Nathan Tran
Dunjiang Zhang
Ryan Schultz
Dylan Cao
Muhaimin Badar

MATH 4323 Project: Final Report

Instructor: Dr. Wenshuang Wang

Introduction

The primary inspiration of this dataset comes from our fascination with mainstream technology prices that have been changing over the decades, with a specific interest in how added features affect the outcome price. The main interest in this paper is to discover what type of relationships exist between the different factors with each component. This way, we can uncover what is truly going on with the justification for its sale prices. With more open information it could become easier for customers to make informed decisions about what they are buying. Whether or not they need to spend a specific amount for the components and attributes they want. With more information about how phones are priced, it can also be used to see if a company has overpriced their merchandise.

Our dataset comprises of 500 observations and 21 variables, with the aim of accurately predicting the price range (Y) based on predictors (X) such as battery power, bluetooth availability, clock speeds (processing speed), dual sim availability, front camera megapixels count, four-g availability, internal memory, phone depth, phone weight, number of cores, pc, pixel height, pixel width, ram, screen height, screen width, battery life time, three-g availability, touch screen availability, and wifi availability. The price ranges that we assign phones are low (0), medium (1), high (2), and very high (3). We believe that understanding these relationships is crucial for both consumers and businesses, as it facilitates informed decision-making regarding mobile phone purchases and allows them to identify the features typically associated with a specific price range. This, in turn, promotes healthier market competition, elevates consumer satisfaction, and ultimately spurs innovation in the mobile phone industry, resulting in better deals tailored to various price brackets. As mobile phones have grown increasingly complex with numerous features, our supervised learning models can provide instantaneous insight into the appropriate price range for a given set of features, saving consumers not only time but also the effort required to manually evaluate these factors. This would also create more room for transparency among tech corporations competing in this space so that more honest marketing practices will be more relevant. Overall, the question we aim to answer in our research is “How much should a phone cost given its technological specifications?”

Methodology

The method we are using for our dataset will be two supervised learning models: k-nearest neighbors and support vector machines. Both are good models used for classification which will

be the goal of our project to create the best model for classifying a phone's price.
For our method A we will perform a KNN function on the dataset. KNN is a non-parametric model that is simple and easily interpretable for inference, but the predictions may not be accurate since the linear algorithm is based on the closest training samples. Both models are non-linear classifiers but KNN is more likely to be less accurate since it is sensitive to irrelevant features.

The K-nearest-neighbors (KNN) classifier works by calculating a pairwise distance metric for all observations to a given observation x. We then use our defined K value to look at the nearest K observations to x based on that distance metric. For our experiment we opted to use Euclidean distance which can be thought of as the length of a line in p dimensions. We call these K nearest values the neighborhood of points N. We then go on to calculate the conditional probability of x belonging to any class j for all classes. We determine this probability by calculating the fraction of points in our neighborhood N belonging to that class over the size of our neighborhood, which is K. We choose the class for x which we have the highest conditional probability in the neighborhood. This class j value would be our predicted value by the KNN algorithm for x.

This can be summarized by the equation: $argmax_j\{1/k\ \mathbf{\Sigma}_{i \epsilon N} I(y_i = j)\}$

For our method B we will perform a SVM function on the dataset. SVM is a non-linear approach and can give more accurate predictions since it is highly flexible but this makes it harder to interpret the results.

The Support Vector Machine (SVM) classifier works by calculating an optimal separating hyperplane with a defined soft margin that is determined by the hyperparameter C, the cost budget, which is the amount of errors the SVM is willing to accept. Due to the nature of most data being non-separable, we often use kernels alongside the SVM which allow us to map our original feature space in a higher dimension with relatively good performance. The kernels can expand the feature space in many different ways such as linear, polynomial, and radial. The kernels work by calculating an inner product . In a binary problem, we assign x the class for which it is on the side of the boundary. To calculate a multi-class prediction like our problem, most SVM implementations use a One-vs-One approach, we fit C(K,2) pairwise SVM classifiers and then assign it to the class for x which it was most frequently present.

