CS36220: Machine Learning – Assignment
Report - Adrian Tukendorf [adt24]

1. *Task 1- Investigation of classifier performance*

In this section, I will analyse the performance of various classifier learners. I will use dataset letterRecog.arff. Classes are distributed equally by 300 data instances in the dataset. There are 17 features, including the decision class label “A” - “Z” in alphabetic order. Data in every learner’s run are divided using Cross-Validation with 10 folds, to assure the dataset is divided into equal proportions. It is also important to note that some of the data instances are missing. The classifiers of my choice that I will exam are: Support Vector Machines (LibSVM in WEKA), NaiveBayes, lazy - IBk, and decision tree-based approach- J48.
Note: all given values are predictions

A. *LibSVM*

Summary of the results for kernel **-** radial basis function

Correctly Classified Instances	7194	92.2308 %
Incorrectly Classified Instances	606	7.7692 %
Kappa statistic	0.9192	
Mean absolute error	0.006	
Root mean squared error	0.0773	
Root relative squared error	40.1995 %	

(Average) Detailed Accuracy by Class:

True Positive Rate 0.922	higher=better
False Positive Rate 0.003	lower=better
Precision 0.923	higher=better
Recall 0.922	higher=better
ROC Area 0.960	higher=better binary
F-Measure 0.923	higher=better*

B. *NaiveBayes*

Summary of the results

Correctly Classified Instances	4788	61.3846 %
Incorrectly Classified Instances	3012	38.6154 %
Kappa statistic	0.5984	
Mean absolute error	0.0341	
Root mean squared error	0.143	
Root relative squared error	74.3468 %	

(Average) Detailed Accuracy by Class:

True Positive Rate 0.614
False Positive Rate 0.015
Precision 0.625
Recall 0.614
ROC Area 0.952
F-Measure 0.609

C. *IBk*

Summary of the results

Correctly Classified Instances	6057	77.6538 %
Incorrectly Classified Instances	1743	22.3462 %
Kappa statistic	0.7676	
Mean absolute error	0.0174	
Root mean squared error	0.1308	
Relative absolute error	23.5462 %	
Root relative squared error	68.0104 %	

(Average) Detailed Accuracy by Class:

True Positive Rate 0.777
False Positive Rate 0.009
Precision 0.786
Recall 0.777
ROC Area 0.886
F-Measure 0.778

