Testing Accuracy for Linear Kernel: 59.82% (7477 correct, 5023 incorrect, 12500 total)

Testing Accuracy for Polynomial Kernel: 61.41% (7676 correct, 4824 incorrect, 12500 total)

Command used to train using Linear Kernel:

./svm_learn -t 0 <training_file> <model_file>

Command used to train using Polynomial Kernel:

./svm_learn -t 1 -d 5 <training_file> <model_file>

Command used to test:

./svm_classify -f 0 <test_file> <model_file> <predictions_output_filename>

Overall Result:

Accuracy/Kernel	Linear Kernel	Polynomial Kernel
Validation Accuracy	60.136%	61.912%
Training Accuracy	60.46%	62.90%
Testing Accuracy	59.82%	61.41%

It can be interpreted from the table above that Polynomial Kernel predicts much better than Linear Kernel since the training accuracy, testing accuracy and validation accuracy of Polynomial Kernel are greater than the corresponding accuracies of Linear Kernel. It can also be seen that the training accuracy for both linear and polynomial kernel is better than the validation and testing accuracy.

Task 3: Adaboost.py

It was found that the number of misclassified labels in each iteration was fluctuating as the attribute chosen during each iteration was different. For example, the number of misclassified labels for the first 3 iterations were 10, 80, 274 respectively and it again dropped to 10 in the next iteration. As a result of this, epsilon and alpha values were also fluctuating. So, a proper (increasing or decreasing) pattern for alpha was not observed. But, it was observed that as the number of training iterations increased, the classifier got trained better and it was able to predict the labels better on the test data. This can be seen from the increase in test accuracy from 99.37% to 99.78% as the number of training iterations was increased from 10 to 20.

Compiling Instructions:

Since the homework description says the second and third command line arguments are just the files and not the path to the training and test data files, both mushroom.train and mushroom.test must be present in the same directory as the Adaboost.py file.

I have also included the output I got after running Adaboost.py for T=10 and T=20. The first line corresponds to the test accuracy and the remaining values refer the alpha values for each classifier in each iteration.

Output for T=10:

99.3719279082
2.18472392623
1.46559687621
0.718801969242
0.81826789037
0.561398352289
0.735695157392
0.452104760663
0.735146988427
0.405999990754
0.521730051702

Output for T=20:

99.781540142

2.18472392623
1.46559687621
0.718801969242
0.81826789037
0.561398352289
0.735695157392
0.452104760663
0.735146988427
0.405999990754
0.521730051702
0.45762626801
0.585397964197
0.63047717014
0.574027055166
0.386101593931
0.426232057216
0.755174613607
0.527608593007
0.380773667968
0.470228747789