# Implementing a Naive Bayesian Classifier

*by Jon Vegard Jansen and Robin Tollisen*

We implemented the classifier using C# and Visual Studio 2010. The program consists of four source files.

When the program starts, it reads 70% of the files it has been given (approx. 14'000). Then it builds up a dictionary using these documents for each word found, counting how many times each word appears. It also builds up 20 separate dictionaries, containing the words for each category, and counts how may times each word shows up.

After the dictionaries have been built you can enter several commands, one of them is "AnalyzeAll". This makes it go through each of the 30% remaining documents (approx. 6'000), and try to put them in one of the 20 categories. In order to calculate the probability for each document, the program goes through every single word found, and checks the probability for them to belong to a proposed category. Then, this value is multiplied by the prior probability, which is found by checking the word against the vocabulary. These two steps are done for each category, and the category with the highest probability is then selected as the most likely.

After the program is done, it checks its own performance against the actual classifications since these are known for the test set, (this is actually done during the calculation), and gives the result as text.

When using 70% of the documents from each category as training, and then analyzing the remaining 30% we get the following result in total:

```
Correctly classified: 5073 of 6000 - 84,55%
```

The highest error rate was in the category sci.electronics, with the following results:

```
Out of 300 documents in sci.electronics.
201 were correctly classified.
```

The lowest error rate was in the category soc.religion.christian with the following results:

```
Out of 300 documents in soc.religion.christian.
299 were correctly classified.
1 were wrongly classified.
```

# Appendix 1 - Main Algorithm

```
private string CalculateCategory(Dictionary<string, int> candidateDocument, DataSet dataset)
      {
         // Vocabulary <- All distinct words in all documents.
         Dictionary<string, int> vocabulary = dataset.Vocabulary;
         Dictionary<string, double> highestEffect = new Dictionary<string, double>();

         double pH = 0;
         string max_group = "";
         double max_p = 0;
         foreach (KeyValuePair<string, Dictionary<string, int>> categoryWords in
dataset.getDataSet)
         {
Dictionary<string, int> categoryWithDocumentCount = dataset.CategoryDocumentCount;
            pH = (double)categoryWithDocumentCount[categoryWords.Key] / (double)
dataset.documentCount;
            Dictionary<string, double> wordWithLikelyhood = new Dictionary<string, double>();
            wordWithLikelyhood = categoryWithWWL[categoryWords.Key];

            //Finds group with max P(O | H) * P(H)
            //Calculates P(O | H) * P(H) for candidate group
            double p = 0;
            foreach (KeyValuePair<string, int> wordPair in candidateDocument)
            {
               if (vocabulary.ContainsKey(wordPair.Key))
               {
                  p += Math.Log(wordPair.Value * (wordWithLikelyhood[wordPair.Key]));
               }
            }
            p *= (pH);
            if (p > max_p || max_p == 0)
            {
               max_p = p;
               max_group = categoryWords.Key;
            }
         }

         return "Category: " + max_group + ". Likelyhood: " + max_p + ".";
      }
```