Natural Language Processing

4 Courses of NLP techniques.

[Natural Language Processing with Classification and Vector Spaces 2](#_Toc64309258)

[Logistic Regression 2](#_Toc64309259)

[Naïve Bayes 3](#_Toc64309260)

[Vector Space Models 6](#_Toc64309261)

[Machine Translation 8](#_Toc64309262)

# Natural Language Processing with Classification and Vector Spaces

## Logistic Regression

### Introduction

NLP has changed a lot over the last several decades. The field started off with primarily rule-based systems before adopting probabilistic systems that perform much better but still require a lot of hand engineering. Now, NLP relies much more on machine learning and deep learning.

### Supervised ML and Sentiment Analysis

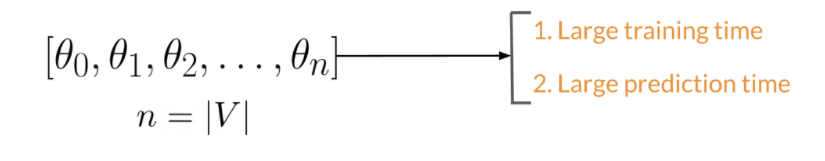
To build a logistic regression classifier capable of predicting sentiments of arbitrary tweets, we will:

* Process the raw tweets in the training set and extract useful features.
* Train a logistic regression classifier while minimising cost.
* Make predictions on a test set.

### Vocabulary and Feature Extraction

A **vocabulary** allows a string of words to be represented as an array of numbers. A string can be broken down into a list of component words and weighed up against the vocabulary. 1s would be assigned to words in the vocabulary which are present in the string, and 0s assigned to absent words. With large vocabularies (i.e. entire languages), most of the vector values would be 0. These types of representations with few non-zero elements are known as **sparse representations**.

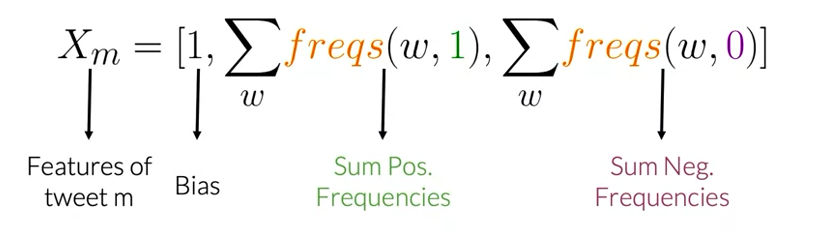
Here, the number of features, , would be equal to the size of the entire vocabulary. Having to train a classifier with parameters would take an excessively long time where the vocabulary is large.

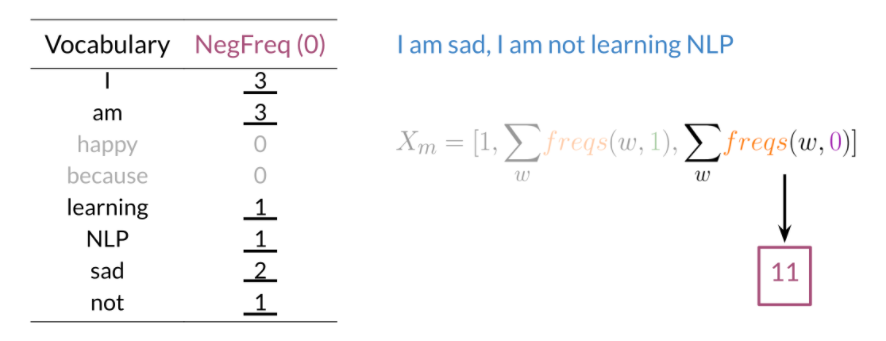


### Feature Extraction with Frequencies

Using the labels assigned to strings of text, we can count the number of times a word from the vocabulary appears in one class of text compared to another class of text. Creating a **frequency dictionary**—mapping a word to the frequency it appears in a certain class—is a useful way of reducing the dimensions for a classifier.

From this frequency dictionary, it is possible to extract useful features. For example, for sentiment analysis with two classes (positive and negative), we could use the sum of words from the frequency dictionary which appear in the text being analysed. Our classifier would therefore only need to train three parameters for three features: a bias term, a positive frequency sum and a negative frequency sum.

