Analysing Wheat Seeds Dataset

Manish Deswal
Student ID: 19968562
Data Science
Western Sydney University

Abstract- This report analyses the wheat seeds dataset which contains seven different measurements of geometrical properties of wheat kernels belonging to three different varieties of wheat: Kama, Rosa and Canadian. Various supervised and unsupervised learning methods has been used in this report to predict the variety of wheat seeds based on these geometric parameters and their performance has been compared.

I. Introduction

Data science is the process of obtaining valuable insights from data by using various tools and techniques. Using these techniques, this paper aim to find the best suited model to fit our data and obtain meaningful analysis out of it. To find the best model, outcomes of various algorithms and techniques will be compared and evaluated against the "ground truth" with the aim of finding an effective technique for forming proper categories of wheat. Also, the main geometric features will be used to identify wheat varieties. The parameters most influential on the variety of wheat will be discovered.

II. Data Description

The dataset contains measurements of geometrical properties of kernels belonging to three different varieties of wheat obtained using soft X-ray technique [1]. The different varieties of wheat are Kama, Rosa and Canadian. The data set contains seven real-valued attributes, one categorical attribute and total of 210 observations and can be used for the tasks of classification and cluster analysis [2]. The description of attributes is stated in the table below:

Attribute	**Data Type**
Area, A	Real
Perimeter, P	Real
Compactness C = 4*pi*A/P^2,	Real
Length of kernel	Real
Width of kernel	Real
Asymmetry coefficient	Real
Length of kernel groove	Real
Class (1 = Kama, 2 = Rosa, 3= Canadian)	Categorical

Table 1

The first seven real attributes will be used as independent variables and will be used to predict the last categorical attribute i.e. Class.

III. Data Pre-Processing

Data pre-processing is an important step to prepare our data so that machine learning models can be built without any issues.

Missing Data

First step would be to deal with any missing data in the dataset. The common practise is to replace the missing value with the mean, median or the most frequent value for all numerical columns. For this dataset, the missing values will be replaced by mean value. The theoretical context of the mean substitution is that the mean is a rational approximation for a randomly chosen observation from a normal distribution.

Categorical Data

Since machine learning models are based on mathematical equations it would cause some problems if the categorical variables are kept in the equations as equations would only allow numbers. Also, the Class attribute values "1", "2" and "3" should not be considered numerical as they prosses a categorical meaning. Therefore, the categorical attribute "Class" is encoded using factors.

Feature Scaling

Many attributes being in different scales can cause issues with some machine learning algorithms which are based on Euclidean distance. As one attribute might dominate in the equations because of being in different scale. Therefore, all attributes are transformed to contain values in the same range for these machine learning algorithms.

	Area	Perimeter	Compactness	Length.of.kernel	Width.of.kernel	Asymmetry.coefficient	Length.of.kernel.groov	Class
4	13.84	13.94	0.8955	5.324	3.379	2.2590	4.805	1
5	16.14	14.99	0.9034	5.658	3.562	1.3550	5.175	1
8	14.11	14.10	0.8911	5.420	3.302	2.7000	5.000	1
11	15.26	14.85	0.8696	5.714	3.242	4.5430	5.314	1

Filter

	Area	Perimeter	Compactness	Length.of.kernel	Width.of.kernel	Asymmetry.coefficient	Length.of.kernel.groov	Class
4	-0.326698807	-0.42858608	1.080517240	-0.61530079	0.26699226	-0.81992550	-1.10768978	1
5	0.402822801	0.30713390	1.443791757	0.07860061	0.72109769	-1.39456093	-0.40729920	1
8	-0.241059314	-0.31647637	0.878187130	-0.41585608	0.07592057	-0.53960004	-0.73856502	1
11	0.123701490	0.20903790	-0.110471364	0.19494336	-0.07296646	0.63191886	-0.14417950	1

Fig 1 Attribute values before and after feature scaling.

Train/Test Split

To avoid over-fitting and to evaluate the model performance the observations are split into train/test sets, test set being 20% of the total observations.

IV. Analysis

Ranking Features by Importance

To visualize the results of different ML models in graphical plots we would need to select top 2 parameters that highly affect the output class for which we would be using a feature selection technique.
The importance of features can be estimated using a ROC curve analysis conducted for each attribute. Learning Vector Quantization (LVQ) model was used to rank the features by their importance with relation to the output class.

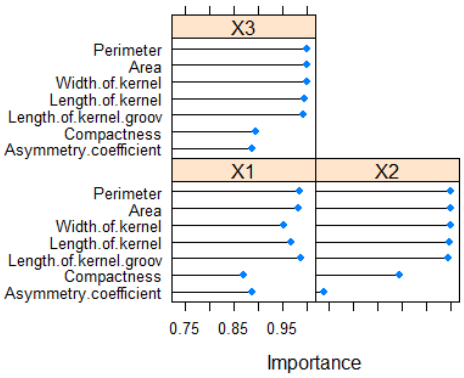


Fig 2 Ranking features by importance

The output shows that Perimeter and Area are the top 2 most important attributes in the dataset and the Asymmetry coefficient and Compactness attributes are the least important attributes across all classes of wheat seeds.

Supervised Learning Techniques used:

A. Support Vector Machines

SVM was originally developed in the 1960s, then in the 1990s they were refined again, and now they are becoming very common in machine learning because they demonstrate that they can be very efficient because they adopt a slightly different approach to other algorithms of machine learning.

A Support Vector Machine is a supervised algorithm for machine learning that can be used for problems with both classification and regression. It follows a technique called the data transformation kernel trick, and it seeks an optimal boundary between the possible outputs based on these transformations. Therefore, they're about finding the best line or the best decision boundary which will help us separate the space into classes.

To find if the dataset is linearly separable SVM classifiers with both linear kernel and radial kernel (also called the Gaussian function) are used and their accuracy on test data is compared.

The support vectors are plotted as crosses and the remaining observations are plotted as circles.

