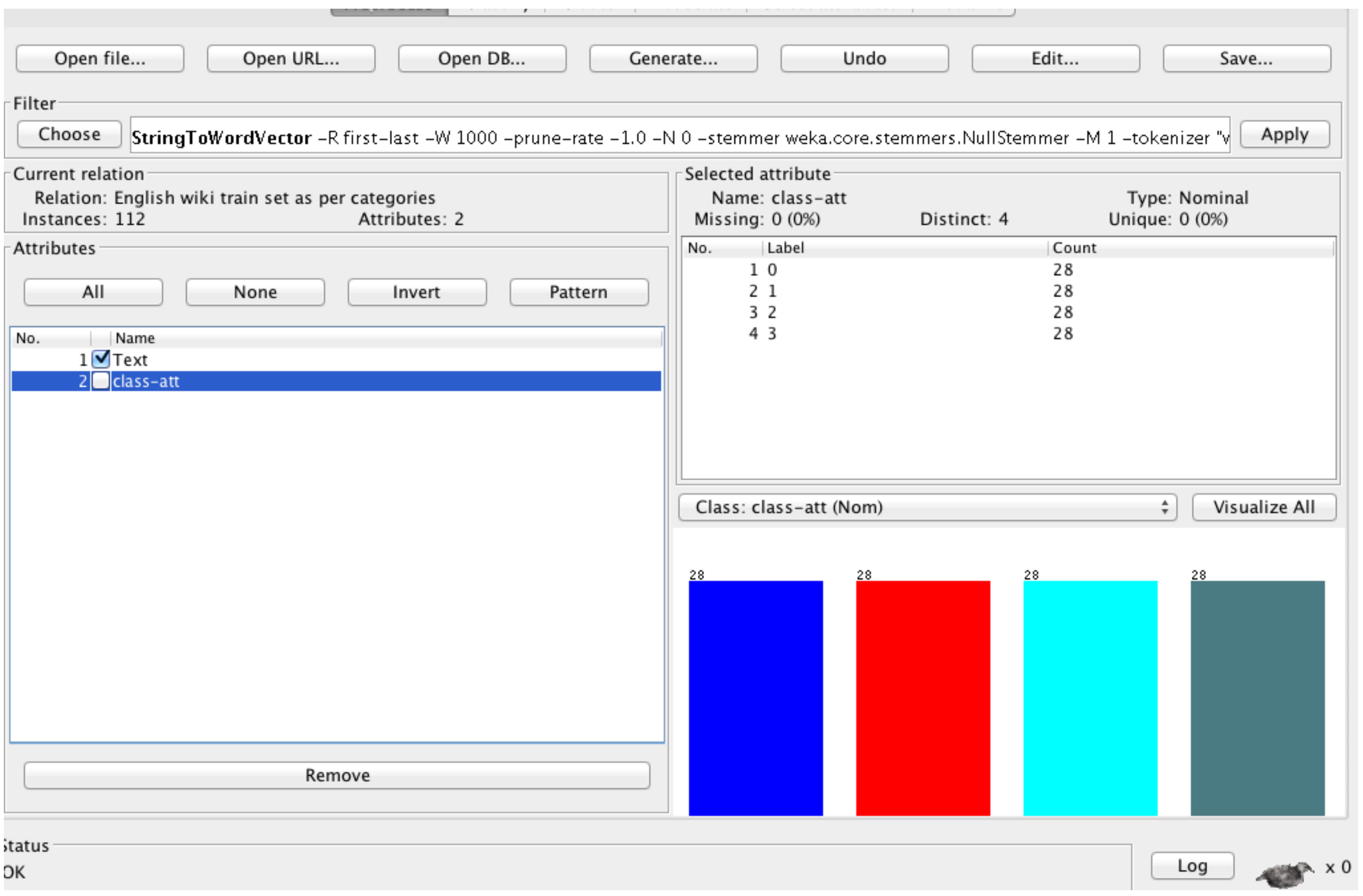
1. Classification Problem

Classify the wikipedia document in following categories. This is a multi class classification problem. The machine learning problem centered around classifying the documents which may belong to one of the following category.

- Arts
- History
- Politics
- Science
- Technology
- Sports

30-40 documents for each category for training set and for test set I am using 5-10 documents for each category.



Filter: StringToWordVector -R first-last -W 1000 -prune-rate -1.0 -N 0 -stemmer weka.core.stemmers.NullStemmer -M 1 -tokenizer "v

Current relation
Relation: English wiki train set as per categories
Instances: 112 Attributes: 2

Attributes

No.	Name
1	✓ Text
2	class-att

Selected attribute
Name: class-att Type: Nominal
Missing: 0 (0%) Distinct: 4 Unique: 0 (0%)

No.	Label	Count
1	0	28
2	1	28
3	2	28
4	3	28

Class: class-att (Nom)

Status
OK

2. Machine Learning models used

1.) Naive Bayes Multinomial
2.) LibSVM

3. Types of test data:

I am running the model on 4 types of data.

