CMM 510 Data Mining - Resit

Vineeth Chimbili

23/07/2021

## Importing the data

We start with downloading the data files propertyMedium.csv and propertyTest.csv from CampusMoodle, which we load into RStudio.

# setting the working directory to import the data  
# setwd('SET WORKING DIRECTORY')  
  
# importing the data files propertyMedium.csv and propertyTest.csv  
propertyMedium <- read.csv('propertyMedium.csv')  
propertyTest <- read.csv('propertyTest.csv')  
  
# setting the seed value  
set.seed(123)

## Question 1

We use univariate statistics to explore the propertyMedium data. By skimming the data, we observe that there are a few missing values for the variables estate\_type. The second table shows the main summary statistics for the numerical variables, such as the mean value, standard deviation, and the quantiles. It also shows a mini-histogram to evaluate the distribution of the data.

We show three visualization plots. The first two plots visualize the numerical variables percent\_mortgage and square\_metres against price\_paid, for every property\_type. The visualization shows what is the relationship between the different variables, and how the relationship is different between different property\_types. For example, we observe that there are a few data points for property\_type O that have a very high price paid for only a small amount of square metres. The third and last visualization is able to plot two categorical variables against each other, namely estate\_type vs. property\_type. We immediately observe that if the estate\_type is L, then there is a high probability that the property\_type is F. Similar plots can be constructed for other variables. These visualizations plots are helpful to uncover the hidden relationships between multiple independent variables and the dependent variable.

## 'data.frame': 10097 obs. of 13 variables:  
## $ instance : int 6966 19822 19361 4528 4650 10690 15844 16976 8586 8465 ...  
## $ price\_paid : int 178000 130000 188000 160000 96000 186500 221500 350000 140000 364500 ...  
## $ percent\_mortgage : int 52 47 34 63 32 58 37 38 56 57 ...  
## $ square\_metres : int 490 75 486 458 451 14 71 29 553 153 ...  
## $ year : chr "Y 2020" "Y 2019" "Y 2020" "Y 2020" ...  
## $ area : chr "AREA\_15" "AREA\_9" "AREA\_8" "AREA\_13" ...  
## $ property\_type : chr "D" "S" "S" "T" ...  
## $ new\_build : chr "N" "N" "N" "N" ...  
## $ estate\_type : chr "F" "F" "F" "F" ...  
## $ district : chr "DISTRICT\_3" "DISTRICT\_3" "DISTRICT\_3" "DISTRICT\_3" ...  
## $ city : chr "Gordonnessburgh" "Gordonnessburgh" "Gordonnessburgh" "Gordonnessburgh" ...  
## $ county : chr "SOUTH\_COUNTY" "SOUTH\_COUNTY" "SOUTH\_COUNTY" "SOUTH\_COUNTY" ...  
## $ transaction\_category: chr "A" "B" "A" "A" ...

Data summary

|  |  |
| --- | --- |
| Name | propertyMedium |
| Number of rows | 10097 |
| Number of columns | 13 |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |
| Column type frequency: |  |
| character | 9 |
| numeric | 4 |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |
| Group variables | None |

**Variable type: character**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| skim\_variable | n\_missing | complete\_rate | min | max | empty | n\_unique | whitespace |
| year | 0 | 1 | 6 | 6 | 0 | 2 | 0 |
| area | 0 | 1 | 6 | 7 | 0 | 24 | 0 |
| property\_type | 0 | 1 | 1 | 1 | 0 | 5 | 0 |
| new\_build | 0 | 1 | 1 | 1 | 0 | 2 | 0 |
| estate\_type | 4 | 1 | 1 | 1 | 0 | 2 | 0 |
| district | 0 | 1 | 10 | 10 | 0 | 5 | 0 |
| city | 0 | 1 | 15 | 15 | 0 | 1 | 0 |
| county | 0 | 1 | 12 | 12 | 0 | 2 | 0 |
| transaction\_category | 0 | 1 | 1 | 1 | 0 | 2 | 0 |