Linear kernel:

```
> summary(classifier)

Call:
svm(formula = Class ~ ., data = training_set, type = "C-classification", kernel = "linear",
    cost = 10, scale = FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  10

Number of Support Vectors:  24

 ( 10 6 8 )


Number of Classes:  3

Levels:
 1 2 3

> confusion_matrix
   y_pred
     1  2  3
  1 12  0  2
  2  0 14  0
  3  1  0 13
> accuracy
[1] 92.85714
```

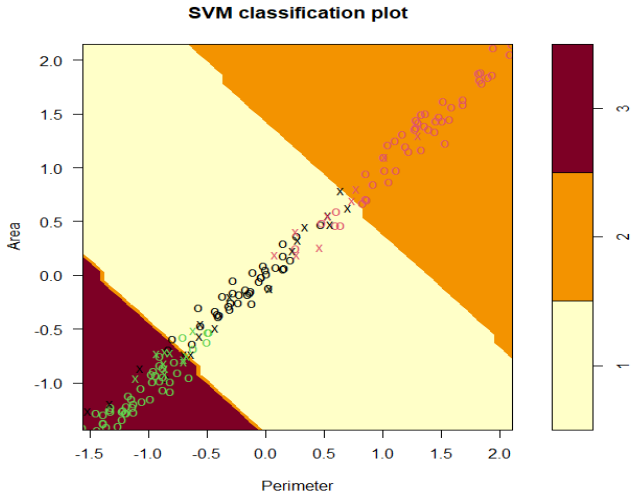


Fig 3 Linear SVM visualization of top 2 important features for training set

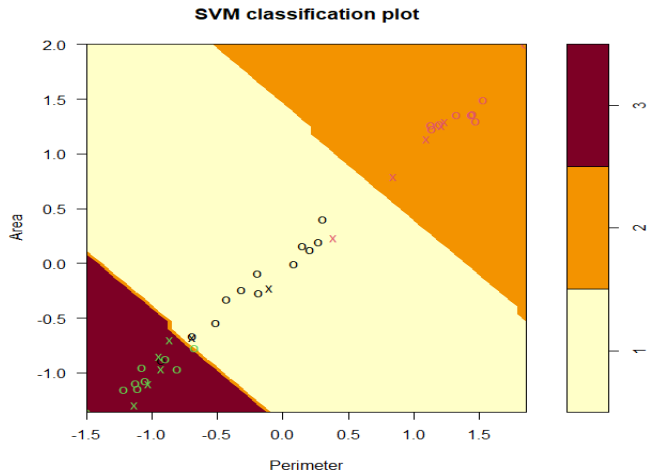


Fig 4 Linear SVM visualization of top 2 important features for test set

Figure 3 visualizes the top two features Area and perimeter and their relationship with each other with respect to this model th ough the classifier inputs all features to predict output. We can observe that linear SVM has fitted the training set separating th e 3 classes by linear separation.

Radial kernel:

```
> summary(classifier)

Call:
svm(formula = Class ~ ., data = training_set, type = "C-classification", kernel = "radial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  62

 ( 27 16 19 )


Number of Classes:  3

Levels:
 1 2 3

> confusion_matrix
    y_pred
      1  2  3
  1 13  0  1
  2  0 14  0
  3  1  0 13
> accuracy
[1] 95.2381
```

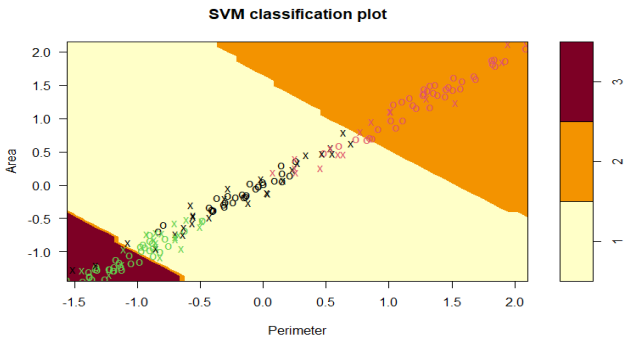


Fig 5 Radial SVM visualization of top 2 important features for traning set

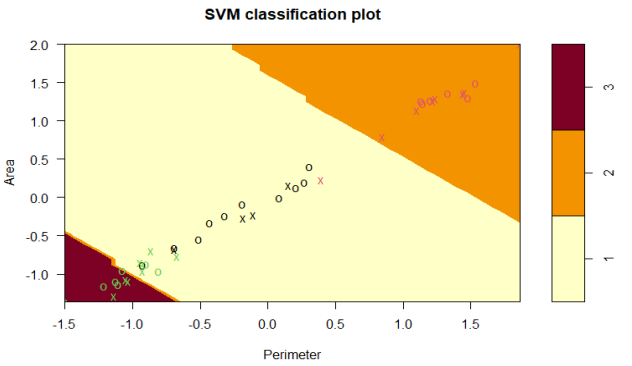


Fig 6 Radial SVM visualization of top 2 important features for test set

It was observed that accuracy was better on test set with radial kernel which is an indication that data might not be linearly separable but by visualizing the plot for Area and Perimeter we can see that radial kernel has only separated the classes linearly which might be the best approach for wheat seed dataset.

Model Evaluation using K-fold cross Validation

Judging the model performance only on accuracy on one test set is not very relevant as one can encounter different accuracy on different test sets. This is known as the variance problem. A more relevant way to evaluate the model performance would be k-fold cross validation which fixes the variance problem by spinning the training set into tenfold as shown in Appendix I. By training the model on nine folds and testing it on the last remaining fold. Hence, it results in training the model and test them on 10 different combinations.
A better performance metric would be taking an average of these 10 evaluated accuracies and computing a standard deviation.

