Alec Yu 993433, Mike Jaworski 833751

Answering: Questions 1, 2, 4, 6

1. Try discretising the numeric attributes in these datasets and treating them as discrete variables in the naive Bayes classifier. You can use a discretisation method of your choice and group the numeric values into any number of levels (but around 3 to 5 levels would probably be a good starting point). Does discretizing the variables improve classification performance, compared to the Gaussian naive Bayes approach? Why or why not?

When discretising the numeric datasets, we chose to do a k-means discretisation with 5 bins. We can see in the results below that using this discretisation technique on a numeric dataset significantly improves our naïve Bayes' model when compared to the Gaussian naïve Bayes approach.

A possible reason for this, but very unlikely, is that our k-means discretisation separated the data's attributes into clusters that well-represent the natural patterns in the data. This would then help our classifier's accuracy as each segregated group would have similar traits.

Another and more probable reason are that the Gaussian Naïve Bayes approach assumes that the data is normally distributed, when it could potentially not be. It seems with the results below, that potentially some of the numeric attributes were not normally distributed, so the Gaussian model would be assigning the wrong relative probability densities, and therefore probabilities, to each class given a set of attributes. If the probabilities assigned to each instance given their attributes is incorrect, then of course this would directly lower the classification accuracy of the Gaussian model.

Dataset	Discretised accuracy	Gaussian accuracy
wdbc	0.94	0.63
wine	0.98	0.40

This is also consistent with the results from the mixed "university" dataset where the discretised model performs better than the Gaussian model. In this case, however, the accuracy between classifiers is a lot more comparable. Perhaps this is due to less columns being converted from numeric to categoric, so less assumptions are being potentially violated in the Gaussian model.

Dataset	Discretised accuracy	Gaussian accuracy
university	0.50	0.44

2. Implement a baseline model (e.g., random or 0R) and compare the performance of the naïve Bayes classifier to this baseline on multiple datasets. Discuss why the baseline performance varies across datasets, and to what extent the naive Bayes classifier improves on the baseline performance

We chose to compare our model with a 0R baseline model. In this baseline model, we are choosing the mode class to classify every instance. As such, we can assume that the performance of our baseline model will vary across different datasets, as the distribution of classes and the percentage of the whole dataset that the mode class occupies will be different for each dataset. For example, when comparing the baseline strategy between "mushroom.data" and "car.data", we can see that the mode class on mushroom makes up ~51% of the dataset while the mode class in car.data makes up ~70% of the dataset. If we chose to classify a portion of each dataset using the 0R strategy, we can assume that roughly our accuracies would be similar to the numbers before.

When compared to this baseline model, our naïve Bayes classifier always outperforms this baseline, shown in the results below. We can attribute this due to the Naïve Bayes classifier being more sophisticated than the 0R baseline model (essentially no sophistication). Rather than picking the most common class, naïve Bayes aims to select the most probable class label for an instance given the membership of its attributes.

Below are the results of all the datasets being run through the model after discretisation. Most notably, the wine numeric dataset performs similarly to the zeroR baseline when not discretised, however achieves an accuracy close to 1 after discretisation. Perhaps, the numeric attributes in wine are much better represented by clusters, rather than a normal distribution.

Dataset	Baseline performance	Naïve bayes performance (5-cv)
mushroom	0.52	0.98
Car	0.70	0.86
Wine	0.40	0.98
Somerville	0.54	0.60
university	0.37	0.50

4. Evaluating the model on the same data that we use to train the model is considered to be a major mistake in Machine Learning. Implement a hold–out or cross–validation evaluation strategy (you should implement this yourself and do not simply call existing implementations from scikit-learn). How does your estimate of effectiveness change, compared to testing on the training data? Explain why. (The result might surprise you!)

We consider training and testing the model on the same dataset as a major mistake in Machine Learning as this tends to overfit our model to the training set and doesn't generalise well to new instances. When testing on the same data we trained on, we expect that the effectiveness and accuracy of our model would be much higher when compared with testing on data the model hasn't seen before (training data). We expect this to be because the model would really just be "memorising" the patterns of each class in the dataset rather than being able to "learn" labels of instances it has never seen before. If we only expose our model to the training data, then our model wouldn't really be predicting anything, as it wouldn't be exposed to anything new when being tested. To compare, we used a 5-cross validation strategy.

As shown below, we can see that our model's accuracy when tested on the same dataset that it trained on is much higher than when we used different datasets for training and testing, with the exception of wine. In fact, the overfitting is so prevalent that in some datasets, our model obtains an accuracy of 1.0 or very close to it!