The kernels can be summarized by the equations below:
Linear: $K(x_i, x_j) = \mathbf{\Sigma}^p_{k=1} x_{ik} x_{ip}$
Polynomial: $K(x_i, x_j) = (1 + \langle x_i, x_j \rangle)^d$
Radial: $K(x_i, x_j) = exp(-\gamma\ \mathbf{\Sigma}^p_{k=1} (x_{ik} - x_{jk})^2),\ \gamma > 0$

Both methods will need to do cross-validation checks on different models to find the best performing model for the training and testing datasets, and also be checked for overfitting.

Data Analysis

Firstly, it is important to consider which predictors (also known as features or predictors) to include in the model. For SVM, it is important to select predictors that are relevant to the problem at hand and that can help distinguish between different classes. Highly correlated predictors may not provide much additional information and can introduce multicollinearity, which can negatively impact the model's performance. Thus, it may be necessary to exclude some predictors. Similarly, for KNN, it's important to select relevant predictors and exclude highly correlated predictors, as mentioned earlier.

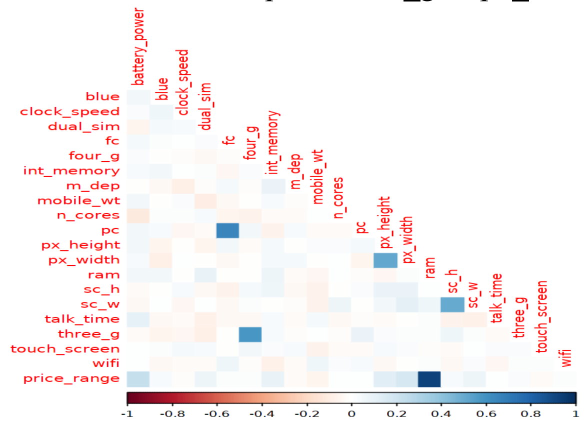
Next, it is important to consider scaling the data for both models. SVM is sensitive to the scale of the features, and if the features are not scaled, some features may dominate others in the calculation of distances. This can result in poor performance. Similarly, for KNN, as it is a distance-based algorithm, scaling can improve its performance.

At the beginning, we randomly subdivide your full data set in 80% for training, and 20% for testing, then proceed to train our model on the training data, and then record its prediction error on left out testing data.

```r
df <- MobilePrice
set.seed(1)
train <- sample(nrow(df), 0.8*nrow(df))
X.train <- df[train,]
X.train$price_range <- NULL
y.train <- as.factor(df[train, "price_range"])
X.test <- df[-train,]
X.test$price_range <- NULL
```

1. KNN (Dunjiang, Muhaimin)

We only scaled the numeric values since most of our features are categorical. We chose to eliminate certain features based on the correlation and distribution analysis. The features we chose to remove were “pc”, “three_g”, “px_width” and “sc_h”