For linear kernel:

```
> cv
$Fold01
[1] 100

$Fold02
[1] 100

$Fold03
[1] 100

$Fold04
[1] 88.23529

$Fold05
[1] 88.23529

$Fold06
[1] 87.5

$Fold07
[1] 88.88889

$Fold08
[1] 94.11765

$Fold09
[1] 94.44444

$Fold10
[1] 93.75

> mean_accuracy = mean(as.numeric(cv))
> mean_accuracy
[1] 93.51716
```

For radial kernel:

```
> cv
$Fold01
[1] 93.33333

$Fold02
[1] 100

$Fold03
```

```
[1] 82.35294

$Fold04
[1] 77.77778

$Fold05
[1] 94.11765

$Fold06
[1] 94.11765

$Fold07
[1] 88.23529

$Fold08
[1] 88.23529

$Fold09
[1] 100

$Fold10
[1] 93.75

> mean_accuracy
[1] 91.19199
```

It is observed that though performance of radial kernel was better than linear kernel on our test set, the mean accuracy obtained by k-fold validation suggests that linear SVM might be a better fit for wheat seed data.

Improving model performance using Grid Search

Machine learning models are composed of two types of parameters. The first type of parameters are the parameters that are learned through the machine learning algorithm and the second type of parameters are the parameters that are chosen at time of model creation called hyper parameters. Grid search helps to find the optimal values for these hyper parameters.

Linear kernel:

```
> classifier
Support Vector Machines with Linear Kernel


168 samples
  7 predictor
  3 classes: '1', '2', '3'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 168, 168, 168, 16
8, 168, 168, ...
Resampling results:

  Accuracy   Kappa
  0.9238827  0.8848199

Tuning parameter 'C' was held constant at
a value of 1

> classifier$bestTune
  C
1 1
```

C: The Penalty Parameter

Radial Kernel:

```
> classifier
Support Vector Machines with Radial Basis
Function Kernel

168 samples
  7 predictor
  3 classes: '1', '2', '3'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 168, 168, 168, 16
8, 168, 168, ...
Resampling results across tuning parameter
s:

  C     Accuracy   Kappa
  0.25  0.9149173  0.8709026
  0.50  0.9187390  0.8765782
  1.00  0.9251701  0.8863247

Tuning parameter 'sigma' was held constant
 at a value of 0.2407934
Accuracy was used to select the optimal mo
del using the largest value.
The final values used for the model were s
igma = 0.2407934 and C = 1.
```

Optimal values for hyper parameters

```
> classifier$bestTune
      sigma C
3 0.2407934 1
```

C: The Penalty Parameter

The best suited values for hyper parameters are same values ch osen at time of model creation. Hence, the above-mentioned ac curacies will be taken into consideration while comparing mod el performances.

B. Decision Trees

A decision tree is an algorithm of supervised machine learning primarily used for Regression and Classification. At the same time as a related decision tree is progressively formed, it breaks down a data set into smaller and smaller subsets. A tree with decision nodes and leaf nodes is the last outcome. Both categorical and numerical data can be treated by the decision tree. The algorithm of the decision tree is like a series of statements of if else and the next statement is based on the previous result.

The split is done in such a way to maximize the number of certain categories in each of these splits. The split is trying to minimize entropy.

Summary of the tree classifier:

```
> summary(classifier)

Classification tree:
```

```
tree(formula = Class ~ ., data = training_
set)
Variables actually used in tree constructi
on:
[1] "Length.of.kernel.groov" "Area"
                "Asymmetry.coefficient"  "Widt
h.of.kernel"
Number of terminal nodes:  8
Residual mean deviance:  0.1296 = 20.73 /
160
Misclassification error rate: 0.02976 = 5
/ 168
```

The misclassification rate being very low 2% corresponds to model performing well.

Confusion matrix and accuracy results on dataset with decision tree model:

```
> cm
   y_pred
     1  2  3
  1 13  0  1
  2  0 14  0
  3  1  0 13
> accuracy
[1] 95.2381
> misclassification_error_rate
[1] 4.761905
```

Plotting the decision Tree:

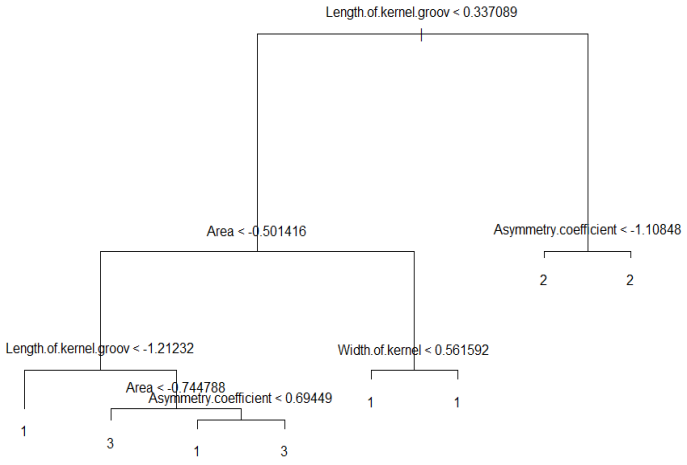


Fig 7 Decision Tree for wheat seed data

Model Evaluation using K-fold cross Validation

```
> cv
$Fold01
[1] 88.09524

$Fold02
[1] 88.09524

$Fold03
[1] 88.09524

$Fold04
[1] 88.09524

$Fold05
[1] 88.09524

$Fold06
[1] 88.09524

$Fold07
[1] 88.09524

$Fold08
[1] 88.09524

$Fold09
[1] 88.09524

$Fold10
[1] 88.09524

> mean_accuracy
[1] 88.09524
```

Cross validation plot:

