Pragmatic Programming Techniques

Thursday, May 17, 2012

Predictive Analytics: Data Preparation

As a continuation of my last post on predictive analytics, in this post I will focus in describing how to prepare data for the training the predictive model., I will cover how to perform necessary sampling to ensure the training data is representative and fit into the machine processing capacity. Then we validate the input data and perform necessary cleanup on format error, fill-in missing values and finally transform the collected data into our defined set of input features.

Different machine learning model will have its unique requirement in its input and output data type. Therefore, we may need to perform additional transformation to fit the model requirement

Sample the data

If the amount of raw data is huge, processing all of them may require an extensive amount of processing power which may not be practical. In this case it is quite common to sample the input data to reduce the size of data that need to be processed.

There are many sampling models. Random sampling is by far the most common model which assign a uniform probability to each record for being picked. On the other hand, stratified sampling allows different probability to be assigned to different record type. It is very useful in case of highly unbalanced class occurrence so that the high frequency class will be down-sampled. Cluster sampling is about sampling at a higher level of granularity (ie the cluster) and then pick all members within that cluster. An example of cluster sampling is to select family (rather than individuals) to conduct a survey.

Sample can also be done without replacement (each record can be picked at most once) or with replacement (same record can be picked more than once)

Here is a simple example of performing a sampling (notice record 36 has been selected twice)

```
> # select 10 records out from iris with replacement
> index <- sample(1:nrow(iris), 10, replace=T)
> index
 [1] 133  36 107 140  66  67  36   3  97  37
> irissample <- iris[index,]
> irissample
     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
133           6.4         2.8          5.6         2.2  virginica
36            5.0         3.2          1.2         0.2     setosa
107           4.9         2.5          4.5         1.7  virginica
140           6.9         3.1          5.4         2.1  virginica
66            6.7         3.1          4.4         1.4 versicolor
67            5.6         3.0          4.5         1.5 versicolor
36.1          5.0         3.2          1.2         0.2     setosa
3             4.7         3.2          1.3         0.2     setosa
97            5.7         2.9          4.2         1.3 versicolor
37            5.5         3.5          1.3         0.2     setosa
>
```

Impute missing data

It is quite common that some of the input records are incomplete in the sense that certain fields are missing or have input error. In a typical tabular data format, we need to validate each record contains the same number of fields and each field contains the data type we expect.

In case the record has some fields missing, we have the following choices

- Discard the whole record if it is incomplete
- Infer the missing value based on the data from other records. A common approach is to fill the missing data with the average, or the median.

About Me

Ricky Ho

I am a software architect and consultant passionate in Distributed and parallel computing, Machine learning and Data mining, SaaS and Cloud computing.

View my complete profile

Popular Posts

MongoDb Architecture
NOSQL has become a very heated topic for large web-scale deployment where scalability and semi-structured data driven the DB requirement tow...

Designing algorithms for Map Reduce
Since the emerging of Hadoop implementation, I have been trying to morph existing algorithms from various areas into the map/reduce model. ...

NOSQL Patterns
Over the last couple years, we see an emerging data storage mechanism for storing large scale of data. These storage solution differs quite...

Couchbase Architecture
After receiving a lot of good feedback and comment on my last blog on MongoDb , I was encouraged to do another deep dive on another popular ...

Predictive Analytics: Overview and Data visualization
I plan to start a series of blog post on predictive analytics as there is an increasing demand on applying machine learning technique to ana...

Predictive Analytics: Generalized Linear Regression
In the previous 2 posts, we have covered how to visualize input data to explore strong signals as well as how to prepare input data to a fo...

BigTable Model with Cassandra and HBase
Recently in a number of "scalability discussion meeting", I've seen the following pattern coming up repeatedly ... To make you...