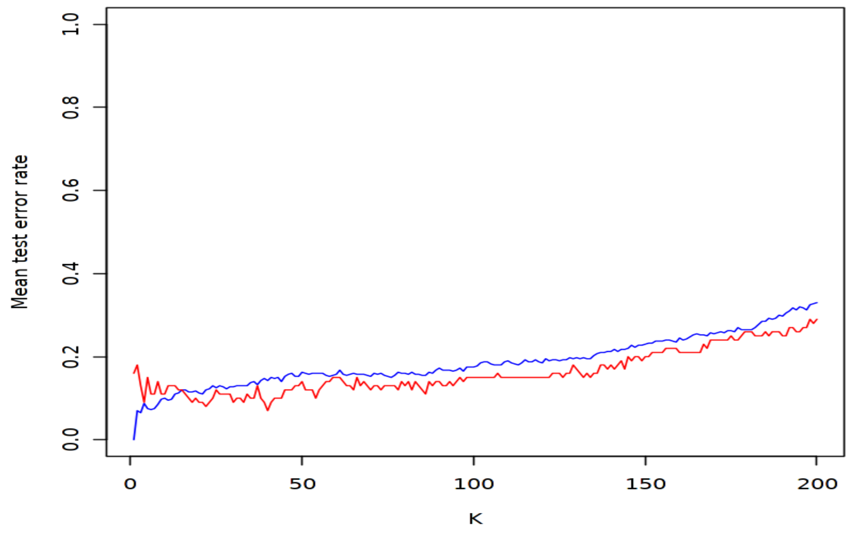
```
# scale numerics columns only
j <- c("clock_speed", "m_dep")

trainS <- scale(X.train[j])
X.train[j] <- trainS
X.test[j] <- scale(X.test[j],
    center = attr(trainS, "scaled:center"),
    scale = attr(trainS, "scaled:scale"))

features <- !names(X.train) %in% c("pc", "three_g", "px_width", "sc_h")
X.train.new <- X.train[, features]
X.test.new <- X.test[, features]
```

For KNN, we need to determine the optimal value of K, which is the number of nearest neighbors to consider when making a prediction. As mentioned earlier, if K is too small, the model will be overly sensitive to noise and outliers, while if K is too large, the model will be overly generalized and may miss important patterns in the data. To select the optimal value of K for KNN, we can use a technique called cross-validation. This involves splitting the data into training and testing sets multiple times and evaluating the model's performance with different values of K. The value of K that produces the highest accuracy (or other performance metric) on the testing set is selected as the optimal value. Using a for loop to iterate K value from 1 to 200, then find the best K value based on the lowest test error rate.

K: 40 Train err: 0.1425 Test err: 0.0700



```
knn.pred <- knn(train=X.train.new, test=X.test.new, cl=y.train, k=40)
knn.test.err <- mean(knn.pred != y.test)

[1] 0.07
```

Confusion matrix for KNN:

		knn.pred			
		0	1	2	3
Y.test	0	22	1	0	0
	1	1	25	2	0
	2	0	0	23	2
	3	0	0	1	23

Apply the model to the full set of data:

```
knn.pred <- knn(train=df[,-21], test=df[,-21], cl=y, k=nm.bestK)
knn.test.err <- mean(knn.pred != y)
table(knn.pred, y)
print(knn.test.err)

[1] 0.08
```

Confusion matrix by applying to full set of data:

		knn.pred			
		0	1	2	3
Y	0	111	3	0	0
	1	1	113	11	0
	2	0	6	108	13
	3	0	0	5	128

2. SVM (Ryan, Nathan, Dylan)

To select the variables with the most important impact on the outcome we used the lm() function to find variables that have the highest correlation with our classifier. This showed that out of the 20 variables correlated to the price_range, only six had a high correlation with the response variable. We then took that information and created a smaller dataframe to work with, with only the highly correlated variables.

```
summary(lm(o.data$price_range~.,data=o.data))

Call:
lm(formula = o.data$price_range ~ ., data = o.data)

Residuals:
     Min       1Q   Median       3Q      Max 
-1.04345 -0.24705  0.00349  0.24601  0.81279 

Coefficients:
                Estimate Std. Error t value Pr(>|t|)    
(Intercept)   -1.575e+00  6.164e-02 -25.553  < 2e-16 ***
battery_power  5.096e-04  1.640e-05  31.071  < 2e-16 ***
blue          -2.032e-03  1.442e-02  -0.141   0.8879    
clock_speed   -1.206e-02  8.814e-03  -1.368   0.1713    
dual_sim      -2.371e-02  1.442e-02  -1.644   0.1004    
fc             9.348e-04  2.166e-03   0.432   0.6660    
four_g        -1.477e-03  1.774e-02  -0.083   0.9337    
int_memory     8.647e-04  3.970e-04   2.178   0.0295 *  
m_dep         -9.963e-03  2.494e-02  -0.399   0.6896    
mobile_wt     -8.801e-04  2.030e-04  -4.335 1.53e-05 ***
n_cores        1.821e-03  3.148e-03   0.579   0.5629    
pc             1.287e-04  1.551e-03   0.083   0.9339    
px_height      2.765e-04  1.891e-05  14.626  < 2e-16 ***
px_width       2.796e-04  1.937e-05  14.437  < 2e-16 ***
ram            9.472e-04  6.638e-06 142.688  < 2e-16 ***
sc_h           1.140e-03  1.982e-03   0.575   0.5651    
sc_w          -3.284e-04  1.915e-03  -0.171   0.8639    
talk_time      3.639e-04  1.319e-03   0.276   0.7827    
three_g        2.705e-02  2.079e-02   1.301   0.1934    
touch_screen  -5.710e-03  1.438e-02  -0.397   0.6914    
wifi          -2.145e-02  1.440e-02  -1.489   0.1367    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3206 on 1979 degrees of freedom
Multiple R-squared:  0.9186,	Adjusted R-squared:  0.9178 
F-statistic:  1117 on 20 and 1979 DF,  p-value: < 2.2e-16
```

