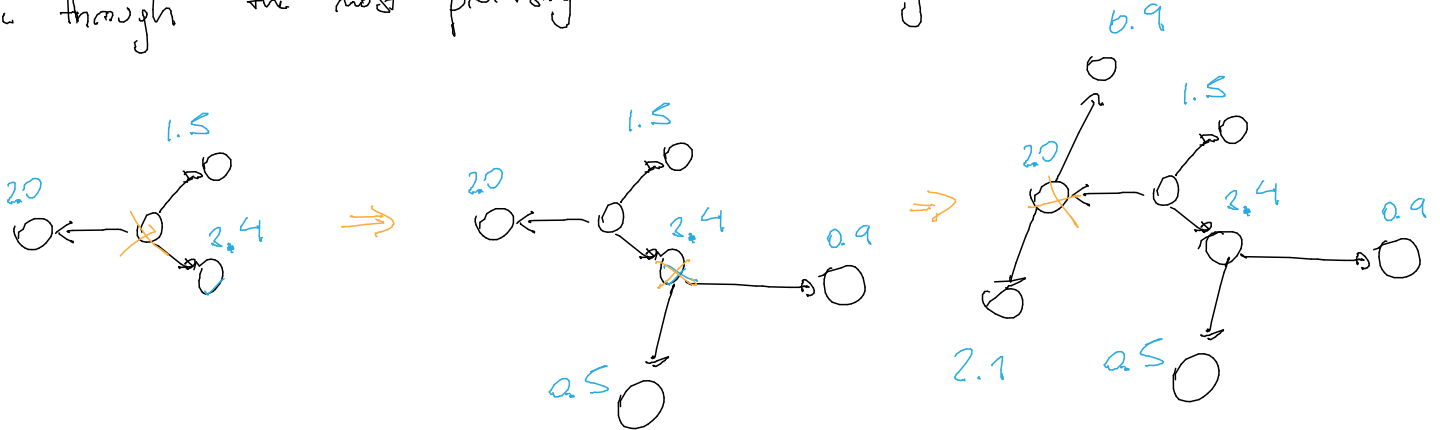


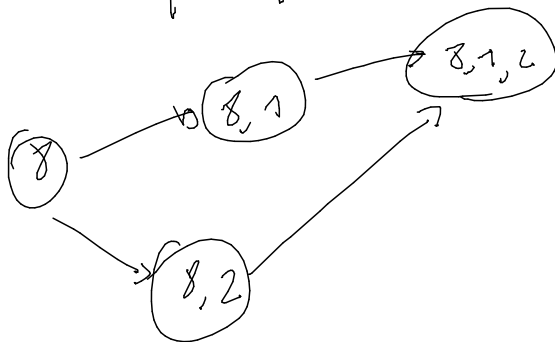
Best First Search

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 Priority Queue, as is done in the WCA 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 expandSubst (subst: Subst, bestSubst: Subst, bestChild: Int, bestChildMerit: Double,
    numOfFails: Int) : Subst {
    if (bestChild == -1) bestSubst // no more feats to search
    else {
        if bestChildMerit > bestSubst.merit
            expand with {bestChild} + subst = bestSubst, numOfFails = 0
        else if numOfFails < maxNumOfFails
            if bestChild.merit > subst.merit
                expand with {bestChild} + subst, bestSubst keeps same, numOfFails + 1
            else
                expandSubst with subst - {head} and in bestChild don't consider {head} or
                bestSubst keep the same, numOfFails + 1 better
        else
            bestSubst
    }
}
```

Tree-based Search

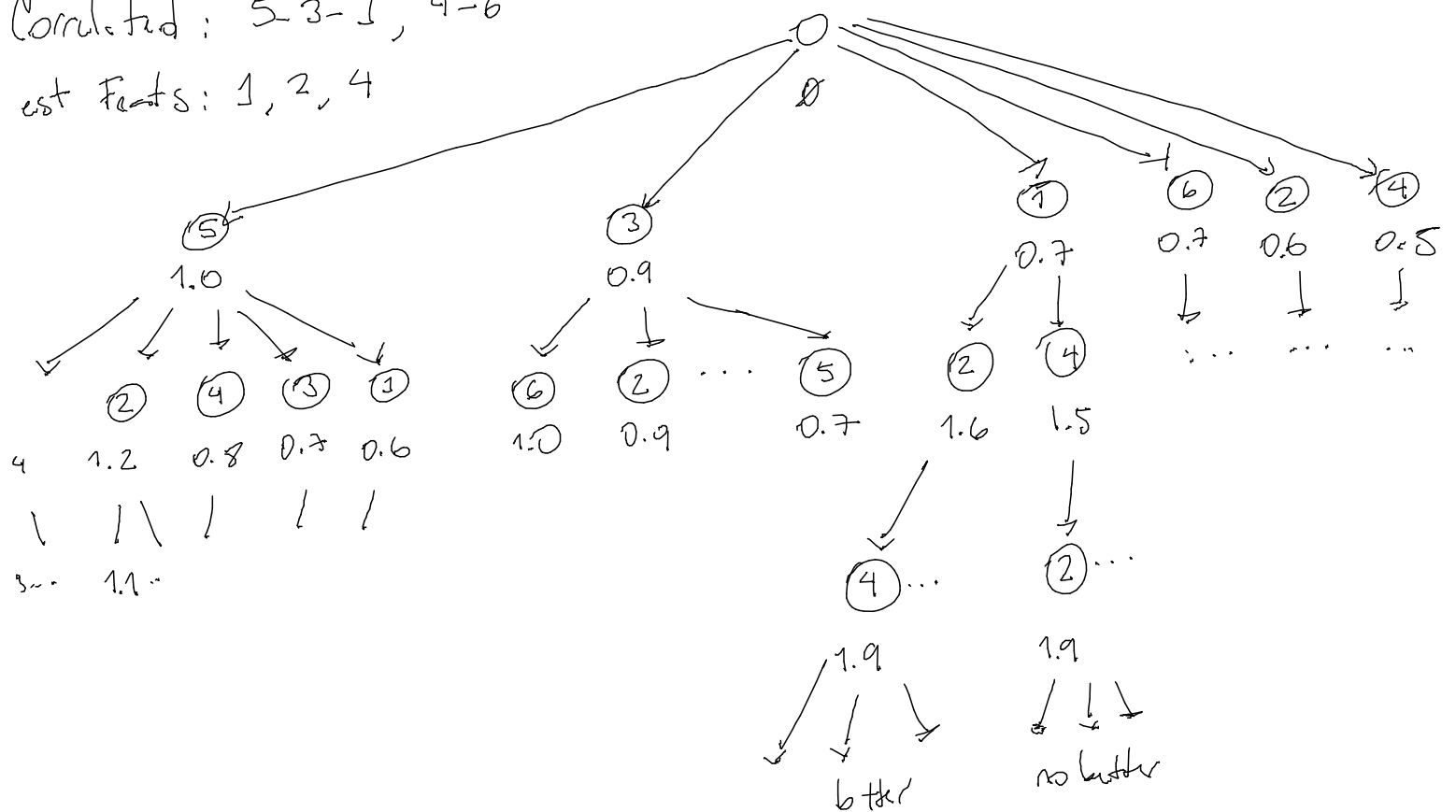
subst	bestSubst	bestChild	fails
[], -∞	[], -∞	5, 1.0	0
[5], 1.0	[5], 1.0	6, 1.4	0
[5,6], 1.4	[5,6], 1.4	3, 1.3	0
[5], 1.0	[5,6], 1.4	2, 1.2	1
[5,2], 1.2	[5,6], 1.4	3, 0.8	2
[8], 1.0	[5,6], 1.4	4, 0.8	3
[], -∞	"	3, 0.9	4