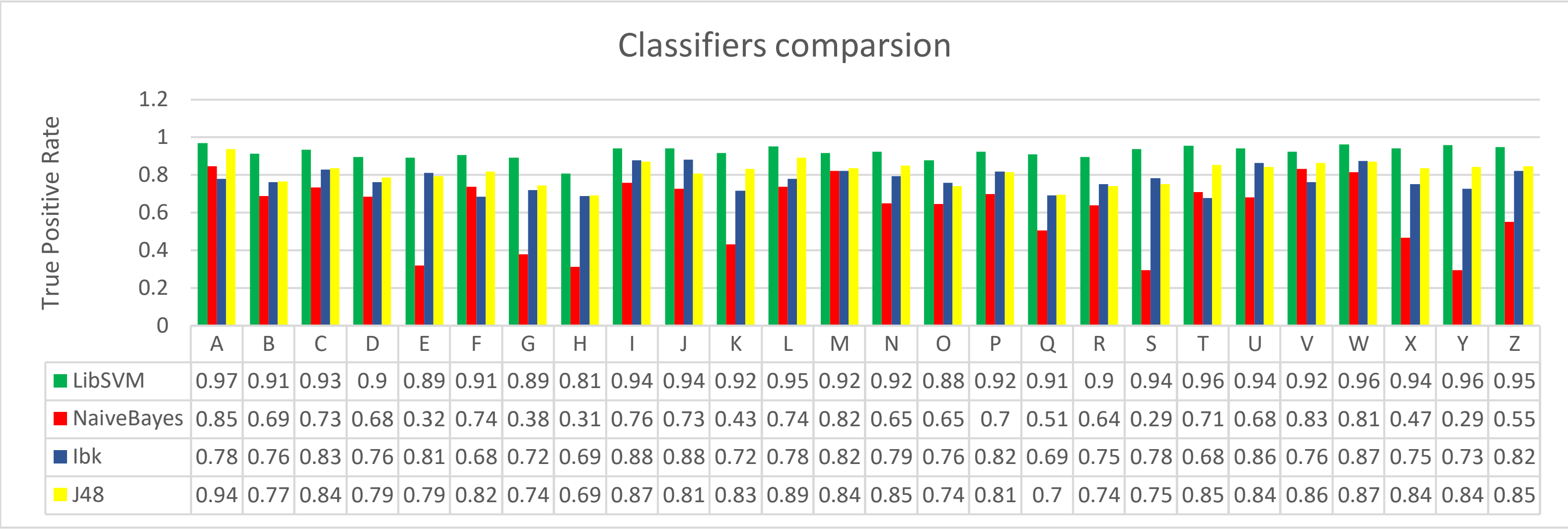
D. *J48*

Summary of the results

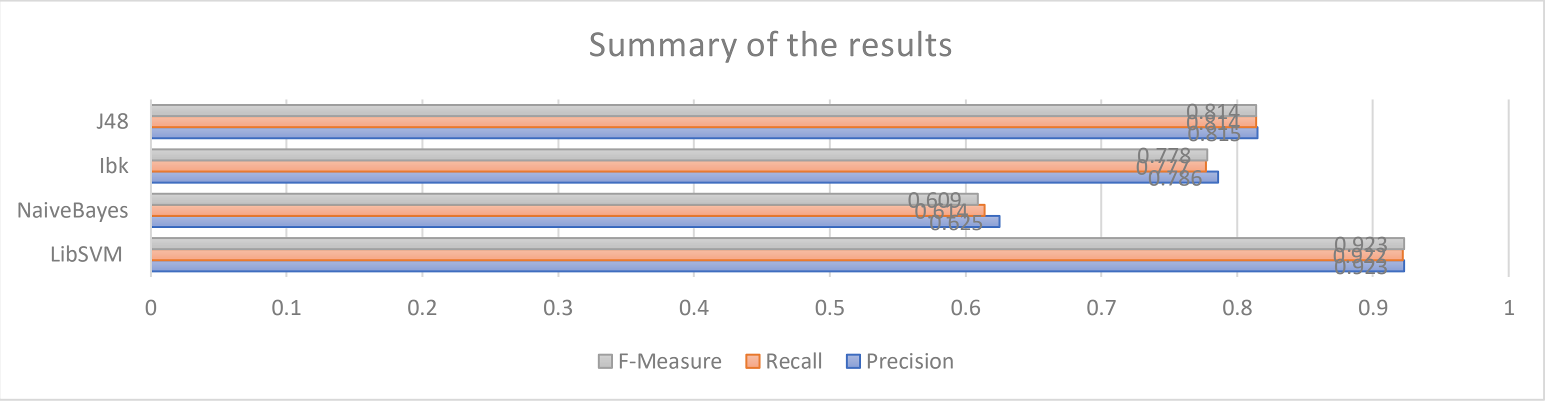
Correctly Classified Instances	6349	81.3974 %
Incorrectly Classified Instances	1451	18.6026 %
Kappa statistic	0.8065	
Mean absolute error	0.017	
Root mean squared error	0.1118	
Relative absolute error	22.9386 %	
Root relative squared error	58.1543 %	

(Average) Detailed Accuracy by Class:

True Positive Rate 0.814
False Positive Rate 0.007
Precision 0.815
Recall 0.814
ROC Area 0.929
F-Measure 0.814