1.) Data downloaded from web and converted to arff without any feature processing.

2.) Data downloaded from web, tokenized, stop words were removed. and all the words put together again to form the text document.

MapReduce program was created for this process. The result was similar to previous arff file however stop words were removed.

3.) N grams without stop word.

Separate MapReduce program was created for this process. The mapReduce program was able to generate bag of words and their frequency for all the documents. The bag of words were fed to hive and stop words were removed and I am using the words were frequency greater than 2 for building my training model.

4.) **Further feature processing** has been done while training the classification model using filter and select attribute properties of weka.

I am running all the three models for all the above four types of data set.

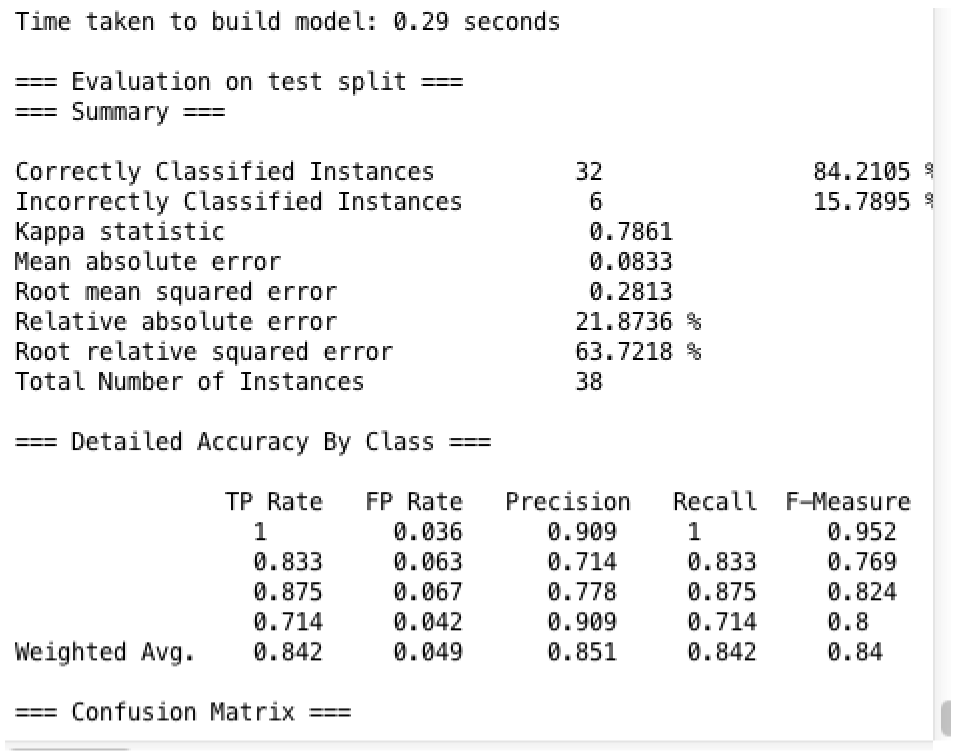
4. Classification with Naive Bayes Multinomial Model

4.1 Reason to use Naive Bayes Model

- Naive Bayes model is a very simple independent probabilistic classifier model
- The strong independent probability assumption makes the feature ordering not related to each other
- This cause the presence of one feature to not cause any effect to presence of another feature.
- Because of the independence probability quality Naive Bayes model works very well with text classification.
- Naive Bayes model also does not require large amount of training data to converge and this property also makes Naive Bayes a good choice for running quick test models