Blog Archive

```
> # Create some missing data
> irissample[10, 1] <- NA
> irissample
     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
133           6.4         2.8          5.6         2.2  virginica
36            5.0         3.2          1.2         0.2     setosa
107           4.9         2.5          4.5         1.7  virginica
140           6.9         3.1          5.4         2.1  virginica
66            6.7         3.1          4.4         1.4 versicolor
67            5.6         3.0          4.5         1.5 versicolor
36.1          5.0         3.2          1.2         0.2     setosa
3             4.7         3.2          1.3         0.2     setosa
97            5.7         2.9          4.2         1.3 versicolor
37             NA         3.5          1.3         0.2     setosa
> library(e1071)
Loading required package: class
Warning message:
package 'e1071' was built under R version 2.14.2
> fixIris1 <- impute(irissample[,1:4], what='mean')
> fixIris1
     Sepal.Length Sepal.Width Petal.Length Petal.Width
133      6.400000         2.8          5.6         2.2
36       5.000000         3.2          1.2         0.2
107      4.900000         2.5          4.5         1.7
140      6.900000         3.1          5.4         2.1
66       6.700000         3.1          4.4         1.4
67       5.600000         3.0          4.5         1.5
36.1     5.000000         3.2          1.2         0.2
3        4.700000         3.2          1.3         0.2
97       5.700000         2.9          4.2         1.3
37       5.655556         3.5          1.3         0.2
> fixIris2 <- impute(irissample[,1:4], what='median')
> fixIris2
     Sepal.Length Sepal.Width Petal.Length Petal.Width
133           6.4         2.8          5.6         2.2
36            5.0         3.2          1.2         0.2
107           4.9         2.5          4.5         1.7
140           6.9         3.1          5.4         2.1
66            6.7         3.1          4.4         1.4
67            5.6         3.0          4.5         1.5
36.1          5.0         3.2          1.2         0.2
3             4.7         3.2          1.3         0.2
97            5.7         2.9          4.2         1.3
37            5.6         3.5          1.3         0.2
>
```

Normalize numeric value

Normalize data is about transforming numeric data into a uniform range. Numeric attribute can have different magnitude based on different measurement units. To compare numeric attributes at the same scale, we need to normalize data by subtracting their average and then divide by the standard deviation.

```
> # scale the columns
> # x-mean(x)/standard deviation
> scaleiris <- scale(iris[, 1:4])
> head(scaleiris)
     Sepal.Length Sepal.Width Petal.Length Petal.Width
[1,]   -0.8976739  1.01560199    -1.335752   -1.311052
[2,]   -1.1392005 -0.13153881    -1.335752   -1.311052
[3,]   -1.3807271  0.32731751    -1.392399   -1.311052
[4,]   -1.5014904  0.09788935    -1.279104   -1.311052
[5,]   -1.0184372  1.24503015    -1.335752   -1.311052
[6,]   -0.5353840  1.93331463    -1.165809   -1.048667
>
```

Reduce dimensionality

High dimensionality is a problem to machine learning task. There are two ways to reduce the number of input attributes. One is about removing irrelevant input variables, another one is about removing redundant input variables.

Looking from the other angle, removing irrelevant feature is same as selecting relevant feature. The general approach is to try different combination of subset of feature to see which combination has the best performance (how well it predicts hold-out test data). One approach is to start with zero feature and pick one feature at a time to see which one give best prediction, and then add the second feature to the best feature to find the best 2 features, so on and so forth. Another approach is to go the opposite direction, start with the full set of feature and throw away one feature to see which one retains best performance.

- ► September (1)
- ► August (2)
- ► July (1)
- ► June (3)
- ▼ May (3)
  - Predictive Analytics: Generalized Linear Regressio...
  - Predictive Analytics: Data Preparation
  - Predictive Analytics: Overview and Data visualizat...
- ► April (3)
- ► March (1)
- ► February (1)
- ► January (1)

► 2011 (6)
► 2010 (18)
► 2009 (31)
► 2008 (22)
► 2007 (11)

Search This Blog

Search

Labels

machine learning data mining map reduce Architecture Design Cloud computing algorithm NOSQL Hadoop scalability Distributed system parallel processing big data predictive analytics Design patterns SOA performance REST ensemble method recommendation engine