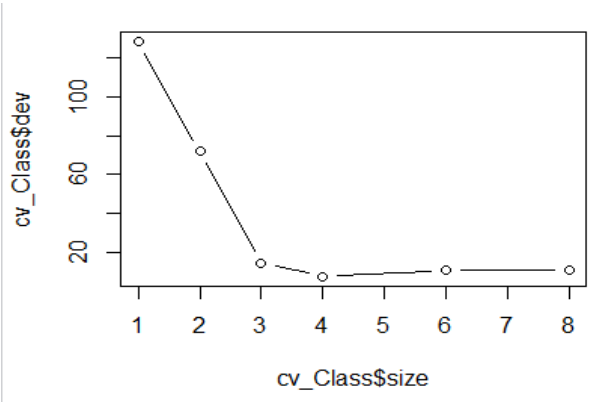


Fig 8 Cross Validation

Pruning the Tree

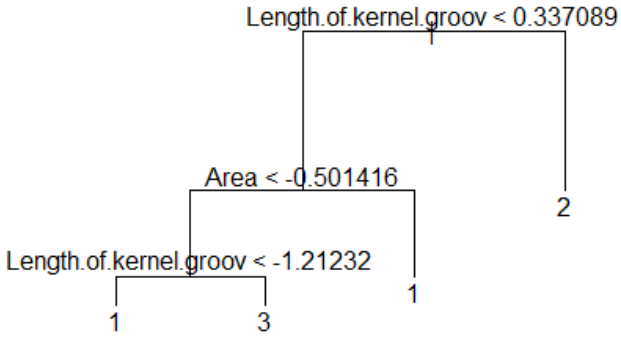


Fig 9 Pruned Tree

Calculating performance after pruning:

```
> new_cm
   tree_predict
     1  2  3
  1 10  0  4
  2  0 14  0
  3  1  0 13
```

```
> new_accuracy
[1] 88.09524
> new_misclassification_error_rate
[1] 4.761905
```

We can observe that while the model performed very well on our dataset with 95.2% accuracy, mean accuracy calculated using k fold cross validation i.e. 88.09 % is a more realist value.
Through the decision tree we can observe that Length of kernel groove is the most important predictor in determining the Class of kernel seeds which alone contributes to diving class 2 from class 1 and 3.
By pruning the decision Tree, the category is being determined in just 3 decisions. Class 1 wheat seeds have the smallest Length of kernel grooves <-1.2 and Class 2 having the highest >0.33. Area of seeds belonging to class 1 is higher than area of seeds belonging to class 3.

Unsupervised Learning Techniques used:

C. *Principal Component Analysis (PCA)*

PCA is a very popular unsupervised dimensionality reduction algorithm for feature extraction. PCA is used for operations such as visualization, feature extraction, noise filtering and can be seen in algorithms used for stock market predictions and gene analysis just to name a few. The goal of PCA is to identify and detect the correlation between variables. If there is a strong correlation and it's found, then it could reduce the dimensionality [3]. I can find the directions of maximum variance in high dimensional data and then project it into a smaller dimensional subspace while retaining most of the information. The goal is to reduce the dimensions of a D dimensional dataset by projecting onto a k dimensional subspace where k is less than D.

The main functions of the PCA algorithm would be to standardize the data ,obtain the eigen vectors and eigen values, to then sort the eigen values in descending order, construct the projection matrix W from the selected K eigen vectors and to transform the original dataset. Rather than attempting to predict
the values PCA is attempting to learn about the relationship between x and y values, quantified by finding a list of principal axes. One disadvantage is that it is highly affected by outliers in the data.

For in given matrix of features has m independent variables PCA will extract (p < = m) a smaller number of independent variables which are new independent variables. These new independent variables extracted explain the most the variance of your data set that is regardless of your dependent variable. PCA is considered an unsupervised mode in the sense that it doesn't consider the dependent variable in the model.

Applying PCA to our dataset and extracting 2 variables.

```
> pca$sdev # standard deviation of the principal components
[1] 3.285319911 1.459265266 0.271348545 0.113523102 0.052423532 0.039628901 0.005445681
> pca$center # mean of the original variables
                Area          Perimeter        Compactness       Length.of.kernel
          14.8475238           14.559285            0.8709986           5.6285333
 Width.of.kernel  Asymmetry.coefficient Length.of.kernel.groov
           3.2586048           3.7002010           5.4080714

> pca$scale # standard deviation of the original variables
[1] FALSE
```

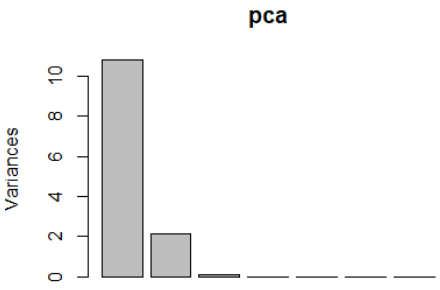


Fig 10 Screen plot of pca object

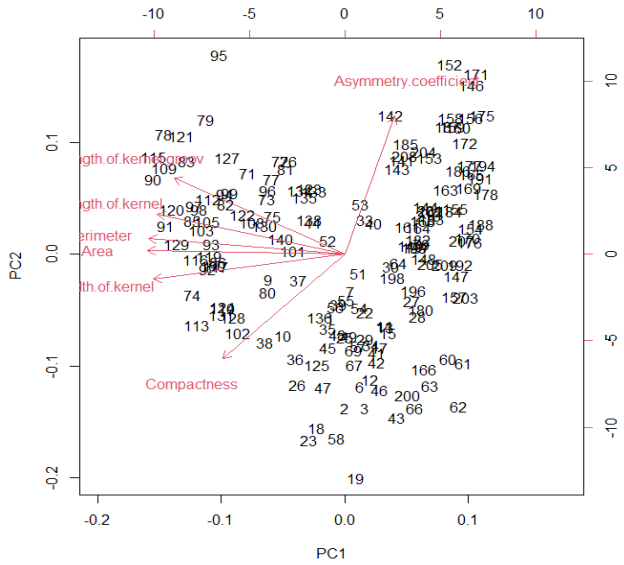


Fig 11 Biplot of pca object

We can observe that Length of kernel, Perimeter, Area and Width of kernel and length of kernel groove are closely related to each other while Asymmetry coefficient is least related with any other parameter.

