Supervised Classification Models for Text Analysis

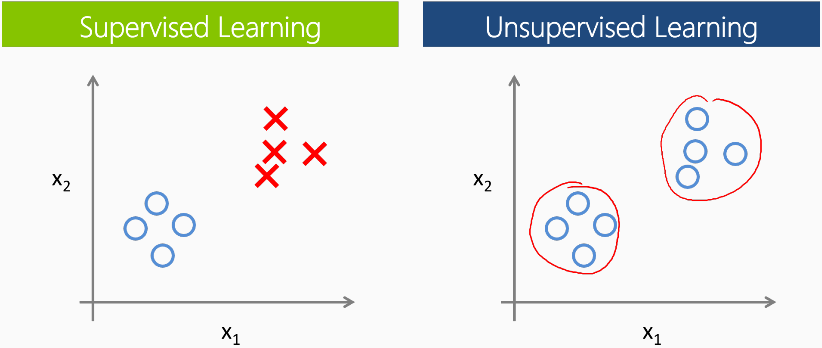
Bamberg Summer Institute in Computational Social Science

Carsten Schwemmer, University of Bamberg

2019-08-01

Supervised text classification

Supervised vs unsupervised



- “all models are wrong but some models are useful.”
- *naive* assumption: features are independent of each other

$$P(\text{label} \mid \text{feature}) = \frac{P(\text{feature} \mid \text{label}) \cdot P(\text{label})}{P(\text{feature})}$$

Reload our DonorsChoose data

```r
library(tidyverse)
library(quanteda)
library(caret)
load('data/dfm_donor.Rdata')
dim(dfm_donor)
```

```
## [1] 10000  6751
```

We want to predict whether a donation request received funding:

```r
docvars(dfm_donor) %>% count(funded)
```

```
## # A tibble: 2 x 2
##   funded     n
##    <dbl> <int>
## 1      0  2409
## 2      1  7591
```

Training and test sets

- in order to train a model, we need to split our data into training and test sets
- the classifier will learn from the training set. The testset is used to evaluate its performance on unseen data
- there is no optimal solution for the proportions to split train and test data (see bias-variance tradeoff).

Training a model

```
set.seed(1337) # for replication

train <- dfm_sample(dfm_donor, size = 8000) # 80% of the data
test <- dfm_donor[!docnames(dfm_donor) %in% docnames(train), ]

# run the model
donor_nb <- textmodel_nb(x = train, y = docvars(train)$funded)
```

Making sense of the model - important features

We create a function to find the most important terms for correctly predicting whether a donation request received funding:

```
imp_features <- function(nb_model, n = 10) {
  # PcGw = probability of class given the word
  features <- t(nb_model$PcGw) %>%  # transpose
    as.data.frame() %>% rownames_to_column('feature') %>%
    gather('label', 'prob_class' , -feature) %>%  # tidy data
    group_by(label) %>% top_n(n, prob_class) %>% # max probability
    slice(1:n) %>%
    ungroup() %>% arrange(label, desc(prob_class)) # sort
  return(features)
}
```

Making sense of the model - important features

```r
imp_features(donor_nb, n = 5)
```

```
## # A tibble: 10 x 3
##    feature   label prob_class
##    <chr>     <chr>      <dbl>
##  1 intuitive 0          0.903
##  2 rated     0          0.879
##  3 france    0          0.875
##  4 associate 0          0.875
##  5 parcc     0          0.875
##  6 harvey    1          0.965
##  7 failing   1          0.896
##  8 teammates 1          0.893
##  9 renewable 1          0.893
## 10 arrived   1          0.893
```

Confusion matrix & accuracy

	Reference	
Predicted	*TRUE*	*FALSE*
TRUE	True Positives	False Positives
FALSE	False Negatives	True Negatives

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

Other metrics for performance evaluation

- accuracy is often not a good measure of performance, especially for imbalanced classes.
- multiple alternatives are available, e.g. balanced accuracy

$$Sensitivity = \frac{TP}{TP + FN}; Specificity = \frac{TN}{TN + FP}$$
$$Balanced\ Accuracy = \frac{Sensitivity + Specificity}{2}$$

Evaluating model performance

```r
pred_labels <- predict(donor_nb, newdata = test) %>% as.factor()
true_labels <- docvars(test)$funded %>% as.factor()
stats <- confusionMatrix(pred_labels, true_labels)
stats$table # confusion matrix
```

```
##           Reference
## Prediction    0    1
##          0  227  502
##          1  244 1027
```

```r
stats$byClass # performance metrics
```

```
##          Sensitivity          Specificity       Pos Pred Value
##            0.4819533            0.6716808            0.3113855
##       Neg Pred Value            Precision               Recall
##            0.8080252            0.3113855            0.4819533
##                   F1           Prevalence       Detection Rate
##            0.3783333            0.2355000            0.1135000
## Detection Prevalence    Balanced Accuracy
##            0.3645000            0.5768171
```

Cross-validation

- repeat the model training on several train/test splits
- assess performance across all runs
- we'll use a simple k-fold variant, where k equalts the number if splits

Running validation over 10 splits

```r
cross_val(input_dfm = dfm_donor, labels = 'funded' , train_size = 9000,
          nr_runs = 10, what = 'Balanced Accuracy', pos_class = '1')
```

```
## [1] "Balanced Accuracy run 1 : 0.589"
## [1] "Balanced Accuracy run 2 : 0.571"
## [1] "Balanced Accuracy run 3 : 0.585"
## [1] "Balanced Accuracy run 4 : 0.603"
## [1] "Balanced Accuracy run 5 : 0.573"
## [1] "Balanced Accuracy run 6 : 0.59"
## [1] "Balanced Accuracy run 7 : 0.544"
## [1] "Balanced Accuracy run 8 : 0.571"
## [1] "Balanced Accuracy run 9 : 0.584"
## [1] "Balanced Accuracy run 10 : 0.581"
## [1] "Average Balanced Accuracy : 0.579"
```

What to do when model performance is bad

- adjust feature space
- adjust model parameters
- try different model(s)
- in our case: think about other factors that might be related to the funding success of donation requests

Other approaches we didn’t cover

- averaging predictions of multiple models (e.g. support vector machines, random forest models)
- optimizing hyperparameters (e.g. distribution assumptions of our naive-bayes model)
- using features beyond bag-of-words (e.g. word embeddings)

When should you use supervised classification?

- if your goal is the best predictive power, supervised models are reasonable choices
- for many social science applications, supervised models are used to infer labels for a larger dataset from a smaller trainingset, which are then used in downstream tasks
- it might be helpful to compare results with dictionary-based approaches

Questions?