4.2 First Step: Use the unprocessed data

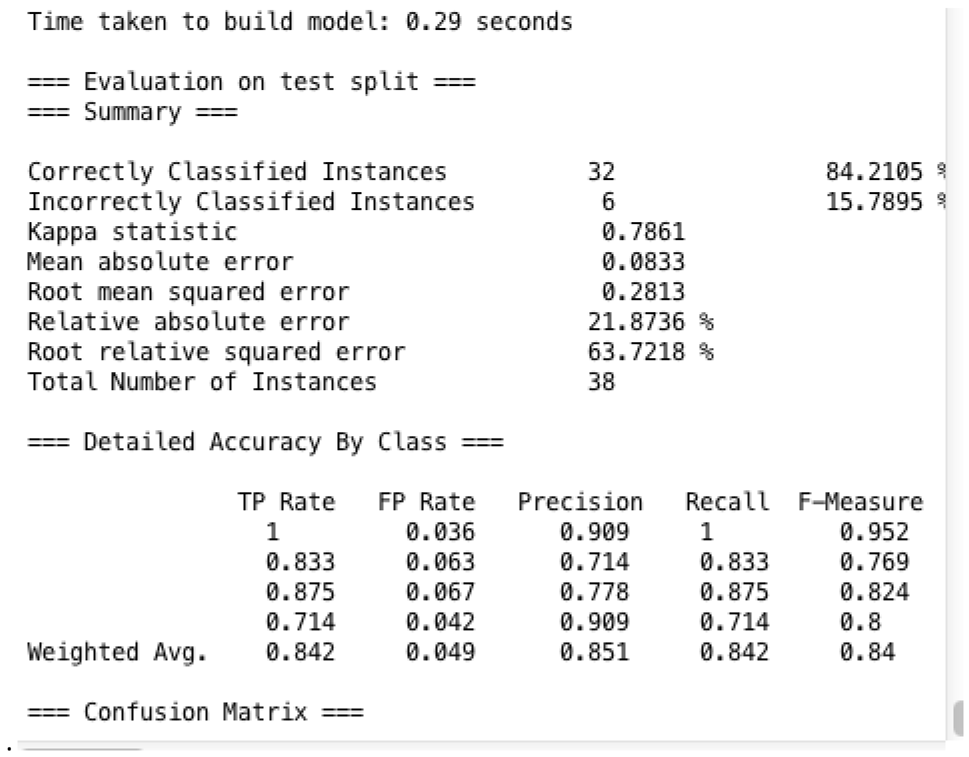
- Data file downloaded from internet was converted to arff file and supplied to train the model.
- StringToWordVector filter applied. This converted all the string to word vector.
- The training model gave 84.2105 % accuracy.
- We see that Naive Bayes Multinomial model is a very powerful model for text classification..



```
Time taken to build model: 0.29 seconds

=== Evaluation on test split ===
=== Summary ===

Correctly Classified Instances          32               84.2105 %
Incorrectly Classified Instances         6               15.7895 %
Kappa statistic                          0.7861
Mean absolute error                      0.0833
Root mean squared error                  0.2813
Relative absolute error                 21.8736 %
Root relative squared error             63.7218 %
Total Number of Instances               38

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure
                 1         0.036     0.909     1         0.952
                 0.833     0.063     0.714     0.833     0.769
                 0.875     0.067     0.778     0.875     0.824
                 0.714     0.042     0.909     0.714     0.8
Weighted Avg.    0.842     0.049     0.851     0.842     0.84

=== Confusion Matrix ===
```

4.3 Following questions arose?

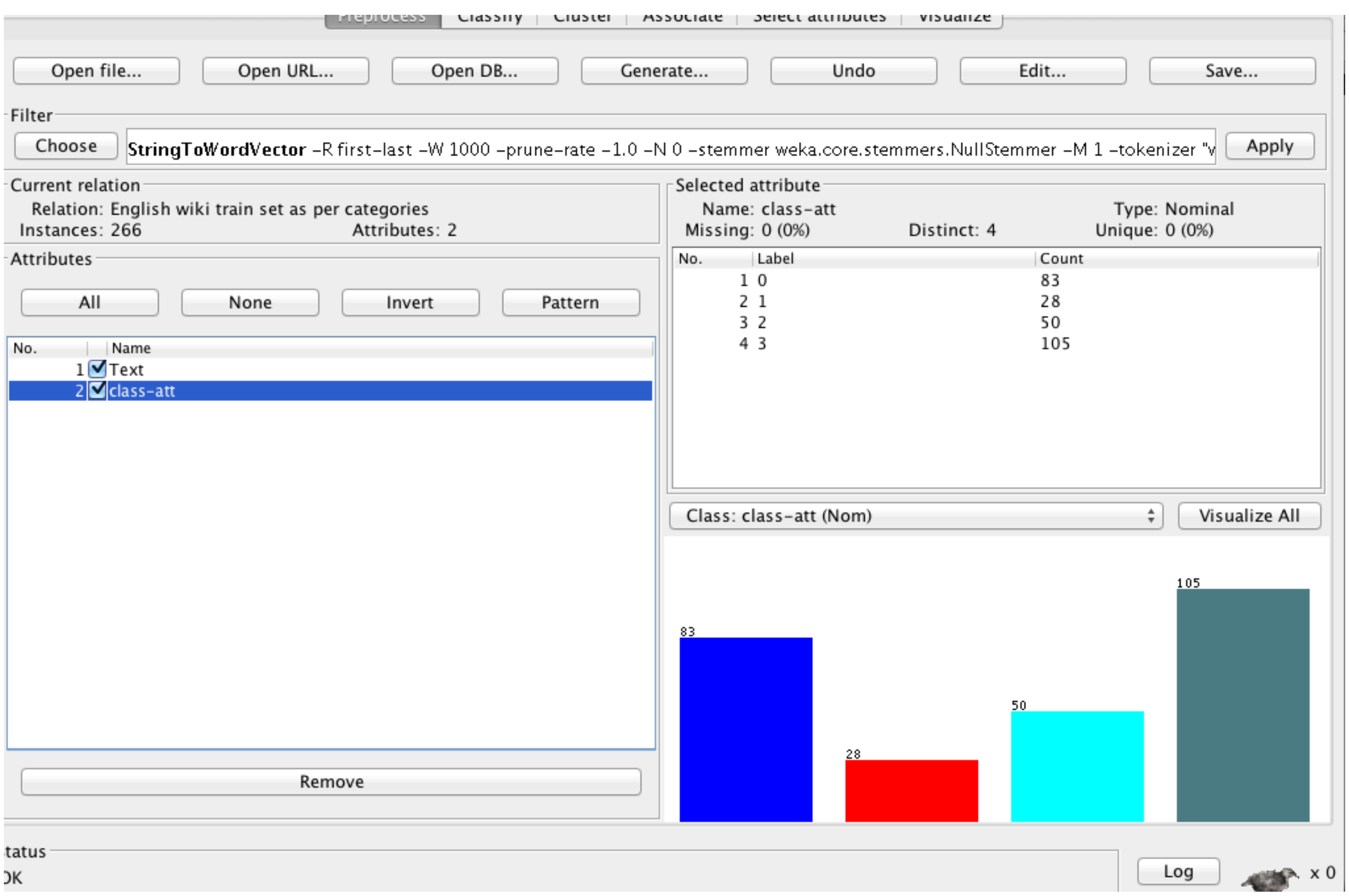
- Is this model overfitting because we have very less training data and generally if the model overfit the best way is to increase the training data.
- Can we optimized this model to 90% accuracy.
- We also see that the text contains lots of numeric word which seems unrelated and stop words such as are, am, the, a. Are these stop words are interfering with the result.



```
Time taken to build model: 0.29 seconds

=== Evaluation on test split ===
=== Summary ===

Correctly Classified Instances          32               84.2105
Incorrectly Classified Instances         6               15.7895
Kappa statistic                          0.7861
Mean absolute error                      0.0833
Root mean squared error                  0.2813
Relative absolute error                 21.8736 %
Root relative squared error             63.7218 %
Total Number of Instances               38

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure
                 1         0.036      0.909     1         0.952
                 0.833     0.063      0.714     0.833     0.769
                 0.875     0.067      0.778     0.875     0.824
                 0.714     0.042      0.909     0.714     0.8
Weighted Avg.    0.842     0.049      0.851     0.842     0.84

=== Confusion Matrix ===
```