To select the optimal tuning parameter values for SVM, we need to determine the values of cost, gamma and degree. The cost parameter determines the tradeoff between a simple decision boundary and correctly classifying training data points, while the gamma parameter determines the width of the radial basis function kernel.
Using the tune() function in R, we cycled through 5 different cost values on a linear kernel: 0.01, 0.1, 1, 10, 100. For our polynomial kernel, we cycled through cost values of 0.1, 1, 10, 100, 1000, and degrees of 2, 3, 4, 5. Finally, we used cost values of 0.1, 1, 10, 100, 1000 and gamma of 0.5, 1, 2, 3, 4. The hyperparameters for the model with the lowest training error rate was selected from each kernel type and are as follows:

Linear:	cost: 100		error: 0.02
Polynomial:	cost: 100	degree: 3	error: 0.21
Radial:	cost: 10	gamma: 0.5	error: 0.1675

We determined that the best model to use would be the linear kernel with cost 100 because it had the lowest training error rate.

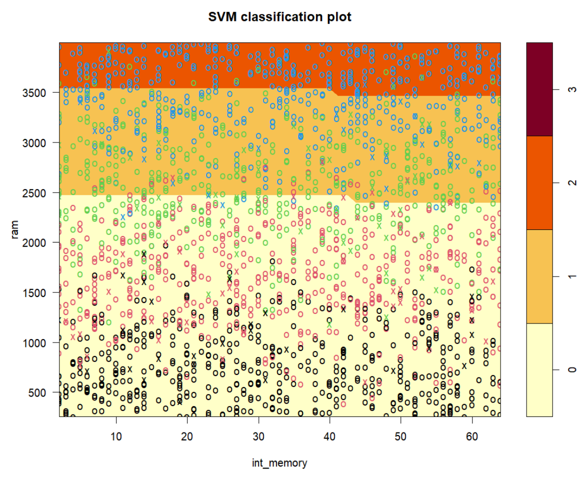
Confusion Matrix:

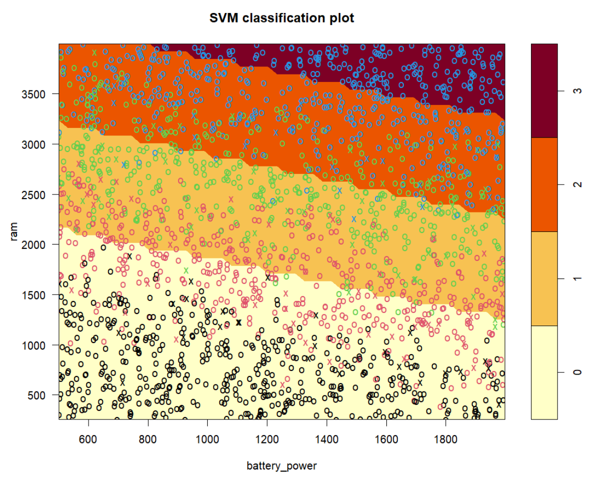
```
table(true=df[-train_sample,"price_range"],pred=ln.pred)
     pred
true  0  1  2  3
   0 29  1  0  0
   1  0 22  0  0
   2  0  1 17  0
   3  0  0  0 30
```

