Sales_Prediction

Bowei.Zhang

Janurary 9, 2017

Background and Introduction

One of my clients is an E-commerce company, they will launch new sales on their site every day. Each sale is a collection of products, basically they want me to do the predictions on their female shoes sales based on their historical data.They will use the prediction to guide their inventory and price in the future sales.

Below is a **quick demo & model prototype** that I built for them base on **3 months** historical data

Data Dictionary:

num_units_sold = This is the target variable. Number corresponds to the number of units per product look that were sold in a single sale.
product_look_id = Identifies each product look. Note that one single product look can be in multiple sales.
sale_id = identifies each sale. There should be multiple product look id's per sale.
id = concatenation of product_look_id and sale_id, serves as unique identifier for each row. Num_units_sold corresponds to this unique id.
sale_start_time = Time that sale begins, in EST
sale_end_time = Time that sale ends, in EST
sale_type_key = Type of sale can be single-brand (entire sale is one brand name), multi-brand (sale is organized around a theme and has multiple brands) or structured (splits sale into sizes or category)
brand_id = identifier for brand name
shoe_type = the style of the shoe
return_policy_id = The type of return policy associated with the product look
base_price = The price E-commerce company paid to buy the product
sale_price = The price that customers purchase the product for
MSRP_price = The manufacturer's suggested retail price. The full, non-discounted price that the brand would typically charge for their product.
initial_sale_price = All of the product looks in this dataset are repeats, meaning they have been on sale on their website before. This is the sale price of the product look the first time it was ever offered.
last_price = the sale price of the product look the previous time it was offered.
material_name = all the materials that the product is made of
color = color of the product
country of origin = country where product was manufactured
num_sizes = number of sizes available for that product look
num_units_available = number of total units per product look that are available for customer to buy

From the data dictionary that we could know we need to predict **num_units_sold** based on other variables

Required Packages:

```
require(Matrix)   # For construting binary variables from categorical variables (One-hot encoding)
require(xgboost)  # For XGBoosting model
require(dplyr)    # For data wrangling
require(caTools)  # For stratify sampling
```

Algorithm Selection

Period is not long enough for time series analysis.

Dependent variable **num_units_sold** is a **continuous** variable, so it is a **regression** question.

Independent variables are multi-types, there might be both linear and non-linear relationships, instead of **pure linear regression**, it is better to use **regression tree**. Tree based model also pretty robust to **multicollinearity**.

In order to improve the accuracy and prediction power of the model, I would like to use assemble method, specifically, **XGBoost**, since its high performance and speed.