4.4 Second Step : Double the no of Training Instances

- Doubled the no of instances and applied Naive Bayes Multinomial.
- Now I am getting 87% accuracy. Increasing the training example increased the accuracy which proves that the model is not overfitting.
- However stop words might be causing the high accuracy.



Filter

StringToWordVector -R first-last -W 1000 -prune-rate -1.0 -N 0 -stemmer weka.core.stemmers.NullStemmer -M 1 -tokenizer "w

Current relation
Relation: English wiki train set as per categories
Instances: 266 Attributes: 2

Attributes

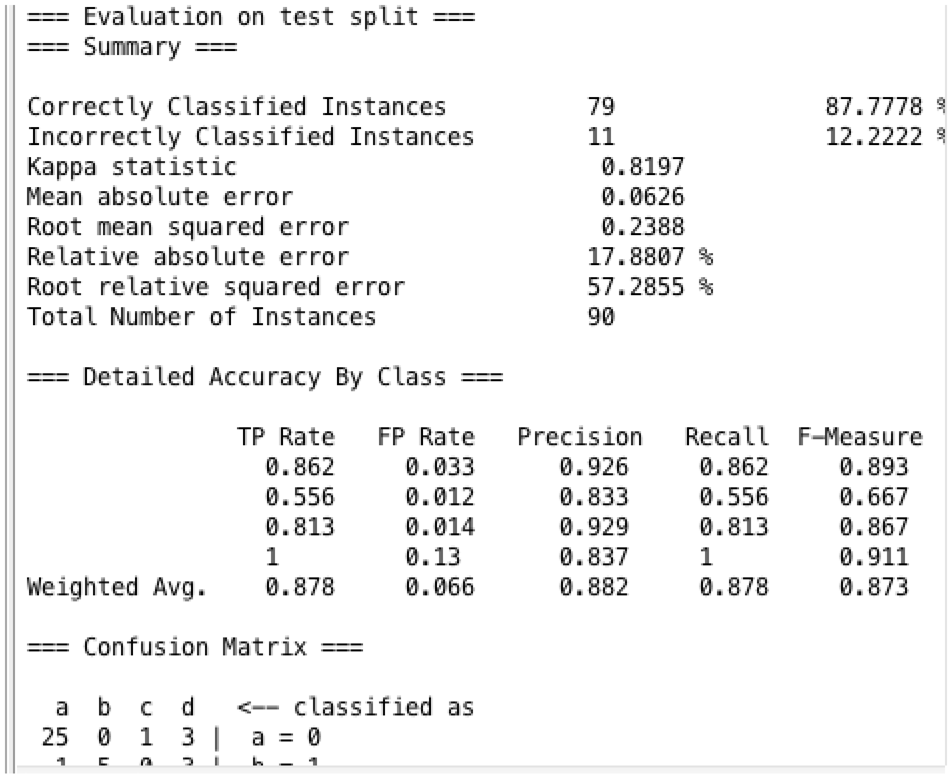
No.	Name
1	✓ Text
2	✓ class-att

Selected attribute
Name: class-att Type: Nominal
Missing: 0 (0%) Distinct: 4 Unique: 0 (0%)

