CS-422
Fall 2018
A20423189
Assignment 2

1) Data Analysis

1.1

IRIS - Number of attributes: 5(including class)

Numeric	4
Nominal	1

Vote - Number of attributes: 17(including class)

Nominal	17

Diabetes - Number of attributes: 9(including class)

Numeric	8
Nominal	1

1.2 For the Iris data set report: Number of attributes: 5

Attribute	Minimum	Maximum	Standard Deviation	Range
Sepal length	4.3	7.9	0.828	3.6
Sepal Width	2	4.4	0.434	2.4
Petal length	1	6.9	1.764	5.9
Petal Width	0.1	2.5	0.763	2.4
Class	-	-	-	-

The range of each attribute is calculated by subtracting the max and min.

1. Range measures the spread of the attributes, which could be useful to check if any outliers exists. In few cases where finding the outliers is the main aim they must be retained (fraud detection) else such data objects must be discarded for getting higher accuracy.
2. Range is not definitive measure to tell if values in this spread(attribute) is closely packed or loosely packed. It doesn't imply anything about the class distribution of the dataset.

Using the examples that we covered at the end of lecture 3 to discuss the data, variables overlap, what performance we can expect by analyzing and visualizing the data.

Iris dataset- The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

No. of instances: 150

No. of attributes: 5

Numeric: 4 & Nominal:1 (class)

Class Distribution: 33.3% for each of 3 classes.

1. Using the sepal-length or sepal-width it's hard to classify any of the groups correctly.
2. Using just the petal-length or petal-width we can classify at least one the class (Iris-setosa) accurately. 100% on petal-length and 98% on petal-width. Hence this would give us better accuracy.

3. The graph of sepal-length shows that Iris-setosa have smaller sepal-length Iris-versicolor have a medium length and Iris-virginica has a bigger sepal-length. But these cannot distinguish by a line of separation and hence classifying purely based on it won't give us good results as there is a variable overlap, when you consider sepal-width again it's not the best attribute to classify on as the variables values are overlapping but they act as a valuable input for the decision tree overall.

   Performance we can expect on by analyzing the data:
   By visualizing the graph, we can decide which attribute might contribute more to the classifier. Using feature like petal-length we can easily get 100% performance on the Iris-setosa, but not for classifying Iris-versicolor or Iris-virginica.
   Considering other features based on how informative they are we can make this a better classifier (i.e. not training it with only the petal-length but using petal-width, sepal-width and sepal-length).

---

2) For each data set use 3 decision tree algorithms: Decision Stump, J48, Random Forest
These are few of the parameters discussed in class and some of which I found important.

- Decision stump (weak learner) (1-level decision tree)

  -batchSize

  The desired batch size for batch prediction (default 100).
  The number of training examples utilized in one iteration.
  The classifier is trained in many iterations where the parameters are updated in each iteration. The batch size gives the number of data objects to be considered for training in such iterations.

- J48

  - unpruned

  Whether pruning is performed. It takes Boolean value for this parameter.
  If we use an unpruned tree it may lead to overfitting. I.e. we learn the training set too well but it doesn't generalize that well on the testing set. Pruning reduces the complexity of the Decision tree, by reducing its size removing the sections of the tree that provides very little information to classify.

  - confidenceFactor <pruning confidence>

  Set confidence threshold for pruning. (default 0.25)
  The confidence is used to compute a pessimistic upper bound on the error rate at a leaf/node.
  Smaller value more pessimistic the error is and higher the pruning.

  - minNumObj <minimum number of instances>

  Set minimum number of instances per leaf. (default 2)
  When the Decision tree goes on making a classification this acts as the stopping condition where it stops when a node/leaf contains only 2 instances. Increasing it heavily can cause underfitting as the classifier doesn't build the tree to its fullest capacity and decreasing it heavily makes the classifier learn one particular type of example which may not be a common case in general or seen again. This parameter acts as a stopping condition as stated in the class. Leading overfitting as well as a complex tree and highly time consuming to train the model when used with very small values.

  - reducedErrorPruning

  Use reduced error pruning instead of C.4.5 pruning.

It uses the error found based on the validation dataset. It removes the subtree at that node, make it a leaf and assign the most common class at that node.

- Random Forest

  - numFeatures

  Numbers of features being used for the classifier.

  Sets the number of randomly chosen attributes. If 0, int(log_2(#predictors) + 1) is used.

  If a dataset contains many features, we can rank them based on the information gain and use the ones with higher information gain and get good predictions as well save the time on training time.

  - numIterations

  Number of iterations. (current value 100)

  It gives the number of trees.