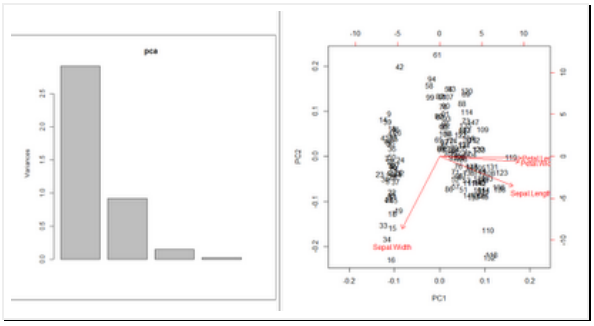
Pages

- Home

In my experience, having irrelevant features sitting around will affect the overall workload of processing but usually won't affect the accuracy of the prediction. This is a lesser problem than redundant features, which will give unequal weights to information that are redundant. Removing redundant features is at a higher priority in my opinion.

Principal Component Analysis is a common technique to look only at the numeric input features to measure the linear-dependency among themselves. It transforms the input feature set into a lower dimensional space while retaining most of the fidelity of data.

```
> # Use iris data set
> cor(iris[, -5])
              Sepal.Length   Sepal.Width  Petal.Length   Petal.Width
Sepal.Length  1.0000000000 -0.1175697841  0.8717537759  0.8179411263
Sepal.Width  -0.1175697841  1.0000000000 -0.4284401043 -0.3661259325
Petal.Length  0.8717537759 -0.4284401043  1.0000000000  0.9628654314
Petal.Width   0.8179411263 -0.3661259325  0.9628654314  1.0000000000
> # Some attributes shows high correlation, compute PCA
> pca <- prcomp(iris[,-5], scale=T)
> summary(pca)
Importance of components:
                          PC1       PC2       PC3       PC4
Standard deviation     1.708361 0.9560494 0.3830886 0.1439265
Proportion of Variance 0.729620 0.2285100 0.0366900 0.0051800
Cumulative Proportion  0.729620 0.9581300 0.9948200 1.0000000
> # Notice PC1 and PC2 covers most variation
> plot(pca)
> pca$rotation
                     PC1            PC2           PC3           PC4
Sepal.Length  0.5210659147 -0.37741761556  0.7195663527  0.2612862800
Sepal.Width  -0.2693474425 -0.92329565954 -0.2443817795 -0.1235096196
Petal.Length  0.5804130958 -0.02449160909 -0.1421263693 -0.8014492463
Petal.Width   0.5648565358 -0.06694198697 -0.6342727371  0.5235971346
> # Project first 2 records in PCA direction
> predict(pca)[1:2,]
             PC1           PC2          PC3           PC4
[1,] -2.257141176 -0.4784238321 0.1272796237 0.02408750846
[2,] -2.074013015  0.6718826870 0.2338255167 0.10266284468
> # plot all points in top 2 PCA direction
> biplot(pca)
```



Add derived attributes

In some cases, we may need to compute additional attributes from existing attributes.

```
> iris2 <- transform(iris, ratio=round(Sepal.Length/Sepal.Width, 2))
> head(iris2)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species ratio
1          5.1         3.5          1.4         0.2  setosa  1.46
2          4.9         3.0          1.4         0.2  setosa  1.63
3          4.7         3.2          1.3         0.2  setosa  1.47
4          4.6         3.1          1.5         0.2  setosa  1.48
5          5.0         3.6          1.4         0.2  setosa  1.39
6          5.4         3.9          1.7         0.4  setosa  1.38
```

Another common use of derived attributes is to generalize some attributes to a coarser grain, such as converting a geo-location to a zip code, or converting the age to an age group.

