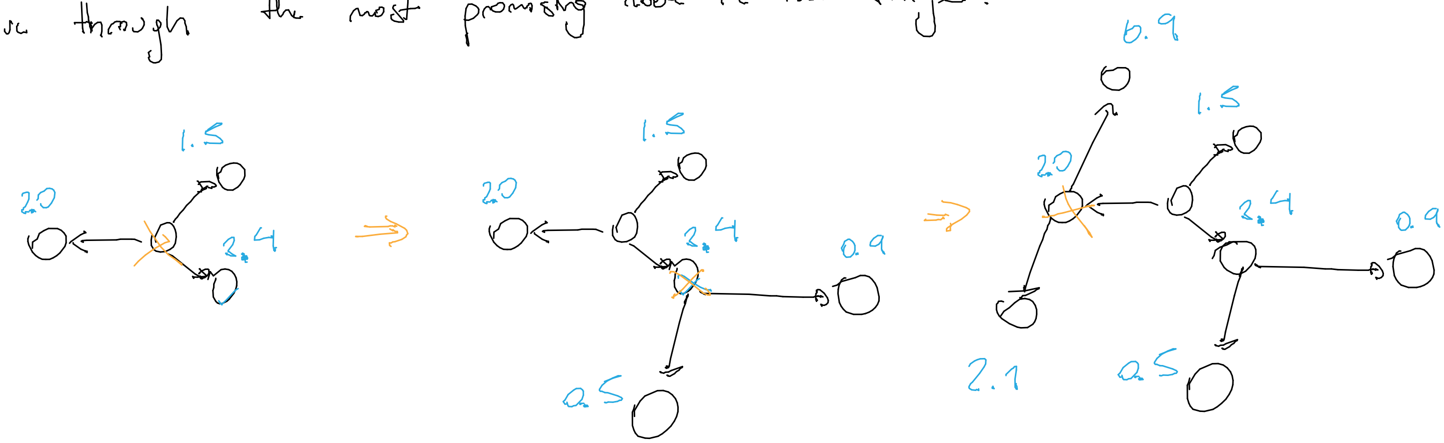
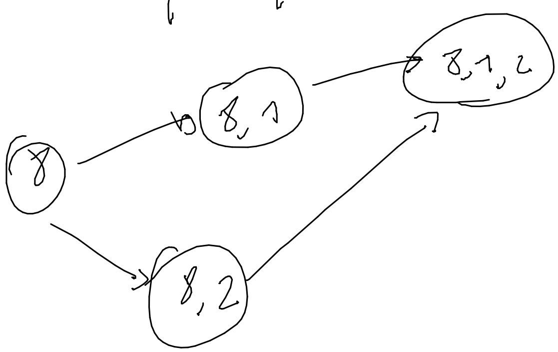
Best First Search

The search done in CFS is correctly represented using a graph with a fringe that represents all the nodes that haven't been expanded, the search will continue through the most promising node in the fringe.



The best way to implement this is by using a PriorityQueue, as is done in the WEKA implementation.

* Note that this cannot be represented by a tree because a state can be reached in many ways:



Best First Search

WRONG TREE-BASED APPROACH : check note below

```
def expandSubset (subset: Subset, bestSubset: Subset, bestChild: Int, bestChildMerit: Double,
        numOfFails: Int
    ...): Subset {
if (bestChild == -1) bestSubset //no more feats to search
else {
  if bestChildMerit > bestSubset.merit
    expand with {bestChild} + subset = bestSubset, numOfFails = 0
  else if numOfFails < maxNumOfFails
    if bestChildrenMerit > subset.merit
      expand with {bestChild} + subset, bestSubset keeps same, numOfFails+1
  else
    expandSubset with subset - {head} and in bestChild don't consider {head} or better
              bestSubset keep the same, numOfFails+1
else
  bestSubset
```