SVM Plot:

```
df.svmfit<-svm(as.factor(price_range)~.,data=o.data,kernel="linear",cost=10
0)
plot(df.svmfit,o.data,formula=ram~int_memory)
plot(df.svmfit,o.data,formula=ram~battery_power)
predict.df<-predict(df.svmfit,o.data)
table(true=as.factor(o.data$price_range),pred=as.factor(predict.df))
mean(predict.df!=o.data$price_range)
```

Linear SVM Classification Plots With Full Dataset





Here we have two plots of the best fitted model on all of the data. One uses the formula ram~int_memory and the other uses ram~battery_power. You can see the decision boundaries for the different price ranges on the graphs. When plotting out the reduced dataset with the variables colored in relation to what price range they are, we can see that the ram variable has the most visual impact as it increases or decreases. Therefore when making the graph we chose ram as a main variable.

Linear kernel SVM summary on full dataset:

```
Call:
svm(formula = price_range ~ ., data = df, kernel = "linear", cost = 100)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  100

Number of Support Vectors:  87

 ( 24 25 17 21 )


Number of Classes:  4

Levels:
 0 1 2 3
```

Confusion Matrix for Linear SVM on Full Dataset

```
> table(true=df$price_range, pred=best.pred)
   pred
true   0   1   2   3
   0 112   1   0   0
   1   1 120   1   0
   2   0   1 120   3
   3   0   0   2 139
> mean(best.pred != df$price_range)
[1] 0.018
```

Result

For our SVM model, the test error rate using our best model (linear kernel, 100 cost) is 0.0180. For our KNN model, the best test error rate using K value equal to 40 is 0.0800.
Based on the analysis, we can conclude that both SVM and KNN models have respectable predictive performances. However, the SVM model seems to have a slightly better performance compared to the KNN model based on the test error rate. The SVM model has a lower test error rate (0.018) than the KNN model (0.08) for this dataset. This means that the SVM model is better at predicting the outcome of new, unknown data points than the KNN model.

Conclusion

In our research, we found it difficult to decide on what features would be useful for modeling, and whether or not we should scale the data. Perhaps we could have performed PCA and used a variance threshold to condense the feature space further, but the loss in interpretability may or may not have been worth it compared to the performance gain. The main question that the research aimed to answer was "How much should a phone cost given its technological specifications?" The analysis also provides insights into product measurements and their respective sale prices. We trained models using some example phones and their respective price ranges, and we ended up with models that have promising, low error rates. We believe that we could use this model to make predictions of the price range a phone should fall in based on its specifications. From our models, it seems that RAM is the important factor in determining the price of a phone. This implies that if a customer is willing to have a slower phone they can get a cheaper phone with the same features. Ultimately, this research can be used to promote more honest market competition where companies can allocate more resources on product design that assists pricing strategies, and most importantly, help consumers make more informed purchases.

Difficulties:

- Data scaling
- Feature selection
- Convert categorical value to numeric value

Procedures to Improve:

- Excluding some features
- Using PCA to reduce data dimension

References

Abhishek, I. (2019). Mobile Price Classification. Kaggle. https://www.kaggle.com/iabhishekofficial/mobile-price-classification

Wang, W. (2023). Homework 2. MATH 4323: Data Science and Statistical Learning. University of Houston.

Wang, W. (2023). Homework 4. MATH 4323: Data Science and Statistical Learning. University of Houston.

Wang, W. (2023). Lab 2. MATH 4323: Data Science and Statistical Learning. University of Houston.