Discretize numeric value into categories

Discretize data is about cutting a continuous value into ranges and assigning the numeric with the corresponding bucket of the range it falls on. For numeric attribute, a common way to generalize it is to discretize it into ranges, which can be either constant width (variable height/frequency) or variable width (constant height).

```
> # Equal width cuts
> segments <- 10
> maxL <- max(iris$Petal.Length)
> minL <- min(iris$Petal.Length)
> theBreaks <- seq(minL, maxL,
                   by=(maxL-minL)/segments)
> cutPetalLength <- cut(iris$Petal.Length,
                        breaks=theBreaks,
                        include.lowest=T)
> newdata <- data.frame(orig.Petal.Len=iris$Petal.Length,
                        cut.Petal.Len=cutPetalLength)
> head(newdata)
  orig.Petal.Len cut.Petal.Len
1            1.4      [1,1.59]
2            1.4      [1,1.59]
3            1.3      [1,1.59]
4            1.5      [1,1.59]
5            1.4      [1,1.59]
6            1.7   (1.59,2.18]
>
> # Constant frequency / height
> myBreaks <- quantile(iris$Petal.Length,
                       probs=seq(0,1,1/segments))
> cutPetalLength2 <- cut(iris$Petal.Length,
                         breaks=myBreaks,
                         include.lowest=T)
> newdata2 <- data.frame(orig.Petal.Len=iris$Petal.Length,
                         cut.Petal.Len=cutPetalLength2)
> head(newdata2)
  orig.Petal.Len cut.Petal.Len
1            1.4       [1,1.4]
2            1.4       [1,1.4]
3            1.3       [1,1.4]
4            1.5     (1.4,1.5]
5            1.4       [1,1.4]
6            1.7     (1.7,3.9]
>
```

Binarize categorical attributes

Certain machine learning models only take binary input (or numeric input). In this case, we need to convert categorical attribute into multiple binary attributes, while each binary attribute corresponds to a particular value of the category. (e.g. sunny/rainy/cloudy can be encoded as sunny == 100 and rainy == 010)

```
> cat <- levels(iris$Species)
> cat
[1] "setosa"     "versicolor" "virginica"
> binarize <- function(x) {return(iris$Species == x)}
> newcols <- sapply(cat, binarize)
> colnames(newcols) <- cat
> data <- cbind(iris[,c('Species')], newcols)
> data[45:55,]
        setosa versicolor virginica
 [1,] 1      1          0         0
 [2,] 1      1          0         0
 [3,] 1      1          0         0
 [4,] 1      1          0         0
 [5,] 1      1          0         0
 [6,] 1      1          0         0
 [7,] 2      0          1         0
 [8,] 2      0          1         0
 [9,] 2      0          1         0
[10,] 2      0          1         0
[11,] 2      0          1         0
```

Select, combine, aggregate data

Designing the form of training data in my opinion is the most important part of the whole predictive modeling exercise because the accuracy largely depends on whether the input features are structured in an appropriate form that provide strong signals to the learning algorithm.

Rather than using the raw data as is, it is quite common that multiple pieces of raw data need to be combined together, or aggregating multiple raw data records along some dimensions.

In this section, lets use a different data source CO2, which provides the carbon dioxide uptake in grass plants.

```
> head(CO2)
  Plant   Type  Treatment conc uptake
1   Qn1 Quebec nonchilled   95   16.0
2   Qn1 Quebec nonchilled  175   30.4
3   Qn1 Quebec nonchilled  250   34.8
```

```
4   Qn1 Quebec nonchilled  350   37.2
5   Qn1 Quebec nonchilled  500   35.3
6   Qn1 Quebec nonchilled  675   39.2
>
```

To select the record that meet a certain criteria

```
> data <- CO2[CO2$conc>400 & CO2$uptake>40,]
> head(data)
   Plant   Type  Treatment conc uptake
12   Qn2 Quebec nonchilled  500   40.6
13   Qn2 Quebec nonchilled  675   41.4
14   Qn2 Quebec nonchilled 1000   44.3
19   Qn3 Quebec nonchilled  500   42.9
20   Qn3 Quebec nonchilled  675   43.9
21   Qn3 Quebec nonchilled 1000   45.5
```