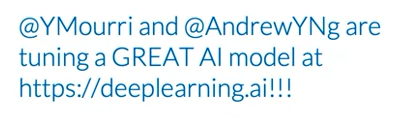
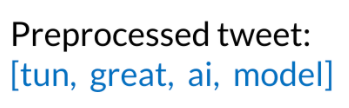




### Preprocessing

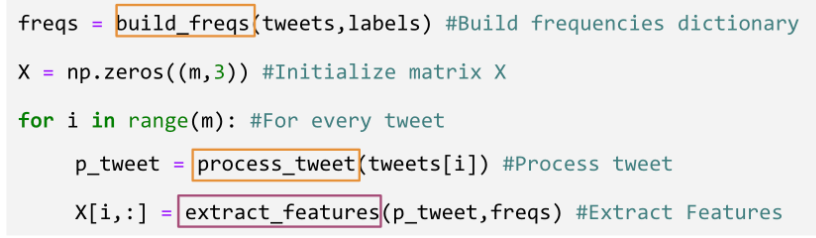
There are five primary components of text preprocessing for ML analysis:

* **Tokenising:** a string must first be processed into a list of words. Python NLP libraries include functions which will split a string into words based the location of spaces.
* **Stop words:** removing all words which don’t add significant meaning to text. Usually performed by comparing a set of words against a list of common words and a list of punctuation marks.
* **Special text:** If processing tweets or specialised text, then handles, URLs and other textual formats may need eliminating as well.
* **Stemming:** transforming any word to its base stem which is the set of characters used to construct the word and its derivatives.
* **Lowercasing**: transforming a word to all lowercase characters.

### Putting it together

Using functions to preprocess and extract features of text strings one at a time, a matrix of features can be built as an input for a classifier. For a training set of examples and number of features , our resulting matrix would have dimensions . An example of code with prewritten processing and feature extraction functions might look like this:



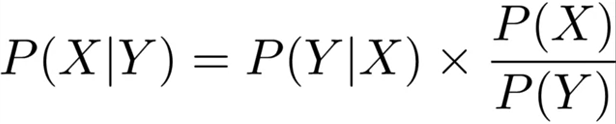
### Logistic Regression

*See notes from Machine Learning (Stanford University)*

## Naïve Bayes

### Bayes’ Rule

Bayes Rule: The probability of X given Y is equal to the probability of Y given X times the ratio of the probability of X over the probability of Y.



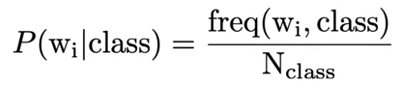
Bayes’ Rule is derived from expressions of conditional probability. Say we have two conditions: the data in question can be X or Y. The intersection of the Venn Diagram represents the set where the data are both X and Y. The probability of being X given Y or vica versa corresponds to the area of the intersection over the area of the corresponding circle.

Since the numerators are the same in both cases, we can substitute to find the relationship of and :

### Naïve Bayes Inference

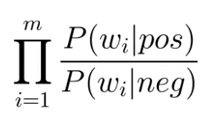
Naive Bayes is an example of supervised machine learning. It's called naive because this method makes the assumption that the features being used for classification are all independent, which in reality is rarely the case.

Naive Bayes works by creating a table of conditional probabilities. Rather than summing the word frequencies as with logistic regression, the frequencies are *divided* by the total number of words associated with a certain class within the corpus. This will produce a table of conditional probabilities: the probability that a word will appear in a text given its label.

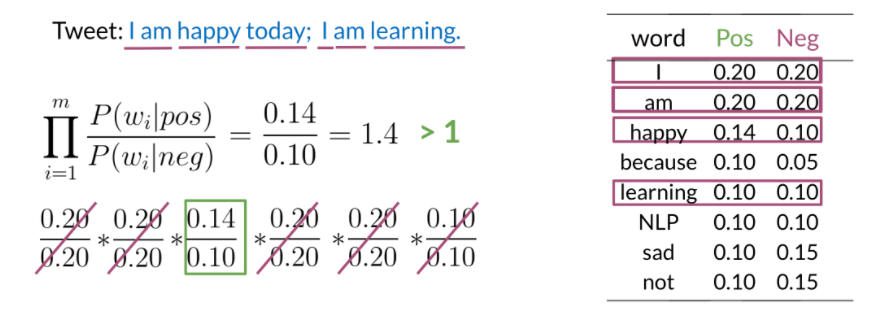


The ML features are generated by multiplying the ratios of the conditional probabilities for words in each class. For sentiment analysis, we find the product of the probability that a word appears in a positive message and divide it by the probability that a word appears in a negative message. This is called Naïve Bayes inference condition rule for binary classification.