[3], 0.9	"	6, 1.0	5
[3,6], 1.0	"	1, 0.9	6
[3], 0.9	"	2, 0.9	7
[], -∞	"	1, 0.7	8
[1], 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 UCKA Code Analysis

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

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

n is the total number of instances

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

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

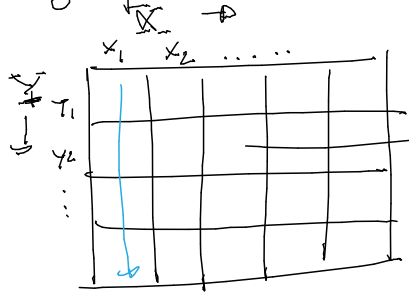
$$= - \frac{1}{n} \left[\sum_{x \in \mathcal{X}} |x| \log_2(|x|) - n \cdot \log_2 n \right] = - \frac{1}{n} \sum_{x \in \mathcal{X}} (|x| \log_2(|x|)) + \log_2 n$$

$$\therefore \frac{1}{n} \sum_{x \in \mathcal{X}} |x| \log_2(|x|) = -(H(X) - \log_2 n)$$

Conditional Entropy of X after denoising Y is defined as:

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

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



$$\|x_3\|_1 = \|y_2\|_1$$

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

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

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

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

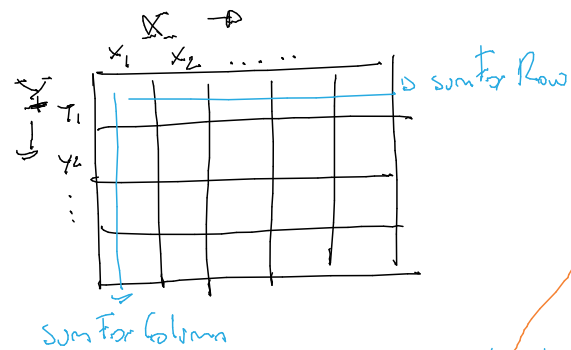
$$= - \frac{1}{n} \sum_{y \in \mathcal{Y}} \sum_{x \in \mathcal{X}} |(x|y)| \cdot \log_2(|(x|y)|) + \frac{1}{n} \sum_{y \in \mathcal{Y}} |y| \cdot \log_2 |y|$$

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

```
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));
}
```



$$\text{columnEntropy} = \sum_{x \in \mathcal{X}} \text{sumForCol} \propto \log_2(\text{sumForCol})$$

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

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

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

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

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

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

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

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

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

$$= \frac{2 (\text{columnEntropy} - \text{entropyConditionRows})}{\text{columnEntropy} + \text{rowEntropy}}$$

Symmetric Uncertainty Scale Implementation

discretizedData

```
def entropy (data: RDD[LabeledPoints], featIndex: int, count: int): Double = {
  data.map(lp => (lp.features(featIndex), 1))
    .reduceByKey(_+_)
```

or a featIndex == lp.features.length to get class entropy.