Wang, W. (2023). Lab 3. MATH 4323: Data Science and Statistical Learning. University of Houston.

Wang, W. (2023). Lab 4. MATH 4323: Data Science and Statistical Learning. University of Houston.

Wang, W. (2023). Lab 5. MATH 4323: Data Science and Statistical Learning. University of Houston.

Wang, W. (2023). Lecture 3 MATH 4323: Data Science and Statistical Learning. University of Houston.

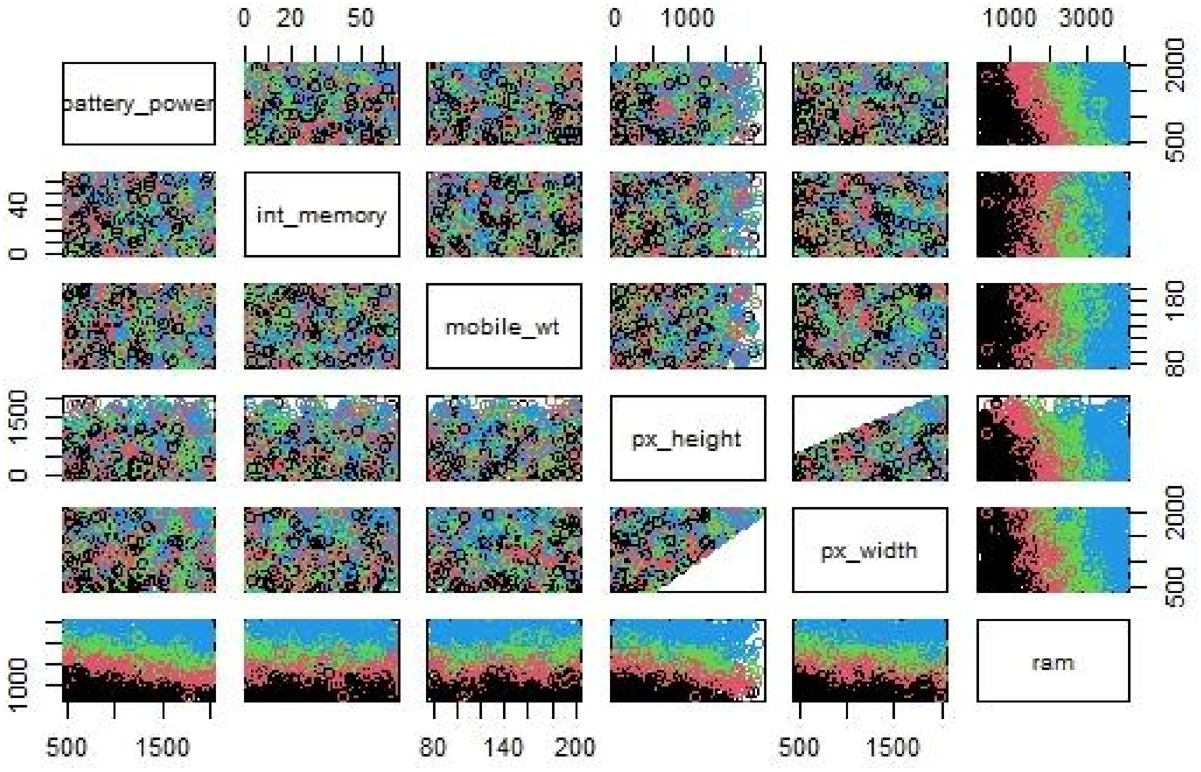
Wang, W. (2023). Lecture 4 MATH 4323: Data Science and Statistical Learning. University of Houston.

Wang, W. (2023). Lecture 5 MATH 4323: Data Science and Statistical Learning. University of Houston.

Wang, W. (2023). Lecture 6 MATH 4323: Data Science and Statistical Learning. University of Houston.

Wang, W. (2023). Lecture 7 MATH 4323: Data Science and Statistical Learning. University of Houston.