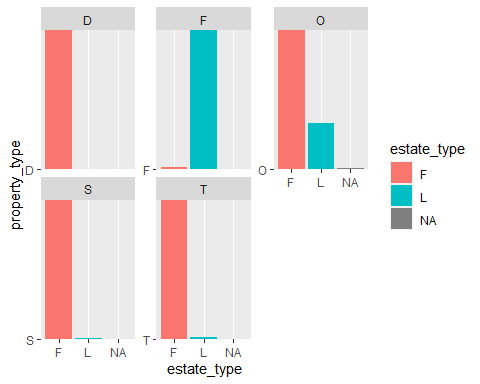
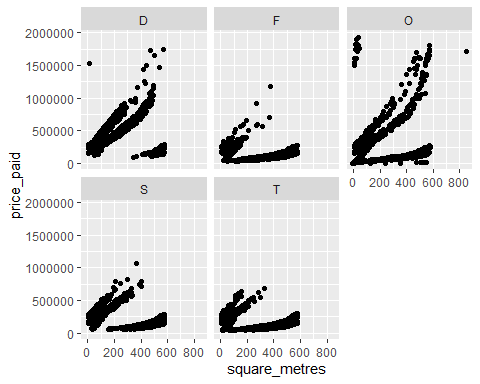
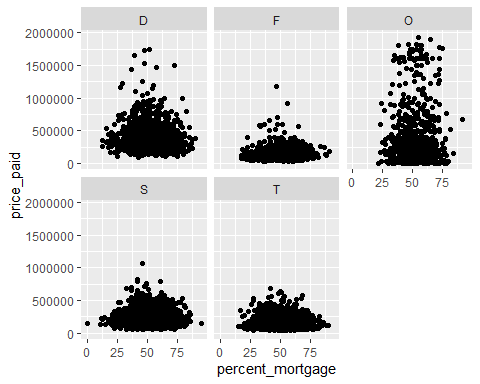
**Variable type: numeric**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| skim\_variable | n\_missing | complete\_rate | mean | sd | p0 | p25 | p50 | p75 | p100 | hist |
| instance | 0 | 1 | 10391.64 | 6014.92 | 5 | 5136 | 10396 | 15632 | 20889 | ▇▇▇▇▇ |
| price\_paid | 0 | 1 | 225568.85 | 180788.89 | 150 | 125000 | 182000 | 267500 | 1932000 | ▇▁▁▁▁ |
| percent\_mortgage | 0 | 1 | 51.77 | 11.54 | 0 | 44 | 52 | 60 | 95 | ▁▂▇▅▁ |
| square\_metres | 0 | 1 | 282.23 | 204.47 | -5 | 71 | 284 | 492 | 853 | ▇▂▅▃▁ |

## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v tibble 3.1.0 v dplyr 1.0.5  
## v tidyr 1.1.3 v stringr 1.4.0  
## v readr 1.4.0 v forcats 0.5.1  
## v purrr 0.3.4

## -- Conflicts ------------------------------------------ tidyverse\_conflicts() --  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag() masks stats::lag()



## Question 2

We take several steps to prepare the propertyMedium dataset for classification. First of all, all character variables are transformed to factor variables. We noted that there are several missing instances of the values of the variable estate\_type. Since there are only 4 missing values, we simply impute the most common value, which is the ‘F’. Finally, we apply normalization to the numerical variables to deal with the different scales.

## Question 3

We randomly sample from the propertyMedium dataset to obtain the datasets as defined in the question. The advantage of a reduced dataset is that algorithm will run more quickly, which will save time. On the other hand, the disadvantage of using a reduced dataset is that we throw away valuable information, which could lead to a reduction in performance. It is clear that there is a trade-off between performance and speed of the model. A possible criteria is to assess the width of confidence intervals of estimated parameters, which should not be too wide.

##   
## Attaching package: 'Momocs'

## The following objects are masked from 'package:dplyr':  
##   
## arrange, combine, filter, mutate, rename, sample\_frac, sample\_n,  
## select, slice

## The following object is masked from 'package:tidyr':  
##   
## chop

## The following object is masked from 'package:stats':  
##   
## filter

## Question 4

We are aiming to build a tree classifier to classify the variable property\_type. We build the same model on the datasets of different sizes, namely property30, property 15 and property5. The model is build using 5-fold cross-validation. Subsequently, we predict the property\_type variable on the full train dataset propertyMedium with the different models. The predictions on the test data will happen only in Question 7. We are using the class accuracy measure to assess the performance of the tree classifier. The class accuracy simply states the percentage of the predictions that were in the right class. The model built on the largest set of data, property30, has an accuracy rate of 0.5618. This is slightly higher than the accuracy rate on the data property15, which is only 0.5439. However, the model with the highest accuracy rate is actually the model built on the smallest set of data, namely property 5. This model has an accuracy rate of 0.5948301. Hence, we prefer the model built on the data property5.