To sort the records, lets say we want to sort by conc (in ascending order) and then by uptake (in descending order)

```
> # ascend sort on conc, descend sort on uptake
> CO2[order(CO2$conc, -CO2$uptake),][1:20,]
   Plant        Type  Treatment conc uptake
15   Qn3      Quebec nonchilled   95   16.2
1    Qn1      Quebec nonchilled   95   16.0
36   Qc3      Quebec    chilled   95   15.1
22   Qc1      Quebec    chilled   95   14.2
8    Qn2      Quebec nonchilled   95   13.6
50   Mn2 Mississippi nonchilled   95   12.0
57   Mn3 Mississippi nonchilled   95   11.3
43   Mn1 Mississippi nonchilled   95   10.6
78   Mc3 Mississippi    chilled   95   10.6
64   Mc1 Mississippi    chilled   95   10.5
29   Qc2      Quebec    chilled   95    9.3
71   Mc2 Mississippi    chilled   95    7.7
16   Qn3      Quebec nonchilled  175   32.4
2    Qn1      Quebec nonchilled  175   30.4
9    Qn2      Quebec nonchilled  175   27.3
30   Qc2      Quebec    chilled  175   27.3
23   Qc1      Quebec    chilled  175   24.1
51   Mn2 Mississippi nonchilled  175   22.0
37   Qc3      Quebec    chilled  175   21.0
58   Mn3 Mississippi nonchilled  175   19.4
>
```

To look at each plant rather than each raw record, lets compute the average uptake per plant.

```
> aggregate(CO2[,c('uptake')], data.frame(CO2$Plant), mean)
   CO2.Plant        x
1        Qn1 33.22857
2        Qn2 35.15714
3        Qn3 37.61429
4        Qc1 29.97143
5        Qc3 32.58571
6        Qc2 32.70000
7        Mn3 24.11429
8        Mn2 27.34286
9        Mn1 26.40000
10       Mc2 12.14286
11       Mc3 17.30000
12       Mc1 18.00000
>
```

We can also group by the combination of type and treatment

```
> aggregate(CO2[,c('conc', 'uptake')],
            data.frame(CO2$Type, CO2$Treatment),
            mean)

     CO2.Type CO2.Treatment conc   uptake
1      Quebec    nonchilled  435 35.33333
2 Mississippi    nonchilled  435 25.95238
3      Quebec       chilled  435 31.75238
4 Mississippi       chilled  435 15.81429
```

To join multiple data sources by a common key, we can use the merge() function.

```
> head(CO2)
  Plant   Type  Treatment conc uptake
1   Qn1 Quebec nonchilled   95   16.0
2   Qn1 Quebec nonchilled  175   30.4
3   Qn1 Quebec nonchilled  250   34.8
4   Qn1 Quebec nonchilled  350   37.2
5   Qn1 Quebec nonchilled  500   35.3
6   Qn1 Quebec nonchilled  675   39.2
> # Lets create some artitificial data
> state <- c('California', 'Mississippi', 'Fujian',
                 'Shandong', 'Quebec', 'Ontario')
> country <- c('USA', 'USA', 'China', 'China',
                  'Canada', 'Canada')
> geomap <- data.frame(country=country, state=state)
> geomap
  country       state
1     USA  California
2     USA Mississippi
3   China      Fujian
4   China    Shandong
5  Canada      Quebec
6  Canada     Ontario
> # Need to match the column name in join
> colnames(geomap) <- c('country', 'Type')
> joinCO2 <- merge(CO2, countrystate, by=c('Type'))
> head(joinCO2)
         Type Plant  Treatment conc uptake country
1 Mississippi   Mn1 nonchilled   95   10.6     USA
2 Mississippi   Mn1 nonchilled  175   19.2     USA
3 Mississippi   Mn1 nonchilled  250   26.2     USA
4 Mississippi   Mn1 nonchilled  350   30.0     USA
5 Mississippi   Mn1 nonchilled  500   30.9     USA
6 Mississippi   Mn1 nonchilled  675   32.4     USA
>
```

Power and Log transformation

A large portion of machine learning models are based on assumption of linearity relationship (ie: the output is linearly dependent on the input), as well as normal distribution of error with constant standard deviation. However the reality is not exactly align with this assumption in many cases.

In order to use these machine models effectively, it is common practice to perform transformation on the input or output variable such that it approximates normal distribution (and in a multi-variate case, multi-normal distribution).