Data Preparation

```
# Set working dictionary to target folder
setwd("C:\\Users\\bowei.zhang\\Desktop\\ME\\Career\\Analytics_Example\\Sale_Pridiction")


# read sample data, and change all blanks into 'NA'
sales <- read.csv("C:\\Users\\bowei.zhang\\Desktop\\ME\\Career\\Analytics_Example\\Sale_Pridiction/sales.csv", header=T, na.
strings=c("","NA"))

# Look at the size of the data and data type of each variables
str(sales)
```

```
## 'data.frame':    19023 obs. of  20 variables:
##  $ num_units_sold     : int  2 0 0 2 0 1 0 0 4 2 ...
##  $ product_look_id    : int  1013583601 1053356454 179777891 1079659830 1085768993 1055208901 1055243709 1072014509 10731
66367 1053839838 ...
##  $ sale_id            : int  1059839361 1071756375 1088474099 1089758407 1089811882 1089811882 1089811882 1089811882 1090
073154 1090261136 ...
##  $ id                 : Factor w/ 19023 levels "1000041475_1096093603",..: 484 5657 18890 13672 16479 5919 6030 11182 116
92 5690 ...
##  $ sale_start_time    : Factor w/ 694 levels "2015-01-01 02:00:00+00",..: 51 13 15 32 5 5 5 5 588 19 ...
##  $ sale_end_time      : Factor w/ 247 levels "2015-01-02 05:00:00+00",..: 40 10 11 29 5 5 5 5 81 15 ...
##  $ sale_type_key      : Factor w/ 4 levels "final","multi-brand",..: 3 3 3 3 2 2 2 2 2 2 ...
##  $ brand_id           : Factor w/ 181 levels "b10","b1038",..: 25 27 29 28 152 176 176 53 146 125 ...
##  $ shoe_type          : Factor w/ 22 levels "ballet flat",..: 18 14 5 10 14 14 14 14 18 4 ...
##  $ return_policy_id   : Factor w/ 2 levels "r-1","r260": 1 1 1 1 1 1 1 1 1 1 ...
##  $ base_price         : num  29.4 161.3 66.5 80.5 297 ...
##  $ sale_price         : num  59 299 119 149 379 499 499 499 89 359 ...
##  $ msrp_price         : num  100 495 190 268 475 ...
##  $ initial_sale_price : num  59 329 119 149 379 529 519 549 119 429 ...
##  $ last_price         : num  59 329 119 149 379 499 499 499 119 359 ...
##  $ material_name      : Factor w/ 1111 levels "-","agnepythonche",..: 829 164 802 1094 248 665 665 665 605 1035 ...
##  $ color              : Factor w/ 38 levels "Assorted Pre Pack",..: 34 23 26 23 4 36 10 4 4 32 ...
##  $ country_of_origin  : Factor w/ 23 levels "-  ","BGD","BRA",..: 4 12 23 4 12 12 12 12 4 12 ...
##  $ num_sizes          : int  1 8 3 7 3 5 2 1 6 3 ...
##  $ num_units_available: int  2 12 9 10 6 5 2 1 90 6 ...
```

```
# Take a look at typical value of each variables
summary(sales)
```

```
##  num_units_sold    product_look_id       sale_id
##  Min.   :  0.000   Min.   :9.878e+06   Min.   :1.060e+09
##  1st Qu.:  0.000   1st Qu.:1.051e+09   1st Qu.:1.095e+09
##  Median :  1.000   Median :1.063e+09   Median :1.098e+09
##  Mean   :  2.142   Mean   :1.039e+09   Mean   :1.097e+09
##  3rd Qu.:  2.000   3rd Qu.:1.080e+09   3rd Qu.:1.100e+09
##  Max.   :121.000   Max.   :1.102e+09   Max.   :1.104e+09
##
##                      id                      sale_start_time
##  1000041475_1096093603:    1   2015-02-26 02:00:00+00: 1508
##  1000041475_1100018125:    1   2015-03-21 16:00:00+00:  863
##  1000305637_1095805477:    1   2015-01-26 17:00:00+00:  643
##  1000684314_1095804575:    1   2015-03-30 16:00:00+00:  571
##  1000684316_1095804575:    1   2015-02-17 17:00:00+00:  386
##  1000684317_1094234363:    1   2015-02-24 17:00:00+00:  383
##  (Other)              :19017   (Other)               :14669
##                 sale_end_time        sale_type_key      brand_id
##  2015-02-27 17:00:00+00: 1304   final       :  437   b1352  :  930
##  2015-03-27 04:00:00+00:  830   multi-brand :15425   b30560 :  806
##  2015-02-02 17:00:00+00:  587   single-brand: 2532   b27053 :  657
##  2015-04-03 16:00:00+00:  571   structured  :  629   b25456 :  480
##  2015-03-19 16:00:00+00:  537                        b29650 :  470
##  2015-02-06 17:00:00+00:  496                        b146   :  358
##  (Other)               :14698                        (Other):15322
##                         shoe_type    return_policy_id   base_price
##  sandals                     :5023   r-1 :18586       Min.   :   4.0
##  pumps                       :4231   r260:  437       1st Qu.:  41.4
##  bootie                      :2500                    Median :  74.8
##  ballet flats                :2012                    Mean   : 105.4
##  sneakers                    : 953                    3rd Qu.: 127.9
##  loafers  drivers and moccasins: 912                  Max.   :1017.0
##  (Other)                     :3392
##    sale_price       msrp_price     initial_sale_price   last_price
##  Min.   :  15.0   Min.   :  20.0   Min.   :  19.0     Min.   :  15.0
##  1st Qu.:  79.0   1st Qu.: 150.0   1st Qu.:  89.0     1st Qu.:  85.0
##  Median : 129.0   Median : 265.0   Median : 149.0     Median : 129.0
##  Mean   : 180.4   Mean   : 363.2   Mean   : 202.4     Mean   : 183.9
##  3rd Qu.: 229.0   3rd Qu.: 495.0   3rd Qu.: 249.0     3rd Qu.: 229.0
##  Max.   :1099.0   Max.   :2740.0   Max.   :1465.0     Max.   :1295.0
##
##       material_name         color      country_of_origin   num_sizes
##  leather      : 3372   Black    :6037   CHN    :9581      Min.   : 1.000
##  suede        : 1549   No Color :1808   ITA    :5271      1st Qu.: 2.000
##  patentleather:  817   Cream Tan:1573   ESP    :1369      Median : 3.000
##  suedeleather :  473   Multi    :1161   BRA    :1270      Mean   : 4.469
##  leathersuede :  378   Red      : 926   PRT    : 467      3rd Qu.: 6.000
##  canvas       :  202   Grey     : 834   VNM    : 462      Max.   :13.000
##  (Other)      :12232   (Other)  :6684   (Other): 603
##  num_units_available
##  Min.   :   1.00
##  1st Qu.:   2.00
##  Median :   7.00
##  Mean   :  24.75
##  3rd Qu.:  21.00
##  Max.   :1533.00
##
```

```
# Data cleaning

unique(sales$shoe_type)
```

```
##  [1] sandals                       pumps
##  [3] cold weather boots            loafers  drivers and moccasins
##  [5] bootie                        ballet flats
##  [7] riding boot                   sneakers
##  [9] boot                          sandal
## [11] flats                         oxfords
## [13] flats pointed toe             slingbacks
## [15] thongs                        pump
## [17] rain boot                     other
## [19] slippers                      flats round toe
## [21] ballet flat                   espadrille
## 22 Levels: ballet flat ballet flats boot bootie ... thongs
```

```
sales$shoe_type[sales$shoe_type == "ballet flat"] <- "ballet flats"
sales$shoe_type[sales$shoe_type == "pump"] <- "pumps"
sales$shoe_type[sales$shoe_type == "sandal"] <- "sandals"
```

