k-Nearest Neighbor and Naive Bayes

1 k-Nearest Neighbor

The k-Nearest Neighbors algorithm (or k-NN for short) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression. Here is this note, we are discussing only the k-NN classification. In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

1.1 Text Classification using k-NN

For the text classification, we consider each text (also known as document) as a point in n dimensional space where n is the number of different words present in training documents and the test document (no test document or all the test documents can also be considered).

There are three different measures for text classification using k-NN.

1. Hamming Distance
2. Euclidean Distance
3. Cosine Similarity

1.2 Hamming Distance

For hamming distance, we consider each document as a vector.

Document 1 (Sports): *I play cricket. I play football.*
Document 2 (Music): *Play this music.*
Document 1 (Music): *I like singing.*
Document 1 (Biology): *Cricket is a very small insect.*

During parsing the document/text, we do not count the prepositions, pronouns, auxiliary verbs etc. So after parsing, we create an object for each document. This object carries the information about the topic name of the document, the list of relevant words and their counts, and other necessary information.

Document 1 (D_1): {topic name: "Sports", word list: [<play,2> <cricket,1> <football,1>]}
Document 2 (D_2): {topic name: "Music", word list: [<play,1> <music,1>]}
Document 3 (D_3): {topic name: "Music", word list: [<like,1> <singing,1>]}
Document 4 (D_4): {topic name: "Biology", word list: [<cricket,1> <very,1> <small,1> <insect,1>]}

Now, if our test document is

Document t: *I want to play music.*

then after parsing it becomes

Document t (D_t): {topic name: "null", word list: [<want,1> <play,1> <music,1>]}

For hamming distance we consider each document as a feature vector. So, below are the feature vectors

$$\begin{array}{cccccc} \text{Feature Space} & D_1 & D_2 & D_3 & D_4 & D_t \\ \begin{bmatrix} play \\ cricket \\ football \\ music \\ like \\ singing \\ very \\ small \\ insect \\ want \end{bmatrix} & \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} & \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \end{array}$$

The hamming distance is actually the distance between two different vectors. For our example, the hamming distance between Document t (D_t) and Document 1 (D_1) is

hd (D_t,D_1)=(0+1+1+1+0+0+0+0+0+1) = 4

Similarly,

hd (D_t,D_2) = (0+0+0+0+0+0+0+0+0+1) = 1
hd (D_t,D_3) = (1+0+0+1+1+1+0+0+0+1) = 5
hd (D_t,D_4) = (1+1+0+1+0+0+1+1+1+1) = 7

Here, the distance from D_t to D_2 is the smallest, so D_2 is the nearest neighbor of D_t. For k=1, we assign the topic name of the test document as the topic name of the nearest neighbor. If k is more than 1 then we take k number of nearest neighbors and consider their voting. During this voting, we can consider either the equal weight of all the neighbors or the weighted voting. The weighted voting is considered through **Radial Basis Functions**. Here, we are considering equal voting, not weighted one. In that case, if k=3, then we select D_2, D_1 and D_3 as our neighbors and our answer through majority voting is "Music". If there is a tie during majority voting or neighbor selection, we may use random function or any other heuristic to break the tie.

1.3 Euclidean Distance

For euclidean distance calculation, we consider each document as a n-dimensional vector where n is the number of different types of words present in training documents and the test document.

Let's consider two 2-D vectors

P_1 = 3x+3y
P_2= 2x+y

The euclidean distance between this two vector is

ed(P_1,P_2) = $\sqrt{(3-2)^2 + (3-1)^2} = \sqrt{5}$

Here, the two dimensions are along x-axis and y-axis. If we replace this two dimensions with $word_1$-axis and $word_2$-axis and represent each document as a vector of this two words. For example, consider the two documents below:

$D_1 = \mathrm{N}_{word_1,D_1} word_1 + \mathrm{N}_{word_2,D_1} word_2$
$D_2 = \mathrm{N}_{word_1,D_2} word_1 + \mathrm{N}_{word_2,D_2} word_2$

where N_{word_j,D_i} is the total number of $word_j$ in document D_i. We can easily find the euclidean distance between D_1 and D_2

$\mathrm{ed}(D_1,D_2) = \sqrt{(N_{word_1,D_1} - N_{word_1,D_2})^2 + (N_{word_2,D_1} - N_{word_2,D_2})^2}$

Similarly, if there are n number of words, then the distance will be in n-dimensional space

$\mathrm{ed}(D_1,D_2) = \sqrt{(N_{word_1,D_1} - N_{word_1,D_2})^2 + (N_{word_2,D_1} - N_{word_2,D_2})^2 + ... + (N_{word_n,D_1} - N_{word_n,D_2})^2}$

Now, it is time to go for an example. Suppose the training documents are the ones discussed in the previous subsection.