No.	Label	Count
1	0	83
2	1	28
3	2	50
4	3	105

Class: class-att (Nom)

tatus
OK



```
=== Evaluation on test split ===
=== Summary ===

Correctly Classified Instances          79               87.7778 %
Incorrectly Classified Instances        11               12.2222 %
Kappa statistic                          0.8197
Mean absolute error                      0.0626
Root mean squared error                  0.2388
Relative absolute error                 17.8807 %
Root relative squared error             57.2855 %
Total Number of Instances               90

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure
                 0.862     0.033      0.926     0.862     0.893
                 0.556     0.012      0.833     0.556     0.667
                 0.813     0.014      0.929     0.813     0.867
                 1         0.13       0.837     1         0.911
Weighted Avg.    0.878     0.066      0.882     0.878     0.873

=== Confusion Matrix ===

  a  b  c  d   <-- classified as
 25  0  1  3 |  a = 0
  1  5  0  3 |  b = 1
```

In text attributes I am seeing lots of stop words. Stop words may be interfering with the accuracy

4.5 3rd Step: Removing the stop words.

- Created a MapReduce program which can remove the stop words from the text documents.
- After removing english stop words I am getting 80% accuracy.
- Which proves that stop words are causing the high accuracy.
- Used the same model with Naive Bayes multinomial text and getting the result as 87%
- However I still see some stop words such as books, arts, technology occurring with high frequency.
- **I used select attributes feature of weka** to check what are the best classifier attributes and I get the
- following results.

  abstract
  act
  arts
  big
  decay
  december
  greece
  historiography
  machines
  performance
  photography
  records
  techniques
  technologies

  - Here december, technologies arts are sop words
  - I removed the stop words and the accuracy decreased to 83%.
  - Which proves that stop words are definitely messing with the accuracy. As the Naive Bayes classifier considers each event independent and calculated probability independently. So these stop words are having high probability which may be causing the model to overfit.

4.6 Fourth Step: N-grams

- MapReduce program was created to convert the text document into bag of words.
- English stop words were removed.
- Fed the data to model
- The Naive Bayes model now gives very poor accuracy.
- All the words are being classified to class D.
- I also see that no of words in class D is the highest.

- This clearly looks like the problem of Hierarchical multi classification.
- I also observe that class D also contains some high frequency words which are occurring in all the other classes too which is causing this issue.

4.7 Optimizing the model for N-grams.

- Removed more stop words. But this also does not improve the accuracy
- Fed the n-gram data without frequency. This improved the result to 43%
- Here we see that frequency of n gram are causing the issue. There still might be some words which are occurring in all the classes.
- Tried tuning different parameters of StringToWordVector such as n gram tokenizer 3 IDF, TDF but accuracy has not been improved.
- The reason is the noisy data which causing the classifier to classify all the test data to a particular class only

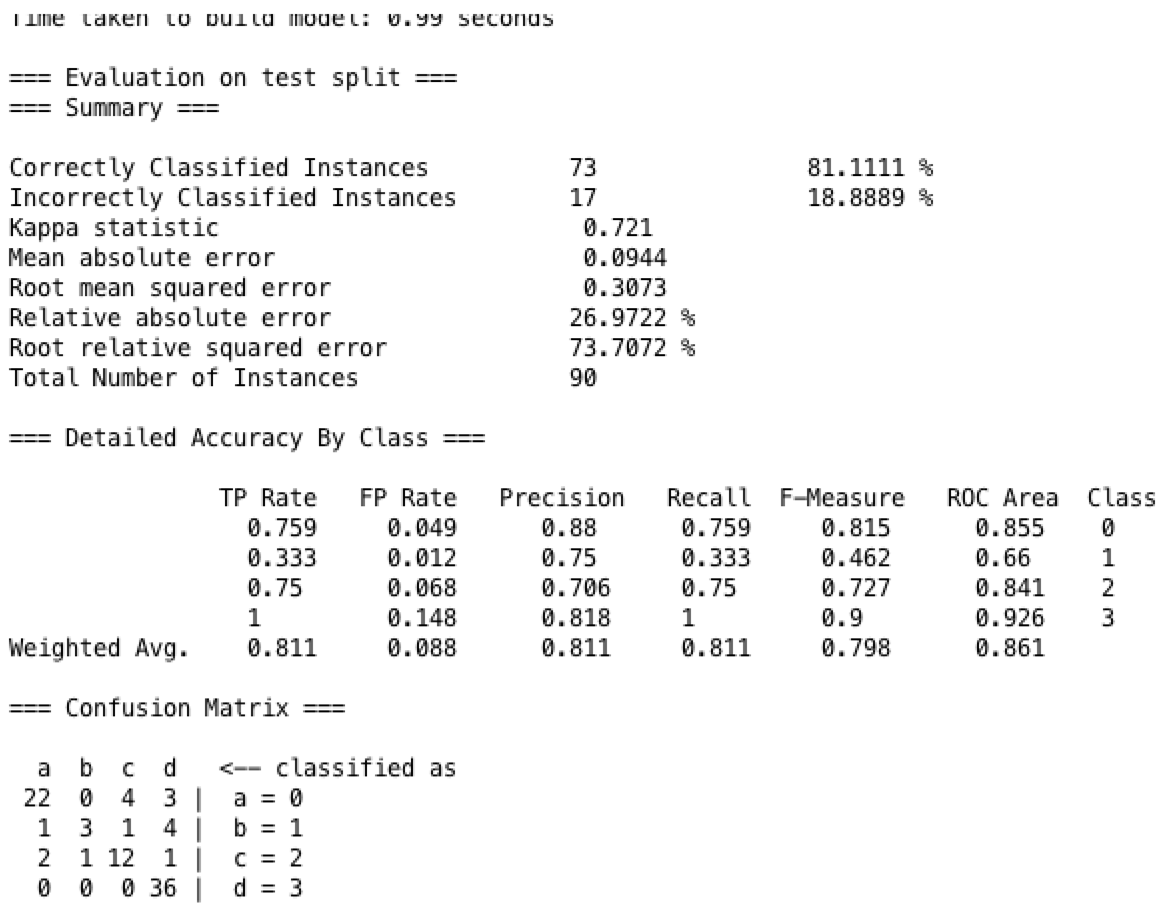
5. Support Vector Machine.