Feature Engineering

Since **XGBoost** only take **numerical variables**, there are lots of categorical variables in the model, I need to do some feature transformations here:
1. For categorical variables with less levels, I will convert them into binary dummy variables;
2. For categorical variables with more levels, I will calculate the average sale of each level and use those to represent that variable.

Create new variables

```
# Price related new variables
sales$msrp_discount <- sales$sale_price/sales$msrp_price

sales$initial_discount <-   sales$sale_price/sales$initial_sale_price

sales$profit <- sales$sale_price - sales$base_price


# Time related new variables
sales$sale_duration <- as.numeric(difftime(as.POSIXlt(sales$sale_end_time, format="%Y-%M-%d %H:%M:%S"), as.POSIXlt(sales$sal
e_start_time, format="%Y-%M-%d %H:%M:%S"), units="hours"))

sales$sale_month <- months(as.Date(sales$sale_start_time))

# New product categories
sales <- mutate(sales,brand_shoe_type = paste(brand_id,shoe_type, sep = '_'))

sales <- mutate(sales,brand_shoe_type_sale_type = paste(brand_id,shoe_type,sale_type_key, sep = '_'))

sales <- mutate(sales,brand_shoe_type_sale_type_color = paste(brand_id,shoe_type,sale_type_key,color, sep = '_'))
```

Does sales time sensitive? (Seasonality)

```
sales %>% group_by(sale_month) %>% summarise(month_avg=mean(num_units_sold))
```

```
## # A tibble: 3 x 2
##   sale_month month_avg
##        <chr>     <dbl>
## 1   February  1.880646
## 2    January  2.609110
## 3      March  2.046989
```

It looks like sales do have some sensitive to month(Jan. is a little bit higher than Feb. and Mar.), so `sale_month` variable should be able to catch this relationship.

Delete useless variables

```
# Drop meaningless (in terms of modeling) variables from dataset
drops <- c("sale_id","id","sale_start_time","sale_end_time")
sales <- sales[ , !(names(sales) %in% drops)]
```

Convert categorical variables into numerical variables

Convert `product_look_id` , `brand_id` , `material_name` , `country_of_origin` , `color` , `brand_shoe_type` , `brand_shoe_type_sale_type` , `brand_shoe_type_sa` into average sales

Before we do this, let’s split the dataset into training (80%) and test (20%) first:

```
# Stratify sampling based on time
train_rows = sample.split(sales$sale_month, SplitRatio=0.80)
train = na.omit(sales[ train_rows,])
test  = na.omit(sales[!train_rows,])
```