  The number of times the classifier learns updating its parameter in each iteration. For a small dataset small value will be enough, but on larger dataset u must experiment.

  - numExecutionSlots <num>

  Number of execution slots. (default 1 - i.e. no parallelism)

  (use 0 to auto-detect number of cores)

  Is useful when training a huge dataset as parallel computing can be utilized to train the classifier decreasing the time taken to train the model.

  - maxDepth <num>

  The maximum depth of the tree, 0 for unlimited. (default 0)

  This parameter can be used to set how many levels we want in the decision tree. As the depth increases the performance and the time taken to train increases, but after a certain depth the performance remain constant or even decrease due to overfitting Hence this is one main parameter in random forest.

  - printClassifiers

  Print the individual classifiers in the output.

  This lets us see how the classifier makes its decision. Just for the understanding purpose and visualizing the how the tree is making its decision. What splits are being used.

-Describe the class distribution for each dataset.

- The iris dataset is uniformly distributed as there are 50 instances of each of the 3 labels.
- The vote dataset is not uniformly distributed (skewed or biased) as there are 267 instances of democrat and 168 instances of republican
- The diabetes dataset is not uniformly distributed as there are 500 instances of tested_negative and 268 instances which is being tested_positive.

Balanced data (uniformly distributed) do well for classification problems, but that's not the case all the time. If we try to balance the data then there is loss of information about frequencies of the data object can occur, which might have a negative impact on accuracy. For example, when we consider the diabetes dataset, it captures the information that there are less people have been tested positive for diabetes overall compared to people who have been tested negative.

Most of the learning algorithm don't do well on the imbalance data as the algorithm tries to focus on abundant class to maximize the accuracy of the classification. Hence, use cost matrix to alter our classifier based on what we want our classifier to do. Those were some of the examples how class distribution matter.

10-fold cross-validation

The training set and test set:
Iris 150/10 = 15 datapoints in each of the 10-bins.

training set	135
test set	15

Vote

training set	391	or	392
test set	44		43

Diabetes

training set	691	or	692
test set	77		76

The class distribution varies from one iteration to another on the chosen samples used for classifier. Hence the classifier gets to learn in different class distributions.
On these training and test data you run 10 separate learning experience and average the test results from each of those training test. Whereas while splitting the data into 80:20 or 60:40 training testing sets there could be some bias in which few kinds of data object is just found on the testing test but not trained on them or vise-versa. This will also help us from not overfitting the data just to the training set and getting a very good accuracy on the training set but not on classifying the new unseen data object in the testing set or validation set.
No, we don't have the same class distribution as in the full dataset. It does matter as it will be an imbalanced set as explained in above about the imbalanced dataset. There are other advantages of using the 10 folds.

pros

- ✓ used all the data for training and as well as for testing,
- ✓ better results.
- ✓ Doesn't increase the bias as all the data is considered for training & evaluation.

cons

- ▪ More processing time,

Default cost matrix
(All the Confusion matrix is by using diabetics dataset)

0	1.0
1.0	0

```
=== Confusion Matrix ===

   a   b   <-- classified as
 418  82 |   a = tested_negative
 104 164 |   b = tested_positive
```

Cost sensitive learning is a way to punish the classifier for making False Positive or False Negatives. By default, it is set to 1 which means it is not going to punish for making any false positive or false negatives and work the normal way it had on learning from the training set.

Applying weights (on FN)

0	2.0
1.0	0

```
=== Confusion Matrix ===

   a   b   <-- classified as
 455  45 |   a = tested_negative
 141 127 |   b = tested_positive
```

When the above weights are applied, the classifier is punished when it makes a False-Negative.
When we run this matrix on the diabetic's dataset the no. of False-Negatives reduces drastically but on the other hand the number of False-Positives increases, the classifier tries to get low instances classified under FN as it has more penalty in this case.
This might not be the case we want as we do not want the classifier to tell a person doesn't have diabetes when he has it.
False-Negatives are related to recall hence this would give us better Recall for the classifier.

Applying weights (on FP)

0	1.0
2.0	0

```
=== Confusion Matrix ===

   a   b   <-- classified as
 384 116 |   a = tested_negative
  80 188 |   b = tested_positive
```

When the above weights are applied, the classifier is punished when it makes a False-Positive.
When we run this matrix on the diabetic's dataset the no. of False-Positive reduces drastically but on the other hand the number of False-Negatives increases, the classifier tries to get low instances classified under FP as it has more penalty in this case.
False-Positives are related to precision hence this would give us better Precision for the classifier.
Nothing can be told about the F1 using just the cost matrix.

