Q2(3)

The training efficiency of Naïve Bayes Classifier is :

92.836% (i.e. 985/1061 correctly classified)

The Test Set efficiency is :

88.9674% (i.e. 629/707 correctly classified) .

Q2(4)

The Naïve Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem with strong (NAÏVE) INDEPENDENCE ASSUMPTION.

The Naive Bayes model assumption of independence is not a very good assumption always. As has often been observed, the independence assumption on which Naïve Bayes classifiers are based almost never hold for natural data sets, and certainly not for textual data.

How so ever it does not really mean that Naïve Bayes classifier is useless. **Actually, when the attributes are nearly conditionally independent or there isn't enough data to learn the correlations, then naive Bayes is quite competitive**.

NB's main strength is its efficiency: Training and classification can be accomplished with one pass over the data. Because it combines efficiency with good accuracy it is often used as baseline in text classification research. It is often the method of choice if (i) squeezing out a few extra percentage points of accuracy is not worth the trouble in a text classification application, (ii) a very large amount of training data is available and there is more to be gained from training on a lot of data than using a better classifier on a smaller training set, or (iii) if its robustness to concept drift can be exploited.

The advantage of NB is that by making the assumption of independence of any word feature with respect to other word feature given the document class can typically reduce the model with much lesser parameters left.

This is also true that NB classifier is not supposed to always under perform the Decision Tree, as we can see in the given example that NB completely outperform decision tree which is quite surprising. Actually, Naïve Bayes is potentially good at serving as a document classification model due to its simplicity.

Q2(5) The extensions to NB classifier are:

1) **Tree Augmented Naïve Bayes**:

For this approach to understood we limit our attention to a class of network structures that are based on the structure of naive Bayes, requiring that the class variable be a parent of every attribute. This ensures that, in the learned network, the probability Pr(**Cj|A1;……..; An**) , the main term determining the classification, will take every attribute into account. Unlike the naïve Bayesian classifier, however, this classifier allows additional edges between attributes that capture correlations among them.

This extension incurs additional computational costs. While the induction of the naive Bayesian classifier requires only simple bookkeeping, the induction of Bayesian networks requires searching the space of all possible networks—that is, the space of all possible combinations of edges. To address this problem, we examine a restricted form of correlation edges. The resulting method, which is known as Tree Augmented Naive Bayes (TAN), approximates the interactions between attributes by using a tree structure imposed on the naive Bayesian structure. This approximation is optimal, in a precise sense; moreover, we can learn TAN classifiers in polynomial time.

Experiments show that TAN maintains the robustness and computational complexity of naive Bayes, and at the same time displays better accuracy.

HOW?

This has been proved that the performance of a Bayesian network as a classifier may improve_if the learning procedure takes into account the special status of the class variable. An easy way to ensure this is to bias the structure of the network, as in the naive Bayesian classifier,_such that there is an edge from the class variable to each attribute. This ensures that, in the_learned network, the probability
P(Cj|A1; : : : ; An) will take all attributes into account. In_order to improve the performance of a classifier based on this bias, it has been proposed to augment the naive Bayes structure with edges among the attributes, when needed, thus dispensing with its strong assumptions about independence. This type of structure is known as structures augmented naïve Bayesian networks and these edges augmenting edges._In an augmented structure, an edge from **Ai** to **Aj** implies that the influence of **Ai** on the_assessment of the class variable also depends on the value of **Aj.**

Adding the best set of augmenting edges is an intractable problem, since it is equivalent to learning the best Bayesian network among those in which C is a root. Thus, even if we could improve the performance of a naive Bayes classifier in this way, the computational effort required may not be worthwhile. However, by imposing acceptable restrictions on the form of the allowed interactions, we can actually learn the optimal set of augmenting edges in polynomial time.

2) Bayesian Multinets as classifiers:

The TAN approach forces the relations among attributes to be the same for all the different instances of the class variable **C** . An immediate generalization would have different augmenting edges (tree structures in the case of TAN) for each class, and a collection of networks as the classifier.

HOW ?

To implement this idea, we partition the training data set by classes. Then, for each class **c**i in Val**(C)** , we construct a Bayesian network **B**i for the attribute variables **{A1; : : : ; A**n**}** .The resulting probability distribution $\mathbf{P}_{Bi}$ **(A1……. A**n**)** approximates the joint distribution of the attributes, given a specific class, that is, $\mathbf{P}_D$**(A1…….. A**n **|C = c**i**)** . The Bayesian network for $\mathbf{c}_i$ is called a local

When learning a multinet, we set $\mathbf{P}_C$**(C)** to be the frequency of the class variable in the training data, that is, $\mathbf{P}_D$**(C)** , and learn the networks **B**i in the manner just described. Once again, we classify by choosing the class that maximizes the posterior probability $\mathbf{P}_M$**(C|A1…….. A**n**)** . By partitioning the data according to the class variable, this methodology ensures that the interactions between the class variable and the attributes are taken into account. The multinet proposal is strictly a generalization of the augmented naive Bayes, in the sense that that a multinet can easily simulate an augmented naive Bayesian network where all the local networks have the same structure. Note that the computational complexity of finding unrestricted augmenting edges for the attributes is aggravated by the need to learn a different network for each value of the class variable. Thus, the search for learning the Bayesian network structure must be carried out several times, each time on a different data set.

