Bayesian decision theory exercises

1 Team members:

- Luis Sierra Muntané
- Àlex Batlle Casellas
- Aleix Torres i Camps

Theoretical exercise:

Exponential family: The definition of what is the exponential family of distributions can be found in many places. For example, Christopher M. Bishop's book *Patter Recognition and Machine Learning* (a.k.a. Bishop's) says that the exponential family of distributions is the set of all distributions that have a density function that can be written as

$$p(x|\boldsymbol{\mu}) = h(x)g(\boldsymbol{\mu})e^{\boldsymbol{\mu}^T \boldsymbol{u}(x)},$$

where $\boldsymbol{\mu}$ is a vector of parameters called *natural parameters* of the distribution. x is a scalar or vector variable, and can be either discrete or continuous. $h(x)$, $g(\boldsymbol{\mu})$ and $\boldsymbol{u}(x)$ are known functions.

In the way that we have defined the exponential family, we don't have to check if it accomplish the properties that all the distributions must have, for example that the integral (or summation) of all its possible values gives exactly one.

Alternatively, other sites like *Wikipedia* give others forms to express the density function. For example:

$$p(x|\boldsymbol{\mu}) = e^{\boldsymbol{M}(\boldsymbol{\mu})\cdot\boldsymbol{u}(x) - A(\boldsymbol{\mu}) + B(x)}$$

Notice that, although being equivalent, the Bishop's one appear explicitly the natural parameters, but in the Wikipedia's one it doesn't. This may cause that the second one gives a bit more freedom in choosing the functions and the natural parameters than the first one. In consequence, may be easier to prove that a random variable form part of the exponential family if it can be written in this alternative form.

The exponential family includes very well known distributions, among others: *normal*, *exponential*, *gamma*, *binomial* (with fixed number of trials), ... and *Poisson* as we will show next.

Poisson distribution: Remember that the Poisson distribution is discrete and starts at $x = 0$, it is frequently used for modeling the number of times that something occurs in a defined interval. It only has one parameter λ and its probability mass function is:

$$p(x|\lambda) = \frac{\lambda^x e^{-\lambda}}{x!}$$

And so we can write it as:

$$p(x|\lambda) = e^{x \log \lambda - \lambda + \log \frac{1}{x!}}$$

Now, associating each term with Wikipedia's expression of the exponential distribution, we have:

$$\begin{aligned} M(\mu) &= \log \lambda \\ u(x) &= x \\ A(\mu) &= \lambda \\ B(x) &= -\log x! \end{aligned}$$

Now, we just have to choose the natural parameter μ. And here we have some freedom of choice because there are many ways to do it. We think that the easiest one is choosing $\mu = \lambda$. Then the functions A and M are the identity and the natural logarithm, respectively.

Therefore, we have shown that the Poisson distribution belongs to the exponential family.

Computer Exercise: Musk

Àlex Batlle, Luis Sierra, Aleix Torres

22/4/2020

Introduction

For this analysis, we will be considering a dataset titled “Musk” created by the AI Group at Arris Pharmaceutical Corporation, consisting of chemical molecules and variables concerning different physical measurements of several low-energy states of those molecules. The aim is to predict whether the aforementioned variables are musk or non-musk, which is a property about the final smell of the molecules, from the 6,598 instances provided. The class of the molecule is given by the final variable, *V169*, and so that is what we will be trying to predict.

To do so, we will apply a Generalised Linear Model using a logit transformation over a set of variables because we considered this to be the most adequate way to model our data given the characteristics of the problem at hand. When trying to explore how other GLMs would fit the data the results were very poor, and the features of the dataset all point towards using this kind of model.

```r
data <- read.table("clean2.data", sep=',')
```

This is what a couple of molecules look like, since there are many configurations for the same molecule. We can see that if one such molecule is marked as a musk, then all configurations are marked as so.

```r
Musk.211 <- data[data$V1 == "MUSK-211",]
Musk.212 <- data[data$V1 == "MUSK-212",]
Musk.213 <- data[data$V1 == "MUSK-213",]
```

The values for the variables from *V3* to *V168* are very irregularly distributed, as can be seen when looking at the boxplots or histograms of the data. Moreover, we can check for missing datapoints, but when doing so we found none, and the authors of the dataset assured that there were no missing attribute fields, so no further treatment of the data was necessary.

Preliminary Considerations

Firstly, it is important to observe that given the 168 attributes (excluding the class) and the relatively high number of datapoints, using a GLM over all of the variables was unfeasible due to the lack of computational power. Despite generating the model without too many issues, when using the *step()* command to eliminate redundant variables the time it took was quite long, and the computer often crashed, and so the command in the chunk below is written as a comment.

```r
variables <- paste("V", 3:168, sep="")
fmla <- as.formula(paste("V169 ~ ", paste(variables, collapse= "+")))
giant_model <- glm(fmla, family = binomial, data = data)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
# summary(giant_model)
# step(giant_model) -- Do not run
```

