# Computational Statistics HW4

*Andre Duarte*

*April 21, 2017*

## Load data and basic pre-analysis

We are working with the Mashable data set in order to build and compare two classification algorithms: standard logistic regression and Bayesian logistic regression.

```
# Get data
train <- read.csv("OnlineNewsPopularityTraining.csv", stringsAsFactors = F)

# Remove variables
train$shares <- NULL
train$url <- NULL
train$timedelta <- NULL

# Verify that the data is clean (ie no NAs)
#sapply(train, function(x) sum(is.na(x))) # not shown for simplicity

# Look at the data (distribution of popular)
table(train$popular)
```

```
##
##     0     1
## 25280  6436
```

```
print(1-sum(train$popular)/nrow(train))
```

```
## [1] 0.797074
```

The set is very unbalanced! If we predict only 0 all the time, we will have an accuracy of approximately 80%! This is exactly what happens when we fit a standard logistic regression model (not shown here). The first thing we need to do is to create a balanced data set. For that, we undersample from the majority category.

```
# Let's create a 50-50 split sample
pos_idx <- which(train$popular == 1)
undersample <- append(pos_idx, sample(which(train$popular == 0), length(pos_idx), F))
train_under <- train[undersample,]

# Look at the data (distribution of popular)
table(train_under$popular)
```

```
##
##    0    1
## 6436 6436
```

```
print(1-sum(train_under$popular)/nrow(train_under))
```

```
## [1] 0.5
```

We can see now that the set is balanced. Let's fit some models.

## Standard logistic regression

We can fit a standard logistic regression model using the `glm` function.

```
# Fit glm using all variables
glm.fit <- glm(popular ~ ., data = train_under, family = binomial)
#summary(glm.fit) # not shown for simplicity

# Predict probabilities
glm.probs <- predict(glm.fit, type = "response")

# Tranform into {0, 1} predictions
glm.pred.train <- ifelse(glm.probs > 0.5, 1, 0)
table(glm.pred.train, train_under$popular)
```

```
##
## glm.pred.train    0    1
##              0 4276 2437
##              1 2160 3999
```

```
# Get misclassification rate
misclass <- sum(glm.pred.train != train_under$popular)/length(train_under$popular)
print(paste('Accuracy', 1-misclass))
```

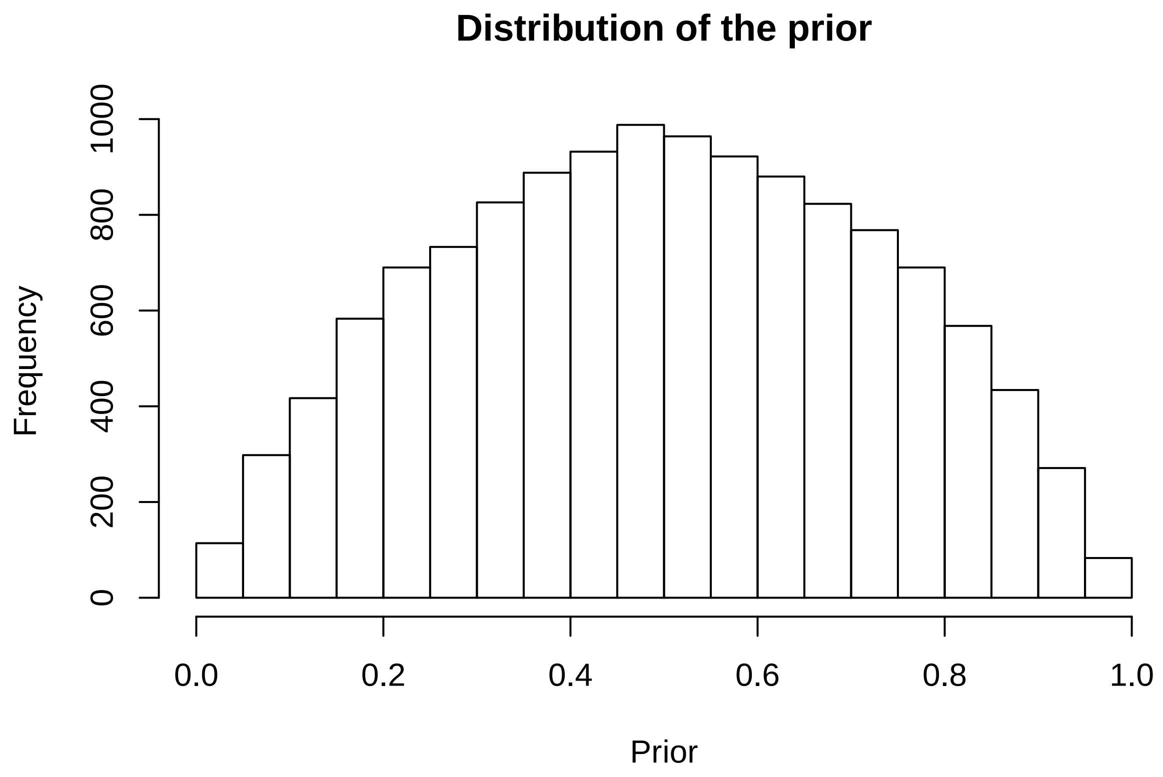
```
## [1] "Accuracy 0.642868241143567"
```