New features extracted from existing features:

	PC1	PC2	Class
1	-0.3216784075	-0.81756507	1
2	0.0143054913	-1.95046041	1
3	0.4840666268	-1.94484474	1
6	0.3554100078	-1.67171421	1
7	0.1483922120	-0.46889230	1
9	-1.7915841099	-0.32144174	1
10	-1.4537401582	-1.03425963	1
12	0.5902594048	-1.59041261	1
13	0.9485844123	-0.91865773	1
14	0.9295566872	-0.91208952	1

Fig 12 variables extracted from PCA

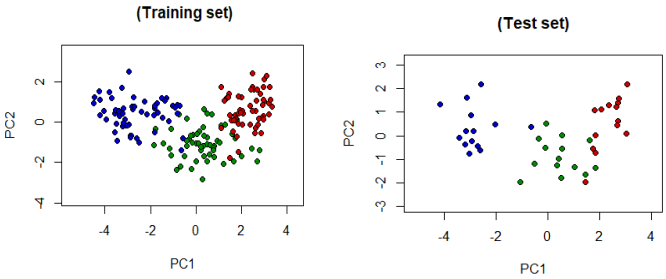


Fig 13 Derived PCA variables for all observations belonging to 3 different classes.

Using svm model with linear kernel to train the new training set and classify the new test set composed of extracted PCA variables.

```
> cm
   y_pred
      1  2  3
  1 13  0  1
  2  0 14  0
  3  2  0 12
> accuracy
[1] 92.85714
```

Comparing the results before and after PCA, the accuracy is retained as 92.85% even after reducing the dimensionality to 2 variables with PCA.

Visualizing the results:

Visualizing the results of training set to observe how well the model has fitted into out dataset, the prediction regions and the prediction boundary. We can observe that these points correspond to the new variables PC1 and PC2.

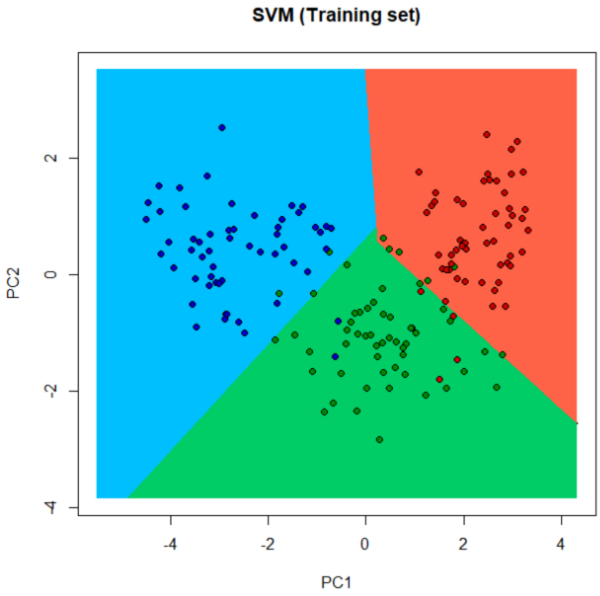


Fig 14 Visualization of results on training set with linear PCA

Here class 1, 2 ,3 represent colour green, blue and red respectively. We can observe that linear SVM has properly fitted the training set though the decision boundary has some outliers.

Visualization of the test set to observe the predicted results:

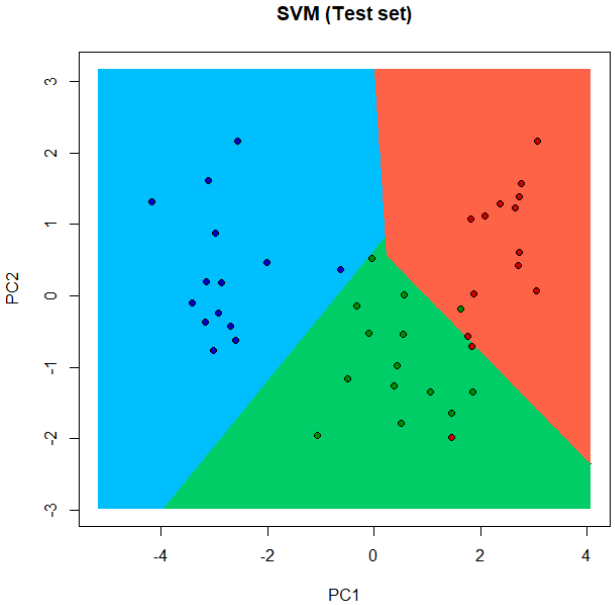


Fig 15 Visualization of results on test set with linear PCA

All observations are correctly predicted expect 3 observations which can be clearly observed in the graph which makes up for the 92.85% accuracy on the test set.

Kernel PCA

Kernel PCA is a version of PCA where the data is mapped to a higher dimension using the kernel trick. By extracting new

principal components, it manages to deal with nonlinear problems i.e. when data is not linearly separable.

Applying Kernel PCA to our dataset and extracting 2 variables:

	V1	V2	Class
1	0.9898089	-6.372318288	1
2	-0.2211292	-7.485969858	1
3	-1.5403493	-7.142923425	1
6	-1.4735604	-7.755338476	1
7	-1.0837237	-5.792602952	1
9	6.7382109	-1.742137242	1
10	5.3578032	-4.688064267	1
12	-2.2814391	-7.317883376	1
13	-3.5262563	-5.069712426	1
14	-3.9211985	-5.666558175	1

Fig 16 New variables extracted from Kernel PCA

Using svm model with radial kernel to train the new training set and classify the new test set respectively composed of extracted Kernel PCA variables.

```
> cm
   y_pred
     1  2  3
  1 13  0  1
  2  0 14  0
  3  3  0 11
> accuracy
[1] 90.47619
```

As we observed before our dataset performs better in linear models therefore, the accuracy is better with linear PCA than Kernel PCA. The linear PCA worked towards creating new variables which are closer to a linearly separable dataset which reflects in linear SVM performing well with the new variables. While kernel PCA resulted in decreasing our accuracy since our dataset works better with linear kernels as observed before.