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
🟩 LibSVM	0.97	0.91	0.93	0.9	0.89	0.91	0.89	0.81	0.94	0.94	0.92	0.95	0.92	0.92	0.88	0.92	0.91	0.9	0.94	0.96	0.94	0.92	0.96	0.94	0.96	0.95
🟥 NaiveBayes	0.85	0.69	0.73	0.68	0.32	0.74	0.38	0.31	0.76	0.73	0.43	0.74	0.82	0.65	0.65	0.7	0.51	0.64	0.29	0.71	0.68	0.83	0.81	0.47	0.29	0.55
🟦 Ibk	0.78	0.76	0.83	0.76	0.81	0.68	0.72	0.69	0.88	0.88	0.72	0.78	0.82	0.79	0.76	0.82	0.69	0.75	0.78	0.68	0.86	0.76	0.87	0.75	0.73	0.82
🟨 J48	0.94	0.77	0.84	0.79	0.79	0.82	0.74	0.69	0.87	0.81	0.83	0.89	0.84	0.85	0.74	0.81	0.7	0.74	0.75	0.85	0.84	0.86	0.87	0.84	0.84	0.85



Analyse of the results

LibSVM predicted every class significantly better than any others of the three classifiers. Values of F-Measure, Recall, and Precision are also the best for LibSVM. On the second place in my comparison of the best classifiers for the given problem is the tree-based approach – J48. IBk is on the third place, but it performed better than J48 in classes: E(+2%), J(+7%), O(+2%), P(+1%), R(+1%), S(+3%), U(+2%) but only class J is significantly better. Naive Bayes performed the worst from all of the tested learners on given data. Its performance was significantly worse than other tested classifiers, except classes where IBk performed worse: A(+7%), T(+3%), V(+7%).

Experimenter test, shows correctly classified instances and standard deviation of the results in 10 folds Cross-Validation, which seems like the best way to compare the learners taking into consideration the distribution of the data.

```
Dataset              (1) functions.LibS | (2) bayes.Naive (3) lazy.IBk '- (4) trees.J48 '
------------------------------------------------------------------------------------------
letterRecog          (100)   92.35(1.00) |   61.42(1.67) *   77.74(1.52) *   81.02(1.30) *
------------------------------------------------------------------------------------------
                          (v/ /*) |       (0/0/1)       (0/0/1)       (0/0/1)
>-<  >  < Resultset
  3  3  0 functions.LibSVM '-S
  1  2  1 trees.J48 '-C 0.25 -
 -1  1  2 lazy.IBk '-K 1 -W 0
 -3  0  3 bayes.NaiveBayes ''
```

Experimenter test confirms my previews insight, LibSVM achieved a classification accuracy of 92.35 (+/-1%) which is significantly better than J46 81.02% (1.30%), IBk 77.74% (+/-1.52%) NaiveBayes 61.42% (+/-1.67%).

Kappa Statistics

```
Dataset              (1) functions.Lib | (2) bayes.Naiv (3) lazy.IBk ' (4) trees.J48
------------------------------------------------------------------------------------
letterRecog           (100)   0.92(0.01) |   0.60(0.02) *   0.77(0.02) *   0.80(0.01) *
------------------------------------------------------------------------------------
                             (v/ /*) |       (0/0/1)        (0/0/1)        (0/0/1)
```

To confirm my findings, I measured the Kappa Statistics. It is less misleading than simply measuring accuracy. It evaluates the classifier not only against other classifiers but also evaluate the classifier itself. Result clearly states that LibSVM is statistically the best.

Speed

Time taken to build model: LibSVM 5.37s, NaiveBayes 0.05s, IBk 0.01s, J48 0.68s

LibSVM takes the longest time to build the model. J48 being just slightly behind with performance takes much less time to build a model. Other classifiers are much faster but for the cost of performance on this specific data.

Conclusion

For the problem of image-analysis, the best performance was achieved by the sub-symbolic learner - LibSVM. As my findings confirm, it is appropriate to use LibSVM and J48 for the problem of letter recognition from the image. They handled well big amount of data instances and the problem of missing data in some features. Where NaiveBayes is just ignoring features with missing values, but it is much faster.

