IR+DM

Jens Kristian R. Nielsen, Thomas D. Vinther

17. maj 2019

1 Abstract

Given a small data set we apply the information gain algorithm to achieve a decision tree. We see that outlook is the most deciding factor, and that temperature is irrelevant according to the data. We developed a small Scala program that can can calculate the entropy of a list of data, and calculate the information gain, these methods work in full generality. However the program does not perform the decomposition's.
We developed a program that runs a 2-means clustering algorithm with Manhatten distance, and found that in the given data 2 nodes had to switch cluster for the data to be optimally partitioned. Finally we compare 3 small documents for similarity using TF-IDF, and find that the first two are quite similar while the last one is completely autonomous. We preprocessed the data by first removing stopwords, then we removed suffix and prefix, we did not find any synonyms. We developed a small program that performs the TF-IDF procedure on 3 preprocessed documents, it is not far off being scale-able to more documents, but we chose not to implement scaleability at this time.

2 Solution

2.1 A

Consider the data in the table about playing tennis. Apply the information gain based algorithm to obtain a decision tree. Hint: the lecture on Data Mining shows how to get started (and you can use the numbers for humidity and wind on the top level to see if you are on the right track).

We use the information gain algorithm as seen in the lectures, where we choose class to be our target/ decision. And so we start to calculate the information gain, for building our decision tree by looking at the different splits as seen in the tables below:

$O_X \backslash C$	P	N
Sunny	2	3
Overcast	4	0
Rain	3	2

$T_X \backslash C$	P	N
Hot	2	2
Mild	4	2
Cool	3	1

$H_X \backslash C$	P	N
High	3	4
Normal	6	1

$W_X \backslash C$	P	N
False	6	2
True	3	3

$$\begin{aligned}\text{informationGain}(T, A) &= \text{Entropy}(T) - \sum_{i=1}^{m} \frac{|T_i|}{|T|} \cdot \text{Entropy}(T_i) \\ &= \text{Entropy}(T) - \sum_{a \in A} P(a) \cdot \text{Entropy}(a)\end{aligned}$$

Where A is a partition of T, and the probability for a in A is $P(a) = |a|/|T| = |a|/(\sum_{a \in A} |a|)$. For our purposes we partition the data such that each partition has the same value of one of the attributes, as we will see later. We will write $IG(T, B) = informationGain(T, A_B)$ with B as an attribute instead of a partition, when we do so A_B is the partition of T on each of the possible values of B.
The entropy is in general calculated by the following forumla, with convention $0 \cdot \log_2(0) = 0$

$$\text{Entropy}(T) = -\sum_{i=1}^{k} p_i \cdot \log_2 p_i$$

In our situation the entropy is with regards to class, so the sum becomes

$$\begin{aligned}\text{Entropy}(T) &= -(p_P \cdot \log_2 p_P + p_N \cdot \log_2 p_N), \text{ where } p_i = |\{t \in T \mid t.class = i\}| \\ &=: E(p_P, p_N)\end{aligned}$$

We start by calculating the entropy for the entire dataset, denoted as X:

$$\text{Entropy}(X) = -((5/14)\ \log_2(5/14) + (9/14)\ \log_2(9/14)) = 0.9403$$

And then the information gain for class and outlook with $X_1 = \{1, 2, 8, 9, 11\}$ the tuples with t.overlook=Sunny, $X_2 = \{3, 7, 12, 13, \}$ the tuples with t.overlook=Overcast and $X_3 = \{4, 5, 6, 10, 14\}$

the tuples with t.overlook=rain:

$$\begin{aligned}
\text{IG}(X, \mathit{Outlook}) =& 0.9403 - ((3+2)/14.0 * E(3,2) + (4+0)/14.0 * E(4,0) + \\
& (2+3)/14.0 * E(2,3)) \\
E(2,3) =& -((2.0/5)\log_2(2.0/5) + (3.0/5)\log_2(3.0/5)) = 0.9710 \\
E(4,0) =& -((4.0/4)\log_2(4.0/4) + 0 = 0 \\
E(3,2) =& -((3.0/5)\log_2(3.0/5) + (2.0/5)\log_2(2.0/5)) = 0.9710 \\
\text{IG}(X, \mathit{Outlook}) =& 0.9403 - ((3+2)/14.0 \cdot 0.9710) + (4+0)/14.0 \cdot 0) + \\
& (2+3)/14.0 \cdot 0.9710)) \\
=& 0.2467 \\
\text{IG}(X, \mathit{temperature}) =& 0.9403 - ((2+2)/14.0 \cdot E(2,2) + (4+2)/14.0 \cdot E(4,2) + \\
& (3+1)/14.0 \cdot E(3,1)) \\
=& 0.9403 - ((2+2)/14.0 \cdot 1.0) + (4+2)/14.0 \cdot 0.9183) + \\
& (3+1)/14.0 \cdot 0.8113)) \\
=& 0292 \\
\text{IG}(X, \mathit{humidity}) =& 0.9403 - ((3+4)/14.0 \cdot E(3,4) + (6+1)/14.0 \cdot E(6,1)) \\
=& 0.9403 - ((3+4)/14.0 \cdot 0.9852) + (6+1)/14.0 \cdot 0.5917)) \\
=& 0.1518 \\
\text{IG}(X, \mathit{windy}) =& 0.9403 - ((6+2)/14.0 \cdot E(6,2) + (3+3)/14.0 \cdot E(3,3)+) \\
=& 0.9403 - ((6+2)/14.0 \cdot 0.8113) + (3+3)/14.0 \cdot 1.0)) \\
=& 0481
\end{aligned}$$

From the information gained we choose outlook as our decision node as it has the highest information gain, and we build our tree accordingly, splitting into three branches: Sunny, overcast and rain. And as we have already calculated the entropy for each branch above, we see that overcast should be a leaf node in the decision tree as its entropy is equal to zero.

We calculate the information gains in table X_1

$T_{X_1} \backslash C$	P	N
Hot	0	2
Mild	1	1
Cool	1	0

$H_{X_1} \backslash C$	P	N
High	0	3
Normal	2	0

$W_{X_1} \backslash C$	P	N
False	1	2
True	1	1

$$\begin{aligned}
\text{IG}(X_1, \textit{temperature}) =& 0.9710 - ((0+2)/5.0 \cdot E(0,2) + (1+1)/5.0 \cdot E(1,1) + (1+0)/5.0 \cdot E(1,0)) \\
=& 0.9710 - ((0+2)/5.0 \cdot 0) + (1+1)/5.0 \cdot 1.0) + (1+0)/5.0 \cdot -0)) \\
=& 0.5710 \\
\text{IG}(X_1, \textit{humidity}) =& 0.9710 - ((0+3)/5.0 \cdot E(0,3) + (2+0)/5.0 \cdot E(2,0)) \\
=& 0.9710 - ((0+3)/5.0 \cdot 0) + (2+0)/5.0 \cdot 0)) \\
=& 0.9710 \\
\text{IG}(X_1, \textit{windy}) =& 0.9710 - ((1+2)/5.0 \cdot E(1,2) + (1+1)/5.0 \cdot E(1,1)) \\
=& 0.9710 - ((1+2)/5.0 \cdot 0.9183) + (1+1)/5.0 \cdot 1.0)) \\
=& 0.0200
\end{aligned}$$

So we split according to the highest information gain, in this case for humidity.

$$X_{(1,1)} = \{1, 2, 3\} = \{x \in X \mid x.\textit{outlook} = \textit{Sunny} \wedge x.\textit{humidity} = High\}$$
$$X_{(1,2)} = \{9, 11\} = \{x \in X \mid x.\textit{outlook} = \textit{Sunny} \wedge x.\textit{humidity} = \textit{normal}\}$$

As we have seen the entropy of both of these sets are 0, so they are leaves in the decision tree.