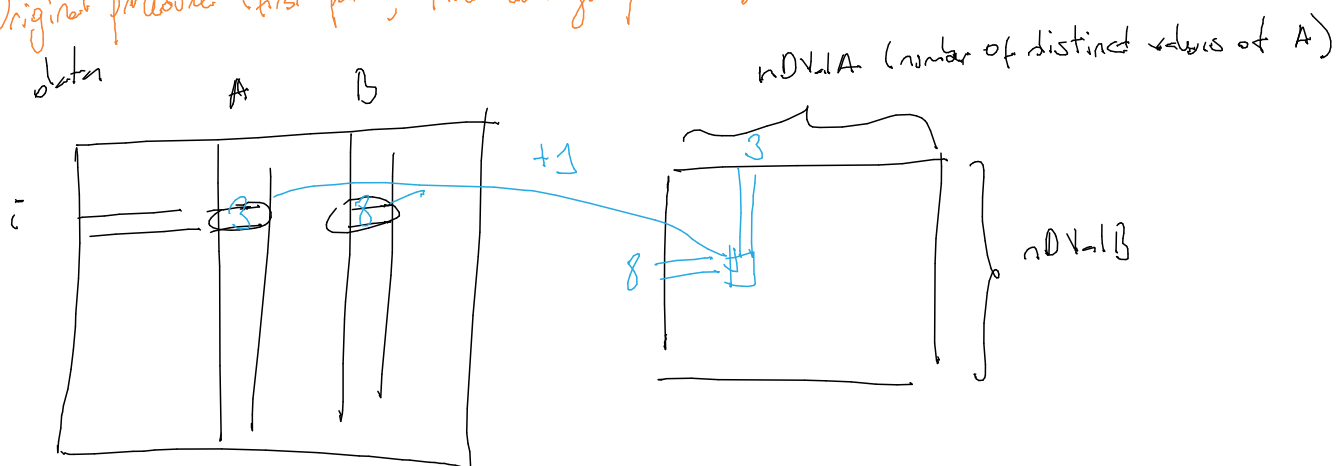
```
    .map((k, v) => v * Math.log(v))
    .sum / (-count) + Math.log(count) => check Eq. (1)
```

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

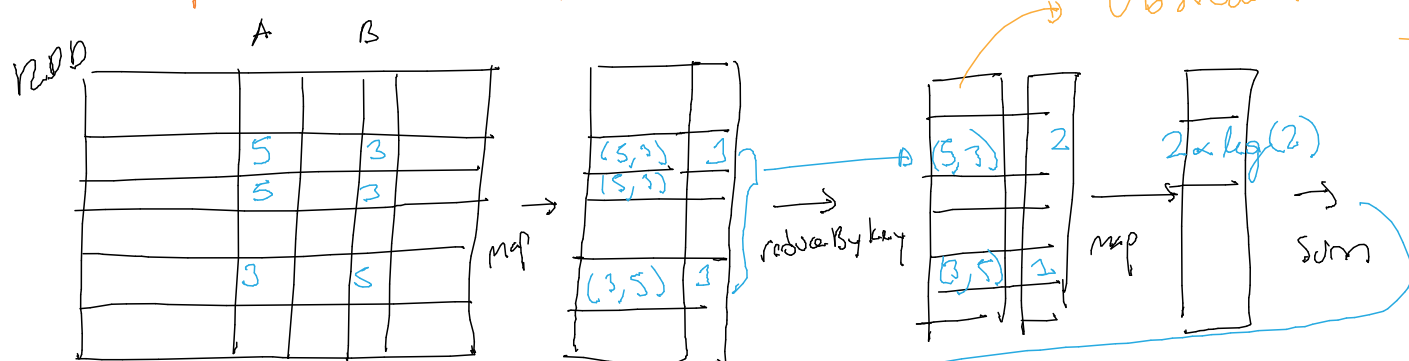
the class is in the last element.

Calc Conditional Entropy

Original procedure (first part, find contingency table):



Functional procedure, find entropy directly



$$\sum_{x \in X} \sum_{y \in Y} |x| \cdot \log_2(|x|)$$

H(X|Y) *H(Y)*

```
def getConditionalEntropy (data: RDD, conditionedFeat: Int, feat: Int, featEntropy: Double, count: Int): Double = {
  data
    .map(lp => (lp.features(conditionedFeat), lp.features(feat), 1))
    .reduceByKey(_+_)
```

Consider the case when one feat is the class!

```
    .map((k, v) => v * Math.log(2))
    .sum * (-1/count) - featEntropy - Math.log(count) (check Eq. (2))
}
```

And finally:

```
def SymmetricUncertainty (data: RDD[LP], feat1: Int, feat2: Int, entropies: Vector[Double]): Double {
```

```
  val InfoGain = entropies(feat1) - getCondEntropy(...)