2. *Task 2 – Tackling the Issue of Missing Data*

After analysing the data, I found out that values are missing in random order to the classes but there is a dependency on attributes. Feat1(65), feat2(1809), feat3(19), feat4(12), feat5(53), feat6(94), feat7(1827), feat8(21), feat9(3). I will try different approaches and try to find the best one for this case.

Approach 1

Using WEKA filters – *ReplaceMissingValues*

Weka filter replaced missing values. They are the same in all data instances for the given feature. For example, all missing values in feat1 now have a value 4.014738. Which is the mean of this attribute.

```
Dataset                 (1) function | (2) trees (3) lazy. (4) bayes
-----------------------------------------------------------------
letterRecog-weka.filters.(100)   92.35 |   80.54 *   88.42 *   60.98 *
letterRecog                    (100)   92.35 |   81.02 *   77.74 *   61.42 *
-----------------------------------------------------------------
                                (v/ /*) |   (0/0/2)   (0/0/2)   (0/0/2)
```

LibSVM performance wasn't impacted by the changes. Although, the performance of tree-based approaches is lower. Probably because J48 deals with missing values during classification. Performance of IBk is 10.68% higher even that it deals with missing values during classification, by setting the distance to the maximum if one of the two attributes being compared has a missing value. NaiveBayes performance is lower after applying the filter that means that it is not a good approach to solve the problem of missing data, results are inconsistent and don't show significant improvement. Especially that most algorithms perform better or the same at not modified dataset

Approach 2

Removing missing values

```
        Dataset                (1) Original | (2) ReplacingFilter (3) Removed
    ----------------------------------------------------------
functions.LibSVM '-S 0 -K(100)   92.35 |   92.35     90.62 *
trees.J48 '-C 0.25 -M 2' (100)       80.54 |  81.02     75.87 *
lazy.IBk '-K 1 -W 0 -A \"(100)      88.42 |   77.74 *   86.80 *
bayes.NaiveBayes '' 59952(100)   60.98 |  61.42     61.23
    ----------------------------------------------------------
                                          (v/ /*) |   (0/3/1)   (0/1/3)
```

Removing missing values seems like one of the worst approaches to the problem. There are fewer data to train the classifier because the attributes with missing values were removed from the dataset. Results are significantly worse than the previews approach (except IBk +9.06% than the previous approach) and the original dataset.

Approach 3

DMI Decision-tree based Regression Imputation technique [*DMI* WEKA plug-in Author: Michael Furner]

I took a different approach, instead of removing missing values, it would be better to predict them using regression and imputation techniques depending on the dataset’s data. The algorithm uses regression in form of a J48 decision tree to split the dataset into horizontal segments to increase attribute correlations for the EMI algorithm which imputes the missing values.

```
                  Dataset         (1) DMI imputation | (2) Original
         ------------------------------------------------
  bayes.NaiveBayes '' 59952(100) 61.20 |   61.42
functions.LibSVM '-S 0 -K(100)   93.25 |   92.35 *
lazy.IBk '-K 1 -W 0 -A \"(100)       89.84 |   77.74 *
trees.J48 '-C 0.25 -M 2' (100)         80.14 |   81.02 v
         ------------------------------------------------
                                         (v/ /*) |       (1/1/2)
```

This approach is so far the best I tested, significantly improved performance of IBk (+12.1%) and less significant LibSVM (+0.9%).

