*1. slide*

Locality Sensitive Hashing (LSH) is a family of hashing methods that tend to produce the same hash (or signature) for similar items. There exist different LSH functions, that each correspond to a similarity metric. For example, the MinHash algorithm is designed for Jaccard similarity (the relative number of elements that two sets have in common).

LSH functions have two main use cases:

* Compute the signature of large input vectors. These signatures can be used to quickly estimate the similarity between vectors.
* With a given number of buckets, bin similar vectors together.

*2. slide*

Shingling is a way to represent document as sets, ideally for the purpose of identifying pairs of documents with similar content. It basically means we produce the set of all the short strings (shingles) that appear in the document

k-shingling: produce all the k-long shingles that appear in the document

There is not an official way to treat white spaces, it depends on the goal. Most usually, we count an any-long whitespace sequence as 1 character

word-level shingling: produce all the substrings of the document that contain a number of continuous words from the document

*3. slide*

The Jaccard Similarity is used to compare the similarity of pairs of documents. It achieves this by calculating the relative number of items the sets have in common: the size of their intersection divided by the size of their union.

*click*

Now to get the exact similarity of two documents we could just get their set-representation using shingling and then compare these two sets using the formula, but that takes too much time and memory.

Instead, we use LSH, which produces a pretty good estimate of the similarity. First, we generate the characteristic matrix, which hold the characteristic vectors of all the sets. The characteristic vector holds compares the items of the sets to the items of the universal set, and has a 1 at every item, which the set holds. It has 0s at items not held by the set. We then use MinHashing: we permute the items of the universal set, and with each permutation store the index of the first row where the set’s characteristic vector holds a 1. We put these indexes into a vector to get the signature. It is important that instead of actually permutating the universal set, we use hash functions to give new logical indexes to its rows.  
Now we could just compare the signatures one by one, but that also takes too much resources. Instead, we group the signature into bands (subgroups), and hash each band to a bucket, in a way that similar bands of different signatures will hash to the same bucket. This produces candidate pairs, which are supposedly similar documents, but there always can be false negatives (not found similars) and false positive (found but actually dissimilar).

*4. slide*

My application uses the open-source medical-nlp dataset, which is a huge set of medical transcriptions and can be found at the above link. The dataset has almost 5000 entries, but to save some time my app only uses 400 of these.

*5. slide*

First of all, I wrote my app in Java. The shingling logic can be found in the Shingler class. First, I split the document (1 entry) based on a number of possible characters into words, after that I remove the stop words. The shingling itself is easily done with 2 nested for-cycles, the first goes through the split document, the second collects the next n words.

*6. slide*

I chose a public Java implementation of the LSH which also included a function for computing Jaccard similarity based on characteristic vectors.

*click*

For the Jaccard similarity the implementation used a pretty basic logic

*click*

Again, for LSH we used at least two types of hash-functions: one for permutating the elements of the universal set, and another for hashing bands of signatures to buckets. For these hash functions, the implementation randomly initializes the parameters every time an LSH object is created, therefore these restrictions live.

*7. slide*

For the interface of my application I used the swing framework. The user first enters the shingling size, before which all functions of the program are disabled. After entering the size, they can read in the data file. Reading progress is shown in the progress bar. After successful read, the left-hand list shows all the entries of the file. Selecting one entry makes its shingles appear in the center text area. Selecting 2 entries using ctrl+click shows both their shingles as well as their estimated and exact Jaccard similarities. Extra features can be found in the menubar, such as computing the highest or the few highest estimated similarities or exact similarities. For these, the whole characteristic matrix needs to be generated, which takes a lot of time. The progress of the generation is shown in the data reading progress bar, since the app can be configured to generate the matrix during file read. After successful generation, the progress of computing the similarities is shown in the progress bar of the dialog box. After successful computation, the results are shown in the text area of the dialog box.

Most features include threading for responsibility.