Appendix



SVM Method Code

```
install.packages("ggplot2")
install.packages("tidyverse")
install.packages("tidymodels")
install.packages("e1071")
library(ggplot2)
library(tidyverse)
library(tidymodels)
library(e1071)

o.data <- read.csv("~/School_2023_Spring/Data Science and Statistical
Learning/project/train.csv")
summary(lm(o.data$price_range~.,data=o.data))
newdata=data.frame(o.data[,c(1,7,9,12,13,14,21)])

n<-nrow(newdata)
set.seed(1)
```

```
training<-sample(1:n,0.8*n)
x=newdata[,1:6]
y=newdata[,7]
plot(x,col=(y))
newdata=data.frame(x=x,y=as.factor(y))
x.train<-newdata[training,-7]
y.train<-newdata[training,"y"]
x.test<-newdata[-training,-7]
y.test<-newdata[-training,"y"]
train.dat=data.frame(x=x.train,y=as.factor(y.train))
test.dat=data.frame(x=x.test,y=as.factor(y.test))

##linear##
set.seed(1)
tune.linear<-tune(svm, y~. ,data=train.dat,kernel='linear',
                  ranges=list(cost=c(0.01,0.1,1,5,10,100,100)))
tune.linear$best.parameters
linear.svm<-svm(y~.,data=train.dat,kernel="linear",cost=100)
table(linear.svm$fitted,train.dat$y)
mean(linear.pred!=train.dat$y)
linear.pred<-predict(linear.svm,newdata=test.dat)
table(linear.pred,test.dat$y)
mean(linear.pred!=test.dat$y)
svmfit.linear<-svm(y~.,data=newdata,kernel="linear",cost=100)
svmfit.ln<-svm(price_range~.,data=o.data,kernel="linear",cost=100)

##radial##
set.seed(1)
tune.radial<-tune(svm,y~.,data=train.dat,

kernel="radial",ranges=list(cost=c(0.01,0.1,1,5,10,100,1000),gamma=c(0.5,1,
2,3,4)))
tune.radial$best.parameters
radial.svm<-svm(y~.,data=train.dat,kernel="radial",cost=1,gamma=0.5)
table(radial.svm$fitted,train.dat$y)
radial.pred<-predict(radial.svm,newdata=test.dat)
table(radial.pred,test.dat$y)
mean(radial.pred!=test.dat$y)
svmfit.radial<-svm(y~.,data=newdata,kernel="radial",cost=1,gamma=0.5)


##polynomial##
set.seed(1)
```

```
tune.poly<-tune(svm,y~.,data=train.dat,kernel="polynomial",ranges=list(cost
=c(0.01,0.1,1,5,10,100),degree=c(2,3,4)))
tune.poly$best.parameters
poly.svm<-svm(y~.,data=train.dat,kernel="polynomial",cost=100,degree=3)
table(poly.svm$fitted,train.dat$y)
mean(poly.pred!=train.dat$y)
poly.pred<-predict(poly.svm,newdata=test.dat)
table(poly.pred,test.dat$y)
mean(poly.pred!=test.dat$y)

plot(x,col=y+1)
plot(svmfit.linear,newdata,formula=x.ram~x.px_height)
plot(svmfit.linear,newdata,formula=x.ram~x.px_width)
plot(svmfit.linear,newdata,formula=x.ram~x.battery_power)
plot(svmfit.linear,newdata,formula=x.ram~x.mobile_wt)
plot(svmfit.linear,newdata,formula=x.ram~x.int_memory)

##complete set##
df.svmfit<-svm(as.factor(price_range)~.,data=o.data,kernel="linear",cost=10
0)
plot(df.svmfit,o.data,formula=ram~int_memory)
plot(df.svmfit,o.data,formula=ram~battery_power)
plot(df.svmfit,o.data,formula = px_width~px_height)
plot(df.svmfit,o.data,formula = battery_power~int_memory)

predict.df<-predict(df.svmfit,o.data)
table(true=as.factor(o.data$price_range),pred=as.factor(predict.df))
mean(predict.df!=o.data$price_range)
```