Visualizing the results:

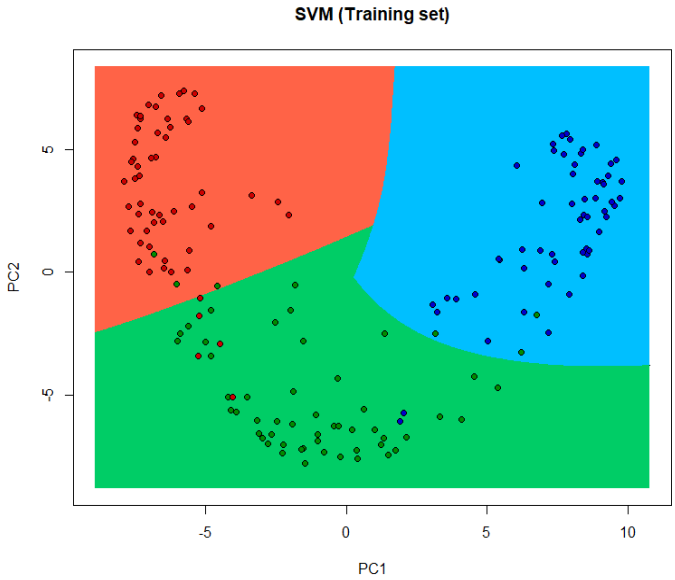


Fig 17 Visualization of results on test set with Kernel PCA

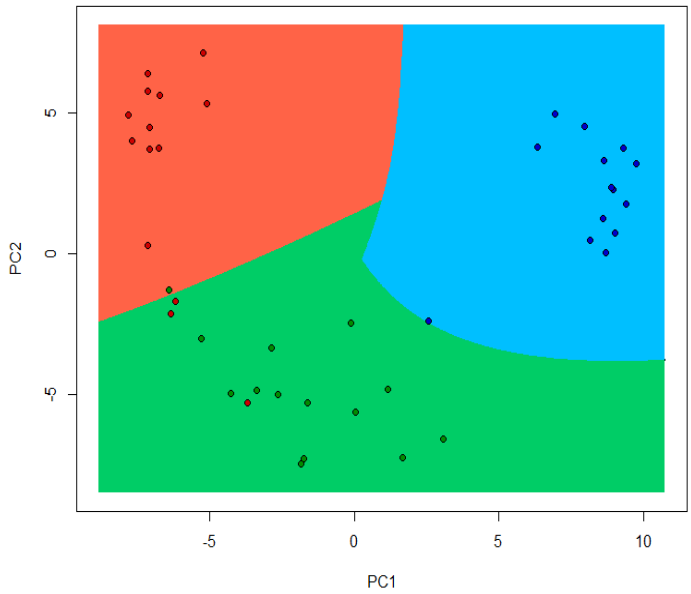


Fig 18 Visualization of results on test set with Kernel PCA

The kernel SVM trying to fit into the new kernel PCA variables with a linear decision boundary in separating class 3 with class 1 while for separating class 1 with class 2 the decision boundary is curved. Though our dataset is linearly separable and works better with linear classifiers Kernel PCA has worked towards creating variables which work well with a kernel approach with a 90.47 % accuracy.

V. Model Comparison

The linear SVM model worked best with our dataset with 92.8 % accuracy for our test set and 93.1 % mean accuracy. Linear PCA corresponded to a better dimensionality reduction technique with retaining the accuracy with linear SVM and reducing the dimensionality to 2 independent variables.

VI. Conclusion:

The proposed SVM algorithm, based on linear kernel, is expected to be an effective technique for wheat variety recognition. The data seems to perform better on linear algorithms than nonlinear algorithms hence data can be almost linearly separable. The data reduced to 2 parameters after applying the Principal Component Analysis, contained apparent clustering structures according to their classes. The amount of 39 kernels, 92.8% of the total of test set, were classified properly. Differences in their primary geometric parameters were shown by the wheat varieties used in the paper. The Rosa variety is better recognized whereas Kama variety and Canadian variety are less successfully predicted. Asymmetry coefficient and Compactness are least contributing across all classes and least related to other parameters.

VII References:

[1] M. Charytanowicz, J. Niewczas, P. Kulczycki, P.A. Kowalski, S. Lukasik, S. Zak, 'A Complete Gradient Clustering Algorithm for Features Analysis of X-ray Images', in: Information Technologies in Biomedicine, Ewa Pietka, Jacek Kawa (eds.), Springer-Verlag, Berlin-Heidelberg, 2010, pp. 15-24.

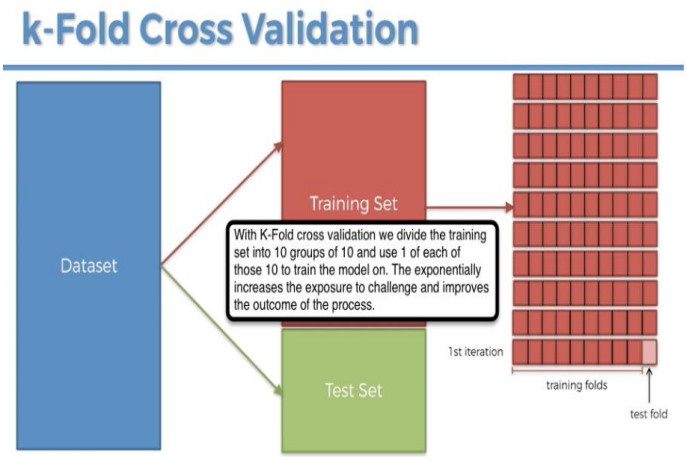
[2] Charytanowicz, Małgorzata & Niewczas, Jerzy & Kulczycki, Piotr & Kowalski, Piotr A. & Łukasik, Szymon & Żak, Sławomir. (2012). Seeds dataset.

[3] Bro, Rasmus & Smilde, Age. (2014). Principal component analysis. Analytical methods.

VIII Appendix:

I.

Source: https://rpubs.com/markloessi/506713



II.

