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 problem

Now we want to consider the problem 'Musk' set in the statement: given a set of molecule descriptions via some chemical and structural parameters, classify them into two different classes: positive (being a Musk molecule) and negative (not being a Musk molecule). All code from now on can be found in the `.Rmd` file included in the compressed file. It also contains some additional chunks not listed here, which were used solely to try and see how different models behaved. In order to do this, we first of all read and pre-process the data:

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

We can't see any missing value or strange behaviour of the variables to our understanding of the topic. Then, we will not comment on anything regarding data cleaning or missing value imputation.

We first of all want to create two different models to gain some insight on the problem. We start our journey inspecting a model using a small number of the variables:

```
mod.simple <- glm(V169 ~ V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 +
                    V11 + V12 + V13 + V14 + V15 + V16 + V17 + V18 +
                    V19 + V20, family = binomial, data = data)
summary(mod.simple)
```

This model appears to ditch V9 as its p-value is quite greater than 0.05: we will use this significance level all through the exercise. We now want to see how can we reduce the dimensionality of the data in order to make simpler models. So, we try to do a PCA and use its results into a GLM:

```
out.PCA <- princomp(data[,3:169])
summary(out.PCA)
```

We can see that using 10-11 variables we get close to the 80% of data variance, so now we will use these 10 first principal components into another GLM:

```
pca_data <- princomp(data[,-1:-2])
ind <- get_pca_ind(pca_data)
coord <- ind$coord
reduced_data <- cbind(data[169], coord[,1:10])
glm_pca <- glm(V169 ~ Dim.1 + Dim.2+ Dim.3 + Dim.4 + Dim.5 + Dim.6
                      + Dim.7 + Dim.8 + Dim.9 + Dim.10,
                      family = binomial, data = reduced_data)
summary(glm_pca)
```

The p-values of the fifth and seventh component are greater that 0.05, and using the `step()` function, only the fifth is left out in the end.

Now we pass on to model testing. We are aware of the fact that each molecule is described by 169 features, although variables 1, 2 and 169 are names and the class, respectively. This is too many variables for a feasible realization of the full model, so we will not consider this one. We will first of all segment the data set into a training set and a testing set. This way, we will be able to assess the different models' performance on this testing set (we set a seed in order to make our results reproducible).

```
set.seed(123)
sample_size <- floor(2/3 * nrow(data))
train_indices <- sample(seq_len(nrow(data)), size = sample_size)
train.data  <- data[train_indices, ]
test.data <- data[-train_indices, ]
```

Now we will define the models using only the training data and we will evaluate their performance through testing. Note that we will always use logistic regression and binomial models, as this is a binary classification problem.

Without dimensionality reduction:
Now we define two different models that do not use dimensionality reduction nor any feature extraction. We are just using some of the variables. For the first one, we use the 10 first variables (excluding V1 and V2, that is, V3-12):

```
simple10_glm_model <- glm(V169 ~ V3 + V4 + V5 + V6 + V7 + V8 + V9
                                 + V10 + V11 + V12, family =
                                 binomial, data = train.data)
```

Now we just make predictions on the testing data,

```
prob_predictions <- simple10_glm_model %>% predict(test.data,
                                                   type = "response")
predicted.classes <- ifelse(prob_predictions > 0.5, 1, 0)
mean(predicted.classes == test.data$V169) # Accuracy of prediction
```

and the results indicate that we have accurate predictions on 85.77% of the individuals in the testing data. We want to see if adding variables to this model makes it be any better; for an example, we are adding 10 more variables, so we are using V3-22:

```
simple20_glm_model <- glm(V169 ~ V3 + V4 + V5 + V6 + V7 + V8 + V9
                                 + V10 + V11 + V12 + V13 + V14 +
                                 V15 + V16 + V17 + V18 + V19 + V20
                                 + V21 + V22, family = binomial,
                                 data = train.data)
prob_predictions <- simple20_glm_model %>% predict(test.data,
                                                   type = "response")
predicted.classes <- ifelse(prob_predictions > 0.5, 1, 0)
mean(predicted.classes == test.data$V169) # Accuracy of prediction
```

This way, we get to 86.00% of the predictions on the testing data being correct. So, we didn't get much better.

Using dimensionality reduction:
Up until this point we have been fitting models that do not use any dimensionality reduction or feature extraction. We now bring this into the model, performing a PCA on the data:
We first perform this analysis and we can observe some things:

```
pca_training_data <- princomp(train.data[-1:-2])
```

First of all, doing a `summary()` of this PCA we can see it is very similar to the one with all the data: the different dimensions entail approximately the same cumulative proportion of variance in both cases. So, we now build the new training data set, where the different observations are explained by the first 10 principal components, and we fit a GLM with these PC's as explanatory variables:

```
principal_coord <- get_pca_ind(pca_training_data)$coord
reduced_training_data <- cbind(train.data[169],
                               principal_coord[,1:10])
pca10_glm_model <- glm(V169 ~ Dim.1 + Dim.2 + Dim.3 + Dim.4 +
                              Dim.5 + Dim.6 + Dim.7 + Dim.8 +
```

```
                                    Dim.9 + Dim.10, family = binomial,
                                    data = reduced_training_data)
```

Now we transform the testing data using the `loadings` attribute of the PCA object in R,

```
pca_test.data <- matrix((as.double(as.matrix(test.data[,-1:-2]))),
                                        nrow = 2200, ncol = 167) %*%
                                        loadings(pca_training_data)[]
colnames(pca_test.data)[1:10] <- colnames(reduced_training_data)[-1]
```

and we make predictions on the testing data and we test its accuracy:

```
prob_predictions_pca <- pca10_glm_model %>%
        predict(as.data.frame(pca_test.data), type = "response")
predicted_pca.classes <- ifelse(prob_predictions_pca > 0.5, 1, 0)
mean(predicted_pca.classes == test.data$V169)
```

We get an accuracy of 79.27%, so not much better also. If we take 11 instead of 10 variables, the accuracy grows to 84.27%, which is a little better. From here on, the accuracy does not seem to grow that much: for example, using 20 PC's it only gets to 87.04%.