We calculate the information gains in table X_3

$T_{X_3} \backslash C$	P	N
Hot	0	0
Mild	2	1
Cool	1	1

$H_{X_3} \backslash C$	P	N
High	1	1
Normal	2	1

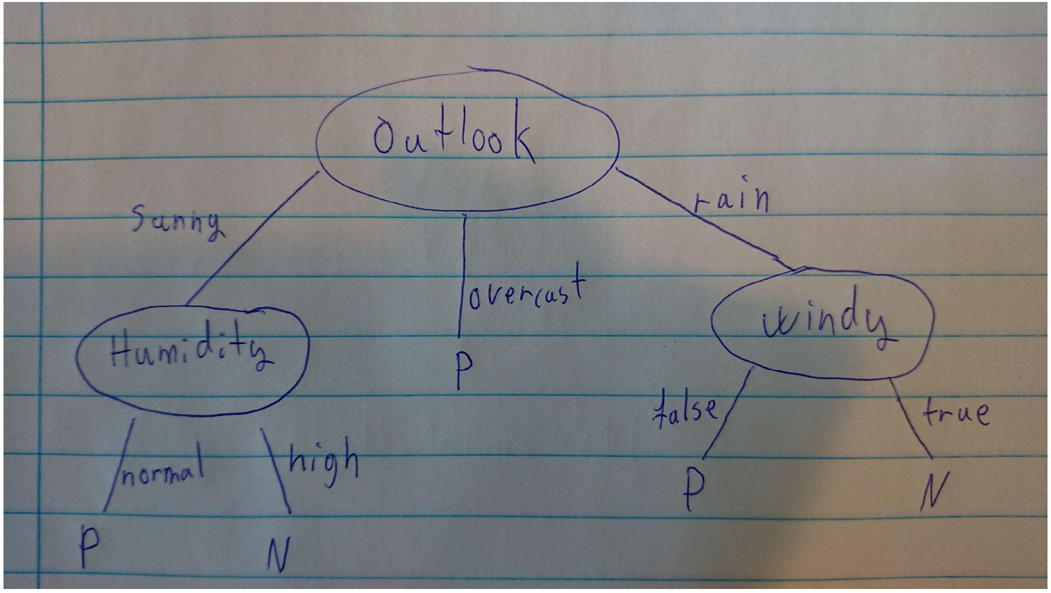
$W_{X_3} \backslash C$	P	N
False	3	0
True	0	2

$$\begin{aligned}
\text{IG}(X_3, \textit{temperature}) =& 0.9710 - ((0+0)/5.0 \cdot E(0,0) + (2+1)/5.0 \cdot E(2,1) + (1+1)/5.0 \cdot E(1,1)) \\
=& 0.9710 - ((0+0)/5.0 \cdot 0) + (2+1)/5.0 \cdot 0.9183) + (1+1)/5.0 \cdot 1.0)) \\
=& 0.0200 \\
\text{IG}(X_3, \textit{humidity}) =& 0.9710 - ((1+1)/5.0 \cdot E(1,1) + (2+1)/5.0 \cdot E(2,1)) \\
=& 0.9710 - ((1+1)/5.0 \cdot 1.0) + (2+1)/5.0 \cdot 0.9183)) \\
=& 0.0200 \\
\text{IG}(X_3, \textit{windy}) =& 0.9710 - ((3+0)/5.0 \cdot E(3,0) + (0+2)/5.0 \cdot E(0,2))E(0,2)) \\
=& 0.9710 - ((3+0)/5.0 \cdot 0) + (0+2)/5.0 \cdot 0)) \\
=& 0.9710
\end{aligned}$$

So we split according to the highest information gain, in this case for windy.

$$X_{(3,1)} = \{6, 14\} = \{x \in X \mid x.\textit{outlook} = \textit{Rain} \wedge x.wind = True\}$$
$$X_{(3,2)} = \{4, 5, 10\} = \{x \in X \mid x.\textit{outlook} = \textit{Rain} \wedge x.wind = False\}$$

As we have seen the entropy of both of these sets are 0, so they are leaves in the decision tree. This was the last decomposition so we are ready to construct the tree:



2.2 B

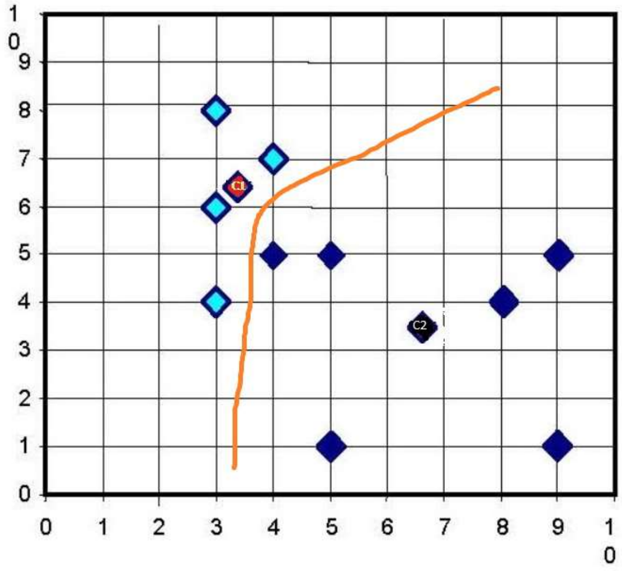
Consider the data and its initial partitions (light blue, dark blue) depicted on the right side. Apply the k-means algorithm to find two clusters. Instead of the (usual) Euclidean Distance between points, use the Manhattan Distance. The Manhattan Distance between two points is the sum of absolute differences in each dimension (so basically, no need to square and take the square root as in Euclidean Distance). Formally, MD(x,y)=, where x=(x1, x2) is a point with 2 dimensions as is the case here. Using Manhattan Distance, it should be relatively easy to sketch the steps and results of k-means in (copies of) the figure.

We use the k-means algorithm to find the two desired clusters. First we calculate the two means for the first two given clusters using:

$$\mu_c = \frac{1}{|C|} \sum_{x_i \in C} x_i$$

$$\text{MD}(x,y) = \sum_{i=1}^{n} |x_i - y_i|, \text{ for } x, y \in \mathbb{R}^n$$

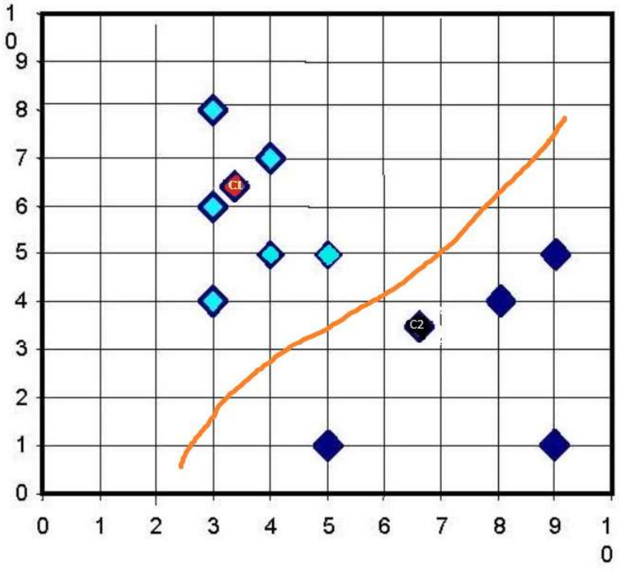
Giving us the two centroids: $c1 = (3.25, 6.25)$, marked with red, and $c2 = (6.\overline{6}, 3.5)$ marked with black.