R CODE:

Data Pre-processing

```
# Importing the dataset
dataset = read.csv('seeds.csv', fileEncoding="UTF-8-BOM")

# Taking care of missing data
dataset$Area = ifelse(is.na(dataset$Area),
                      ave(dataset$Area, FUN = function(x) mean(x,
na.rm = TRUE)),
                      dataset$Area)
dataset$Perimeter = ifelse(is.na(dataset$Perimeter),
                    ave(dataset$Perimeter, FUN = function(x)
mean(x, na.rm = TRUE)),
                    dataset$Perimeter)
dataset$Compactness = ifelse(is.na(dataset$Compactness),
                      ave(dataset$Compactness, FUN = function(x)
mean(x, na.rm = TRUE)),
                      dataset$Compactness)
dataset$Length.of.kernel =
ifelse(is.na(dataset$Length.of.kernel),
                        ave(dataset$Length.of.kernel, FUN =
function(x) mean(x, na.rm = TRUE)),
                        dataset$Length.of.kernel)
dataset$Width.of.kernel =
ifelse(is.na(dataset$Width.of.kernel),
                         ave(dataset$Width.of.kernel, FUN =
function(x) mean(x, na.rm = TRUE)),
                         dataset$Width.of.kernel)
dataset$Asymmetry.coefficient =
ifelse(is.na(dataset$Asymmetry.coefficient),
                        ave(dataset$Asymmetry.coefficient, FUN
= function(x) mean(x, na.rm = TRUE)),
                        dataset$Asymmetry.coefficient)
dataset$Length.of.kernel.groov =
ifelse(is.na(dataset$Length.of.kernel.groov),
                           ave(dataset$Length.of.kernel.groov,
FUN = function(x) mean(x, na.rm = TRUE)),
                           dataset$Length.of.kernel.groov)

# Encoding categorical data
dataset$Class = factor(dataset$Class,
                       levels = c('1', '2','3'), # convert categories to
numbers
                       labels = c('1', '2','3'))

# Splitting the dataset into the Training set and Test set
# install.packages('caTools')
library(caTools)
set.seed(123)
split = sample.split(dataset$Class, SplitRatio = 0.8)
#dependent variable
training_set = subset(dataset, split == TRUE)
test_set = subset(dataset, split == FALSE)

# Feature Scaling
training_set[, 1:7] = scale(training_set[, 1:7]) #as 2 cloumns
are non numberials but factorial
test_set[, 1:7] = scale(test_set[, 1:7])
```

Feature Importance

```
#Feature Selection top 2
library(mlbench)
# prepare training scheme
control <- trainControl(method="repeatedcv", number=10,
repeats=3)
# train the model
model <- train(Class~., data=dataset, method="lvq",
preProcess="scale", trControl=control)
# estimate variable importance
importance <- varImp(model, scale=FALSE)
# summarize importance
print(importance)
# plot importance
plot(importance)
```

Linear SVM

```
# Fitting SVM to the Training set
library(e1071)
classifier = svm(formula = Class ~ .,
                 data = training_set,
                 type = 'C-classification',
                 kernel = 'linear')
summary(classifier)
# Predicting the Test set results
y_pred = predict(classifier, newdata = test_set[-8])
# Making the Confusion Matrix
cm = table(test_set[, 8], y_pred)
confusion_matrix = cm
n = sum(cm) # number of instances
nc = nrow(cm) # number of classes
diag = diag(cm) # number of correctly classified instances per
class
accuracy = sum(diag) / n *100
#visluzaing traning set
plot(classifier,training_set,Area~Perimeter)
#visluzaing test set
plot(classifier,test_set,Area~Perimeter)


# Applying k-Fold Cross Validation for linear kernel SVM
# install.packages('caret')
library(caret)
folds = createFolds(training_set$Class, k = 10)
cv = lapply(folds, function(x) {
  training_fold = training_set[-x, ]
  test_fold = training_set[x, ]
  classifier = svm(formula = Class ~ .,
                   data = training_fold,
                   type = 'C-classification',
                   kernel = 'linear')
  y_pred = predict(classifier, newdata = test_fold[-8])
  cm = table(test_fold[, 8], y_pred)
  n = sum(cm) # number of instances
  diag = diag(cm) # number of correctly classified instances per
class
  accuracy = sum(diag) / n *100
  return(accuracy)
})
mean_accuracy = mean(as.numeric(cv))

# Applying Grid Search to find the best parameters for Linear
SVM
# install.packages('caret')
library(caret)
classifier = train(form = Class ~ ., data = training_set, method
= 'svmLinear')
classifier
classifier$bestTune
```

Radial Kernel SVM

```
classifier = svm(formula = Class ~ .,
                 data = training_set,
                 type = 'C-classification',
                 kernel = 'radial')# Predicting the Test set results
summary(classifier)
y_pred = predict(classifier, newdata = test_set[-8])

# Making the Confusion Matrix
cm = table(test_set[, 8], y_pred)
confusion_matrix = cm
n = sum(cm) # number of instances
nc = nrow(cm) # number of classes
diag = diag(cm) # number of correctly classified instances per
class
accuracy = sum(diag) / n *100
#visluzaing traning set
plot(classifier,training_set,Area~Perimeter)
#visluzaing test set
plot(classifier,test_set,Area~Perimeter)

# Applying k-Fold Cross Validation
# install.packages('caret')
library(caret)
folds = createFolds(training_set$Class, k = 10)
cv = lapply(folds, function(x) {
  training_fold = training_set[-x, ]
  test_fold = training_set[x, ]
  classifier = svm(formula = Class ~ .,
                   data = training_fold,
                   type = 'C-classification',
                   kernel = 'radial')
  y_pred = predict(classifier, newdata = test_fold[-8])
  cm = table(test_fold[, 8], y_pred)
  n = sum(cm) # number of instances
  diag = diag(cm) # number of correctly classified instances per
class
  accuracy = sum(diag) / n *100
  return(accuracy)
})
```

```
mean_accuracy = mean(as.numeric(cv))
```