Then average the corresponding variables for both train and test data

```
################################################ Train dataset ################################################################
# Make lookup tables
product_look_id_lookup <- train %>% group_by(product_look_id) %>% summarise(product_look_id_avg=mean(num_units_sold))

brand_id_lookup <- train %>% group_by(brand_id) %>% summarise(brand_id_avg=mean(num_units_sold))

material_name_lookup <- train %>% group_by(material_name) %>% summarise(material_name_avg=mean(num_units_sold))

country_of_origin_lookup <- train %>% group_by(country_of_origin) %>% summarise(country_of_origin_avg=mean(num_units_sold))

color_lookup <- train %>% group_by(color) %>% summarise(color_avg=mean(num_units_sold))

brand_shoe_type_lookup <- train %>% group_by(brand_shoe_type) %>% summarise(brand_shoe_type_avg=mean(num_units_sold))

brand_shoe_type_sale_type_lookup <- train %>% group_by(brand_shoe_type_sale_type) %>% summarise(brand_shoe_type_sale_type_av
g=mean(num_units_sold))

brand_shoe_type_sale_type_color_lookup <- train %>% group_by(brand_shoe_type_sale_type_color) %>% summarise(brand_shoe_type_
sale_type_color_avg=mean(num_units_sold))


# Join lookup tables with train data
lookups <- list(train, product_look_id_lookup, brand_id_lookup,material_name_lookup,country_of_origin_lookup,color_lookup,br
and_shoe_type_lookup,brand_shoe_type_sale_type_lookup,brand_shoe_type_sale_type_color_lookup)

train <- Reduce(inner_join, lookups)

average_drops <- c("product_look_id","brand_id","material_name","country_of_origin","color","brand_shoe_type","brand_shoe_ty
pe_sale_type","brand_shoe_type_sale_type_color")
train <- train[ , !(names(train) %in% average_drops)]

################################################ Test dataset ##############################################################
# Make lookup tables
product_look_id_lookup <- test %>% group_by(product_look_id) %>% summarise(product_look_id_avg=mean(num_units_sold))

brand_id_lookup <- test %>% group_by(brand_id) %>% summarise(brand_id_avg=mean(num_units_sold))

material_name_lookup <- test %>% group_by(material_name) %>% summarise(material_name_avg=mean(num_units_sold))

country_of_origin_lookup <- test %>% group_by(country_of_origin) %>% summarise(country_of_origin_avg=mean(num_units_sold))

color_lookup <- test %>% group_by(color) %>% summarise(color_avg=mean(num_units_sold))

brand_shoe_type_lookup <- test %>% group_by(brand_shoe_type) %>% summarise(brand_shoe_type_avg=mean(num_units_sold))

brand_shoe_type_sale_type_lookup <- test %>% group_by(brand_shoe_type_sale_type) %>%
summarise(brand_shoe_type_sale_type_avg=mean(num_units_sold))

brand_shoe_type_sale_type_color_lookup <- test %>% group_by(brand_shoe_type_sale_type_color) %>% summarise(brand_shoe_type_s
ale_type_color_avg=mean(num_units_sold))


# Join lookup tables with test data
lookups <- list(test, product_look_id_lookup, brand_id_lookup,material_name_lookup,country_of_origin_lookup,color_lookup,bra
nd_shoe_type_lookup,brand_shoe_type_sale_type_lookup,brand_shoe_type_sale_type_color_lookup)

test <- Reduce(inner_join, lookups)

average_drops <- c("product_look_id","brand_id","material_name","country_of_origin","color","brand_shoe_type","brand_shoe_ty
pe_sale_type","brand_shoe_type_sale_type_color")
test <- test[ , !(names(test) %in% average_drops)]
```

