AGENDA FOR TODAY

1. quality evaluation of information retrieval (last part of the vsm notebook)
2. introduction to the POS-tagging notebook
3. your questions

QUALITY EVALUATION
OF
INFORMATION RETRIEVAL

1. the task of IR
2. definition of similarity
3. metrics
4. notebook functions
    - 4.1. closest_n_documents()
    - 4.2. compute_average_results()

THE TASK OF IR

Information Retrieval (IR) - the task of finding the document *d* from the *D* documents in some collection that best matches a query *q*

IN THE NOTEBOOK

IDEALLY: find N songs from the collection that belong to the same artist as *q*
***d* - a song represented as a vector**
***q* - a new song**

we're testing if it makes sense to try and build a recommendation system based on lyrics only. we test if it works for finding songs by the same artist because these are easy to obtain 'true labels'. but other 'true labels' could be similar artists according to some listeners.

this approach might also be used for attributing anonymous or pseudonymous works to an author (who might have written this song?)

DEFINITION OF SIMILARITY 1

we represent songs as vectors where dimensions are related to words used in these song.
our premise is that similar songs use similar vocabulary.

to test how similar the values across those dimensions are we use cosine. it is dot product normalised by vector lengths (so that songs with lots of words don't get the advantage)

song1 = [3,0,5] (8 words) |s1|= sqrt(3*3+0*0+5*5) = sqrt(34) = 5.83
song2 = [3,7,5] (15 words) |s2|= sqrt(3*3+7*7+5*5) = sqrt(83) = 9.11
song3 = [5,1,4] (10 words) |s3|= sqrt(5*5+1*1+4*4) = sqrt(42) = 6.48

dot(s1,s2) = 3*3+0*7+5*5=34
dot(s1,s3) = 3*5+0*1+5*4=35

cosine(s1,s2) = 34/(5.83*9.11)=0.64
cosine(s1,s2) = 35/(5.83*6.48)=0.93

DEFINITION OF SIMILARITY 2

to find N most similar songs to a query q:
1. compare q to every song
2. sort the songs by their similarity
3. choose N most similar

collection [s1,s2,s3], query q

cosine(q,s1)=0.05
cosine(q,s2)=0.5
cosine(q,s3)=0.09

sorted_collection [s2,s3,s1]

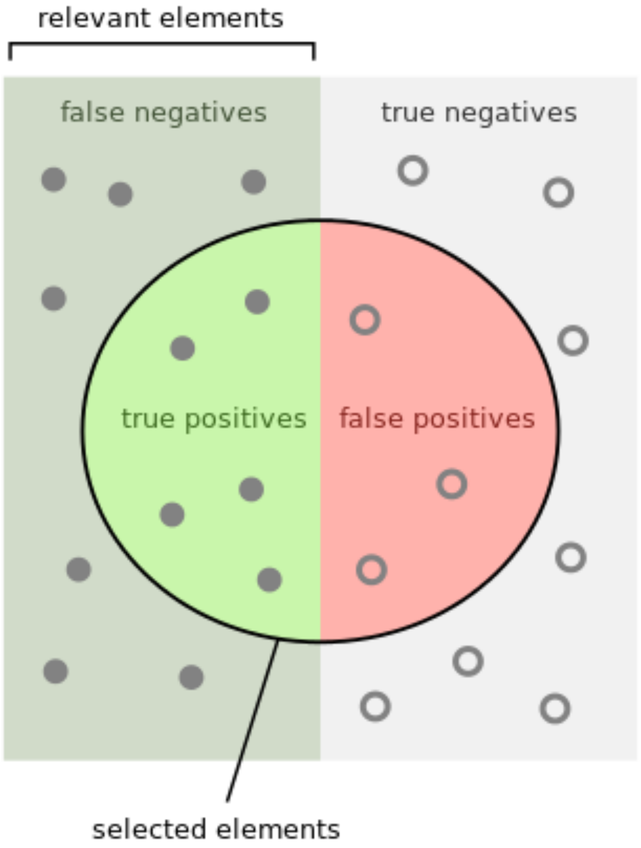
top_2 [s2,s3]

METRICS 1

we have N most similar songs for every song
in a test set of T songs
how do we know if a system is good?

STEP1 for every song count:

True Positives (TP) - were chosen and right
True Negatives (TN) - were not chosen and are not relevant
False Positives (FP) - were chosen, but are not relevant
False Negatives (FN) - were not chosen, but relevant



thanks god oh no

METRICS 2

we have N most similar songs for every song in a test set of T songs. how do we know if a system is good?