Using this model, we get a training accuracy of around 64%! This means that the model is actually learning something, since the accuracy is greater than 50%. Let's see if we can improve on this baseline by using a Bayesian approach to this problem.

## Bayesian logistic regression

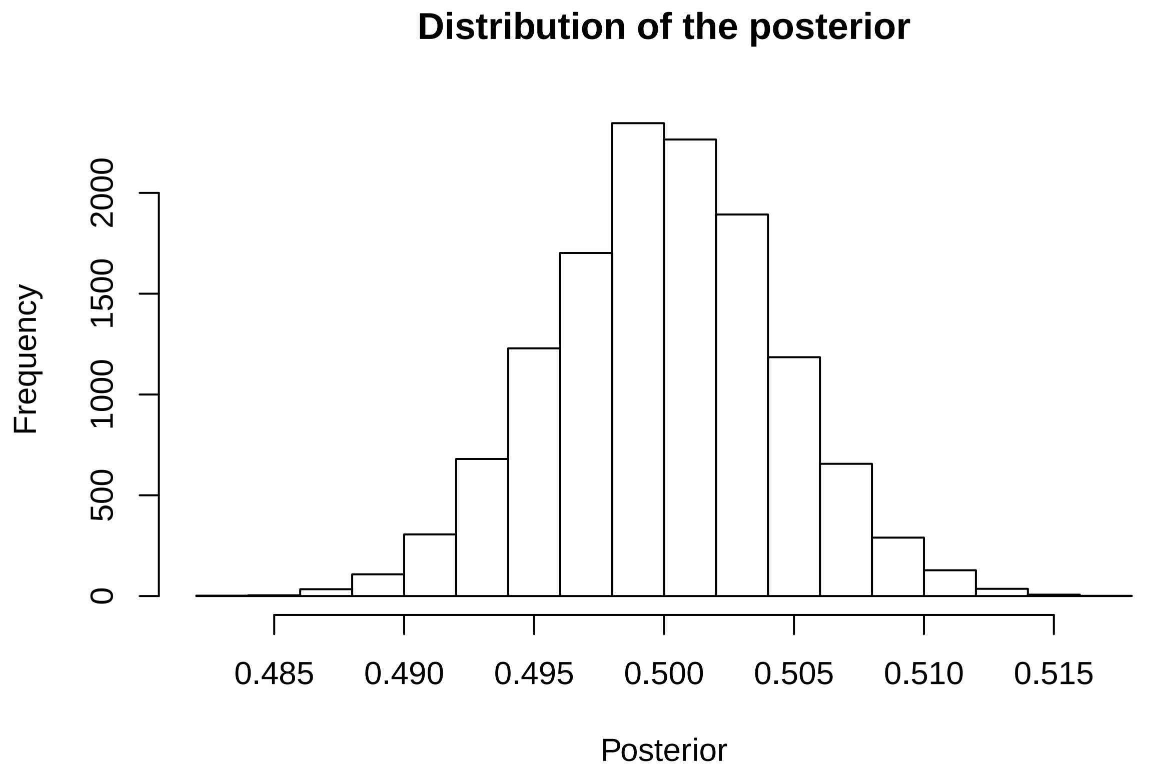
A $Beta(2, 2)$ has expected mean of 0.5, which is what we want since the data set is now balanced (this is an uninformative prior). We will create $n =$ '$nrow(train_under)$' priors from a $\beta(2, 2)$ distribution.

```
alpha <- 2
gamma <- 2
n <- nrow(train_under)
prior <- rbeta(n, 2, 2)
hist(prior, main="Distribution of the prior", xlab="Prior", ylab="Frequency")
```