5.1 Reason to use Support Vector Machine Model

- Support Vector machine has been classified as one of the very powerful machine learning algorithm.
- Support vector machine works on reducing the multiple local minima issue.
- Support vector machines also uses complex kernel functions which can guarantee the lowest true error
-

5.2 First Step with the unprocessed arff file

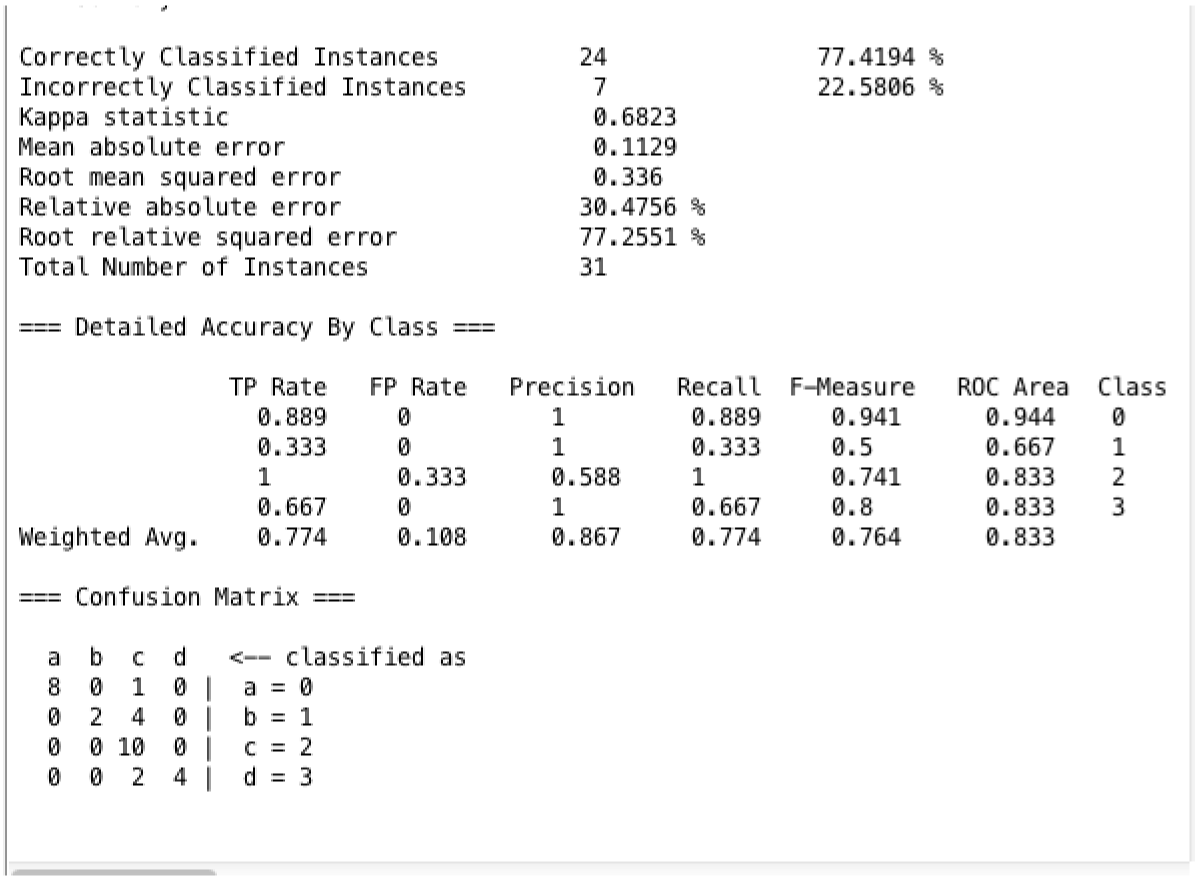
- This gives only 41.11 % accuracy with svm model.
- However changing the Kernel function to **linear u*v drastically improves the result to 81%** accuracy.
- Other kernel functions were also tested but linear function provides the best accuracy.



```
Time taken to build model: 0.99 seconds

=== Evaluation on test split ===
=== Summary ===

Correctly Classified Instances          73               81.1111 %
Incorrectly Classified Instances        17               18.8889 %
Kappa statistic                          0.721
Mean absolute error                      0.0944
Root mean squared error                  0.3073
Relative absolute error                 26.9722 %
Root relative squared error             73.7072 %
Total Number of Instances               90

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.759     0.049     0.88      0.759     0.815      0.855    0
                 0.333     0.012     0.75      0.333     0.462      0.66     1
                 0.75      0.068     0.706     0.75      0.727      0.841    2
                 1         0.148     0.818     1         0.9        0.926    3
Weighted Avg.    0.811     0.088     0.811     0.811     0.798      0.861

=== Confusion Matrix ===

  a  b  c  d   <-- classified as
 22  0  4  3 |  a = 0
  1  3  1  4 |  b = 1
  2  1 12  1 |  c = 2
  0  0  0 36 |  d = 3
```

