Introduction:

This report describes the implementation of three algorithms for image retrieval, presents the *best* algorithm based on *Latent Semantic Indexing* (*MAP = 78%*) and compares it to baseline.

Description of the overlap and tf.idf algorithms:

I implemented the *overlap* algorithm in a naive way – for each query computing the overlap with each document. I implemented the *tf.idf* algorithm in two ways – in a naive way (for each query and document pair computing the entire *tf.idf* formula), and using the sparse matrices (similar to the *LSI* algorithm, but without *SVD* decomposition. The time taken was reduced from 4 minutes to 12 seconds by using sparse matrices and pre-computing the *tf.idf* for each word-document pair.

Description of the Latent Semantic Indexing algorithm (best):

- Vocabulary of all the words is constructed by taking a set of words from all the documents.
- Then a sparse matrix (vocabulary **x** documents) *lookup* (such that each *(i,j)* entry corresponds to the count of occurrences of the word *i* in document *j*) is constructed.
- *tf.idf* is precomputed for each element (for each word for each document) in *lookup* and *lookup* is transformed using an *SVD* decomposition up to *k* significant eigenvalues (uses the following library function: *scipy.sparse.linalg.svds*).
- The queries are transformed in the decomposed space (where each *(i,j)* entry corresponds to the count of occurrences of the word *i* in the query *j*).
- Then the matrix of queries is simply multiplied with the *lookup* matrix to obtain a similarity value for each query and document pair.

Discussion:

I chose to implement the *LSI* algorithm after having a look at the documents. They contain words in multiple languages, so the 'real' vector space is smaller than the vector space span by the vocabulary, as a same semantic token is represented by multiple languages as different words. *LSI* reduces the vector space, therefore I thought it is a suitable algorithm to implement for this task.
I chose the parameter *k* (number of dimensions in transformed vector space) to be *140*, which gave the best performance on the training set. *Figure 1* shows that there is hardly any difference for the values of *k* between *120* and *160*. The Sign test was also insignificant for a comparison of *k=120* and *k=140* (*p= .09*). Note that as *k* approaches the vocabulary size, the *MAP* converges to the *MAP* value of the *tf.idf* algorithm.

To test the significance of my results I used a *sign test* (as described in the lecture). The null hypothesis (that the best algorithm is no better than the baseline (*tf.idf*)) was rejected with very high certainty ($p < 10^{-12}$). Therefore the *best* algorithm is significantly better than the baseline.

The algorithm could be further improved by a combination of multiple prediction systems, such as Pseudo-relevance feedback and statistical synonyms.

Results:

The Mean Average Precision for the three algorithms implemented is displayed in *Table 1*:

Algorithm	*MAP [%]*
overlap	18.72
tf.idf	34.33
best (k=140)	77.81

Table 1: Mean Average Precision

Recall-Precision plot for those algorithms is displayed in *Figure 2*.