Tree-based Search

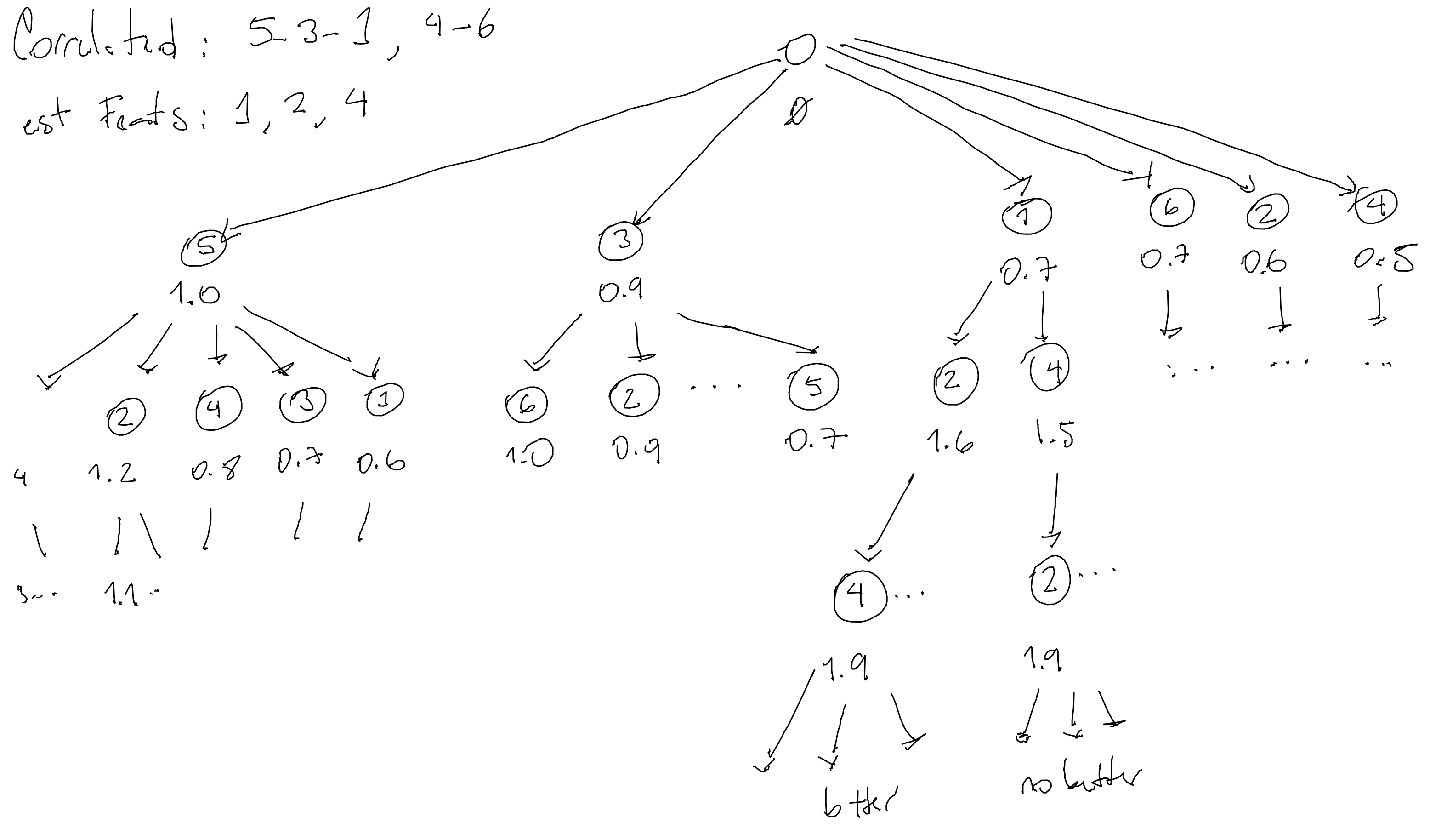
subset	bestSubset	bestChild	fails
[], -∞	[], -∞	5, 1.0	0
[5], 1.0	[5], 1.0	6, 1.4	0
[5,6], 1.4	[5,6], 1.4	3, 1.3	0
[5], 1.0	[5,6], 1.4	2, 1.2	1
[5,2], 1.2	[5,6], 1.4	3, 0.8	2
[5], 1.0	[5,6], 1.4	4, 0.8	3
[], -∞	"	3, 0.9	4
[3], 0.9	"	6, 1.0	5
[3,6], 1.0	"	1, 0.9	6
[3], 0.9	"	2, 0.9	7
[], -∞	"	1, 0.7	8
[1], 0.7	"	2, 1.6	9
[1,2], 1.6	[1,2], 1.6	4, 1.9	0

The problem with this approach is that we can grow from any of the visited states, we have a fringe, not only from the recently visited

Even when searching through 2 is a fail, it is our best option.

Correlated: 5-3-1, 4-6

est Feats: 1, 2, 4



Symmetric Uncertainty WEKA Code Analysis

Entropy of X is defined as: $H(X) = -\sum_{x \in X} p(x) \cdot \log_2(p(x))$

$p(x) = \frac{|x|}{n}$, $\quad H(X) = -\frac{1}{n} \sum_{x \in X} |x| \log_2\left(\frac{|x|}{n}\right)$

n is the total number of instances

$$= -\frac{1}{n} \sum_{x \in X} |x| \left(\log_2(|x|) - \log_2 n\right)$$

$$= -\frac{1}{n}\left[\sum_{x \in X} |x| \log_2(|x|) - \log_2 n \cdot \underbrace{\sum_{x \in X} |x|}_{= n}\right]$$

$$= -\frac{1}{n}\left[\sum_{x \in X} |x| \log_2(|x|) - n \cdot \log_2 n\right] = -\frac{1}{n} \sum_{x \in X}\left(|x| \log_2(|x|)\right) + \log_2 n \quad (D)$$