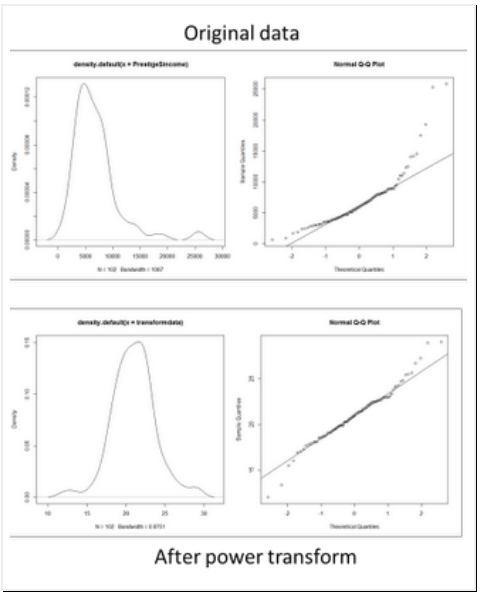
The commonly use transformation is a class call Box-Cox transformation which is to transform a variable x to (x^k - 1) / k where k is a parameter that we need to determine. (when k = 2, this is a square transform, when k = 0.5, this is a square root transform, when k = 0, this will become log y, when k = 1, there is no transform)

Here is how we determine what k should be for input variable x and then transform the variable.

```
> # Use the data set Prestige
> library(cat)
> head(Prestige)
                    education income women prestige census type
gov.administrators      13.11  12351 11.16     68.8   1113 prof
general.managers        12.26  25879  4.02     69.1   1130 prof
accountants             12.77   9271 15.70     63.4   1171 prof
purchasing.officers     11.42   8865  9.11     56.8   1175 prof
chemists                14.62   8403 11.68     73.5   2111 prof
physicists              15.64  11030  5.13     77.6   2113 prof
> plot(density(Prestige$income))
> qqnorm(Prestige$income)
> qqline(Prestige$income)
> summary(box.cox.powers(cbind(Prestige$income)))
bcPower Transformation to Normality

   Est.Power Std.Err. Wald Lower Bound Wald Upper Bound
Y1    0.1793   0.1108          -0.0379           0.3965

Likelihood ratio tests about transformation parameters
                           LRT df         pval
LR test, lambda = (0)  2.710304  1 9.970200e-02
LR test, lambda = (1) 47.261001  1 6.213585e-12
> transformdata <- box.cox(Prestige$income, 0.18)
> plot(density(transformdata))
> qqnorm(transformdata)
> qqline(transformdata)
```



Hopefully I have covered the primary data transformation tasks. This should have set the stage for my next post, which is to train the predict model.

Posted by Ricky Ho at 2:35 PM

+5 Recommend this on Google

Labels: data mining, data preparation, machine learning, predictive analytics, R

7 comments:

BR Deshpande said...

Good summary.

For dimension reduction, we would like to invite you and your readers to beta test our cloud based tool. This works better than PCA for many cases.

Read here for details
http://www.simafore.com/blog/bid/105297/feature-selection-with-mutual-information-part-1-pca-disadvantages

May 21, 2012 at 6:04 AM

Богунов Илья said...

joinCO2 <- merge(CO2, countrystate, by=c('Type'))

should be

joinCO2 <- merge(CO2, geomap, by=c('Type'))

May 23, 2012 at 8:34 AM

mikky99 said...

Reading you blog was a great experience. you always equipped with creative ideas and i love the way you have explained the things..

michel
Trader finance

July 19, 2012 at 2:41 AM

H Vasudev said...

Great post.

There is a great article on multiple imputation on missing values.

It is also about a package in R, called 'Amelia'. (install.packages("Amelia", repos="http://r.iq.harvard.edu", type = "source"))

http://gking.harvard.edu/amelia/

July 22, 2012 at 8:23 AM

H Vasudev said...

Also, try mutate() instead of transform(), for modifying as well as adding derived var. I found the former to be a lot faster than transform() with large datasets.

Cheers

July 22, 2012 at 9:18 AM

Patrick said...

This is a very informative sequence of posts.

Within the last box of R code you write 'load(cat)' when you mean 'load(car)'.

February 14, 2013 at 11:54 PM

Rohit said...

Thanks for the series of articles on Data Science. Really helps

June 30, 2013 at 9:10 AM

Post a Comment

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)