Naïve Bayes inference condition rule for binary classification



The idea is that neutral words which are equally likely to appear in either class of message will not affect the product since their ratio is equal to one. Words with a strong bias towards one class will impact the product disproportionately. In the example below, a product of more than one would indicate a positive message and less than one a negative message.

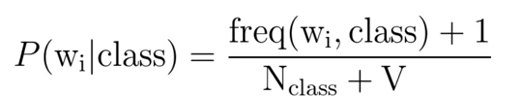


*Note: gradient descent is not required for a Naïve Bayes learning model since there are no function parameters to learn. We are dealing solely with probabilities here.*

### Laplacian Smoothing

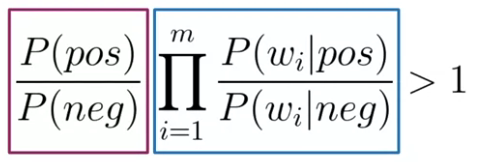
Naïve Bayes runs into problems when probabilities equal zero. In order to avoid this occurrence, the probability function is smoothed by:

1. Adding 1 to all frequencies to ensure no conditional probability is zero.
2. Adding the number of unique words in the vocabulary, , to the total frequency of words in a class in order to normalise the probability (such that the sum of probabilities in the table will still be 1 for each class).

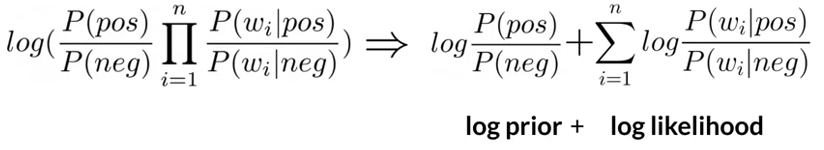
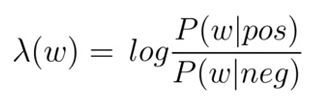


### Log Likelihood

When working with an unbalanced training set, the inference condition rule must be adapted to account for the probability that a message falls under a certain class. With a balanced dataset, this **prior probability** ratio is equal to 1 and can be ignored; otherwise it should be factored into the feature generation. Adding this prior probability term (boxed in purple below) to the likelihood (boxed in blue) give the full Naïve Bayes formula.



This Naïve Bayes model requires the multiplication of lots of small numbers which carries the risk of n**umerical underflow:** the return of a number so small that it can’t be stored on a device. Applying logarithms to the prior and likelihood means that the product becomes a sum of the *logarithms* of the conditional probability ratios. These logarithmic likelihoods are sometimes represented as .

A neutral ratio of 1 corresponds to a logarithmic ratio of 0. Therefore, positive sums are indicative of positive sentiment and negative sums of negative sentiment in this case.





### Naïve Bayes Classification

The process of training a Naïve Bayes model can be broken down into four steps following collection and preprocessing:

1. Compute
2. Get
3. Get
4. Compute

Once a model has been trained by calculating these probabilistic features, unseen text can be analysed using the Naïve Bayes formula to find , and classified using the decision threshold of 0. Accuracy of the model can be calculated the same as any classifier.

### Assumptions of Naïve Bayes

The main assumptions of the Naïve Bayes model are:

1. Independence between the predictors or features associated with each class.
2. Relative frequencies in corpus are representative of reality.

The first assumption could lead to overestimating or underestimating the conditional probabilities of individual words. Contextual information around a word is neglected.

The second issue is that Naive Bayes relies on the distribution of the training data sets. A good data set will contain the same proportion of positive and negative tweets as a random sample would. However, most of the available annotated corpora are artificially balanced. The weighting of classes in a training set could bias the model to a certain class.

### Error Analysis

Possible errors in model prediction can be caused by:

1. Semantic meaning lost in the preprocessing step.
2. Syntactic choices affecting the meaning of a sentence.
3. Adversarial attacks: quirks of language which confuse algorithms but are understood naturally by humans, such as sarcasm, irony and euphemisms.

These can be ironed out in the preprocessing and feature generation steps using mathematical and computational tricks. Some of these techniques will be looked at later.