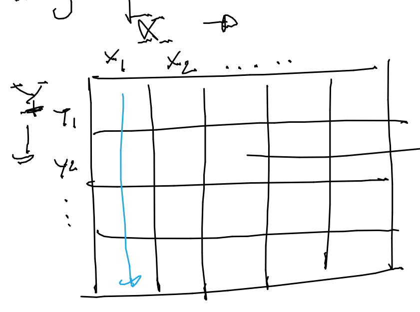
$\therefore \frac{1}{n} \sum_{x \in X} |x| \log_2(|x|) = -\left(H(X) - \log_2 n\right) \quad (C)$

(A)

Conditional Entropy of X after observing Y is defined as:

$$H(X|Y) = -\sum_{y \in Y} p(y) \sum_{x \in X} p(x|y) \log_2(p(x|y))$$

If we define X as being the columns and Y as the rows of a contingency matrix:



$\| x_3 | y_2 \| = \| y_2 | x_3 \|$

$|x| \cdot \log_2 (\ldots)$

$$H(X|Y) = -\sum_{y \in Y}\left[p(y) \sum_{x \in X} p(x|y) \cdot \log_2 p(x|y)\right] = -\frac{1}{n} \sum_{y \in Y} |y| \sum_{x \in X} \frac{|(x|y)|}{|y|} \cdot \log_2\left(\frac{|(x|y)|}{|y|}\right)$$

$$= \frac{-1}{n} \sum_{y \in Y}\left[\sum_{x \in X} |(x|y)| \cdot \log_2(|(x|y)|) - \sum_{x \in X} |(x|y)| \cdot \log_2 |y|\right]$$

$$= \frac{-1}{n} \sum_{y \in Y}\left[\sum_{x \in X} |(x|y)| \cdot \log_2(|(x|y)|) - |y| \cdot \log_2 |y|\right]$$

$$= \frac{-1}{n}\left[\sum_{y \in Y} \sum_{x \in X} |(x|y)| \cdot \log_2(|(x|y)|) - \sum_{y \in Y} |y| \cdot \log_2 |y|\right]$$

$$= \frac{-1}{n} \sum_{y \in Y} \sum_{x \in X} |(x|y)| \cdot \log_2(|(x|y)|) + \frac{1}{n} \sum_{y \in Y} |y| \cdot \log_2 |y| \quad (C)$$

check up there

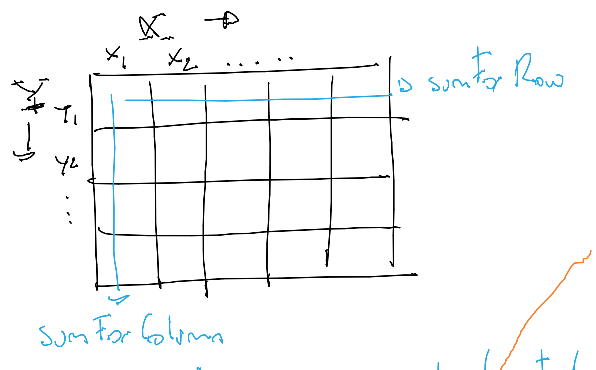
$$= \frac{-1}{n} \sum_{y \in Y} \sum_{x \in X} |(x|y)| \cdot \log_2(|(x|y)|) + \left(-H(Y) + \log_2 n\right)$$

```java
public static double symmetricalUncertainty(double matrix[][]) {

    double sumForColumn, sumForRow, total = 0, columnEntropy = 0,
      rowEntropy = 0, entropyConditionedOnRows = 0, infoGain = 0;

    // Compute entropy for columns
    for (int i = 0; i < matrix[0].length; i++) {
      sumForColumn = 0;
      for (int j = 0; j < matrix.length; j++) {
    sumForColumn += matrix[j][i];
      }
      columnEntropy += lnFunc(sumForColumn);
      total += sumForColumn;
    }
    columnEntropy -= lnFunc(total);

    // Compute entropy for rows and conditional entropy
    for (int i = 0; i < matrix.length; i++) {
      sumForRow = 0;
      for (int j = 0; j < matrix[0].length; j++) {
        sumForRow += matrix[i][j];
        entropyConditionedOnRows += lnFunc(matrix[i][j]);
      }
      rowEntropy += lnFunc(sumForRow);
    }
    entropyConditionedOnRows -= rowEntropy;
    rowEntropy -= lnFunc(total);
    infoGain = columnEntropy - entropyConditionedOnRows;
    if (Utils.eq(columnEntropy, 0) || Utils.eq(rowEntropy, 0))
      return 0;
    return 2.0 * (infoGain / (columnEntropy + rowEntropy));
}
```



(A)