```

```
  2 * InfoGain / (entropies(feat1) + entropies(feat2))
```

```
}
```

Covariance Matrix

Problem:

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

$$\text{Req_Mem} = (\text{Num_of_feats})^2 \times \underbrace{64 \text{ bits}}_{\text{assuming 64 bits float features.}} \times \frac{1 \text{ byte}}{8 \text{ bits}} \times \frac{1 \text{ Gigabyte}}{(1024)^3 \text{ byte}}$$

$$\text{Req_Mem Gb} = (\text{Num_of_feats})^2 \times \frac{8}{(1024)^3} \text{ Gb}$$

$$\text{If Num_of_feats} = 1 \times 10^5$$

$$\text{Req_Mem Gb} = \frac{8 \times 10^{10}}{(1024)^3} \text{ Gb} \approx \underline{74.505 \text{ Gb}}$$

Solutions Ideas.

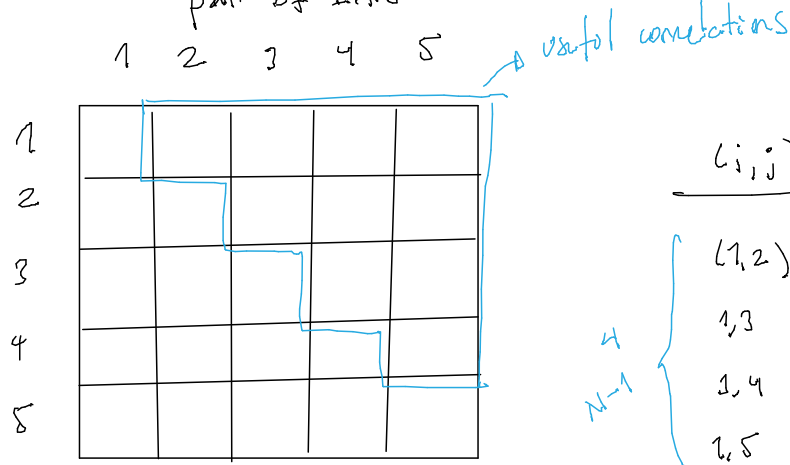
- Use a distributed matrix. (Simple but not a significant contrib.) ✓
- Try with some other thing:
 - Eliminate the need to store the whole matrix.
 - Study what percentage of the matrix is really calculated in the search (should depend on the number of attempts, default: 5).
 - 1/ the percentage is small, an sparse representation should work.
- * Note that the corr. matrix is symmetric, so has then half this space is really needed (the diagonal is not needed)

Distributed Correlation Matrix

We really don't need a matrix, only an storage.

If we update 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 used, 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 dimensionality datasets (num of feats $\sim 10^4$ 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 Partitioner for the keys. Keys will be implemented using an ~~immutable~~ Bitset. A Partitioner assigns an id to a key.

pair of Ints



Mapping to a linear array.