## Loading required package: lattice

##   
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':  
##   
## lift

##   
## D F O S T   
## 0 1743 0 4948 3406

##   
## D F O S T   
## 0 1743 0 4216 4138

##   
## D F O S T   
## 2012 1743 0 2810 3532

## [1] 0.561751

## [1] 0.5439239

## [1] 0.5948301

## Question 5

We have previously applied normalization to the numerical variables to scale them from 0 to 1. We must always normalize the data in order to create unbiased results from an instance-based classifier. We choose to use the K-Nearest Neighbors (KNN) algorithm. Again, we use 5-fold cross-validation and evaluate based on the accuracy of the model on the full train data (accuracy on the test data will be evaluated later). The model on property30 has an accuracy rate of 0.6420, while property15 and property5 have accuracy rates of 0.5864 and 0.5099, respectively. Hence, the highest accuracy rate is obtained by the most data.

We use Fisher’s Exact Test to test if the samples come from a binomial distribution with a different parameter. The test shows a p-value close to 0, indicating that the parameter differs significantly between the two models. Hence, we prefer the model built on the biggest dataset, which is the model built on property30.

## k-Nearest Neighbors   
##   
## 3029 samples  
## 10 predictor  
## 5 classes: 'D', 'F', 'O', 'S', 'T'   
##   
## No pre-processing  
## Resampling: Cross-Validated (5 fold)   
## Summary of sample sizes: 2424, 2423, 2423, 2423, 2423   
## Resampling results across tuning parameters:  
##   
## k Accuracy Kappa   
## 5 0.6044803 0.4614706  
## 7 0.5985424 0.4505878  
## 9 0.5929340 0.4400926  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was k = 5.

## k-Nearest Neighbors   
##   
## 1514 samples  
## 10 predictor  
## 5 classes: 'D', 'F', 'O', 'S', 'T'   
##   
## No pre-processing  
## Resampling: Cross-Validated (5 fold)   
## Summary of sample sizes: 1211, 1209, 1213, 1212, 1211   
## Resampling results across tuning parameters:  
##   
## k Accuracy Kappa   
## 5 0.5739263 0.4174284  
## 7 0.5779258 0.4181961  
## 9 0.5693644 0.4023707  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was k = 7.

## k-Nearest Neighbors   
##   
## 499 samples  
## 10 predictor  
## 5 classes: 'D', 'F', 'O', 'S', 'T'   
##   
## No pre-processing  
## Resampling: Cross-Validated (5 fold)   
## Summary of sample sizes: 400, 399, 399, 398, 400   
## Resampling results across tuning parameters:  
##   
## k Accuracy Kappa   
## 5 0.5031105 0.3186374  
## 7 0.4949299 0.3022352  
## 9 0.4988503 0.3049968  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was k = 5.

##   
## D F O S T   
## 1119 1566 587 3730 3095

##   
## D F O S T   
## 1064 1408 393 3984 3248

##   
## D F O S T   
## 1247 1248 482 3437 3683

## [1] 0.64227

## [1] 0.5776963

## [1] 0.5077746

##   
## Fisher's Exact Test for Count Data  
##   
## data: matrix(c(round(0.6419729 \* nrow(propertyMedium)), round((1 - 0.6419729) \* nrow(propertyMedium)), round(0.5864118 \* nrow(propertyMedium)), round((1 - 0.5864118) \* nrow(propertyMedium))), ncol = 2)  
## p-value = 5.585e-16  
## alternative hypothesis: true odds ratio is not equal to 1  
## 95 percent confidence interval:  
## 1.194338 1.339085  
## sample estimates:  
## odds ratio   
## 1.264613

## Question 6

The instance-based classifier does a better job in predicting on the full train data than the tree classifier. The accuracy is higher with 0.6420 compared to 0.5948 of the decision tree in question 4. We note that the KNN model, which was chosen as the instance-based classifier, needed more data to train effectively than the decision tree. The decision tree did not perform any better when feeding it more data, it actually did slightly worse on our data.