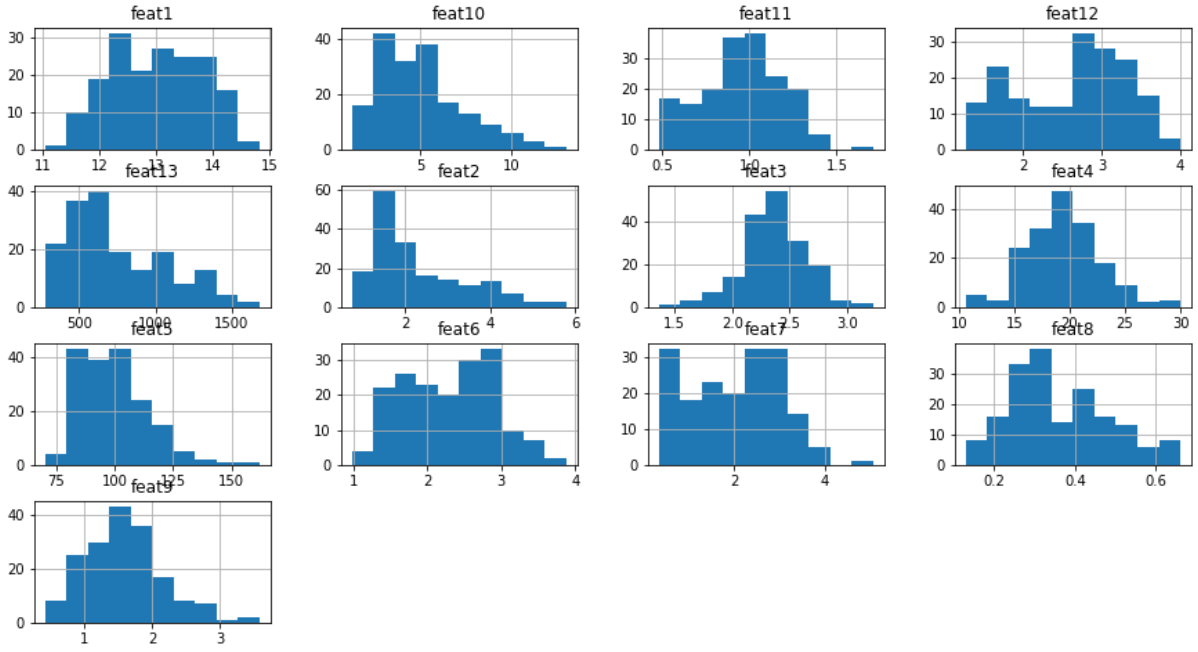
Dataset	Test set == Train set accuracy (Using $1/5^{th}$ of the training set as test set)	5-Cross validation average accuracy
Breast-cancer-wisconsin	0.993	0.977
Car	0.922	0.863
wine	0.401	0.400
University	1.000	0.443

6. The Gaussian naive Bayes classifier assumes that numeric attributes come from a Gaussian distribution. Is this assumption always true for the numeric attributes in these datasets? Identify some cases where the Gaussian assumption is violated and describe any evidence (or lack thereof) that this has some effect on the NB classifier's predictions

Not all the numeric attributes in the numeric datasets were distributed normally. This assumption was violated more predominantly in the wine dataset than the wdbc dataset, with more attributes being not normally distributed in wine.data and some attributes even being extremely skewed or following the shape of a seemingly entire different distribution. Of course, when our assumption of the distribution of data is incorrect, we are going to be assigning incorrect probability densities to certain attribute values. Since a Gaussian naïve Bayes classifier makes predictions based on relative probability densities between certain attribute's values, if we get these wrong then this will have a direct impact on our classification accuracy. Shown below, we can see a few examples of attributes in these datasets having a non-normal distribution. While we didn't run an experiment to isolate the direct impact of assuming a normal distribution for the below attributes, we did run

a discretised version of the numeric datasets and compared it with running it using a Gaussian naïve Bayes classifier. In the table of results below, we can obviously see that discretising the numeric data before running a Naïve Bayes classifier performed much better than using the Gaussian version. While this doesn't imply much about the effect of the attributes below in isolation, it implies that running a Gaussian Naïve Bayes model for non-normally distributed data is non-optimal for a classification task and there are much better options available.

Feature distributions for wine: maybe features 3, 4 and 9 look normally distributed. The other features look as if they violate the assumption of normality.



Dataset	Accuracy Gaussian	Accuracy Discretised (NB)	Fold number (cross-validation)
wdbc.data	0.649122	0.912280	0
	0.596491	0.947368	1
	0.526315	0.938596	2
	0.699115	0.982300	3
	0.672566	0.911504	4
	0.628721	**0.938410**	**averaged**
wine.data	0.444444	0.972222	0
	0.416666	1.0	1
	0.457142	0.971429	2
	0.400000	0.942857	3
	0.285714	1.0	4
	0.400792	**0.977295**	**averaged**