STEP2 for every song compute:

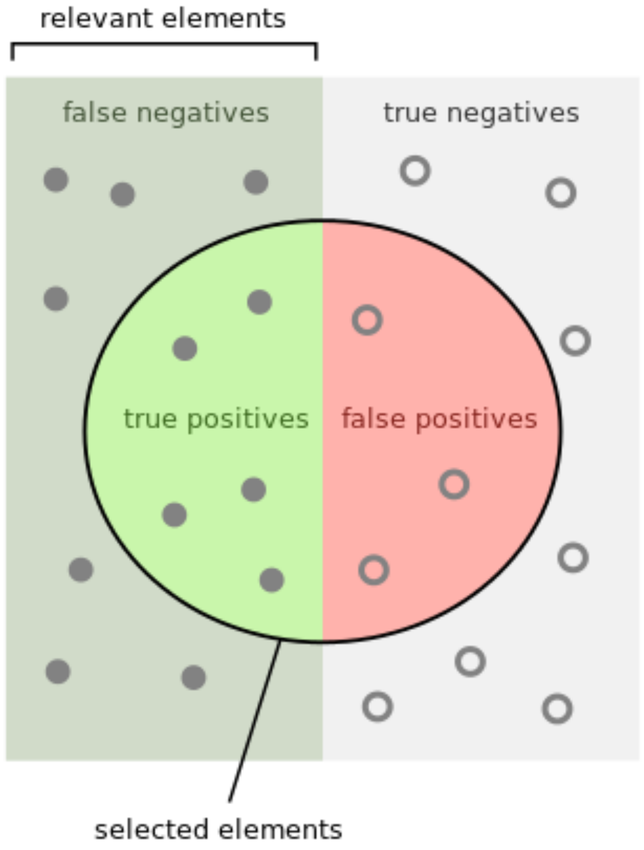
Precision = $\frac{TP}{TP + FP} = \frac{TP}{N}$

Recall = $\frac{TP}{TP + FN} = \frac{TP}{numofartistsongs}$

Accuracy = $\frac{TP + TN}{collectionlen}$

Error = $\frac{FP = FN}{collectionlen} = 1 - Accuracy$

F-measure = $\frac{1}{\alpha \frac{1}{Precision} + (1 - \alpha) \frac{1}{Recall}}$



METRICS 3

we have N most similar songs for every song
in a test set of T songs
how do we know if a system is good?

STEP3 get an average performance according to every metric:

average_accuracy = $\frac{acc_1 + acc_2 + acc_3 + \ldots + acc_T}{T}$

average_recall = $\frac{rec_1 + rec_2 + rec_3 + \ldots + rec_T}{T}$

and so on …

NOTEBOOK FUNCTIONS
closest_n_documents()

matrix_collection 3x5
matrix_query 3x3
n 2

closest_docs [[1,2],[1,2],[1,2]]
a list of 3 lists with 2 indices

```
def closest_n_documents(matrix_collection, matrix_queries, n):
    """Finds N closest documents from a training collection to every song in a test collection

    this function takes in original document collection, new document collection,
    computes cosine similarity between documents in old and new collection,
    and outputs the list of n-closest documents to each new song
    when a vector in a query has only zeros,
        the closeness to it should be determined by index of a song from a matrix_collection:
        the closest document for a zero vector has index 0
        the second closest document for a zero vector has index 1 and so on

    Parameters
    ----------
    matrix_collection : numpy array
        a term-document matrix of songs in training collection
        songs are columns
    matrix_queries : numpy array
        a term-document matrix of query songs
        songs are columns
    n : int
        a number of closest documents to return

    Returns
    -------
    closest_docs : a list of lists
        a list of length equal to the number of songs in a query matrix
        each element is, in turn, a list of n imdices of documents in matrix_collection that were closest to the query
        for n=2 and query matrix with 3 songs, the out put should look like so [[1,2],[1,2],[1,2]]
    """
```

NOTEBOOK FUNCTIONS
compute_average_results()

closest_songs [[1,2],[1,2],[1,2]]
a list of 3 lists with 2 indices
comes from closest_n_documents()

train_index
{
artist1:[0,1,2]
artist2:[3,4]
}
tells what songs from matrix_collection are by who

test_index
{
artist1:[0,1,2]
artist2:[]
}
tells what songs from matrix_query are by who

```
def compute_average_results(closest_songs, train_index, test_index):
    """Computes average metrics for a model based on n closest songs for a test set

    Parameters
    ----------
    closest_songs : a list of lists
        a list of length equal to the number of songs in a query matrix
        each element contain n closest songs from a training collection to that song
    train_index : dict {atrist:lits of song indices}
        indices of songs in training collection assigned to artists
    test_index : dict {atrist:lits of song indices}
        indices of songs in test collection assigned to artists

    Returns
    -------
    average_precision: float
        average precision based on all test songs
    average_recall: float
        average recall based on all test songs
    average_accuracy: float
        average accuracy based on all test songs
    average_error: float
        average error based on all test songs
    average_f_measure: float
        average f_measure based on all test songs
    """
```