3. Task 3 – Feature Selection

Feature	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Standard Deviation	1.916	3.304	2.013	2.263	2.228	2.022	2.336	2.705	2.35	2.516	2.584	2.074	2.356	1.528	2.593	1.635
Mean-avg.	4.033	7.075	5.139	5.386	3.551	6.886	7.503	4.615	5.208	8.319	6.448	7.914	3.071	8.332	3.747	7.821
Min.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Max.	14	15	14	14	15	15	15	15	15	15	15	15	15	15	15	14

Every feature has a very similar if not the same Range, which is the minimal and maximal value the feature can represent. The standard deviation of the features is not very different in each of them. That means that data in the features are spread at a quite similar distance from their average. Observed Mean, which is the average of all the data instances of the attribute, suggests that data in features luckily for the classifier are not redundant, that is because there is a high variation of this value in presented data. If the standard deviation and the mean are different for each of the attributes. This is a sign of the uniqueness of the features when we look at correlations between them. In my opinion, the best feature selection method for this problem would be CfsSubsetEval because it assesses the predictive ability individually for each of the attributes and the degree of redundancy among the features, preferring sets of attributes that are highly correlated with the class but with low correlation with other features. But to rise the prediction performance I will use the wrapper because it uses cross-validation, the learner to estimate the accuracy of each tested set and chooses the best one. In the table, I also noticed two very similar features that might be redundant. Its features 3 and 4 because they have similar std., mean, and also range. But I would evaluate my observation with an automated feature selection method especially that it has an option to treat missing data separately.

To visualise it more I decided to generate the plot, it is on the last page of this document.

Now, I will perform two attribute selection techniques on *letterRecogTrain.arff* dataset:

Using *CfsSubsetEval* with Greedy Stepwise search algorithm to generate feature ranking:

```
Ranked attributes:
16 0.233  13 feat13
14 0.312  11 feat11
13 0.359  12 feat12
12 0.394  14 feat14
9 0.423  15 feat15
7 0.445   9 feat9
5 0.454   8 feat8
1 0.46   10 feat10
2 0.46    7 feat7
3 0.458  16 feat16
4 0.457   6 feat6
6 0.446   4 feat4
8 0.436   3 feat3
10 0.422   1 feat1
11 0.403   5 feat5
15 0.268   2 feat2

Selected attributes: 13,11,12,14,15,9,8,10,7,16,6,4,3,1,5,2 : 16
```

Now, I will perform attribute selection using *Wrapper* with different classifiers and search methods I find appropriate.

*-features appearing in every tried approach

Wrapper method:

LibSVM, Greedy Stepwise forwards

Selected attributes: 6,8,9,10,11,12,13,14,15,16 : 10

LibSVM, Greedy Stepwise backwards:

Selected attributes: 1,4,5,6,8,9,10,11,12,13,14,15,16 : 13

J48, Greedy Stepwise forwards:

Selected attributes: 8,9,10,11,12,13,14,15,16 : 9

J48, Greedy Stepwise backwards:

Selected attributes: 1,2,3,4,6,8,9,10,11,12,13,14,15,16 : 14

J48, Greedy Stepwise backwards using 10 folds Cross-validation:

```
number of folds (%)  attribute
      3( 30 %)    1 feat1
      6( 60 %)    2 feat2
      5( 50 %)    3 feat3
      4( 40 %)    4 feat4
      2( 20 %)    5 feat5
      4( 40 %)    6 feat6
      7( 70 %)    7 feat7
     10(100 %)    8 feat8
     10(100 %)    9 feat9
     10(100 %)   10 feat10
     10(100 %)   11 feat11
     10(100 %)   12 feat12
     10(100 %)   13 feat13
     10(100 %)   14 feat14
     10(100 %)   15 feat15
     10(100 %)   16 feat16
```

Conclusion

Because in every method, features: 8,9,10,11,12,13,14,15,16 were selected, I will reduce my dataset maintaining only mentioned features. I will compare two datasets, original and with reduced features.

```
Dataset                    (1) Original | (2) Reduced Dataset
---------------------------------------------------
functions.LibSVM '-S 0 -K(100)   92.35 |   92.66
bayes.NaiveBayes '' 59952(100)   61.42 |   62.52 v
lazy.IBk '-K 1 -W 0 -A \"(100)   77.74 |   90.97 v
trees.J48 '-C 0.25 -M 2' (100)   81.02 |   81.20
---------------------------------------------------
                              (v/ /*) |   (2/2/0)
```