Therefore, we can consider models using only a subset of the variables in question, like for instance the first n variables, as in the following chunk. Below is a GLM using a small number of variables.

```r
# Choose a small (<20) number of variables
n <- 20

# Formula for succint writing, excluding the factor variables due being the names
```

```r
variables <- paste("V", 3:(n + 2), sep="")
fmla <- as.formula(paste("V169 ~ ", paste(variables, collapse= "+")))

# Simple GLM explaining V169 with respect to the first n variables
mod.simple <- glm(fmla, family = binomial, data = data)
summary(mod.simple)
```

```
##
## Call:
## glm(formula = fmla, family = binomial, data = data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7676  -0.5273  -0.3432  -0.1859   2.9185
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.7236674  0.2990495 -12.452  < 2e-16 ***
## V3          -0.0217651  0.0013534 -16.081  < 2e-16 ***
## V4          -0.0049715  0.0005827  -8.532  < 2e-16 ***
## V5           0.0042726  0.0010899   3.920 8.85e-05 ***
## V6          -0.0084594  0.0008033 -10.531  < 2e-16 ***
## V7          -0.0066820  0.0007916  -8.441  < 2e-16 ***
## V8          -0.0037878  0.0005481  -6.910 4.84e-12 ***
## V9           0.0004460  0.0006717   0.664 0.506701
## V10         -0.0027446  0.0009693  -2.831 0.004635 **
## V11          0.0032039  0.0008226   3.895 9.84e-05 ***
## V12          0.0106293  0.0011152   9.531  < 2e-16 ***
## V13          0.0101855  0.0013441   7.578 3.51e-14 ***
## V14          0.0050708  0.0013191   3.844 0.000121 ***
## V15         -0.0131365  0.0012921 -10.167  < 2e-16 ***
## V16         -0.0063111  0.0017425  -3.622 0.000293 ***
## V17         -0.0036467  0.0016683  -2.186 0.028823 *
## V18         -0.0080221  0.0009275  -8.650  < 2e-16 ***
## V19          0.0027026  0.0007848   3.444 0.000574 ***
## V20          0.0015303  0.0009443   1.621 0.105113
## V21         -0.0037363  0.0013177  -2.835 0.004577 **
## V22         -0.0014068  0.0007692  -1.829 0.067414 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 5671.9  on 6597  degrees of freedom
## Residual deviance: 4361.7  on 6577  degrees of freedom
## AIC: 4403.7
##
## Number of Fisher Scoring iterations: 6
```

That being said, the previous choice of variables is rather arbitrary, and so we could check all possible combination of n variables and compare the models that result from such choices, with a total of ${}_{167}C_n$ models, which is a relatively large amount. Alternatively, we can consider techniques of dimensionality reduction in order not to have to resort to arbitrary choices of variables for the model. For instance, we could randomly sample the n vaiables, or we could use principal component analysis, as in the following chunk.

```r
# PCA decomposition of the data
pca_data <- princomp(data[, -1:-2])
```

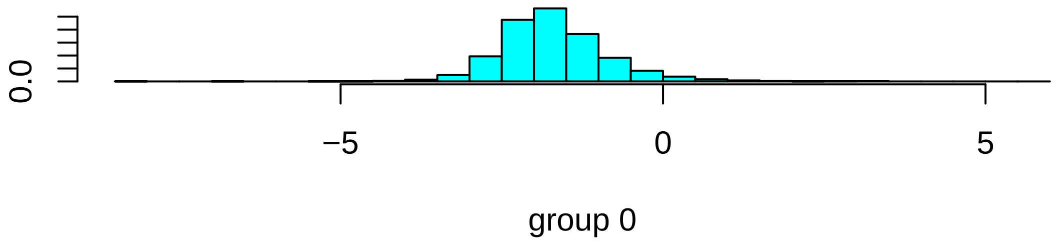
We can see from the results of the PCA that the first 35 principal components account for just over 95% of the total variability in the data. As such, we can produce a GLM using the principal components as variables to predict the value for the final *V169* variable.

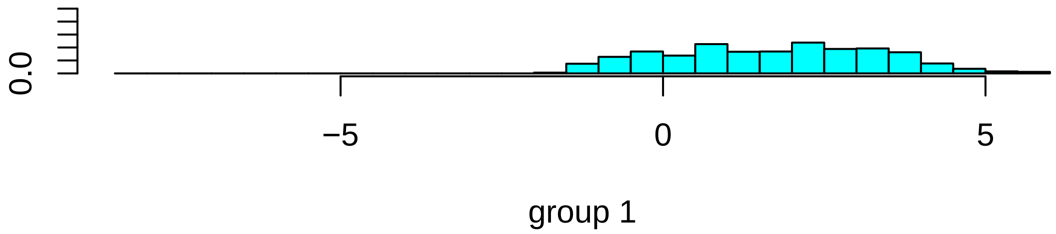
It is important to take into account the fact that PCA by design is a dimensionality reduction method based around extracting the maximum variance and correlations between the datapoints at each step. As such, it is possible that the principal components are not the best at predicting the value for a single variable, such as *V169*, especially if there are many other variables which are highly correlated between them but which are not strongly related to out target variable. Below is a preliminary GLM from the results of a GLM using *dim*=35 dimensions.

```r
indices <- get_pca_ind(pca_data)

# Coordinates to use as variables
dim <- 35
coord <- indices$coord
reduced_data <- cbind(data[169], coord[,1:dim])
dimensions <- paste("Dim.", 1:dim, sep="")
pca_fmla <- as.formula(paste("V169 ~ ", paste(dimensions, collapse= "+")))

# GLM model from principal components using dim principal components
glm_pca <- glm(pca_fmla, family = binomial, data = reduced_data)

# Eliminating redundancy using Akaike Information Criterion (AIC)
pca_out <- capture.output(step(glm_pca))
```