Document 1 (D_1): {topic name: "Sports", word list: [<play,2> <cricket,1> <football,1>]}
Document 2 (D_2): {topic name: "Music", word list: [<play,1> <music,1>]}
Document 3 (D_3): {topic name: "Music", word list: [<like,1> <singing,1>]}
Document 4 (D_4): {topic name: "Biology", word list: [<cricket,1> <very,1> <small,1> < insect,1>]}

and the test one is

Document t (D_t): {topic name: "null", word list: [<want,1> <play,1> <music,1>]}

According to euclidean distance,

$\mathrm{ed}(D_t,D_1) = \sqrt{(1-0)^2 + (1-2)^2 + (1-0)^2 + (0-1)^2 + (0-1)^2} = \sqrt{5}$

(the words/dimensions considered here are "want","play","music","cricket","football". The values for rest of the words/along rest of the dimensions are 0 for both of the documents.)

Similarly,

$\mathrm{ed}(D_t,D_2) = \sqrt{(1-0)^2 + (1-1)^2 + (1-1)^2} = \sqrt{1}$
$\mathrm{ed}(D_t,D_3) = \sqrt{(1-0)^2 + (1-0)^2 + (1-0)^2 + (0-1)^2 + (0-1)^2} = \sqrt{5}$
$\mathrm{ed}(D_t,D_4) = \sqrt{(1-0)^2 + (1-0)^2 + (1-0)^2 + (0-1)^2 + (0-1)^2 + (0-1)^2 + (0-1)^2} = \sqrt{7}$

So, the nearest neighbor of D_t is D_2 and "Music" will win. Again, for k=3, the nearest neighbors will be D_2, D_1 & D_3 and the result will be same.

1.4 Cosine Similarity

In cosine similarity, we consider how similar the documents are by determining the angle between the vectors representing the documents.

Let's start with two vectors P_1 and P_2
P_1 = 3x+3y
P_2= 2x+y

$$P_1.P_2 = |P_1|.|P_2|.Cos\theta \tag{1}$$

$$Cos\theta = \frac{P_1.P_2}{|P_1|.|P_2|} = \frac{(3*2+3*1)}{\sqrt{3^2+3^2}.\sqrt{2^2+1^2}} \tag{2}$$

$Cos\theta = 1$, when $\theta = 0$
$Cos\theta = 0$, when $\theta = 90$

So, the more the value of $Cos\theta$, the more similar the two vectors are.

Here, the two dimensions are along x-axis and y-axis. If we replace these two dimensions with $word_1$-axis and $word_2$-axis, and represent each document as a vector of these two words, then the two documents will be

$D_1 = \mathrm{N}_{word_1,D_1} word_1 + \mathrm{N}_{word_2,D_1} word_2$
$D_2 = \mathrm{N}_{word_1,D_2} word_1 + \mathrm{N}_{word_2,D_2} word_2$

where N_{word_j,D_i} is the total number of $word_j$ in document D_i. Now, we can easily find the cosine similarity between D_1 and D_2

$$\begin{aligned} cs(D_1, D_2) &= Cos\theta \\ &= \frac{D_1.D_2}{|D_1|.|D_2|} \\ &= \frac{(N_{word_1,D_1} * N_{word_1,D_2} + N_{word_2,D_1} * N_{word_2,D_2})}{\sqrt{{N_{word_1,D_1}}^2 + {N_{word_2,D_1}}^2} * \sqrt{{N_{word_1,D_2}}^2 + {N_{word_2,D_2}}^2}} \end{aligned} \tag{3}$$

If there are n number of words, then the distance will be in n-dimensional space.

Wait. This is not final. Here, we are using N_{word_j,D_i} as the co-efficient of $word_j$ in Document D_i. But in our cosine similarity, we shall use TF-IDF$_{word_j,D_i}$ instead of N_{word_j,D_i}.

TF: TF stands for "Term Frequency". This means the count of the word in a specific document. There are several heuristics for TF calculation.

Heuristic	**Formula**
Using N_{word_j,D_i} directly	$\mathrm{TF}_{word_j,D_i} = \mathrm{N}_{word_j,D_i}$
Normalizing by the maximum word count in that document	$\mathrm{TF}_{word_j,D_i} = \mathrm{N}_{word_j,D_i}/\max\{\mathrm{N}_{word_k,D_i}:\mathrm{k} \in D_i\}$
Normalizing by the total number of words in that document	$\mathrm{TF}_{word_j,D_i} = \mathrm{N}_{word_j,D_i}/\mathrm{N}_{D_i}$

We can use any one among these. In our experiment, we shall use the last one.

IDF: IDF stands for "Inverse Document Frequency". This is to determine how specific the term/word is. If a word appears in almost all the documents, then it is general, not specific to a topic. On the other hand, if it appears only in a few number of documents, then it may be specific and relevant to a topic.

There are several heuristics for IDF also but we shall use the following one:
$\mathrm{IDF}_{word_j} = \log(\mathrm{D}/\mathrm{d}_{word_j})$

where D is total number of documents in the training sets and d_{word_j} is the total number of documents in the training sets where $word_j$ appears. Keep in mind, training sets mean all the training documents whatever the topic is.