The reduced dataset is performing much better than the original. IBk classifier performance on the reduced dataset is 13.23% higher. Which is the most significant improvement among others. NaiveBayes noted a 1.1% improvement in classification. The other two learners also performed better but not statistically significant. I think the difference in performance was caused by the deletion of redundant and missing values. Less redundant data means that learner is not making a decision that is based on noise. I noted also improvement in building the model, LibSVM used to take 5.39s and after reducing the data it takes 3.18s. Feature selection also reduces the chance of the overfitting problem.

The tree generated by J48 for letterRecogTrain has features 13, 15, 11, 12, 14 ,8, 10 on the root nodes. That means these features are the most important to distinguish letters during classification. The tree generated for the test data would be similar. Features on the top would be very similar because these features are the most valuable in distinguishing the classes, that's why they always would be on the top of the tree. Both of the trees use features with the highest information gain metrics to build upon them.

4. Summary of results/findings

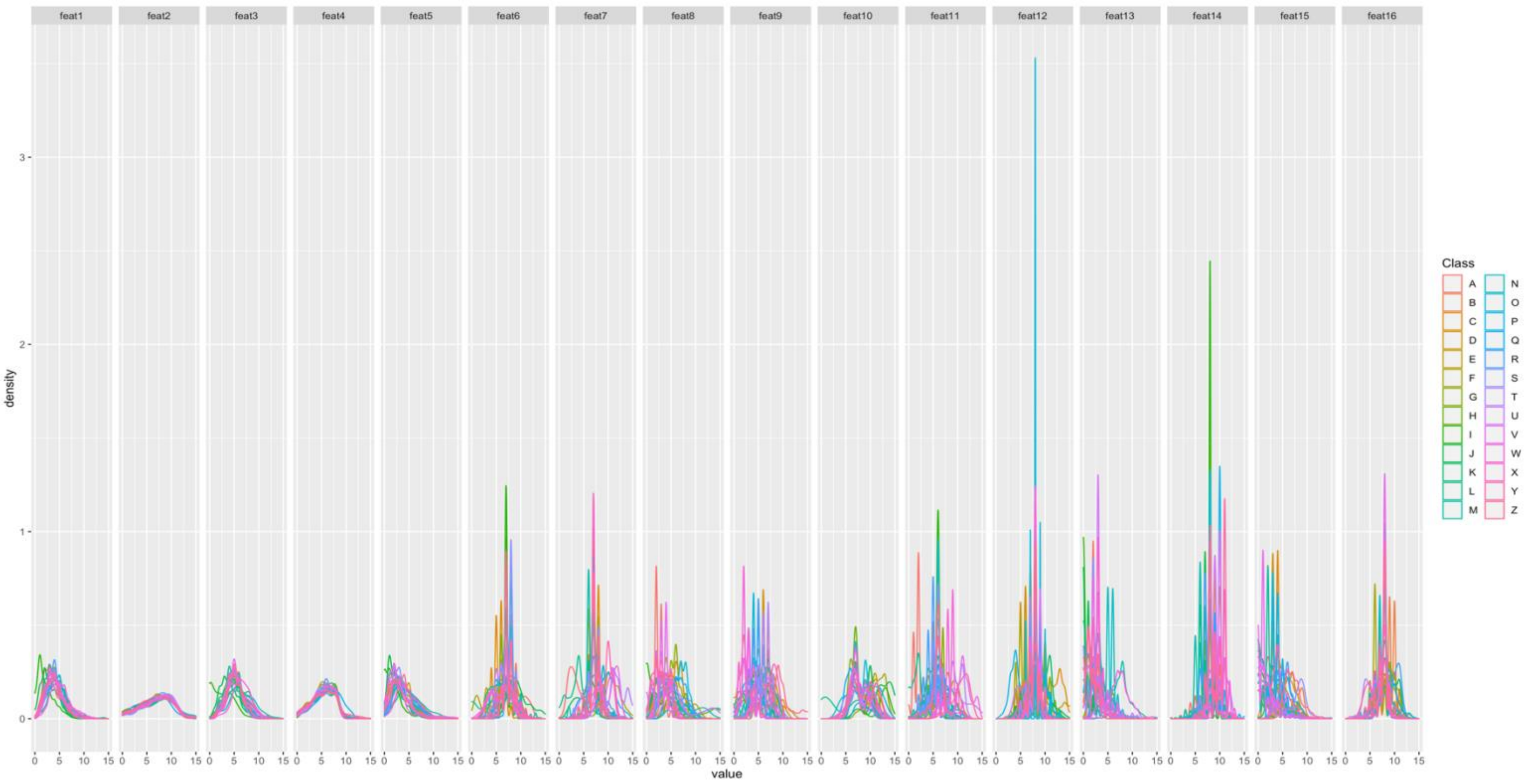
1. Good model for the full dataset, using cross-validation is LibSVM classifier and features 8-16. It has the best performance in prediction on the dataset when comparing to NaiveBayes, IBk, J48. It classifies the data with really high performance. It was confirmed in my experiments in WEKA Experimenter using a corrected k-paired test. Features selected in the task above were determined from the full dataset by using the wrapper method which examines a variety of combinations of feature sets and returns the one where the classifier achieves the best accuracy. Additionally, to confirm the results of my experiment, I used different classifiers and also opposite directions of search in the Greedy Stepwise algorithm. That gives me confidence that this classifier together with selected features builds a good model for given problem of letter recognition.