5.4 Second Step with the tokenized documents after removing the stop words.

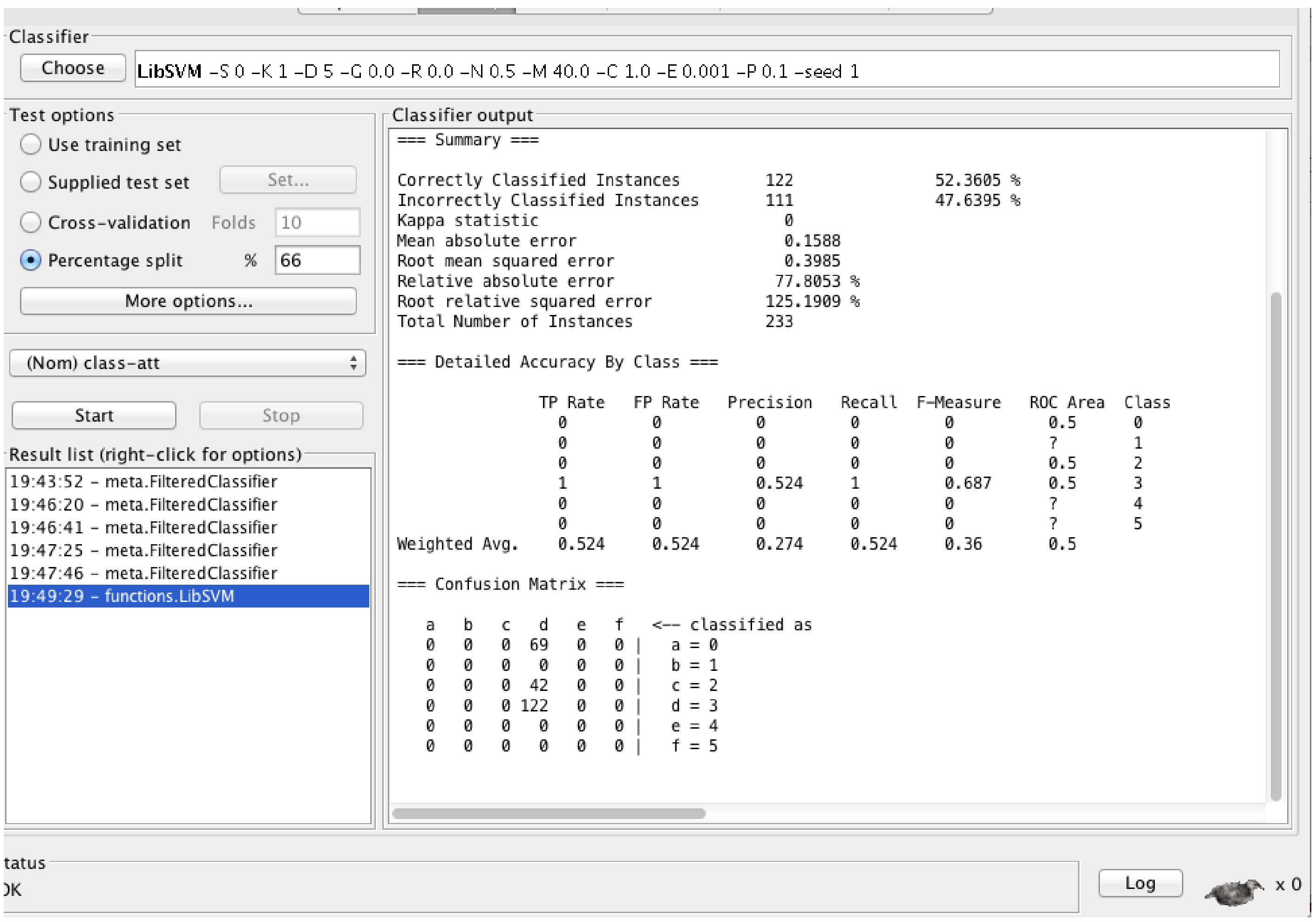
- This step has reduced the accuracy to 77.419%
- From the confusion matrix we also see that class c is correctly classified for all the instances. And all the incorrectly classification is to C class only.
- We also observe that the no of instances for C is greater in training set.
- Either we are having hierarchical multi classification issue
- Or the reason can be that the no of training data for C is greater.



```
Correctly Classified Instances          24               77.4194 %
Incorrectly Classified Instances         7               22.5806 %
Kappa statistic                          0.6823
Mean absolute error                      0.1129
Root mean squared error                  0.336
Relative absolute error                 30.4756 %
Root relative squared error             77.2551 %
Total Number of Instances               31

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.889     0         1          0.889     0.941      0.944    0
                 0.333     0         1          0.333     0.5        0.667    1
                 1         0.333     0.588      1         0.741      0.833    2
                 0.667     0         1          0.667     0.8        0.833    3
Weighted Avg.    0.774     0.108     0.867      0.774     0.764      0.833

=== Confusion Matrix ===

  a  b  c  d   <-- classified as
  8  0  1  0 |  a = 0
  0  2  4  0 |  b = 1
  0  0 10  0 |  c = 2
  0  0  2  4 |  d = 3
```