Convert `sale_type_key` , `return_policy_id` , `sale_month` , `shoe_type` into binary dummy variables.

```
train_sparse_matrix <- sparse.model.matrix(num_units_sold~.-1, data = train)
test_sparse_matrix <- sparse.model.matrix(num_units_sold~.-1, data = test)
head(train_sparse_matrix)
```

```
## 6 x 47 sparse Matrix of class "dgCMatrix"
##
## 1 . . 1 . . . . . . . . . . . . . . . . . 1 . . . . .  29.40  59 100  59
## 2 . . 1 . . . . . . . . . . . . . 1 . . . . . . . . . 161.27 299 495 329
## 3 . . 1 . . . . 1 . . . . . . . . . . . . . . . . . .  66.50 119 190 119
## 4 . . 1 . . . . . . . . . 1 . . . . . . . . . . . . .  80.50 149 268 149
## 5 . 1 . . . . . . . . . . . . . . 1 . . . . . . . . . 339.15 499 660 529
## 6 . 1 . . . . . . . . . . . . . . 1 . . . . . . . . . 339.15 499 660 519
##
## 1  59 1  2 0.5900000 1.0000000  29.60 36 1 . 2.0000000 3.5546875 1.403509
## 2 329 8 12 0.6040404 0.9088146 137.73 36 1 . 0.5000000 0.7198582 1.333333
## 3 119 3  9 0.6263158 1.0000000  52.50 36 1 . 0.1666667 1.5534591 1.132353
## 4 149 7 10 0.5559701 1.0000000  68.50 63 1 . 0.6666667 1.4583333 1.481481
## 5 499 5  5 0.7560606 0.9432892 159.85 27 1 . 2.0000000 0.8552632 2.955224
## 6 499 2  2 0.7560606 0.9614644 159.85 27 1 . .         0.8552632 2.955224
##
## 1 3.0652912 2.500000 3.3636364 1.5000000 2.0000000
## 2 0.9046367 2.006529 0.4578313 0.9090909 0.8888889
## 3 2.1393443 1.890636 1.6137931 3.6774194 6.2727273
## 4 3.0652912 2.006529 1.0000000 1.4000000 1.4000000
## 5 0.9046367 1.216561 0.9056604 0.8372093 1.7777778
## 6 0.9046367 2.699367 0.9056604 0.8372093 .
```

```
# Create the dependent vector
train_output_vector <- train[,"num_units_sold"]
test_output_vector <- test[,"num_units_sold"]
```

Build the model

```
set.seed(100)

param <- list("objective" = "reg:linear",
              "nthread" = 4,
              "eta" = 0.1,
              "subsample"=0.8,
              "gamma" = 1,
              "min_child_weight" = 2,
              "max_depth"= 15
)

# Construct a watch list, use test error to measure the model quality to avoid overfitting during model training (similar to
 cross validation)
dtrain <- xgb.DMatrix(data = train_sparse_matrix, label=train_output_vector)
dtest <- xgb.DMatrix(data = test_sparse_matrix, label=test_output_vector)
watchlist <- list(train=dtrain, test=dtest)

bst <- xgb.train(param=param,data = dtrain, nrounds=25,watchlist=watchlist )
```

```
## [0]  train-rmse:4.535104 test-rmse:3.950115
## [1]  train-rmse:4.188813 test-rmse:3.653950
## [2]  train-rmse:3.879241 test-rmse:3.379133
## [3]  train-rmse:3.585112 test-rmse:3.122602
## [4]  train-rmse:3.327773 test-rmse:2.890170
## [5]  train-rmse:3.093598 test-rmse:2.699106
## [6]  train-rmse:2.877600 test-rmse:2.548073
## [7]  train-rmse:2.683579 test-rmse:2.394692
## [8]  train-rmse:2.503203 test-rmse:2.280750
## [9]  train-rmse:2.341190 test-rmse:2.184124
## [10] train-rmse:2.193161 test-rmse:2.092247
## [11] train-rmse:2.057976 test-rmse:1.996615
## [12] train-rmse:1.932040 test-rmse:1.919639
## [13] train-rmse:1.825776 test-rmse:1.857282
## [14] train-rmse:1.720383 test-rmse:1.807788
## [15] train-rmse:1.626139 test-rmse:1.768529
## [16] train-rmse:1.543834 test-rmse:1.737269
## [17] train-rmse:1.464945 test-rmse:1.707038
## [18] train-rmse:1.393475 test-rmse:1.680869
## [19] train-rmse:1.329797 test-rmse:1.660667
## [20] train-rmse:1.268299 test-rmse:1.646839
## [21] train-rmse:1.215782 test-rmse:1.636878
## [22] train-rmse:1.163557 test-rmse:1.624092
## [23] train-rmse:1.121457 test-rmse:1.618191
## [24] train-rmse:1.079435 test-rmse:1.621443
```

From my own experiment running on large rounds number that I know if the model running more than around 25 rounds,it will decrease `train-rmse` but increase `test-rmse` which means overfitting.