Decision Trees

```
# Fitting Decision Tree Classification to the Training set
# install.packages('rpart')
library(ISLR)
library(tree)
classifier = tree(formula = Class ~ .,
                  data = training_set)

# Plotting the tree #without feature scaling
plot(classifier, training_set,Area~Perimeter)
text(classifier,pretty = 0 )

# check how model is doing
summary(classifier)

# Predicting the Test set results
y_pred = predict(classifier, newdata = test_set[-8], type =
'class')

# Making the Confusion Matrix
cm = table(test_set[, 8], y_pred)
n = sum(cm) # number of instances
nc = nrow(cm) # number of classes
diag = diag(cm) # number of correctly classified instances per
class
accuracy = sum(diag) / n * 100
misclassification_error_rate = mean(y_pred != test_set[,
8])*100



# Applying k-Fold Cross Validation
# install.packages('caret')
library(caret)
folds = createFolds(training_set$Class, k = 10)
cv = lapply(folds, function(x) {
  training_fold = training_set[-x, ]
  test_fold = training_set[x, ]
  classifier = rpart(formula = Class ~ .,
                     data = training_set)
  y_pred = predict(classifier, newdata = test_set[-8], type =
'class')
  cm = table(test_set[, 8], y_pred)
  n = sum(cm) # number of instances
  diag = diag(cm) # number of correctly classified instances per
class
  accuracy = sum(diag) / n *100
  return(accuracy)
})
mean_accuracy = mean(as.numeric(cv))

# cross - validation plot
set.seed(3)
cv_Class<-cv.tree(classifier, FUN = prune.misclass)
names(cv_Class)
plot(cv_Class$size, cv_Class$dev, type = "b")

#pruing the tree
pruned_class=prune.misclass(classifier,best=4)
plot(pruned_class)
text(pruned_class,pretty=0)

tree_predict<-predict(pruned_class, test_set, type="class")
new_cm = table(test_set[, 8], tree_predict)
n = sum(cm) # number of instances
nc = nrow(cm) # number of classes
diag = diag(cm) # number of correctly classified instances per
class
new_accuracy = sum(diag) / n * 100
new_misclassification_error_rate = mean(y_pred != test_set[,
8])*100
```

Linear Principal Component Analysis

```
# Applying PCA
# install.packages('caret')
library(caret)
# install.packages('e1071')
library(e1071)
#bilot
pca = prcomp(training_set[,1:7],rank = 2) # perform PCA
screeplot(pca)
biplot(pca)
pca$sdev # standard deviation of the principal components
pca$center # mean of the original variables
pca$scale # standard deviation of the original variables
#interpret the results

training_set_pca = as.data.frame(predict(pca, training_set))
training_set_pca$Class = training_set$Class
test_set_pca = as.data.frame(predict(pca, test_set))
test_set_pca$Class = test_set$Class

#SVM linear
# Fitting SVM to the Training set
library(e1071)
classifier = svm(formula = Class ~ .,
                 data = training_set_pca,
                 type = 'C-classification',
                 kernel = 'linear')

# Predicting the Test set results
y_pred = predict(classifier, newdata = test_set_pca[-3])

# Making the Confusion Matrix
```

```
cm = table(test_set_pca[, 3], y_pred)
n = sum(cm) # number of instances
nc = nrow(cm) # number of classes
diag = diag(cm) # number of correctly classified instances per class
accuracy = sum(diag) / n * 100



# Visualising the Training set results
library(ElemStatLearn)
set = training_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('PC1', 'PC2')
y_grid = predict(classifier, newdata = grid_set)
plot(set[, -3],
     main = ' (Training set)',
     xlab = 'PC1', ylab = 'PC2',
     xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 2, 'deepskyblue', ifelse(y_grid == 1, 'springgreen3', 'tomato')))
points(set, pch = 21, bg = ifelse(set[, 3] == 2, 'blue3', ifelse(set[, 3] == 1, 'green4', 'red3')))

# Visualising the Test set results
library(ElemStatLearn)
set = test_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('PC1', 'PC2')
y_grid = predict(classifier, newdata = grid_set)
plot(set[, -3], main = '(Test set)',
     xlab = 'PC1', ylab = 'PC2',
     xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 2, 'deepskyblue', ifelse(y_grid == 1, 'springgreen3', 'tomato')))
points(set, pch = 21, bg = ifelse(set[, 3] == 2, 'blue3', ifelse(set[, 3] == 1, 'green4', 'red3')))
```

Kernel Principal Component Analysis

```
# Applying Kernel PCA
# install.packages('kernlab')
library(kernlab)
kpca = kpca(~., data = training_set[-8], kernel = 'rbfdot', features = 2)
training_set_pca = as.data.frame(predict(kpca, training_set))
training_set_pca$Class = training_set$Class
test_set_pca = as.data.frame(predict(kpca, test_set))
test_set_pca$Class = test_set$Class

#SVM linear
# Fitting SVM to the Training set
library(e1071)
classifier = svm(formula = Class ~ .,
                 data = training_set_pca,
                 type = 'C-classification',
                 kernel = 'radial')

# Predicting the Test set results
y_pred = predict(classifier, newdata = test_set_pca[-3])

# Making the Confusion Matrix
cm = table(test_set_pca[, 3], y_pred)
n = sum(cm) # number of instances
nc = nrow(cm) # number of classes
diag = diag(cm) # number of correctly classified instances per class
accuracy = sum(diag) / n * 100



# Visualising the Training set results
library(ElemStatLearn)
set = training_set_pca
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('V1', 'V2')
y_grid = predict(classifier, newdata = grid_set)
plot(set[, -3],
     main = 'SVM (Training set)',
     xlab = 'PC1', ylab = 'PC2',
     xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 2, 'deepskyblue', ifelse(y_grid == 1, 'springgreen3', 'tomato')))
points(set, pch = 21, bg = ifelse(set[, 3] == 2, 'blue3', ifelse(set[, 3] == 1, 'green4', 'red3')))

# Visualising the Test set results
library(ElemStatLearn)
set = test_set_pca
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('V1', 'V2')
y_grid = predict(classifier, newdata = grid_set)
plot(set[, -3], main = 'SVM (Test set)',
     xlab = 'PC1', ylab = 'PC2',
```

```
    xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1),
length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 2,
'deepskyblue', ifelse(y_grid == 1, 'springgreen3', 'tomato')))
points(set, pch = 21, bg = ifelse(set[, 3] == 2, 'blue3',
ifelse(set[, 3] == 1, 'green4', 'red3')))
```