$\text{columnEntropy} = \sum_{x \in X} \text{sumForColumn} \times \log_2(\text{sumForColumn})$

$= \sum_{x \in X} |x| \cdot \log_2 |x| - n \log_2(n)$

(B)

$\text{rowEntropy} = \sum_{y \in Y} |y| \cdot \log_2 |y| - n \log_2(n)$

$\text{entropyCondRows} = \sum_{x \in X} \sum_{y \in Y} |(x|y)| \log_2(|(x|y)|) - \sum_{y \in Y} |y| \log_2 |y|$

$$\text{InfoGain} = H(X) - H(X|Y)$$

$$= -\frac{1}{n}\left[\sum_{x \in X} |x| \log_2(|x|) - n \cdot \log_2 n\right]$$

$$+ \frac{1}{n}\left[\sum_{y \in Y} \sum_{x \in X} |(x|y)| \cdot \log_2(|(x|y)|) - \sum_{y \in Y} |y| \cdot \log_2 |y|\right]$$

$$= -\frac{1}{n}\left[\underbrace{\sum_{x \in X} |x| \log_2 |x| - n \log_2 n}_{\text{columnEntropy}} - \underbrace{\left(\sum_{y \in Y} \sum_{x \in X} |(x|y)| \log_2(|(x|y)|) - \sum_{y \in Y} |y| \cdot \log_2 |y|\right)}_{\text{entropyCondRows}}\right]$$

$$H(X) + H(Y) = -\frac{1}{n}\left[\sum_{x \in X} |x| \log_2 |x| + \sum_{y \in Y} |y| \log_2 |y| - 2n \log_2 n\right]$$

$$\text{Symmetric Uncertainty} = \frac{2 \times \text{InfoGain}}{H(X) + H(Y)} = 2 \times \frac{-\frac{1}{n}(\text{columnEntropy} - \text{entropyCondRows})}{-\frac{1}{n}(\text{columnEntropy} + \text{rowEntropy})}$$

$$= \frac{2\,(\text{columnEntropy} - \text{entropyCondRows})}{\text{columnEntropy} - \text{rowEntropy}}$$

Symmetric Uncertainty Scala Implementation

discretizedData — num of instances (n)

```
def entropy(data: RDD[LabeledPoints], featIndex: int, count: int): Double = {
    data.map(lp => (lp.features(featIndex), 1))
        .reduceByKey(_+_)
        .map((k,v) => v * Math.log(v))
        .sum / (-count) + Math.log(count)        => check Eq. (D)
}
```

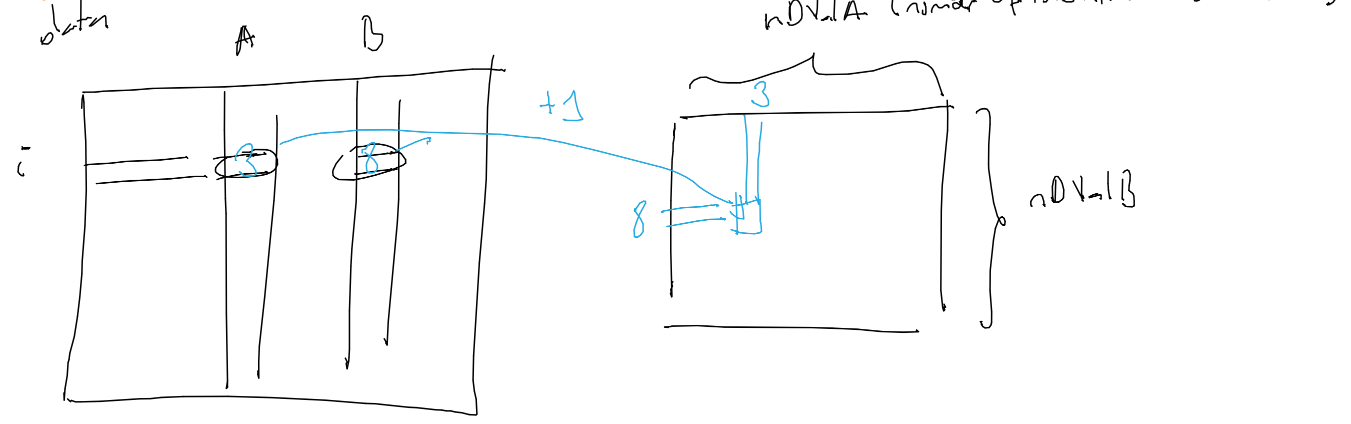
⊗ use a featIndex == lp.features.length to get class entropy.

```
def getEntropies(data: RDD[LabeledPoint]): Vector[Double] = {
    (0 to data.take(1).features.length)
        .map(f => entropy(f))
        .collect
}
```

↑ the class is in the last element.