We can also create models reducing the dimensionality of the data by performing a Linear Discriminant Analysis (LDA) to project the data onto a lower-dimensional space, as shown in the following chunk.

```r
lda_model <- lda(V169 ~ ., data = data[,-1:-2])
plot(lda_model)
```





Model Evaluation:

The previously exposed models were all trained using the entirety of the data, and so they are prone to overfitting problems, given that there may be patterns in the data which are due to random chance rather than there being meaningful relationships between the variables. To sidestep such problems, we will use a method of evaluation of the models using K-fold cross validation.

This method is superior to Leave-One-Out Cross Validation (LOOCV) in that it is computationally less expensive, and given the relatively large amount of datapoints being handled, it preferable to prioritise computational considerations over precision ones. Moreover, James et al. (2014)[1] showed that K-fold CV often leads to more accurate values of the test error than LOOCV.

Below, we separate the data into training and learning to evaluate the performance of the models, in a 2:1 training/testing ratio, and after that perform the K-fold CV over the learning data. After experimenting with the method for a while, it seemed clear that using a value of $K = 5$ was large enough, since larger values did not produce better models.

```
# Randomly separate into learning data and test data
set.seed(123)
sample_size <- floor(2/3 * nrow(data))
learning_indices <- sample(seq_len(nrow(data)), size = sample_size)
learning.data  <- data[learning_indices, ]
test.data <- data[-learning_indices, ]

# Using learning data for cross validation
K <- 10
```

```r
# Transforming V169 to factor variable to perform cv
glm_learning.data <- learning.data
glm_learning.data$V169 <- factor(glm_learning.data$V169)

glm_model <- train(form = V169 ~ .,
               data = glm_learning.data[,-1:-2],
               trControl = trainControl(method = "cv", number = K),
               method = "glm",
               family = "binomial")
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
# Table of results along with accuracy
table(glm_learning.data$V169, predict(glm_model))
```

```
##
##          0    1
##    0 3668   53
##    1  114  563
```

```r
glm_model
```

```
## Generalized Linear Model
##
## 4398 samples
##  166 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 3959, 3958, 3958, 3959, 3959, 3957, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9447473  0.7820201
```

```r
# Real accuracy as calculated over test data
glm_predicted <- glm_model %>% predict(test.data, type = "raw")
glm_table <- table(test.data$V169, glm_predicted)
glm_accuracy <- (glm_table[1,1] + glm_table[2,2])/sum(glm_table)
glm_table
```

```
##    glm_predicted
##        0    1
##   0 1820   40
##   1   60  280
```

```r
glm_accuracy
```

```
## [1] 0.9545455
```

We can do the same to the data decomposed using PCA, as shown below.

```r
# Choice of principal components to be used
dim <- 20

# PCA on data
pca_learning.data <- princomp(learning.data[-1:-2])
principal_coord <- get_pca_ind(pca_learning.data)$coord

# Adjoining target variable to matrix of principal components
reduced_learning_data <- cbind(learning.data[169], principal_coord[,1:dim])

# Using learning data for cross validation
K <- 10

# Transforming V169 to factor variable to perform cv
reduced_learning_data$V169 <- factor(reduced_learning_data$V169)

pca_glm_model <- train(form = V169 ~ .,
               data = reduced_learning_data,
               trControl = trainControl(method = "cv", number = K),
               method = "glm",
               family = "binomial")

# Changing names of columns of learning data to those of the PCA
pca_test.data <- matrix((as.double(as.matrix(test.data[,-1:-2]))), nrow = 2200, ncol = 167) %*% loading
colnames(pca_test.data)[1:n] <- colnames(reduced_learning_data)[-1]

# Real accuracy as calculated over test data
pca_glm_predicted <- pca_glm_model %>% predict(pca_test.data, type = "raw")
pca_glm_table <- table(test.data$V169, pca_glm_predicted)
pca_glm_accuracy <- (pca_glm_table[1,1] + pca_glm_table[2,2])/sum(pca_glm_table)
pca_glm_table
```

```
##    pca_glm_predicted
##        0    1
##   0 1860    0
##   1  302   38
```

```r
pca_glm_accuracy
```

```
## [1] 0.8627273
```

It is interesting to see that it is the case that the final GLM model is better than the final model after the dimensionality reduction, which makes sense given that there is a loss of information when considering only a subset of all principal components.

References

[1] James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2014. An Introduction to Statistical Learning: With Applications in R. Springer Publishing Company, Incorporated.