Model Evaluation

Predicting on test data

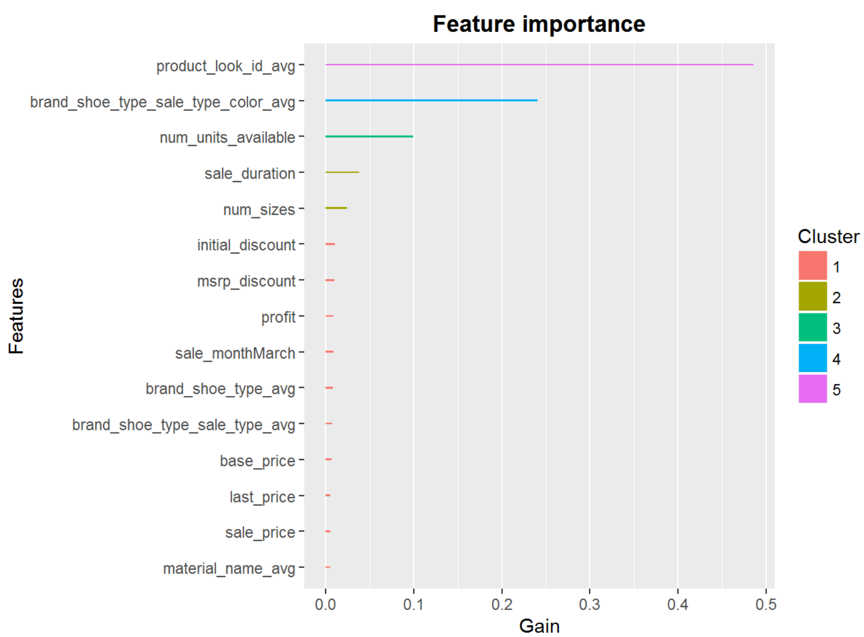
We will apply this model to our test dataset to evaluate the model

```
y_pred <- predict(bst, test_sparse_matrix)

# Round the sales prediction into integer
y_pred <- round(y_pred)

# Predicted sales should not larger than number of units available.
real_pred <- ifelse(y_pred > test$num_units_available,test$num_units_available,y_pred)

r2 <- sum((real_pred - mean(test$num_units_sold))^2)/sum((test$num_units_sold - mean(test$num_units_sold))^2)


print(paste("R square is ", r2))
```

```
## [1] "R square is  0.801239440421171"
```

Feature Importance

```
# get the trained model
model = xgb.dump(bst, with.stats=TRUE)
# get the feature real names
names = dimnames(train_sparse_matrix)[[2]]
# compute feature importance matrix
importance_matrix = xgb.importance(names, model=bst)

# plot the top 10 features
print(xgb.plot.importance(importance_matrix[1:15,]))
```



I only plot top 15 importance features, from the plot we could see that **prodcut_look_id_avg** is definitely the most important variable in terms of prediction power which make sense. Other important variables other average sale variables, **num_units_available**, **sale_duration**, **num_sizes**, **discount** also make sense.

View the trees structure

```
#xgb.dump(bst)

#xgb.plot.tree(model = bst)
```

Save the model

```
xgb.save(bst, "xgboost.model")
```

```
## [1] TRUE
```

```
#bst2 <- xgb.load("xgboost.model")
#pred2 <- predict(bst2, test$data)
```

Other tries:

1. Above is the model that I built in **tree booster**, I've tried **linear booster** as well which I did not list here since the R square is around 15% less than tree based model, I think that is because linear based model cannot catch some non-linear relationship between sales and independent variables;
2. From the complete feature importance plot that I find **shoe_type** almost have no importance in terms of Model gain. Therefore I remove it and re-built the model,whose prediction error is among the same as my previous model, for the sake of simplicity of the model, we could take **shoe_type** out if we want.

Further Modeling suggestions:

1. Change the numerical conversion type of some categorical variables, for example: we could also change `color` into binary dummy variables instead of replacing it by its average sales;
2. Use statistical tests to test on important variables to see if the result make sense;
3. Tuning more on XGBoost model parameters to get better prediction models.