As in the case of augmented naive Bayes, we can address this problem by constraining the class of local networks we might learn to be treelike. Indeed, the construction of a set of trees that minimizes the log likelihood score was the original method used by Chow and Liu (1968) to build classifiers for recognizing handwritten characters. They reported that, in their experiments, the error rate of this method was less than half that of naive Bayes.

As with TAN models, we apply smoothing to avoid unreliable estimation of parameters. Note also that we partition the data further, and therefore run a higher risk of missing the accurate weight of some edge (in contrast to TAN). On the other hand, TAN forces the model to show the same augmenting edges for all classes. As can be expected, multinets perform as well as TAN, and that neither approach clearly dominates.

Q2(6)Which Performs best? :

BRIEF:

To say that one particular approach is better than other is often difficult because it totally depends on the data set. For instance decision tree generally will give better result always because it takes into account the dependencies of the word features but NB will be better if the word features have less dependency on each other. NB also more efficient if the training data do not contain all the possibilities .NB perform better in given question because the data set is small and we have been asked to classify the words in two different classes which are alt.atheism and comp.graphics which are two classes with nearly no dependency. However, this is not always true. A detail discussion of both approaches is as follows:

DETAIL:

Decision Trees are very flexible, easy to understand, and easy to debug. They will work with classification problems and regression problems. So if we are trying to predict a categorical value like (red, green, up, down) or if we are trying to predict a continuous value like 2.9, 3.4 etc Decision Trees will handle both problems. Probably one of the coolest things about Decision Trees is they only need a table of data and they will build a classifier directly from that data without needing any up front design work to take place. To some degree properties that don't matter won't be chosen as splits and will get eventually pruned so it's very tolerant of nonsense. To start it's set it and forget it.

However, the downside is that Simple decision trees tend to **over fit** the training data more so that other techniques, which mean you generally, have to do tree pruning and tune the pruning procedures. You didn't have any upfront design cost, but you'll pay that back on tuning the trees performance. Also simple decision trees divide the data into squares so building clusters around things means it has to split a lot to encompass clusters of data. Splitting a lot leads to complex trees and raises probability you are over fitting. Tall trees get pruned back so while you can build a cluster around some feature in the data it might not survive the pruning process. Other techniques like boosting and random forest decision trees can perform quite well, and some feel these techniques are essential to get the best performance out of decision trees. Again this adds more things to understand and use to tune the tree and hence more things to implement. In the end the more we add to the algorithm the taller the barrier to using it.

Naive Bayes requires you build a classification by hand. There's not way to just toss a bunch of tabulare data at it and have it build the classifier by picking and choosing the right pieces of data for you. If there were you'd be getting close to using the same techniques that make decision trees work like that. It'd be interesting, but it'd be a decision tree of Bayes classifiers and that might not perform any better. So you'll need to design the probabilities you'll keep track of and how each node contributes to the combined probability. Naive Bayes will answer as a continuous classifier. There are techniques to adapt it to categorical prediction however they will answer in terms of probabilities like (A 90%, B 5%, C 2.5% D 2.5%) Bayes can perform quite well, and **it doesn't over fit nearly as much so there is no need to prune or process the network. That makes them simpler algorithms to implement**. However, they are **harder to debug and understand because it's all probabilities getting multiplied 1000's of times so we have to be careful to test it's doing what you expect. Naive Bayes does quite well when the training data doesn't contain all possibilities so it can be very good with low amounts of data.**

Naive Bayes is used a lot in robotics and computer vision, and does quite well with those tasks. Decision trees perform very poorly in those situations. Teaching a decision tree to recognize poker hands by looking a millions of poker hands does very poorly because royal flushes and quads occurs so little it often gets pruned out. If it's pruned out of the resulting tree it will misclassify those important hands (recall tall trees discussion from above).

Decision trees are neat because they tell us what inputs are the best predicators of the outputs so often decision trees can guide you to find if there is a statistical relationship between a given input to the output and how strong that relationship is. Often the resulting decision tree is less important than relationships it describes. So decision trees can be used a research tool as you learn about your data so you can build other classifiers.