Now, two problems can arise in this equation.

- if $\mathrm{d}_{word_j} = 0$, then there will be division by zero problem. To avoid the problem, we can either use $(1+\mathrm{d}_{word_j})$ as the denominator or just put $\mathrm{d}_{word_j} = 1$ only when $\mathrm{d}_{word_j} = 0$.

- if D = d_{word_j}, then the value of IDF will be 0. Again, if we use (1+d_{word_j}) as denominator then when D = d_{word_j}, there can be a negative value. In both of the cases, we can use a very small positive constant value instead of 0 or negative.

Time to go for an example. The training documents are

Document 1 (D_1): {topic name: "Sports", word list: [<play,2> <cricket,1> <football,1>, total_word=4]}
Document 2 (D_2): {topic name: "Music", word list: [<play,1> <music,1>, total_word=2]}
Document 3 (D_3): {topic name: "Music", word list: [<like,1> <singing,1>, total_word=2]}
Document 4 (D_4): {topic name: "Biology", word list: [<cricket,1> <very,1> <small,1> <insect,1>, total_word=4]}

and the test one is

Document t (D_t): {topic name: "null", word list: [<want,1> <play,1> <music,1>, total_word=3]}

Now For D_1,

TF-IDF$_{play,D_1}$ = (2/4)*(log(4/3)) = 0.5*0.1249 = 0.06247

Note that, we are using (1+d_{word_j}) here as the denominator in IDF. Definitely, you can use d_{word_j} but in that case carefully handle the Division-by-Zero problem.

TF-IDF$_{cricket,D_1}$ = (1/4)*(log(4/3)) = 0.25*0.1249 = 0.03123
TF-IDF$_{football,D_1}$ = (1/4)*(log(4/2)) = 0.25*0.301 = 0.07525
Similarly, for D_t,

TF-IDF$_{want,D_t}$ = (1/3)*(log(4/1)) = 0.333*0.602 = 0.20069
TF-IDF$_{play,D_t}$ = (1/3)*(log(4/3)) = 0.333*0.1249 = 0.04165
TF-IDF$_{cricket,D_t}$ = (1/3)*(log(4/3)) = 0.333*0.1249 = 0.04165

So, In vector space, the vectors would be

D_1 = 0want + 0.06247play + 0.03123cricket + 0.07525football
D_t = 0.20069want + 0.04165play + 0.04165cricket + 0football

Now, using the above described formula, we can easily compute the cosine similarity between D_1 and D_t. We can also calculate the similarity of D_2, D_3 and D_4 with our test document D_t, and take the one which gives the maximum similarity, i.e., maximum score.

2 Naive Bayes

If D_t means test document and C_m is the mth topic/class name, then

$$P(C_m|D_t) = \frac{P(D_t|C_m) * P(C_i)}{\sum_{k=1}^{x} P(D_t|C_k)} \tag{4}$$

The denominator of equation4 is constant for every class. So, we can safely omit it to reduce the calculation overhead.
Now, if D_t consists of n number of words, such as, w_1, w_2,..., w_n

$$\begin{aligned} P(D_t|C_m) &= P(w_1, w_2, ..., w_n|C_m) \\ &= P(w_1|w_2, ..., w_n, C_m)P(w_2|w_3, ..., w_n, C_m)...P(w_n|C_m) \\ &= P(w_1|C_m)P(w_2|C_m)...P(w_n|C_m) \end{aligned} \tag{5}$$

In the last line of equation5, we assume that the words are independent. This may not be correct theoretically but works well in practice during text classification. Merging all we get,

$$P(C_m|D_t) = P(w_1|C_m) * P(w_2|C_m) * ... * P(w_n|C_m) * P(C_m) \tag{6}$$

Now, P(w_j | C_m) means the probability of word w_j in class/topic C_m. For this, we combine all the training documents under a specific class/topic into a single document. Now, calculate P($w_j|C_m$).

$$P(w_j|C_m) = \frac{N_{w_j,C_m}}{N_{C_m}} \tag{7}$$

where N_{w_j,C_i} is the total number of word w_j under the class/topic C_i and N_{C_1} is the total number of words under the class/topic C_m.

If N_{w_j,C_m}=0, then P(w_j | C_m) would be 0 leading to P(C_m | D_t)=0. To avoid this problem we use smoothing factor.

$$P(w_j|C_m) = \frac{N_{w_j,C_m} + \alpha}{N_{C_m} + \alpha * |V|} \tag{8}$$

where, |V| is total number of different words present in all the training documents and α is the smoothing factor. If $\alpha = 1$, then we call it Laplace Smoothing Factor. If α <1, then we call it Lidstone Smoothing Factor.

3 References

1. Class Lectures
2. Wikipedia

4 Version and Errors

- This version may contain some grammatical and logical errors. Increment of the version number indicates changes.