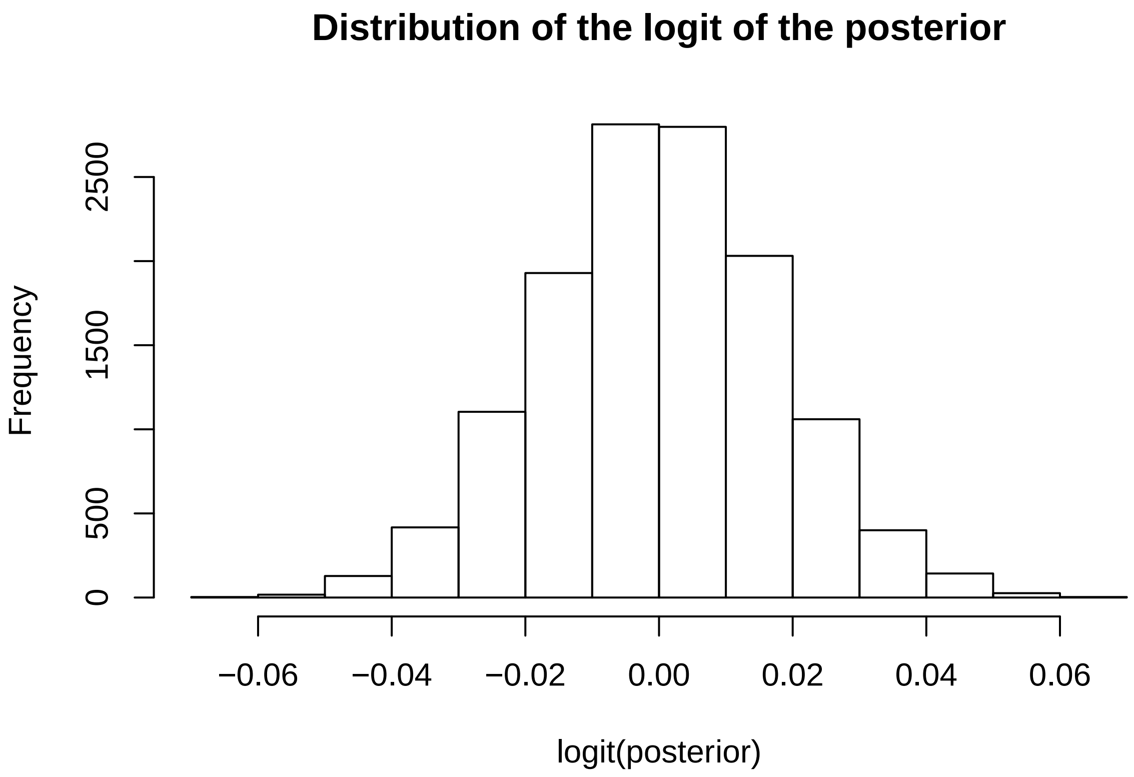


We can see that the distribution is centered around 0.5 and is "fat", which is what we want for this non-informative prior. The posterior will follow a Beta distribution with parameters $\alpha + \frac{n}{2}$ and $\gamma + n - \frac{n}{2} = \gamma + \frac{n}{2}$. We can therefore construct $n$ posteriors and take the logit of each, then look at their distribution.

```
posterior <- rbeta(n, alpha+n/2, gamma+n/2)
hist(posterior, main="Distribution of the posterior", xlab="Posterior", ylab="Frequency")
```



```r
logits <- log(posterior/(1-posterior))
hist(logits, main="Distribution of the logit of the posterior", xlab="logit(posterior)", ylab="Frequenc
```



We can see that the distribution is not very "thin" and centered around 0! In other words, the model will most likely always predict 0. Indeed, we have $\text{logit}(p_j) \simeq y_j = X\beta$. This is due to the sample size, which is very large. In this condition, the Bayesian approach closely mirrors the frequentist approach of always choosing one outcome.