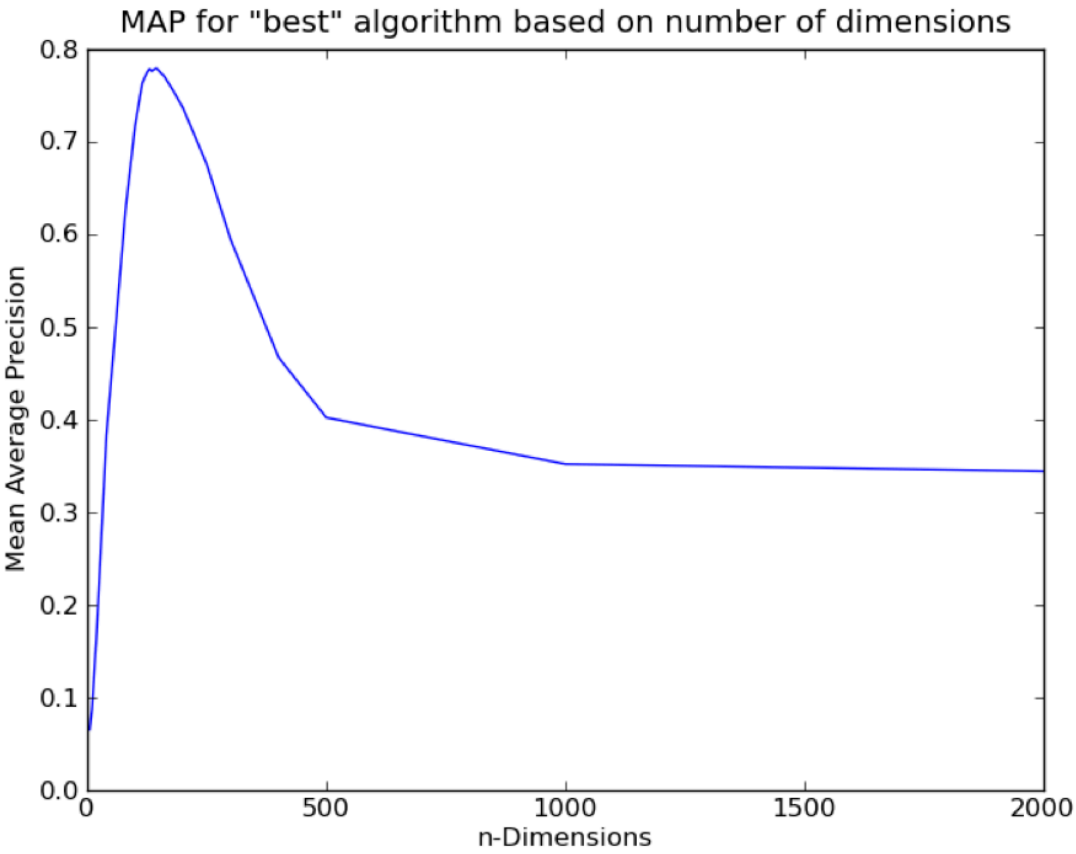

Figure 1: Different values of *k* for *best* algorithm. Optimal value seems to be between *120* and *160*.

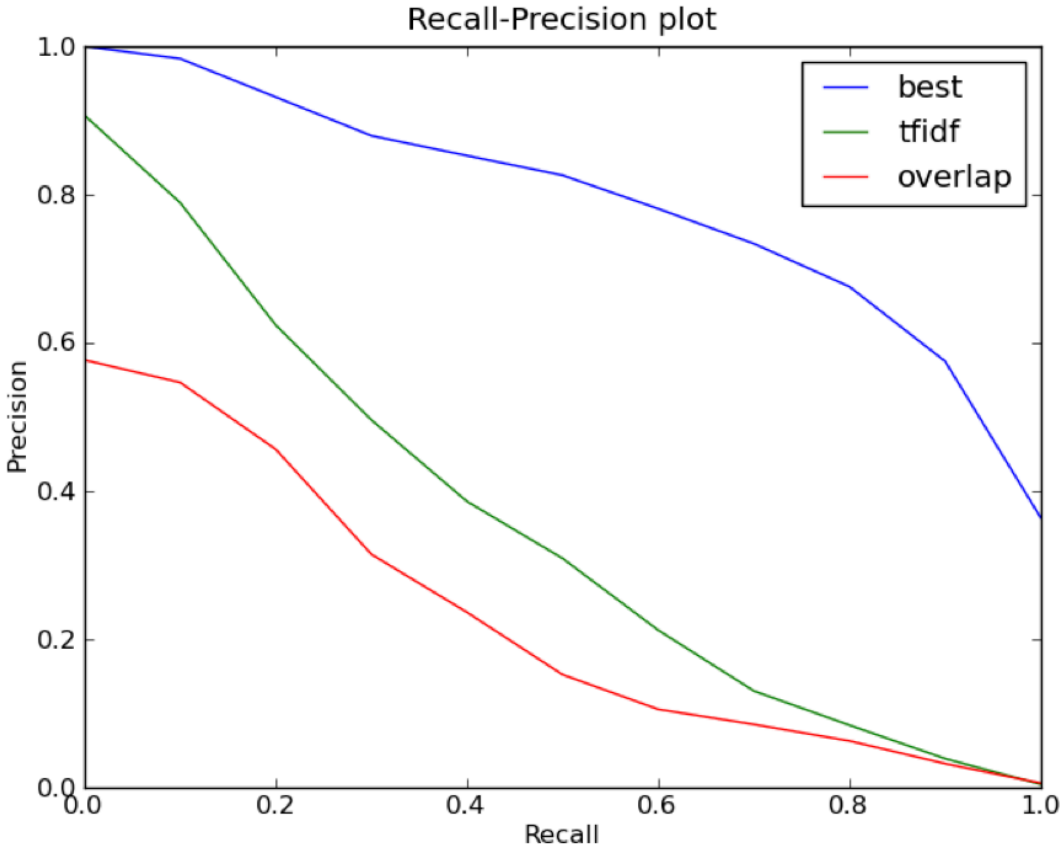

Figure 2: Recall-Precision plot for the three algorithms implemented. Algorithm *best* uses *k=140*.