## Vector Space Models

### Vector Space Models

**Vector space model**: an algebraic model for representing text documents as vectors of identifiers.

VSMs are useful for:

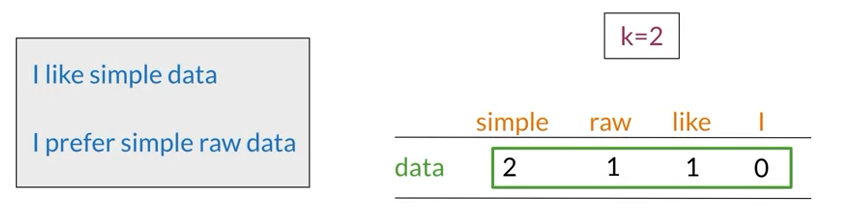
* Identifying semantic similarities between sentences, paraphrasing and summarisation.
* Capturing dependencies between words

### Co-occurrence & Word by Document

Vector spaces can be represented by co-occurrence matrices or word by document matrices.

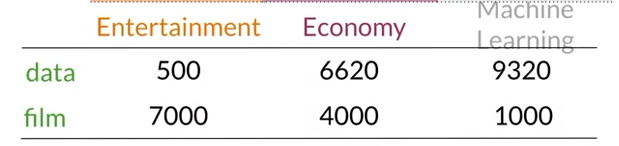
**Co-occurrence**: the number of times that two different words appear in your corpus together within a certain word distance, .

The row of the co-occurrence matrix corresponding to the word data would look like this if you consider the co-occurrence with the words ‘simple’, ‘raw’, ‘like’, and ‘I’. With a word-by-word design, you can get a representation with entries, with between one and the size of your entire vocabulary.



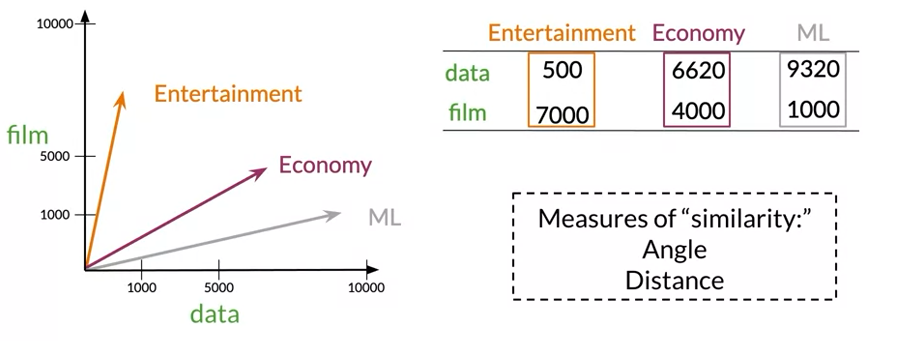
A simple Word by Word co-occurrence matrix

A Word by Document design counts the number of times a word occurs within a certain category of text within a corpus.



A simple Word by Document matrix

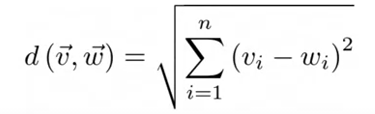
This matrix represents a vector space. Taking the representation for each category of documents as dimensions (i.e. the words ‘data’ and ‘film’) give us a 2D vector space for visual clarity. The vector representations as each column of the matric are plotted in the vector space. Within this vector space, it is possible to make comparisons using cosine similarity and Euclidean distance. Using these metrics, its possible to get a sense of how similar two documents or words are within a corpus.



### Similarity Functions

**Euclidean distance**: the length of a straight-line segment between two points in Euclidean space. The equation is an example of the Pythagorean Theorem.

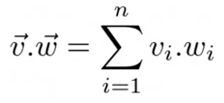
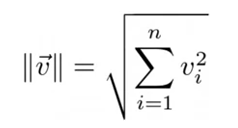
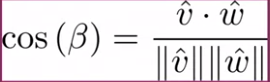
Euclidean distance formula for n-dimensional vectors

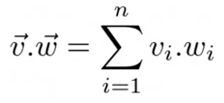


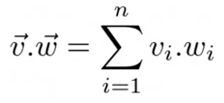
Euclidean distance comparisons can be problematic, however, when there is a large size discrepancy in the corpora being compared. For example, words with similar ratios of co-occurrences but high variability in usage may be judged to be less similar than words with very different co-occurrence ratios but have similar degrees of usage. In this case, using the cosine similarity—making use of the angle between the two vectors—resolves the issue. The same is true for corpora of different sizes.