Conclusion: We can use the cost matrix to make the classifier train based on our reasoning of if we want the classifier to be more careful on making a false positive or false positive as these are subjective to each application. Ex classifier shouldn't classify a person with diabetes as negative when it is positive, but we can relay on a classifier which does the opposite as further test is carried on if the chances are high.

	Training set	66-44 split (on test set)	10-fold
Accuracy(default):	100	68.1992	75.7813
Accuracy FN:	97.3958	68.1992	75.7813
Accuracy FP:	99.8698	31.8008	74.4792

For the following example it doesn't make that big of a difference on accuracy as 10-fold cross validation was used but that is not the case always, when traditionally splitting the data into training and testing set the accuracy on the test set is always much lower than the training set on which it been trained. It does matter as in the test set there might be certain instances that is never seen by the model and hence accuracy on the test set decreases. When cost matrix is introduced on the 66split the data is all classified under single class using all the 3-cost matrix previously used, which isn't a good classifier even though we get accuracy like close to 100% on the training set but reduces to 68% or 31% on the test set.

Iris

	Decision Stump	**J48**	**Random Forest**
no. of leaves	3	5	9
Size of the tree	4	9	17

vote

no. of leaves	3	6	48
Size of the tree	4	11	95

Diabetics J48

no. of leaves	3	20	96
Size of the tree	4	39	191

Size of tree: Gives the total no. of nodes in the tree.

No. of leaves: Gives the total no. of nodes in the tree that have exactly on incoming edge and no outgoing edge. Such leaf nodes are assigned a label in the decision tree.

Random Forest - iris

Depth (maximum depth of the tree)		0 (unlimited) **DEFAULT**	1	5
	accuracy	96.0784	62.7451	96.0784
	Size of tree	7-17		
I (Number of iterations)		100 **DEFAULT**	1	1000
	accuracy	96.0784	96.0784	96.0784
	Size of tree	9-19		

As the Depth (maximum depth of the tree) increased up to a certain point the accuracy increased, after the limit the accuracy remained constant.

I dint find any change on this dataset when I changes the number of Iteration, but it did change when I used it on diabetic's dataset i.e. increased when I changed the no. of iteration. The time taken to train the model also increases with increase in number of iterations

Random forest classifier to analyze the number of trees.

(Iris dataset)

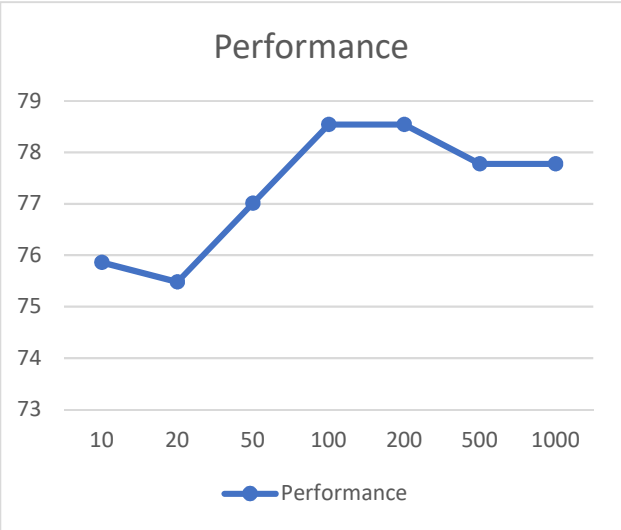
	10	20	50	100	200	500	1000
Performance	95.333	95.333	94.6667	95.3333	95.3333	95.3333	95.3333
Time to train(S)	0.04	0.03	0.03	0.05	0.08	0.13	0.18

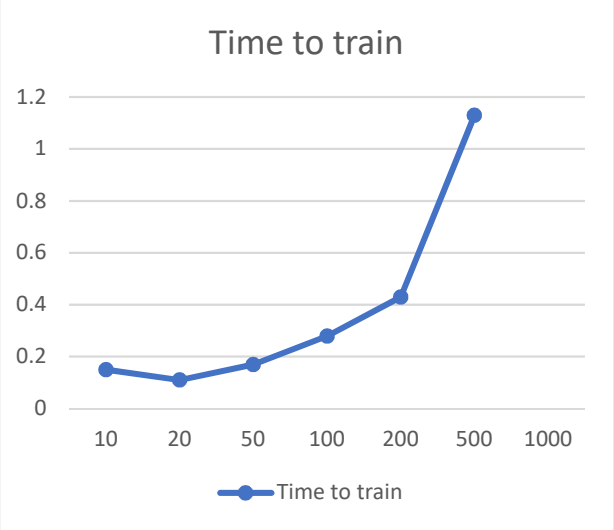
(Vote dataset)