## Question 7

We take the same steps to prepare the propertyTest dataset for classification as we did for the train data.

The best model that was trained previously was the KNN model on the property30 dataset, which we saved as the object knn\_property30. We validate this model by testing it with the propertyTest dataset.

# making predictions with the KNN model on the property30 dataset  
propertyTest$pred\_knn\_property30 <- predict(knn\_property30, type ='raw', newdata = propertyTest[, c(yy, xx)])  
  
# show the predictions on the test data  
table(propertyTest$pred\_knn\_property30)

##   
## D F O S T   
## 1265 1743 305 4698 2782

# calculate the accuracy of the predictions on the test data  
mean(propertyTest$property\_type == propertyTest$pred\_knn\_property30) # 0.5999259

## [1] 0.5999259

## Question 8

We add a new attribute called house to property30 which states whether the property is a house (detached, semidetached or terraced) or not. We call this new dataset houseflat and subsequently remove the property\_type attribute from it.

## Question 9

We perform the same procedure as previously for the property\_type attribute, but now for the house attribute. We compare the tree-based classifier with the instance-based classifier. Again, we use the accuracy rate of the class predictions as the measure of interest. We find that the tree-based classifier has an accuracy rate of 0.955761, while the instance-based classifier has an accuracy rate of 0.9564213. Hence, the performance of both classifiers is fairly similar, although the instance-based classifier slightly outperforms the instance-based classifier.

##   
## FALSE TRUE   
## 582 2447

##   
## FALSE TRUE   
## 590 2439

## [1] 0.955761

## [1] 0.9564213

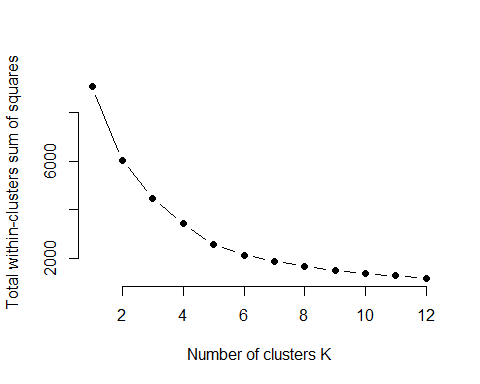
## Question 10

We use the k-means clustering algorithm via principal component analysis (PCA) to cluster the property30 dataset. It is a common practice to apply PCA before a clustering algorithm, because is believed that it improves the clustering results in practice (noise reduction).

We only use the numerical attributes price\_paid, percent\_mortgage and square\_metres in the clustering process, as they are the most appropriate given the distance metrics. We fit the k-means clustering algorithm after applying centering, scaling and principal component analysis. We first use the elbow method to determine a good value for the number of clusters k. Since this plot gives unclear results, we apply the silhouette method. This indicates that k = 3 is the optimal value.

The resulting clusters are correlated to the variables price\_paid and square\_metres, but not so much to percent\_mortgage. Cluster 3 has higher values for price\_paid compared to clusters 1 and 2, while cluster 1 has much lower values for square\_metres compared to cluster 2 and 3.

# apply centering, scaling and principal component analysis  
pca\_property30 <- preProcess(property30[,2:4], method = c("pca"))  
  
# predict using the principle component analysis on the property30 data  
property30\_v2 <- predict(pca\_property30, newdata = property30)  
  
# For each value of k, k-means is applied.  
# we first use the Elbow Method to determine a good k  
k.max <- 12  
wss <- sapply(1:k.max,   
 function(k){kmeans(property30\_v2[, 11:13], k, nstart=25,iter.max = 15 )$tot.withinss})  
plot(1:k.max, wss,  
 type="b", pch = 19, frame = FALSE,   
 xlab="Number of clusters K",  
 ylab="Total within-clusters sum of squares")



# since the results are unclear, the average silhouette is calculated.  
library(cluster)

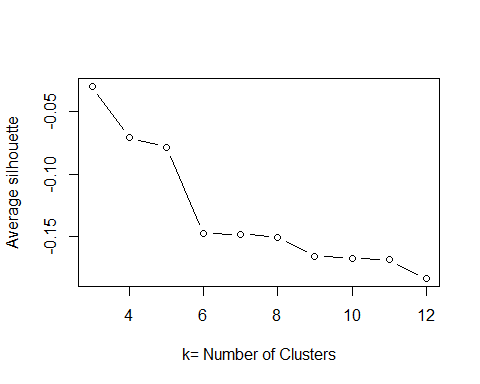
##   
## Attaching package: 'cluster'