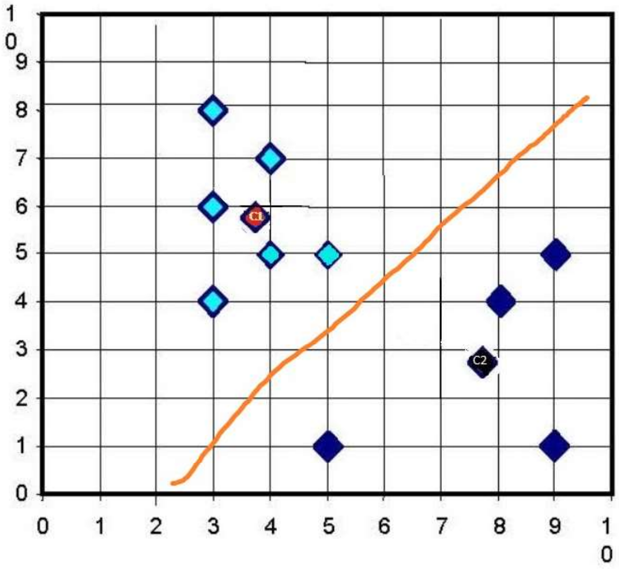
We then calculate which points are closest w.r.t. the Manhatten Distance to each centroid and divide into two new clusters.

p	$d(p, c1)$	$d(p, c2)$	new cluster
$(3, 4)$	2.5	$4.1\overline{6}$	1
$(3, 6)$	0.5	$6.1\overline{6}$	1
$(3, 8)$	2.0	$8.1\overline{6}$	1
$(4, 7)$	1.5	$6.1\overline{6}$	1
$(4, 5)$	2.0	$4.1\overline{6}$	1
$(5, 1)$	7.0	$4.1\overline{6}$	2
$(5, 5)$	3.0	$3.1\overline{6}$	1
$(8, 4)$	7.0	$1.8\overline{3}$	2
$(9, 1)$	11.0	$4.8\overline{3}$	2
$(9, 5)$	7.0	$3.8\overline{3}$	2

Giving us cluster1= $\{(5, 5), (4, 5), (4, 7), (3, 8), (3, 6), (3, 4)\}$ and cluster2=$\{(9, 5), (9, 1), (8, 4), (5, 1)\}$ As seen in the picture below.



We repeat the procedure, once again calculating the two centroids, which gives us $(3.\overline{6}, 5.8\overline{3})$ and $(7.75, 2.75)$ we calculate the distances for each point and partition, with no change, the clusters remain the same. We iterate through the algorithm once again, when calculating the centroids, we get the same as before and the algorithm stops.



2.3 C

C. Take the following three text examples:
“Unlike classification or prediction, which analyzes data objects with class labels, clustering analyzes data objects without consulting a known class label.”
“Classification can be used for prediction of class labels of data objects. However, in many applications, prediction of missing values is performed to fit data objects into a schema.”
“Sun Salutation, a ritual performed in the early morning, combines seven different postures. The sun, the life generator, is invoked by this Yogic exercise.”
Using word frequency (simple word counts), Euclidean Distance and Cosine Similarity, as well as the following “stop words”, which of these are most similar? Stopwords = { a, an, are, be, because, by, can, for, however, in, into, is, keep, many, not, of, or, rather, than, the, they, this, to, unlike, used, way, which, with, without }

We preprocess the data
Step 1: removing stop words

“Unlike classification or prediction, which analyzes data objects with class labels, clustering analyzes data objects without consulting a known class label.”

“Classification can be used for prediction of class labels of data objects. However , in many applications, prediction of missing values is performed to fit data objects into a schema.”

“Sun Salutation, a ritual performed in the early morning, combines seven different postures. The sun, the life generator, is invoked by this Yogic exercise."

Step 2: stemming, trimming the suffix and prefix of an original word

“Unlike classification prediction, analyzes data objects class labels, clustering analyzes data objects consulting known class label.”

“Classification used prediction class labels data objects. applications, prediction missing values performed fit data objects schema.”

“Sun Salutation, ritual performed early morning, combines seven different postures. sun, life generator, invoked Yogic exercise."

Step 3: eliminate synonyms, we did not find any. The preprocessed data is:

“Unlike classification prediction, analyze data object class label, clustering analyze data object consult known class label.”

“Classification used prediction class label data object. application, prediction missing value perform fit data object schema.”

“Sun Salutation, ritual perform early morning, combine seven different posture. sun, life generator, invoke Yogic exercise."