	10	20	50	100	200	500	1000
Performance	96.3218	95.3333	96.5517	96.092	96.5517	96.3218	96.5517
Time to train(S)	0.09	0.03	0.08	0.12	0.17	0.35	0.57

(Diabetes dataset)

	10	20	50	100	200	500	1000
Performance	75.8621	75.4789	77.0115	78.5441	78.5441	77.7778	77.7778
Time to train(s)	0.15	0.11	0.17	0.28	0.43	1.13	2.22





The graph above(left) shows the performance(y-axis) v/s number of trees(x-axis) for diabetics' dataset.
The graph above(right) shows the time to train(y-axis) v/s number of trees(x-axis) for diabetic's dataset.

Conclusion: The performance of the classifier increases with increase in no. of tree up to a certain number (in this case 200) but then gradually decreases after that as it can lead to overfitting.
The time to train increases as the no of tree increases. This trend is seen all the dataset.

Iris dataset

No. of tree	1 DEPTH	10 DEPTH	0(unlimited)
10	70.6667 (0s)	95.3333 (0.01s)	95.3333 (0.01s)
100	83.3333 (0.02s)	95.3333 (0.03s)	95.3333 (0.04s)
500	93.3333 (0.13)	95.3333 (0.12s)	95.3333 (0.13s)

Vote dataset

No. of tree	1 DEPTH	10 DEPTH	0(unlimited)
10	93.7931 (0s)	96.5517 (0.02)	96.3218 (0.02s)
100	92.8736 (0.03)	96.3218 (0.12)	96.092 (0.12s)
500	93.3333 (0.12)	96.092 (0.3)	96.3218(0.3s)

Diabetics dataset

No. of tree	1 DEPTH	10 DEPTH	0(unlimited)
10	72.3958 (0.01)	75.1302 (0.04)	74.349 (0.05)
100	72.7865 (0.08)	75.5208 (0.25)	75.7813 (0.24)
500	72.526 (0.24)	75.7813 (0.93)	75.7813 (1.02)

Conclusion:

- As the number of tree and depth increases the time taken to train the model also increases.
- Increasing the depth increases the accuracy as the classifier gets one level extra to learn but giving it an extreme value has a negative impact as the classifier is overfitted.

Use the option to print the classifier (for a small number of trees),

It shows the classifier criteria for dataset and threshold chosen to partition the node. As classifier runs from one iteration to another we can notice how the parameters are getting updated. I.e. how the classifier learns.

---

3) Feature Selection

	CorrelationAttributeEval	**InfoGainAttributeEval**
Iris	0.615 3 petallength 0.592 4 petalwidth 0.478 1 sepallength 0.397 2 sepalwidth	1.418 3 petallength 1.378 4 petalwidth 0.698 1 sepallength 0.376 2 sepalwidth
Vote	0.9096 4 physician-fee-freeze 0.7343 3 adoption-of-the-budget-resolution 0.6837 5 el-salvador-aid 0.6666 12 education-spending 0.6283 9 mx-missile 0.617 8 aid-to-nicaraguan-contras 0.6063 14 crime 0.5268 13 superfund-right-to-sue 0.5127 15 duty-free-exports 0.5045 7 anti-satellite-test-ban 0.413 6 religious-groups-in-schools 0.3931 1 handicapped-infants 0.3669 11 synfuels-corporation-cutback 0.3519 16 export-administration-act-south-africa 0.0838 10 immigration 0.011 2 water-project-cost-sharing	0.7078 4 physician-fee-freeze 0.4185 3 adoption-of-the-budget-resolution 0.4028 5 el-salvador-aid 0.3403 12 education-spending 0.3123 14 crime 0.3095 8 aid-to-nicaraguan-contras 0.2856 9 mx-missile 0.2121 13 superfund-right-to-sue 0.2013 15 duty-free-exports 0.1902 7 anti-satellite-test-ban 0.1404 6 religious-groups-in-schools 0.1211 1 handicapped-infants 0.1007 11 synfuels-corporation-cutback 0.0529 16 export-administration-act-south-africa 0.0049 10 immigration 0.000011 2 water-project-cost-sharing
Diabetics	0.4666 2 plas 0.2927 6 mass 0.2384 8 age 0.2219 1 preg 0.1738 7 pedi 0.1305 5 insu 0.0748 4 skin 0.0651 3 pres	0.1901 2 plas 0.0749 6 mass 0.0725 8 age 0.0595 5 insu 0.0443 4 skin 0.0392 1 preg 0.0208 7 pedi 0.014 3 pres