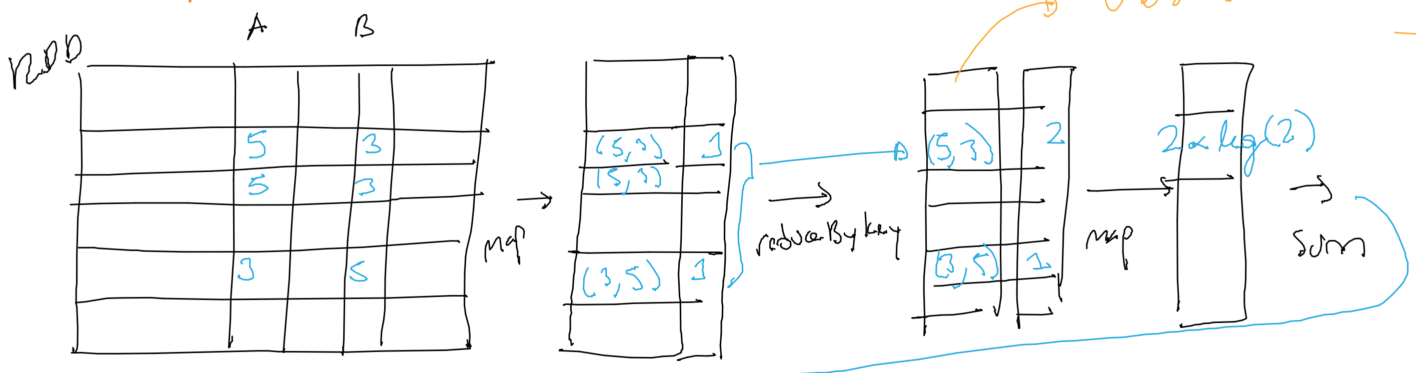
Calc Conditional Entropy

Original procedure (first part, find contingency table):



Functional procedure, find entropy directly

Observe that (5,3) != (3,5)



=> $\sum_{x\in X}\sum_{y\in Y} |x|y| \cdot \log_2 |x|y|$

H(X|Y) — X — Y — H(Y)

```
def getConditionalEntropy(data: RDD, conditionalFeat: Int, feat: Int, featEntropy: Double, count: Int): Double = {
    data
        .map(lp => ((lp.features(condFeat), lp.features(feat)), 1))
        .reduceByKey(_+_)
        .map((k, v) => v * Math.log(2))
        .sum * (-1/count) - featEntropy - Math.log(count)        (check Eq. C)
}
```

⊗ Consider the case when one feat is the class!

And finally:

```
getSimmetricUncertainty(data: RDD[LP], feat1: int, feat2: int, entropies: Vector[Double]): Double {
    val InfoGain = entropies(feat1) - getCondEntropy(....)
    2 * InfoGain / (entropies(feat1) + entropies(feat2))
}
```

Correlations Matrix

Problem:

the whole CFS correlation matrix won't fit in memory if the number of feats is in 1×10^5 order:

→ assuming 64 bits float features.

$$Req\_Mem = (Num\_of\_feats)^2 \; 64 \cancel{bits} \times \frac{1 \cancel{byte}}{8 \cancel{bits}} \times \frac{1 \text{ Gigabyte}}{(1024)^3 \cancel{byte}}$$

$$Req\_Mem\ Gb = (Num\_of\_feats)^2 \times \frac{8}{(1024)^3} Gb$$

If $Num\_of\_feats = 1\times10^5$

$$Req\_Mem\ Gb = \frac{8 \times 10^{10}}{(1024)^3} Gb \approx 74.505\ Gb$$

Solutions Ideas.

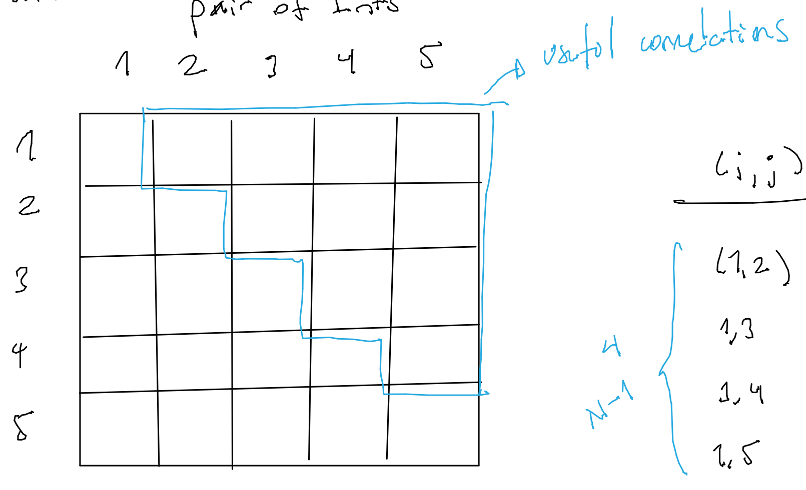
- Use a distributed matrix (Simple but not a significant contrib.) ✓
- Try with some other thing:
  - Eliminate the need to store the whole matrix.
  - Study what percentage of the matrix is really calculated in the search (should depend on the number of attempts, default: 5).