Word frequency: We take the set of all the observed words after preprocessing and make a vector v_0 to indicate which indicies represent which words.
v_0 =(unlike,classification,prediction,analyze,data,object,class,label,clustering,consult, known,used,missing,value,fit,schema,sun,salutation,ritual,perform,early,morning, combine,seven,different,posture,life,generator,invoke,yogic,exercise)
We then count the word counts for each document for each index and get

$$\begin{aligned}
f_1 &= (1,1,1,2,2,2,2,2,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)\\
f_2 &= (0,1,2,0,2,2,1,1,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)\\
f_3 &= (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
\end{aligned}$$

We now calculate the TF-IDF scores:

$$\begin{aligned}
\mathrm{TF}_{(i,j)} &= \frac{f_{(i,j)}}{\max\{f_{(k,j} \mid 1 \leq k \leq N\}}\\
\mathrm{IDF}_i &= \log\left(\frac{N}{|\{d \mid t_i \in d\}|}\right) = \log\left(\frac{N}{|\{j \mid f_{(i,j)} > 0\}|}\right)\\
w_{(i,j)} &= \mathrm{TF}_{(i,j)} \cdot \mathrm{IDF}_i
\end{aligned}$$

The IDF calculation denominator is the number of documents in which word i occurs.
See the appendix for the scala program we wrote to perform these calculations.

$$\begin{aligned}
w_1 = (&0.5493, 0.2027, 0.20273, 1.0986, 0.4055, 0.4055, 0.4055, 0.4055, 0.5493, 0.5493, 0.5493,\\
&0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.0, 0, 0, 0, 0, 0, 0, 0, 0, 0)\\
w_2 = (&0, 0.2027, 0.4055, 0, 0.4055, 0.4055, 0.2027, 0.2027, 0, 0, 0, 0.5493, 0.5493, 0.5493, 0.5493, 0.5493,\\
&0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.0, 0, 0, 0, 0, 0, 0, 0, 0, 0)\\
w_3 = (&0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,\\
&1.0986, 0.5493, 0.5493, 0.5493, 0.5493, 0.5493, 0.5493, 0.5493, 0.5493,\\
&0.5493, 0.5493, 0.5493, 0.5493, 0.5493, 0.5493)
\end{aligned}$$

Recall Cosine similarity:

$$\mathrm{CosSim}(A, B) := \frac{A \cdot B}{||A||\ ||B||}$$

Now we calculate using euclidean distance for all pairs, up to permutation as CosSim(A,B) = CosSim(B,A)

$$\begin{aligned}
\mathrm{CosSim}(w_1, w_2) &= \frac{v_1 \cdot v_2}{||v_1||\ ||v_2||} = \frac{0.6165}{1.7759 \cdot 1.4578} = 0.2381\\
\mathrm{CosSim}(w_1, w_3) &= \frac{v_1 \cdot v_3}{||A||\ ||B||} = \frac{0}{1.7759 \cdot 2.3305} = 0 && \text{the dot product is clearly 0}\\
\mathrm{CosSim}(w_2, w_3) &= \frac{v_2 \cdot v_3}{||A||\ ||B||} = \frac{0}{||A||\ ||B||} = 0 && \text{the dot product is clearly 0}
\end{aligned}$$

We conclude that document 1 and 2 are decently similar and both are completely different to document 3.

3 Summary

In this assignment we looked at different sorts of data and using different algorithms to retrieve information from the data.

First we looked at building a decision tree from a table using the information gain algorithm. We decided to use the class attribute as our decision/target. We quickly noticed we needed to do a lot of similar calculations and therefore decided to implement the algorithm in Scala, as to hopefully avoid calculation errors. We repeated the procedure and ended up with a decision tree, making it faster and easier to find out which class we would wind up in, depending on the other attributes. For instance, if the outlook was overcast we would always end up in class P, no matter the other conditions.

In the second part we looked at the k-means cluster for two given clusters, again we wrote the algorithm in Scala as to avoid calculation errors. The algorithm only needed a few iterations, and the clusters were quite intuitive when looking at the figure. The algorithm made it very easy to decide which cluster the given points belonged to.