We now construct our model to verify this hypothesis.

```r
train_bayes <- train_under
train_bayes$popular <- logits
glm.fit.bayes <- glm(popular~., data=train_bayes)
#summary(glm.fit.bayes) # not shown for simplicity

glm.probs.bayes <- predict(glm.fit.bayes, type = "response")

# Tranform into {0, 1} predictions
glm.pred.train.bayes <- ifelse(glm.probs.bayes > 0.5, 1, 0)
table(glm.pred.train.bayes, train_under$popular)
```

```
##
## glm.pred.train.bayes    0    1
##                    0 6436 6436
```

```r
# Get misclassification rate
misclass.bayes <- sum(glm.pred.train.bayes != train_under$popular)/length(train_under$popular)
print(paste('Accuracy', 1-misclass.bayes))
```

```
## [1] "Accuracy 0.5"
```

As we can see, the final training accuracy using this model is exactly 0.5! We always predict 0.

## Testing the models using test data

We now test both models using test data. First, we load the data and remove the variables we don't want.

```r
# Test
test <- read.csv("OnlineNewsPopularityTest.csv", stringsAsFactors = F)

# Remove variables
test$shares <- NULL
test$url <- NULL
test$timedelta <- NULL

table(test$popular)
```

```
##
##    0    1
## 6285 1643
```

```r
print(1-sum(test$popular)/nrow(test))
```

```
## [1] 0.7927598
```

This set is also unbalanced (around 80-20). It will be interesting to see whether our standard model actually learned useful features, or if it overfit.

```r
glm.probs.test <- predict(glm.fit, test[,1:ncol(test)-1], type="response")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

```r
# Tranform into {0, 1} predictions
glm.pred.test <- ifelse(glm.probs.test > 0.5, 1, 0)
table(glm.pred.test, test$popular)
```

```
##
## glm.pred.test    0    1
##             0 4176  623
##             1 2109 1020
```

```r
# Get misclassification rate
misclass.test <- sum(glm.pred.test != test$popular)/length(test$popular)
print(paste('Accuracy', 1-misclass.test))
```

```
## [1] "Accuracy 0.65539858728557"
```

We can see that we get an accuracy of around 65%, which is lower than we would expect. The features that were "learned" in the training data do not reflect the testing data, so we likely overfit. Let's see how the Bayesian model fares.

```r
glm.probs.test.bayes <- predict(glm.fit.bayes, test[,1:ncol(test)-1], type="response")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

```r
# Tranform into {0, 1} predictions
glm.pred.test.bayes <- ifelse(glm.probs.test.bayes > 0.5, 1, 0)
table(glm.pred.test.bayes, test$popular)
```

```
##
## glm.pred.test.bayes    0    1
##                     0 6285 1643
```

```
# Get misclassification rate
misclass.test.bayes <- sum(glm.pred.test.bayes != test$popular)/length(test$popular)
print(paste('Accuracy', 1-misclass.test.bayes))
```

```
## [1] "Accuracy 0.792759838546922"
```

Here, the Bayesian framework again predicted only 0 (as we were expecting). So the test accuracy is only as high as the unbalanced-ness of the data set.

## Last words

In conclusion, we have seen that due to the amount of data here, the Bayesian method on the balanced data set ends up being equivalent to the standard method on the entire data set. Although it seemed like the standard logistic regression model had learned some useful features from the training data, we saw that it was actually overfitting, and the performance on the test set was not great. Both methods have their own merits, but I think the Bayesian approach is more data-driven. We use some prior knowledge of the distributions in order to try to get more accurate predictions. In this case, the Bayesian approach did not yield conclusively better results than a naive logistic regression model without undersmapling of the data, but it also did not overfit to the training data.