    If the percentage is small, an sparse representation should work.

* Note that the corr. matrix is symmetric, so less than half this space is really needed (the diagonal is not needed)

Distributed Correlation Matrix

We really don't need a matrix, only a storage. If we evaluate the correlations lazily, the search algorithm in the driver will be constantly asking for unevaluated correlations in a serialized fashion. So, even when not all the correlations are need, I believe that calculating all of them is more efficient. Also this is more advantageous because this way a RDD of correlations can be created, allowing to process higher dimensionaly datasets (num of feats ~ 10^4 or more). Moreover, for smaller datasets, a local matrix is better. Now, I'll examine the distributed case, the first thing needed for efficient access via the lookup action, is to implement a Partioner for the keys. Keys will be implemented using an ~~immutable BitSet~~ pair of Ints. A Partitioner assigns an id to a key.



Mapping to a linear array.

N is the num of features

	(i,j)	z	
4, N-1	(1,2)	0	
	1,3	1	
	1,4	2	
	1,5	3	
3, N-2	2,3	4	$(N-1)+j-i-1$
	2,4	5	
	2,5	6	
2, N-3	3,4	7	$(N-2)+(N-1)+j-i-1$
	3,5	8	$\sum_{k=1}^{i-1}(N-k)+j-i-1$
1, N-4	4,5	9	

$$\sum_{k=1}^{i-1}(N-k)+j-i-1$$

$$=\sum_{k=1}^{i-1}N-\sum_{k=1}^{i-1}k+j-i-1$$

$$=N(i-1)+j-i-1-\frac{(i-1)(\cancel{(i-1)}\cancel{+1})}{2}$$

$$=N(i-1)+j-i-1-\frac{i}{2}(i-1)$$

This function can be used to map two indexes (i,j) to a linear array. $(i<j)$

The steps for partitioning are:

1. Mapping to a linear array. ✓
2. Determine in which partition of the array the key falls.

The size of the linear array will be given by the position of the last element $+1$, the last element is (i,j); $i=N-1$, $j=N$

$$last\_pos=N(i-1)+j-i-1-\frac{i}{2}(i-1)\quad+1$$

And substituting:

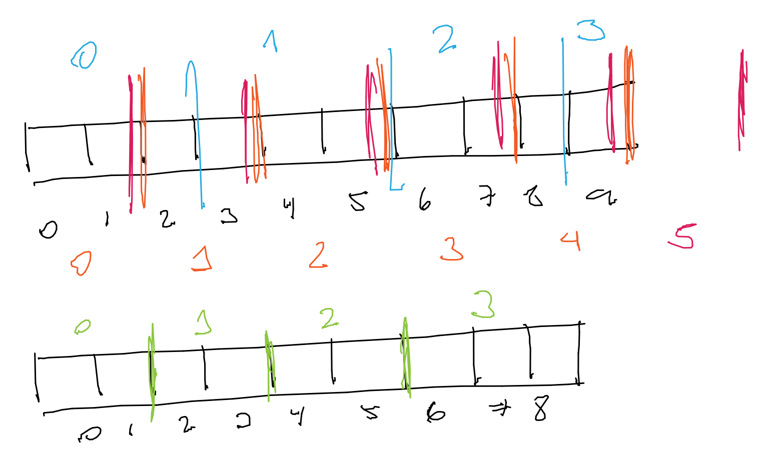
$$length=N(N-1-1)+N-(N-1)-1-\frac{(N-1)\cdot(N-1-1)}{2}$$

$$=(N-2)\cdot N+(2-N)\cdot(N-1)/2\quad+1$$

Now, when partitioning the linear array, with this formula:

$$partitionSize=round\left(\frac{length}{numPartitions}\right)$$

4 cases can happen, with respect to the last partition size:



1.) $round\left(\frac{10}{4}\right)=3$

$last\_part\_size < part\_size$

2.) $round\left(\frac{10}{5}\right)=2$

$last\_part\_size = part\_size$

3.) $round(10/6)=2$

$last\_part\_size = 0$

4.) $round(9/4)=2$

$last\_part\_size > part\_size$

For finding the correct partitionId $[0, numPartitions-1]$, I defined the following property:

pi : partitionId

ps : partitionSize

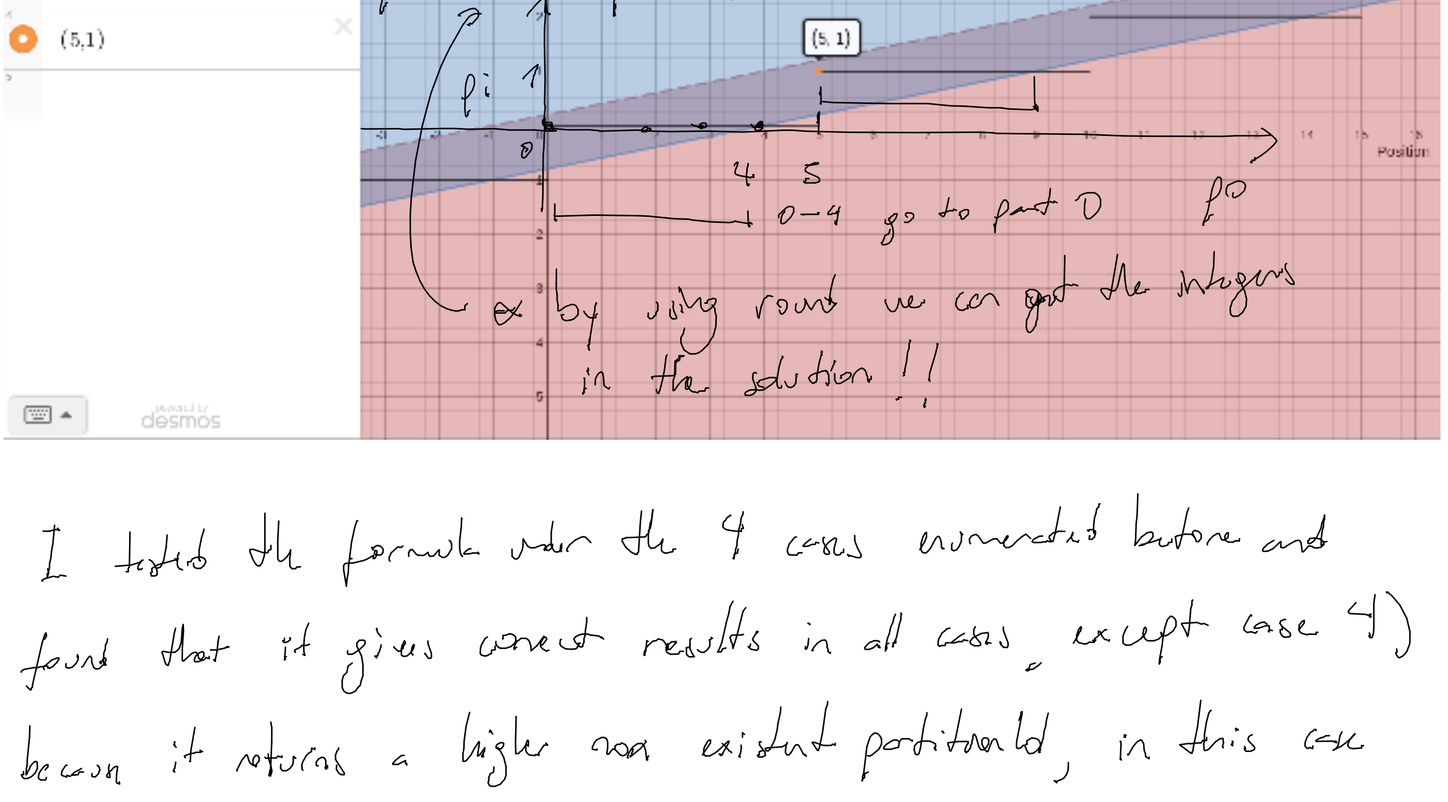
po : actual position $[0, length-1]$

nP: numPartitions $[1,\infty[$

$$pi * ps < po+1 \leq (pi+1)*ps, \quad 0\leq pi\leq nP+1$$

i.e. $pi<\frac{po+1}{ps}$, $pi\geq\frac{po+1}{ps}-1$, $pi\geq 0$, $pi\leq nP+1$

And then, by graphing, found the following solution:



I tested the formula under the 4 cases enumerated before and found that it gives correct results in all cases, except case 4) because it returns a higher non existent partitionId, in this case the maxPartitionId = numPartitions − 1 must be assigned.

Analysis of Heuristic Merit Subset Formula

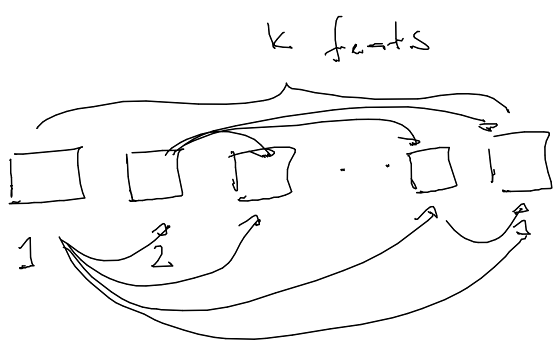
Equation 1 (Ghiselli, 1964) formalizes the heuristic:

$$Merit_s = \frac{k\overline{r_{cf}}}{\sqrt{k + k(k-1)\overline{r_{ff}}}} \qquad (1)$$

where $Merit_S$ is the heuristic "merit" of a feature subset S containing k features, $\overline{r_{cf}}$ the average feature-class correlation, and $\overline{r_{ff}}$ the average feature-feature intercorrelation. Equation 1 is, in fact, Pearson's correlation, where all variables have been standardized.

