Additional Material for Supervised Learning

This Additional Material will use real life examples to deep-dive into concepts that were briefly explained in the Supervised Learning Spoken Tutorial.

Information about various packages:

- e1071 - This library contains various miscellaneous functions for statistical operations like clustering, support vector machines etc. It is a dependency for using the caret package.
- caret - This package is a comprehensive framework for building machine learning models in R. It contains the function to obtain the confusion matrix of a machine learning predictor.

Train-test split:

This procedure involves splitting the original data into two distinct components.

- train set - It is used to fit the parameters of the classification model.
- test set - Trained model is used to predict responses for the observations in the test set.

The test set is used to evaluate the performance of the model. It is also known as a validation set.

Bayes Theorem:

Anish wanted to check the probability of him carrying an umbrella to work, given the fact that it was raining. What he already knew was the following:

- He knew how often it rained i.e $P(Rain)$ is given.
- He knew how often he carried his umbrella to work - $P(Umbrella)$.
- Being a Math aficionado, he also knew how often it rained when he carried his umbrella to work - $P(Rain|Umbrella)$.

What came to Anish's rescue here is a very simple formula devised by Thomas Bayes:

$$P(Umbrella|Rain) = \frac{P(Rain|Umbrella) \times P(Umbrella)}{P(Rain)}$$

With a very important condition that - $P(Rain)$ should never be to equal zero.

However, this is a small problem with Bayes Theorem. As the number of features grow, it becomes increasingly harder to compute these probabilities. It was then that a Naive assumption was made. According to the assumption all the classes are conditionally independent and are equal in value. This brings us to a simple solution - Naïve Bayes.

Naïve Bayes:

Let us assume that we have built a model that will classify news headlines into Business or Non-Business. For example, we have the following sentences:

- Jindal completes BFC takeover (Business).
- Chettri becomes leading goal-scorer (Not Business).

Now, the assumption that Naïve Bayes makes is that all the words in these sentences are independent of each other and are of equal worth. To illustrate:

$$P(Tatamountstakeover|Business) = P(Tata|Business) * P(mounts|Business) * P(takeover|Business)$$

As we can see, we calculate the probability that each word belongs to a headline, given that it is a business headline. Then we multiply each of these probabilities to get our final result.

Here we face a potential problem - $P(Tata|Business)$ and $P(mounts|Business)$ are zero because our model hasn't seen these words before. This can be resolved by using `Laplace smoothing`, where we add 1 to the count of every unique word that we currently have, so that the probability never becomes zero. Using this, we can predict the class of the headline even if some of the words were not seen before.

We will now implement a simple `naiveBayes()` command using the `e1071` package, which we used in the Spoken Tutorial as well.

naiveBayes() in e1071 package:

In this Spoken Tutorial we ran a command to create a Naïve Bayes classifier using the parameters, formula and data. Here is another example, using the built-in iris dataset. Here we have included two other parameters which will be explained below:

```r
library(e1071)
data(iris)
model <- naiveBayes(Species ~ ., data = iris, laplace = 2, na.action(na.omit))
pred <- predict(model, iris[,-c(5)])
table(pred, iris$Species)
```

```
##
## pred         setosa versicolor virginica
##   setosa         50          0         0
##   versicolor      0         47         3
##   virginica       0          3        47
```

- `laplace`: A positive double value that controls `Laplace smoothing`. The default value is zero, which disables smoothing.
- `na.action`: Specifies action to be taken if NA values are found. Here we have used na.omit, which omits NA values if they are found.

Now that we have explored the `naiveBayes()` function, let us turn our attention to the `confusion matrix`. We have used the `confusion matrix` to estimate the performance of our model.

Confusion Matrix:

A `confusion matrix`, also known as an `error matrix`, is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one. Each row of the matrix represents the instances in an actual class while each column represents the instances in a predicted class, or vice versa.

Let us understand this definition with an example. We will import the caret library and construct the `confusion matrix` for our model that we trained above:

```
library(caret)
confusionMatrix(pred,iris$Species)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   setosa versicolor virginica
##   setosa         50          0         0
##   versicolor      0         47         3
##   virginica       0          3        47
##
## Overall Statistics
##
##                Accuracy : 0.96
##                  95% CI : (0.915, 0.9852)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.94
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: setosa Class: versicolor Class: virginica
## Sensitivity                 1.0000            0.9400           0.9400
## Specificity                 1.0000            0.9700           0.9700
## Pos Pred Value              1.0000            0.9400           0.9400
## Neg Pred Value              1.0000            0.9700           0.9700
## Prevalence                  0.3333            0.3333           0.3333
## Detection Rate              0.3333            0.3133           0.3133
## Detection Prevalence        0.3333            0.3333           0.3333
## Balanced Accuracy           1.0000            0.9550           0.9550
```

Apart from accuracy, there are a lot other parameters that we can use to measure how good our model is. Let us understand each one of them:

- `Recall (aka Sensitivity)`: Fraction of correct instances that are retrieved by the model. Represented by
$$\frac{TP}{TP + FN} \tag{1}$$
where TP is true positive and FN is false negative. It is used to check number of positive instances that were correctly detected.

- `Precision`: Fraction of instances that are correctly classified by the model. Represented by

$$\frac{TP}{TP + FP} \tag{2}$$

where TP is true positive and FP is false positive. It is used to check accuracy of positive predictions.

- `Specificity`: Measures how exact the assignment to the positive class is. Represented by

$$\frac{TN}{FP + TN} \tag{3}$$

where TN is true negative and FN is false positive. It is used when we need to check the accuracy of negative predictions.

We now know that these measures can give us a better idea of whether our model succeeds or not. Let us now explore one of the fundamental paradigms - the Precision-Recall tradeoff, and learn about when we need higher precisions and when we need higher recalls.

Precision Recall Tradeoff:

The way precision and recall are structured is such that if we want to increase the `precision` of our model, its `recall` drops and vice versa. This is called the precision recall tradeoff. Understanding where `precision` is important and where `recall` is important is fundamental in building an accurate and robust model.

High Precision:

Supposing we are watching a movie on an OTT platform like PrimeVideo. The age classifier in PrimeVideo must be able to classify with high precision whether the movie is children friendly or not. Here we are not worried so much about the false negatives i.e. if the model wrongly classify a movie as being adult-only, it would be erring on the side of caution. But if the model wrongly classifys an adult-only movie as children friendly (false-positive), it can have disastrous consequences. Hence, in scenarios like these, we should prioritise `precision` over `recall`.

High Recall:

Consider a cancer detection model, that classifies tumours as either malignant or non-malignant. Here, the prioirity should be about reducing false negatives i.e. classifying a tumour as non-malignant when it is infact malignant, because if this happens lives could be lost. But if we wrongly classify a benign tumour as malignant, then the cost of such a mistake is much lesser. Thus, in scenarious where there are output sensitive predictions, we must prioritise high `recall`.