## The following object is masked from 'package:Momocs':  
##   
## flower

set.seed(123)  
sil <- NULL  
for (i in 3:12) {   
 res <- kmeans(property30\_v2[, 11:13], centers = i, nstart = 25)  
 ss <- silhouette(res$cluster, dist(property30\_v2))  
 sil[i-2] <- mean(ss[, 3])  
}

## Warning in dist(property30\_v2): NAs introduced by coercion  
  
## Warning in dist(property30\_v2): NAs introduced by coercion  
  
## Warning in dist(property30\_v2): NAs introduced by coercion  
  
## Warning in dist(property30\_v2): NAs introduced by coercion  
  
## Warning in dist(property30\_v2): NAs introduced by coercion  
  
## Warning in dist(property30\_v2): NAs introduced by coercion  
  
## Warning in dist(property30\_v2): NAs introduced by coercion  
  
## Warning in dist(property30\_v2): NAs introduced by coercion  
  
## Warning in dist(property30\_v2): NAs introduced by coercion  
  
## Warning in dist(property30\_v2): NAs introduced by coercion

# plotting the average silhouettes  
plot(3:12, sil, type="b", xlab="k= Number of Clusters", ylab="Average silhouette")



# the silhouette plot reaches it maximum at k=3, which is therefore the optimal value of k  
  
# optimal model  
kmeans\_model <- kmeans(property30\_v2[, 11:13], centers = 3, nstart = 25)  
  
# predict the clusters from the model  
property30\_v2$cluster <- kmeans\_model$cluster  
  
# look at correlation for the variables: price\_paid  
mean(property30$price\_paid[property30\_v2$cluster == 1])

## [1] 0.5426336

mean(property30$price\_paid[property30\_v2$cluster == 2])

## [1] 0.1385215

mean(property30$price\_paid[property30\_v2$cluster == 3])

## [1] 0.07279544

# look at correlation for the variables: percent\_mortgage  
mean(property30$percent\_mortgage[property30\_v2$cluster == 1])

## [1] 0.5434654

mean(property30$percent\_mortgage[property30\_v2$cluster == 2])

## [1] 0.5536549

mean(property30$percent\_mortgage[property30\_v2$cluster == 3])

## [1] 0.5438014

# look at correlation for the variables: square\_metres  
mean(property30$square\_metres[property30\_v2$cluster == 1])

## [1] 0.4587229

mean(property30$square\_metres[property30\_v2$cluster == 2])

## [1] 0.1036987

mean(property30$square\_metres[property30\_v2$cluster == 3])

## [1] 0.5497193

## Question 11

Association rules can be helpful for the propertyMedium dataset to unravel knowledge about the property markets and structure of the dataset. For example, the apriori algorithm can be used to understand the relationship between the attributes area and district. This is similar to the groceries dataset, where we investigated which set of groceries are often bought together. A difference between the datasets is that we have several numerical variables in propertyMedium, which cannot be used initially as the apriori algorithm only takes categorical variables. However, it is possible to use a binning strategy to categorize the numerical variables.

## Question 12

So far, we have only considered classification models built from a single model with the objective to classify the property\_type variable. Obviously, different classification models may give different results for same problem. The idea of meta-learners is to combine multiple of these models to exploit the joint knowledge of all of them. The great advantage of these meta-learners is that they often improves predictive performance. One of the disadvantages is that the model can become increasingly hard to interpret when we combine the predictions of multiple models.

There are several methods that can be applied to build meta-learners for classification. The bagging approach is the simplest way to combine predictions. In bagging, each classifications model receives equal weight. The performance of a classifier can be improved by bagging, as it reduces the variance of the prediction.

Another method to improve the accuracy of class predictions is to use randomization. A lot of algorithms do not have a deterministic outcome, the results can depend for example on the seed. Another way is to randomly select some attributes in a decision tree. Randomization is widely applicable and can be combined with bagging (as happens for example with a random forest).

Finally, boosting is an iterative procedure whereby new models can be improved by the performance of previous models. For classification purposes, boosting is often used to give different models a different vote weight. Generally, boosting is more accurate than bagging, although it tends to overfit slightly.