$$\overline{r_{cf}} = \sum_{i=1}^{n} r_{ci}/k$$, r_{ci} is the correlation of the i-esim feat with the class.

$$\overline{r_{ff}} = \sum_{\forall i,j \; i\neq j} r_{ij}/w$$, r_{ij} is the correlation of feat i with feat j



$$w = (k-1) + (k-2) + \cdots + 1$$

$$w = \frac{(k-1)\cdot(k-1+1)}{2}$$

$$w = (k-1)\cdot k/2$$

$$Merit_s = \frac{\cancel{k}\cdot\sum_{i=1}^{n} r_{ic}/\cancel{k}}{\sqrt{k + \cancel{k(k-1)}\sum_{\forall i\neq j} r_{ij} \Big/ \cancel{(k-1)\cdot k}/2}} = \frac{\sum_{i=1}^{n} r_{ic}}{\sqrt{k + 2\cdot\sum_{\forall i\neq j} r_{ij}}}$$

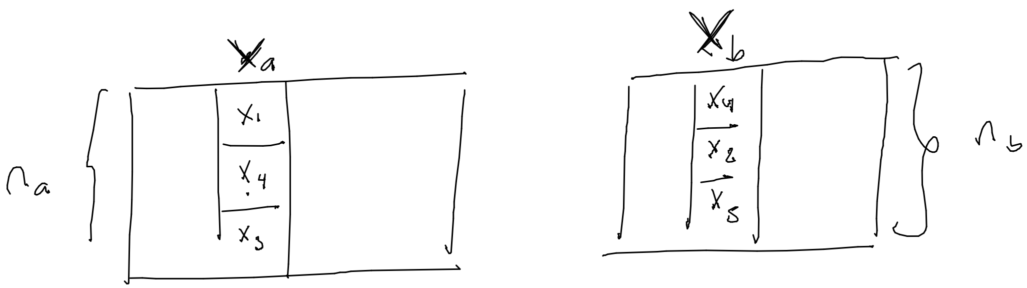
Testing if entropy function can be calculated in a "reduce fashion"

→ R: It can't

$$\binom{nFeats}{2}$$

$\uparrow$

contingencyTables



$$H(X) = -1/n \sum_{x \in X} |x| \log\left(\frac{|x|}{n}\right) = -\frac{1}{n} \sum_{x \in X} (|x_a| + |x_b|) \cdot \log\left(\frac{|x_a| + |x_b|}{n}\right)$$

$$|x| = |x_a| + |x_b|$$

$$= -\frac{1}{n} \sum |x_a| \log\left(\frac{|x_a| + |x_b|}{n}\right) - \frac{1}{n} \sum |x_b| \log\left(\frac{|x_a| + |x_b|}{n}\right)$$

And since this is different to $= -\frac{1}{n} \sum |x_a| \log(|x_a|/n) - \frac{1}{n} \sum |x_b| \log(|x_b|/n)$

Obtaining a contingency Table via an aggregate action.

ContingencyTablesMatrix : Map[(Int, Int), Map[(Int, Int), Int]]
(ctm)

featX feat Y

Cont. Table

values of X and Y

count

fixed size

it is unknown which combinations will emerge from data

Should have a default value of Ø

Use the map function to merge the partial results.

```
def contingencyTablesAccumulator(ctm, lp: LabeledPoint) = {
    (0 to nFeats).combinations(2).map { feats =>
```

this will always generate pairs (i, j) where i < j

```
        ctm(feats(i), feats(j))(lp.features(i), lp.features(j)) += 1
    }
}

def contingencyTablesMerger(ctm1, ctm2) = {
    var ctm = ctm1.map { case (featsIndexes, valuesMap) =>
        valuesMap.map { case (featsValues, count) =>

            ctm2(featsIndexes)(featsValues) + count

        }
```

This should be Ø by default

```
    ctm.featValuesCounts = ctm1.map(k => ctm1(k) + ctm2(k))
    ctm
}

def initContingencyTablesMatrix() = {
```

from Vector to Tuple

```
    val c = (Ø to nFeats).combinations(2).map { case Vector(i, j) => (i, j) }
    val maps = Vector.fill(c.size)((new Map[(Int, Int), Int]).withDefaultVal(0))
    c.zip(maps).toMap.withDefaul
}
```

featIndex featValue count

```
val featsValuesCounts: Map[(Int, Int), Int]
    = new Map[   ].withDefaultValue(Ø)
```