**Cosine similarity**: the cosine of the angle between two non-zero vectors.

Cosine similarity formula for n-dimensional vectors

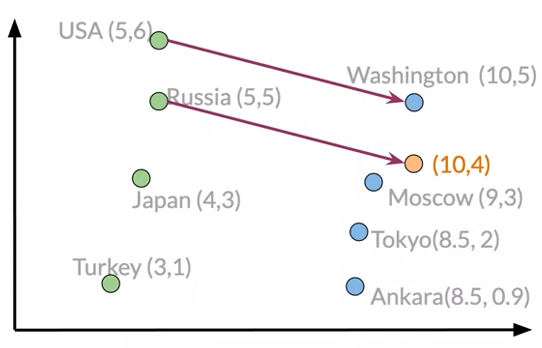


where 

and 

A cosine similarity of 1 indicates the vectors point in exactly the same direction in Euclidean space and overlay one another perfectly. A cosine similarity of zero indicates vectors are orthogonal, so are as different as can possibly be.

### Manipulating Vector Spaces

Vector spaces can be employed to predict and deduce information from the known relationships of other words. A vector space which captures relative meaning between a country and its capital city, for instance, can be used to identify unknown relationships.

For example, using the vector between a known country-capital pair, USA and Washington, I could plot this vector from the point Russia and search for the closest possible city in the vector space to learn that the capital is Moscow.

### Visualisations and PCA

It is often the case that vector spaces will end up having very high dimensions. To reduce the dimension of these vectors to two dimensions where plotting data on an XY axis is possible, **Principal Component Analysis** is employed. PCA is an algorithm used for dimensionality reduction that can find uncorrelated features for your data. It is helpful for visualising data to check if the representation is capturing relationships among words.

The PCA algorithm can be broken down broadly into three parts:

1. Mean-normalise the data.
2. Obtain a covariance matrix of the data.
3. Perform a singular value decomposition (SVD) to obtain a set of three matrices [U,S,V].

Following this, the dimensions reduction can be performed by projecting the data to a new set of features utilising the eigenvector and eigenvalue matrices, U and S.

**Eigenvectors**: correspond to direction of uncorrelated features which define the data subspace.

**Eigenvalues**: correspond to the variance of the new features. These will be organised in descending order.

*For more notes on PCA see Machine Learning (Stanford University)*

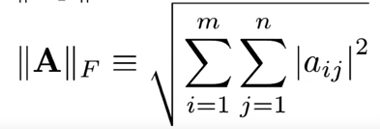
## Machine Translation

### Transforming Word Vectors

Word vectors can capture important properties of words, with information about in which contexts they appear with respect to other words and within corpora.

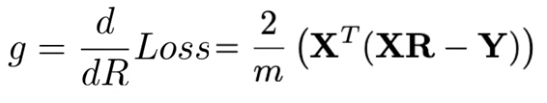
For a machine to translate one word from English to French, for instance, the word embeddings associated with both languages would need to be calculated. A method of translating the word embedding in the vector space of one language to the vector space of the other is tantamount to a basic translation tool.

The transformation method takes the form of a matrix, . The transformation matrix is found by training a supervised learning model on a subset of words and phrases common to both languages. If is our English subset vector space and Y is our French subset vector space, we need to minimise the Frobenius Norm which can be found by square rooting the sum of squares for all elements of the matrix:

In practice, the square of the Frobenius Norm is used since it is easier to take the derivative of the squared expression than to deal with the square roorts.

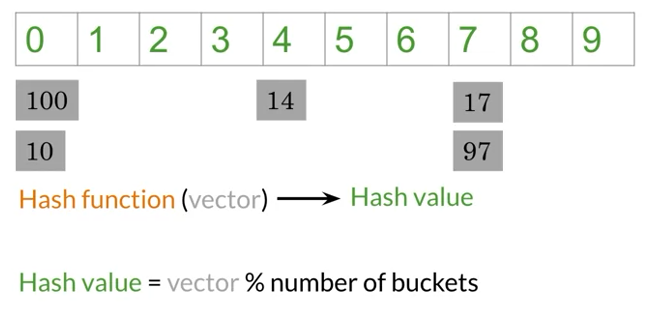
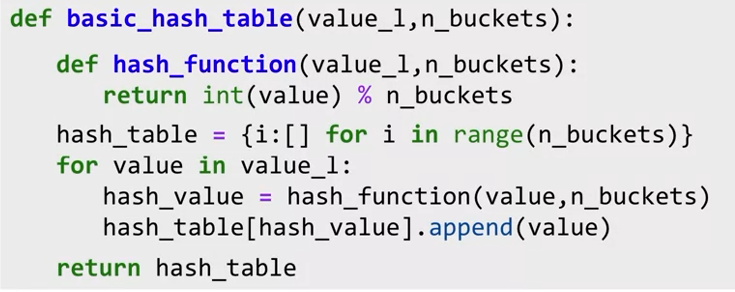
Optimal can be found by applying gradient descent as with logistic regression. The gradient can be calculated as:



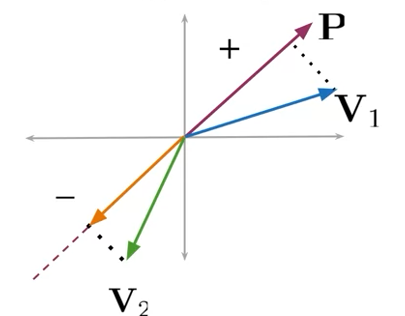
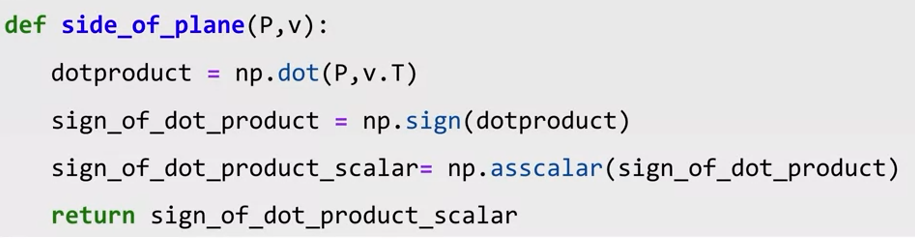
The transformation of to will never be perfect for all words, so the predicted word vector will end up being close to the actual word vector in another language. To find the closest matches, we can perform a K-nearest neighbours-type analysis to find an appropriate translation.

### Hash Tables

Hash tables are essentially buckets of similar words denoted by a numerical key. A **hash function** is employed to assign a word to a hash table. For example, the hash function below takes the remainder of the one-dimensional vectors below divided by the number of buckets and assigns them a hash value equal to that remainder.

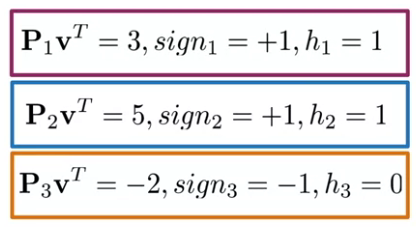
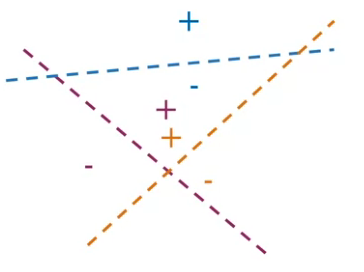
To assign word vectors based on distance in space rather than something arbitrary like remainders of division, **locality sensitive hashing** is performed. Locality sensitive hashing works by dividing the vector space with hyperplanes. Calculating the dot product or **projection** of the normal vector of a plane and word vectors will inform which side of the plane the words fall on, depending on the sign of the dot product. Below, represents the normal vector of a 1D plane and are word vectors.

### Multiple Planes

Dividing the vector space into multiple planes will create regions. Within these regions, word vectors will have discrete combinations of positive and negative values of projections against the planes.

A hash value can be calculated in binary terms by combining the signals from multiple planes. For a vector space with three planes, a word vector has a possible regions in which it could sit. For a word vector which has a positive projection along the normal of the first two planes but a negative projection along the third, the total hash value calculated is . A positive sign is associated with a hash value of and a negative sign with a hash value of .



### Approximate Nearest Neighbours

It’s impossible to know for sure which set of planes is the best way to divide up the vector space. Creating multiple sets of random planes for locality-sensitive hashing signifies a more robust way of searching the vector space for a set of vectors that are possible candidates to be nearest neighbours. This is called **approximate nearest neighbours**; in this case we are searching not for a specific nearest neighbours, just an approximate number over a subset of space. We therefore trade some precision for search efficiency and speed. This KNN-type search can be performed much faster than analogous Bayesian calculations.

### Searching Documents

The same technique can be implemented for searching for pieces of text related to a query in a large collection of documents. For this task, documents need to be represented as vectors which can be achieved, at a basic level, by summing the vectors corresponding to the individual words in the document.

### Summary

* Machine translation is performed by minimising the error when transforming one set of word vectors into another.
* Grouping word vectors can be performed by a nearest neighbour analysis using multiple planes and devising hash tables.
* Searching documents for content can be achieved by summing word vectors.

The general principle of embedding text in vector space is used throughout modern NLP. Using models to identify text with similar meaning, we can mathematically translate and identify words within documents. There are more advanced ways of embedding text which will provide more possibilities and functionalities.