2. I tried several approaches to deal with missing data and the one which for me makes the most sense is DMI Decision-tree based Regression Imputation technique. That is because this algorithm determines the missing values using regression by looking at the data in the dataset and basing on their correlation replaces them. Filters in WEKA were replacing missing values with the mean of the feature where values were missing. That didn't bring the best results during my testing. That is why I think regression is an appropriate approach in this case.

3.Features from 8 to 16 are the most useful in the dataset. Feature selection was proven appropriate for this problem through experiments that show that the model is built faster, and accuracy is visibly improved. Feature selection also lowers the risk of overfitting and noise in the data caused by irrelevant or redundant features. Trees build upon the dataset shows which features are relevant and bring the most information to the model. Findings during analyses of the tree are consistent with the results from the wrapper method, showing relevant to the problem features. Analysis of the single attributes in my case wasn't much help, Although I was able to predict which features might be redundant by analysing Mean, Standard deviation, and Range. If two features have a similar mean and standard deviation that proves they might not contribute with relevant information to class prediction.

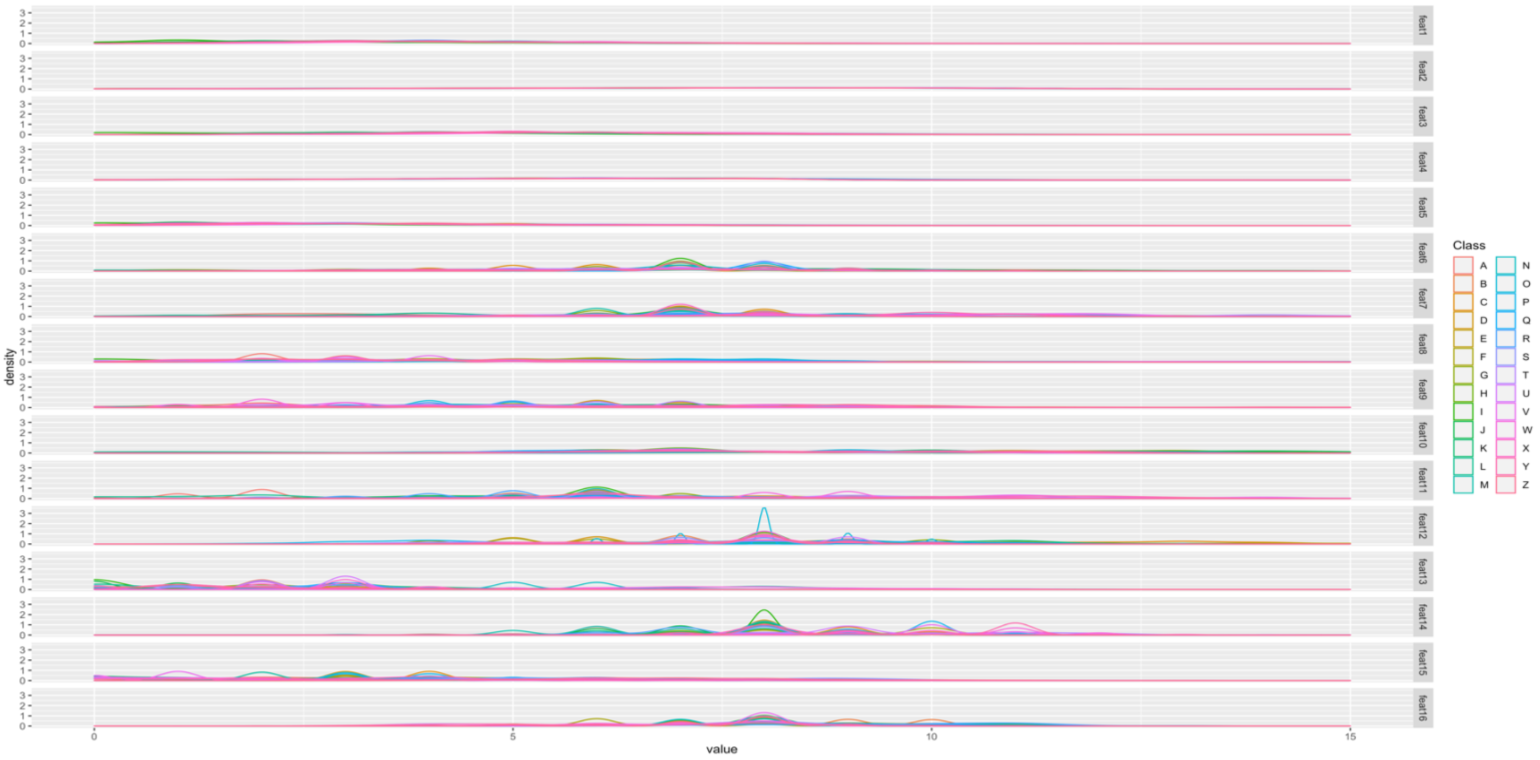
4.Yes, it reduced overfitting in the model, lowers the time needed to train the classifier, lowers the amount of redundant and not relevant attributes. As a result, it causes the model to focus on signal – relevant information, instead of using noise to make predictions. That all together improve the model and its prediction accuracy during dealing with unseen data. I proved this during conducted experiments.

Additional resource to analyse the features (ggplot2):



The plot shows features contribution to the class prediction, it helps to visualize the dataset. Feature 1, 3, 5 look very similar, that might be a sign that they are redundant. It is the same with features 2 and 4. The plot also shows how classes overlap on features 1,2,3,4,5. They will not contribute much to the model because of the lack of uniqueness. The rest of the features looks promising.

It is also easy to notice which features might be relevant and unique.
To visualize it better, I plotted the data in the way shown below, the graph of the first 5 features is mostly flat. That means they will not contribute much to the model, because none of the classes are dominating, that is why it would be harder to determine them from this feature.


feat1
feat2
feat3
feat4
feat5
feat6
feat7
feat8
feat9
feat10
feat11
feat12
feat13
feat14
feat15
feat16
density
value
Class
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z