1. Iris- petal length is the best feature according to my intuition, as it can classify all the Iris-setosa correctly. But it is hard to classify the other 2 classes. The next best would be the petal width if the features are considered independently as it can classify all data-instances of one class expect one. So, I would go with the same one the feature selection attribute selected based on data analysis. On this dataset both the feature selector ranks the attribute in the same manner.
2. Vote- physician-fee-freeze would be my first preference as well, as majority of the democrats can be easily classified by considering the split at this attribute. It has a pure node which can classify all the data-instances correctly. It also classifies Republicans correctly majority of the time but has a few instances of the Democrats which can be further classified considering the next best features. So, I would go with the same parameter as the feature selector. On this dataset there is

a few changes in the order which the attributes are ranked. As many attributes exists in this dataset we can consider dropping few which have don't impact much on the classifier.

3. Diabetics- plas which stands for plasma glucose concentration, is the best attribute as we can concluded that the chances of person being diabetics is less when the attribute value is below 100 as most of the value is blue in the graph but as the value of this attribute increases the chances of him being diabetic also increases. The same trend goes with the mass. There is on trend seen with pregnancy or other attributes. So, I would consider the same feature as the feature selector.

As we consider features which are more important (feature selection) the accuracy keeps increasing, but if the dataset contains too many attributes when we select too many features the accuracy drops as it leads to overfitting where the classifier tries to learn specific instances which may not be a general case and making the classifier hard to classify on an unseen data. We need to consider this while building a classifier and find the optimal value for the no. of features and minimum number of instances required to split to be considered in the model training process. We can also remove features which do not contribute to the classification process or duplicate attributes in the training and test data as removing these records can make the feature space smaller and increase the efficiency. *Choosing few features using feature selection can also reduce the time taken to train the model.*

Example: Vote dataset (random forest- 10-fold CV)

No. of attributes chosen	**7**	**10**	**13**	**17 (All)**
accuracy	94.9425	95.1724	95.1724	96.092
Training time	0.07	0.11	0.11	0.23s

Above u can see the Training time increase with increase in number of features.

---

4) Missing values - Iris dataset

(Using decision stump)

Before		**5%**	**10%**	**50%**
66.6667	accuracy	66.6667	66.6667	64
0s	Training time	0	0	0

(using J48)

Before		**5%**	**10%**	**50%**
96	accuracy	95.33	93.33	91.33
0.02s	Training time	0.01	0.0	0.0

(using random forest)

Before		**5%**	**10%**	**50%**
95.333	accuracy	96	96	92
0.05s	Training time	0.08	0.03	0.06

To introduce the missing values, I had introduced "?" in the dataset for a few data Object as told above (5% around 8 values, 10% around 15values, 50% around 70values) on random attributes. With the introduction of missing value, initially it didn't matter much on the dataset but eventually as the number of missing values increased the accuracy decreased and the training time increased in few cases.

Data object of missing value can be removed if it is a huge dataset as removal of these points wont effect the dataset. There are other options like using the mode or average values for these missing feature values.

Noise:
(Using decision stump)

Before		**5%**	**10%**	**50%**
66.6667	accuracy	63.3333	59.3333	50.6667
0s	Training time	0	0	0

(using J48)

Before		**5%**	**10%**	**50%**
96	accuracy	88	82.6667	43.3333
0.02s	Training time	0	0	0

(using random forest)

Before		**5%**	**10%**	**50%**
95.333	accuracy	88	82	41.33333
0.05s	Training time	0.04	0.04	0.06

To introduce noise, I had to misclassify the classes of the data object. As the noise increased there is noticeable decrease in the accuracy of the classifier and increase in the training time. The accuracy has dropped from 95% to 41% on the introduction of noise using random forest classifier. Hence, we must take measures to reduce as much noise as possible. Noise have higher impact on the classifier accuracy when compared to missing values.
No real dataset can be noise free as there can be some mistake like typo or interchanging of variables. Missing values and Noise are few of the things to be taken care off before training the model. These are usually included in the preprocessing steps.

By changing the range of the attribute by making it 1000 times larger then the other feature range didn't affect the classifier accuracy as we have multiplied each of the values by 1000, the distribution remained the same but only the range changed hence there is no difference in the classifier accuracy.

Correctly Classified Instances 143 95.3333 %

```
=== Confusion Matrix ===

  a  b  c   <-- classified as
 50  0  0 |  a = Iris-setosa
  0 47  3 |  b = Iris-versicolor
  0  4 46 |  c = Iris-virginica
```