In the last part of this assignment, we looked at data as three text examples and compared them. We processed the data, first by removing all stop words, then trimming suffixes and prefixes and then using a simple word count, before calculating the TF-IDF scores. Again we wrote the algorithm in Scala to do the heavy calculations of the TD-IDF scores, before doing the Cosine Similarity. This was quite easy when calculating with the last document, as the Euclidean Dot Product was zero, and therefore very easy to calculate. When comparing the first and second document, we found that they were decently similar, which generally made good sense.

All in all we have processed a lot of different data using different algorithms, and learned the implementation of these. It was nice to actually experience some implementation of real algorithms, instead of just doing them by hand. We would like to experience more of this.

4 Appendix

Listing 1: Task A

```scala
import scala.math
object DB6A {

  def main(args: Array[String]): Unit = {
    println("outlook and class")
    println(gainz(List(5,9),List((3,2),(4,0),(2,3))))
    println("temp and class")
    println(gainz(List(5,9),List((2,2),(4,2),(3,1))))
    println("hum and class")
    println(gainz(List(5,9),List((3,4),(6,1))))
    println("windy and class")
    println(gainz(List(5,9),List((6,2),(3,3))))

    println()
    println("This is the new shit: ")
    println()
    println("Sunny and temperature")
    println(gainz(List(2,3),List((0,2),(1,1),(1,0))))

    println("Sunny and Humidity")
    println(gainz(List(2,3),List((0,3),(2,0))))

    println("Sunny and windy")
    println(gainz(List(2,3),List((1,2),(1,1))))

    println("Rain:")
    println("temp and rain")
    println(gainz(List(3,2),List((0,0),(2,1),(1,1))))
    println("hum and rain")
    println(gainz(List(3,2),List((1,1),(2,1))))
    println("wind and rain")
    println(gainz(List(3,2),List((3,0),(0,2))))

  }
  def entropy(prg : List[Int]): Double = {
        var sum :Double = 0
        val prgSum = prg.reduce((x,y) => x+y)
        var strg = "-("
        for (i <- prg){
          if(i==0){
              strg += "0+"
          }
          else{
          sum = sum + (i.toDouble/prgSum)*math.log(i.toDouble/prgSum)/math.log(2)
          strg += " ("+i.toDouble +" / " +prgSum +""") \text{ log}_2( """+ i.toDouble+"/"+ prgSum+")+"}
        }
      //println(strg+")="+ -sum)
    -sum
  }

  def gainz(T : List[Int],A : List[(Int,Int)]):Double ={
    var res : Double = entropy(T)
    var strg : String = res.toString+ "-("
    var strg2 = strg
    val sum:Double = A.reduce((x,y)=> (x._1+x._2+y._1+y._2,0))._1
    for(i <- A){
      res += -(i._1+i._2).toDouble/sum * entropy(List(i._1,i._2))
      strg = strg +s"("+i._1+"+"+i._2+")/"+sum+""" \cdot"""+" E("+i._1+","+i._2+")+"
      strg2 = strg2 +s"("+i._1+"+"+i._2+")/"+sum+""" \cdot """+entropy(List(i._1,i._2))+")+"
    }
    strg +=")"
    println(strg)
    println(strg2+")")
    res
  }
}
```