5.5 Third Step- N gram

- **N-grams** data were used to create the learning model.
- The SVM also gives very low accuracy for n-grams
- Linear kernel is giving 43% accuracy
- Various parameter tuning was used such as Ngramtokenizer attribute of StringToWordVector, TDF, IDF.
- Lastly created the model with very less amount of training data without word frequency.
- However the accuracy was improved to 52% only for the test data without frequency.



LibSVM -S 0 -K 1 -D 5 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 1.0 -E 0.001 -P 0.1 -seed 1

Cross-validation Folds 10

[✓ Percentage split] % 66

(Nom) class-att

```
19:43:52 - meta.FilteredClassifier
19:46:20 - meta.FilteredClassifier
19:46:41 - meta.FilteredClassifier
19:47:25 - meta.FilteredClassifier
19:47:46 - meta.FilteredClassifier
19:49:29 - functions.LibSVM
```

```
=== Summary ===

Correctly Classified Instances         122               52.3605 %
Incorrectly Classified Instances       111               47.6395 %
Kappa statistic                          0
Mean absolute error                      0.1588
Root mean squared error                  0.3985
Relative absolute error                 77.8053 %
Root relative squared error            125.1909 %
Total Number of Instances              233

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0         0          0         0         0          0.5      0
                 0         0          0         0         0          ?        1
                 0         0          0         0         0          0.5      2
                 1         1          0.524     1         0.687      0.5      3
                 0         0          0         0         0          ?        4
                 0         0          0         0         0          ?        5
Weighted Avg.    0.524     0.524      0.274     0.524     0.36       0.5

=== Confusion Matrix ===

   a   b   c   d   e   f   <-- classified as
   0   0   0  69   0   0 |   a = 0
   0   0   0   0   0   0 |   b = 1
   0   0   0  42   0   0 |   c = 2
   0   0   0 122   0   0 |   d = 3
   0   0   0   0   0   0 |   e = 4
   0   0   0   0   0   0 |   f = 5
```

```
tatus
)K
```

6.Comparison between LibSVM and Naive Bayes

- Here we see that both provide very good accuracy for unprocessed text document.
- Naive Bayes is much simpler than SVM
- Both Naive Bayes and LibSVM are sensitive to parameter tuning
- Different parameters gives different results. for ex: changing the kernel function from Radial to Linear chabes the result drastically
- Also using n-grams have vastly changed the accuracy
- However SVM works well with n-grams as compared to Naive Bayes.
- Naive Bayes is more faster than SVM
- Both the models are sensitive to overlapping features.

8. Issues

- The main issue with the data set is overlapping features which is causing hierarchical classification of data.
- Noisy features which are causing over fitting of the model.

9. Future work for final report

- Using All vs all multiclass classification in SVM
- Reduce the noise in tokenized data by training the model using Tri gram as the tri gram will less likely induce the noise.
- Reduce the overlapping features
- Understand the relation between the features to further optimize the model