KNN Method Code

```r
#KNN
# -- Load libraries
library(e1071) # svm
library(ggplot2)
library(reshape2)
library(plyr); library(dplyr) # pipes
library(class) # classification lib
library(corrplot) # correlation plot lib
#library(caret) # more advanced classification lib
#library(hrbrthemes)

# Sampling settings
RNGkind(sample.kind = "Rounding")
MobilePrice <- read.csv("mobile_project.csv")
dim(MobilePrice)
head(MobilePrice)
df <- MobilePrice
set.seed(1)
train <- sample(nrow(df), 0.8*nrow(df))
str(df)
X.train <- df[train,]
X.train$price_range <- NULL
y.train <- as.factor(df[train, "price_range"])

dim(X.train)
length(y.train)
X.test <- df[-train,]
X.test$price_range <- NULL
y.test <- as.factor(df[-train, "price_range"])

dim(X.test)
length(y.test)# scale numerics columns only
j <- c("clock_speed", "m_dep")

trainS <- scale(X.train[j])
X.train[j] <- trainS
X.test[j] <- scale(X.test[j],
    center = attr(trainS, "scaled:center"),
    scale = attr(trainS, "scaled:scale")
)
features <- !names(X.train) %in% c("pc", "three_g", "px_width", "sc_h")

X.train.new <- X.train[, features]
```

```r
X.test.new <- X.test[, features]
kvalues = 1:50
knn.train.err <- c()
knn.test.err <- c()
for(x in kvalues) {

    knn.pred <- knn(train=X.train, test=X.train, cl=y.train, k=x)
    knn.train.err[x] <- mean(knn.pred != y.train)

    knn.pred <- knn(train=X.train, test=X.test, cl=y.train, k=x)
    knn.test.err[x] <- mean(knn.pred != y.test)

    #cmd <- sprintf("(k=%i) Train err: %f, Test err: %f", x,
knn.train.err[x], knn.test.err[x])
    #print(eval(cmd))
}
bestK = max(which(knn.test.err == min(knn.test.err)))

print(c(bestK, knn.test.err[bestK]))
plot(knn.train.err, ylim=c(0,1), xlab="K", ylab="Mean test error rate",
type = "l", col="blue")
lines(knn.test.err, type='l', col="red")

kvalues = 1:200
knn.train.err <- c()
knn.test.err <- c()
for(x in kvalues) {
    set.seed(1)
    knn.pred <- knn(train=X.train.new, test=X.train.new, cl=y.train, k=x)
    knn.train.err[x] <- mean(knn.pred != y.train)

    knn.pred <- knn(train=X.train.new, test=X.test.new, cl=y.train, k=x)
    knn.test.err[x] <- mean(knn.pred != y.test)

    #cmd <- sprintf("(k=%i) Train err: %f, Test err: %f", x,
knn.train.err[x], knn.test.err[x])
    #print(eval(cmd))
}
nm.bestK = max(which(knn.test.err == min(knn.test.err)))

print(c(nm.bestK, knn.test.err[nm.bestK]))
plot(knn.train.err, ylim=c(0,1), xlab="K", ylab="Mean test error rate",
type = "l", col="blue")
```

```r
lines(knn.test.err, type='l', col="red")

knn.pred <- knn(train=X.train.new, test=X.test.new, cl=y.train, k=40)
knn.train.err <- mean(knn.pred != y.train)
knn.test.err <- mean(knn.pred != y.test)
table(y.test,knn.pred)
knn.train.err
knn.test.err
print(nm.bestK)
set.seed(1)
knn.pred <- knn(train=df[,-21], test=df[,-21], cl=y, k=nm.bestK)
knn.test.err <- mean(knn.pred != y)
table(knn.pred, y)
print(knn.test.err)
```