INF3800/INF4800

Assignment E

WHAT TO DELIVER:
Please submit a zip containing the implementation task (the entire project, there are no pencil-and-paper tasks for this assignment.

DEADLINE:
The assignment must be submitted, using Devilry, by 24.04.2017 at the latest.

In code

There are two implementation tasks in this assignment. Task A will count for 25% and Task B will account for 75%. This should also reflect the respective workload for each task. I recommend that you set aside plenty of time for Task B.

Task A

The previous assignments used exact matching on the query terms, i.e., the query term *backgammon* only matches *backgammon* and not variants such as, e.g., *bcakgammon* or *back-gammon*. There are many techniques available to do fuzzy or approximate matching, and in this assignment you will be tasked with implementing a system based on *k*-grams.

A *k*-gram is simply a window over *k* characters in a string. For example, the set of 3-grams from the string *backgammon* would be the set {*bac*, *ack*, *ckg*, *kga*, *gam*, *amm*, *mmo*, *mon*}. In some applications one might want to append special markers to the start and/or end of the string (e.g., ^*backgammon*$) and use the *k*-grams over this instead of the original string.

Obviously, in a *k*-gram based retrieval task one wants the items in the result set to contain as many of the query *k*-grams as possible. Note the resemblance with the *N*-of-*M* matching task from Assignment C. The overlap between these two sets can then be calculated via the Jaccard coefficient.

The code to complete is divided in two files: ShingleGenerator.java and ShingleRanker.java. Complete the missing code in these two classes, and verify the correctness of your implementation using the two JUnit tests made available in ObligETest.

Task B

Your task is to complete the implementation of a Naive Bayes classifier. We again base our implementation on the SimpleSearch architecture, so that you can relate to a familiar framework. But there is quite a lot of unnecessary overhead (the architecture is actually not really suited for this task), so the construction of the

inverted indexes will be relatively slow. I recommend that you only index part of the data made available, so it's a little bit faster at runtime. You can adjust this in NaiveBayesClassifier.FolderReader (e.g., lines 78-79). But test with all the data once you're done!

You have to complete several methods in the class MultinomialNaiveBayes. The training of the classifier is performed in MultinomialNaiveBayes, where the prior and likelihood distributions are estimated. The prior corresponds to the distribution *P(c)*, whereas the likelihood is defined as *P(w|c)*, where *w* is a word and *c* is a class.

You should carefully read the pseudo-code in 13.2 on page 260 in the textbook for details. However, I recommend using the formula in 13.7, that is, with add-one (Laplace) smoothing. If this is too simple, you can do more advanced smoothing if you want.

The classification is then made in accordance with Fig. 13 on page 260 in the textbook.

Optional: I have not tested, but I think the classification will get far better results if we had used a stop dictionary. You may want to extract a stop dictionary and test this!

You can test your implementation using the third test method of ObligETest. The method trains a Naive Bayes model given the data in data/train, and subsequently tests the model with the held-out data in data/test. The data collection is composed of 20 distinct newsgroups, each with a number of posts. The task is to automatically infer the correct newsgroup from the content of the post.