Listing 2: Task B

```scala
object DB6b {
  def main(args: Array[String]): Unit = {
    println(mean(List((3,4),(3,6),(3,8),(4,7))))
    println(mean(List((4,5),(5,1),(5,5),(8,4),(9,1),(9,5))))
    println(manD((1,2),(1,3)))
    println(loopy(List((3,4),(3,6),(3,8),(4,7)),List((4,5),(5,1),(5,5),(8,4),(9,1),(9,5))))
  }

  def mean(prg: List[(Int,Int)]): (Double, Double) = {
    val c : Double =1.0/ prg.length
    val res = prg.foldLeft(0,0)((a:(Int,Int),b:(Int, Int)) =>
      (a._1+b._1,a._2+b._2))
    (res._1.toDouble*c,res._2.toDouble*c)
  }

  def td(prg: List[(Int,Int)]): Double = {
      val centroid = mean(prg)
      val res = prg.foldLeft(0.0)((a:(Double),b:(Int, Int)) =>
        a+math.pow(manD(b,centroid),2))
      math.pow(res,0.5)
  }

  def manD(point : (Int,Int), centroid: (Double,Double)):Double ={
    math.abs(point._1-centroid._1)+math.abs(point._2-centroid._2)
    //Flying is for droids
  }

  def update(prg : List[(Int,Int)],c1 : (Double,Double), c2: (Double,Double)): (List[(Int,Int)],List[(Int,Int)])={
    var res1= List[(Int,Int)]()
    var res2 = List[(Int,Int)]()
    for(a <- prg){
      println(a+" &"+manD(a,c1)+" &"+manD(a,c2))
      if(manD(a,c1)<=manD(a,c2)){
        res1 =  a :: res1
      }
      else{
        res2 = a :: res2
      }
    }
    (res1,res2)
  }

  def loopy(prg : (List[(Int,Int)],List[(Int,Int)])): (List[(Int,Int)],List[(Int,Int)]) ={
    val fullList = prg._1 ++ prg._2
    var c1new = mean(prg._1)
    println("First Centroid:"+c1new)
    var c1old = (0.0,0.0)
    var c2new = mean(prg._2)
    println("Second Centroid:"+c2new)
    var c2old = (0.0,0.0)
    var res = prg
    while(!compare(c1new._1, c1old._1,0.0001) && !compare(c1new._2, c1old._2
        ,0.0001) && !compare(c2new._1, c2old._1,0.0001)
      && !compare(c2new._2, c2old._2,0.0001)){
      res = update(fullList,c1new,c2new)
      println("Print res: "+res)
      c1old = c1new
      c1new = mean(res._1)
      println("c1new: "+c1new)
      c2old = c2new
      c2new = mean(res._2)
      println("c2new: "+c2new)
    }
    res
  }

  def compare(x: Double, y: Double, precision: Double)={
    if(math.abs(x-y)<precision){ true}
    else{ false}
  }
}
```

Listing 3: Task C

```
object DB6c{
def main(args: Array[String]): Unit = {
  val print = TFIDFweights(List(1,1,1,2,2,2,2,2,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
                              List(0,1,2,0,2,2,1,1,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
                              List(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1))
  println(CosSim(print._1,print._2))
  println(CosSim(print._1,print._3))
  println(CosSim(print._2,print._3))
}
  def TFIDFweights(list1 : List[Int],list2 : List[Int], list3: List[Int]): (List[Double],List[Double],List[Double]) = {
    var res1: List[Double] = List()
    var res2: List[Double] = List()
    var res3: List[Double] = List()
    val max1: Double = list1.foldLeft(0)((a, b) => math.max(a, b))
    val max2: Double = list2.foldLeft(0)((a, b) => math.max(a, b))
    val max3: Double = list3.foldLeft(0)((a, b) => math.max(a, b))

    for (i <- list1.indices) {
      var b = 0
      if (list1(i) > 0) {
        b += 1
      }
      if (list2(i) > 0) {
        b += 1
      }
      if (list3(i) > 0) {
        b += 1
      }
      res1 = list1(i).toDouble / max1 * math.log(3.0 / b) :: res1
      res2 = list2(i).toDouble / max2 * math.log(3.0 / b) :: res2
      res3 = list3(i).toDouble / max3 * math.log(3.0 / b) :: res3
    }
    (res1.reverse, res2.reverse, res3.reverse)
  }
  def CosSim(list1: List[Double],list2: List[Double]): Double = {
    val length1:Double = math.pow(list1.foldLeft(0.0)((a,b)=>a+math.pow(b,2)),1/2)
    val length2:Double = math.pow(list2.foldLeft(0.0)((a,b)=>a+math.pow(b,2)),1/2)
    val dotproduct:Double = list1.zip(list2).foldLeft(0.0)((a,b)=> a+b._1*b._2)
    dotproduct/(length1*length2)
  }
}
```