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

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

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

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

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

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

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 $(i, j); i = N-1, j = N$

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

And substituting:

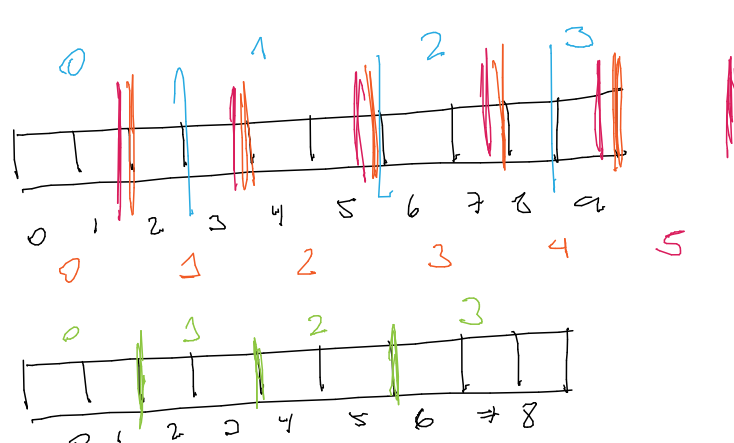
$$\text{length} = N(N-1-1) + N - (N-1) - 1 - \frac{(N-1) \cdot (N-1-1)}{2} + 1$$

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

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

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

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



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

$$\text{last_part_size} < \text{part_size}$$

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

$$\text{last_part_size} = \text{part_size}$$

$$3.) \text{round}\left(\frac{10}{6}\right) = 2$$

$$\text{last_part_size} = 0$$

$$4.) \text{round}\left(\frac{9}{4}\right) = 2$$

$$\text{last_part_size} > \text{part_size}$$

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

p_i : partitionId

p_s : partitionSize

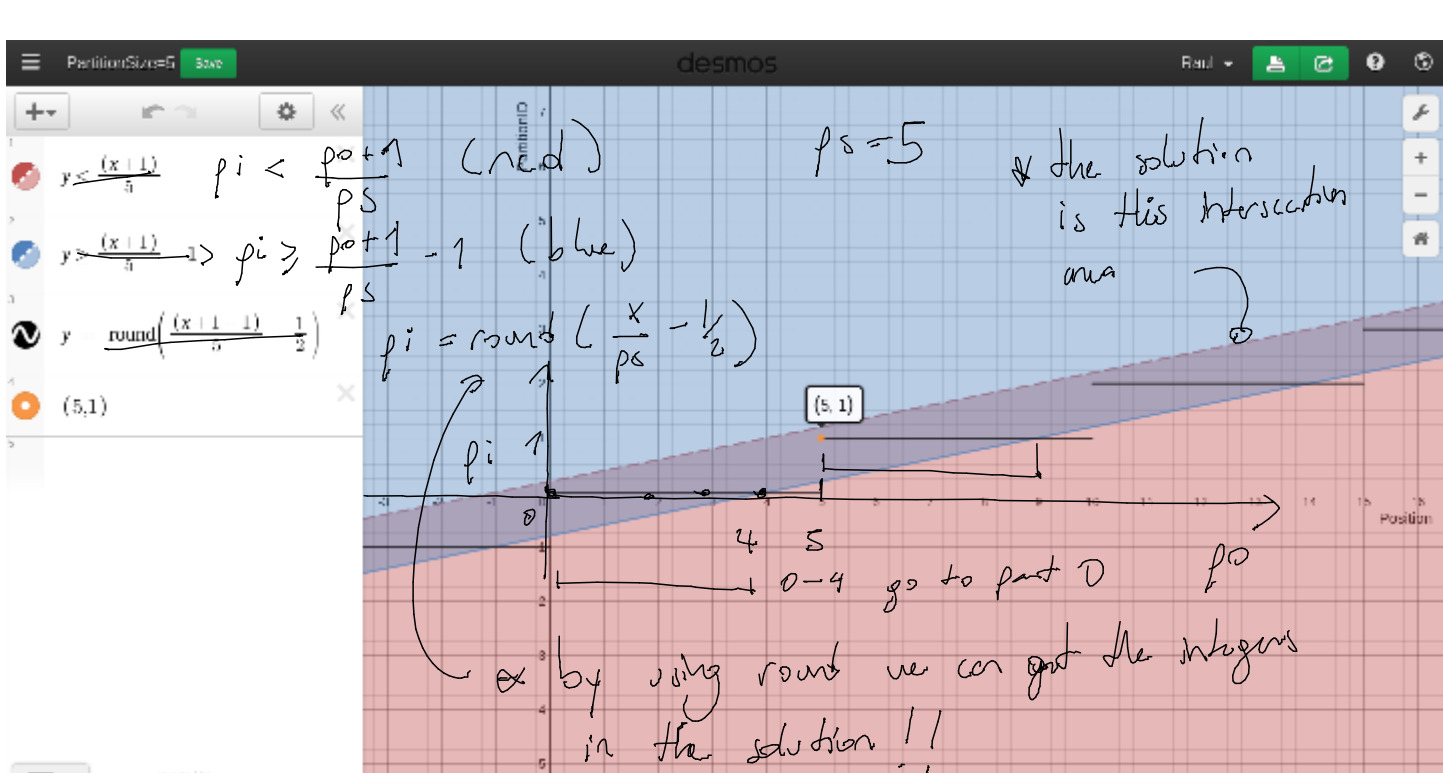
p_o : actual position $[0, \text{length}-1]$

nP : numPartitions $[1, \infty]$

$$p_i \times p_s < p_o + 1 \leq (p_i + 1) \times p_s, \quad 0 \leq p_i \leq nP + 1$$

$$\text{i.e. } p_i < \frac{p_o + 1}{p_s}, \quad p_i \geq \frac{p_o + 1}{p_s} - 1, \quad p_i \geq 0, \quad p_i \leq nP + 1$$

And then, by graphing, found the following solution:



I tested the formula when 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 $\text{maxPartitionId} = \text{numPartitions} - 1$ must be assigned.

Analysis of Heuristic Merit Subset Formula

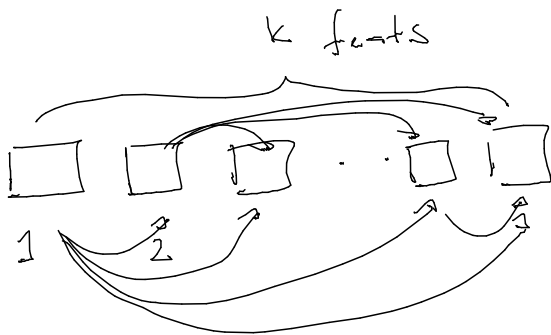
Equation 1 (Ghiselli, 1964) formalizes the heuristic:

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

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.

$$\overline{r_{cf}} = \sum_{i=1}^k r_{ci} / k \quad , \quad r_{ci} \text{ is the correlation of the } i\text{-th feature with the class.}$$

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



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

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

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

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

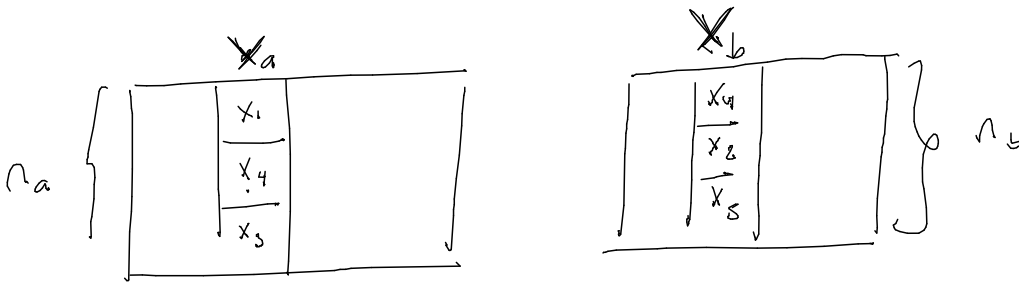
Testing if entropy function can be calculated in a "naive fashion"

↳ It can't

$\binom{n \text{ traits}}{2}$



contingency tables



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

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

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

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

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.

Contingency Tables Matrix: $\text{Map}[(\text{Int}, \text{Int}), \text{Map}[(\text{Int}, \text{Int}), \text{Int}]]$
 (ctm)
 fixed size
 feat X
 feat Y
 Cont. Table
 values of X and Y
 count
 it is unknown
 which combinations will emerge from data
 Should have a default value of 0

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 (featIndex, valuesMap) =>

valuesMap.map { case (featValues, count) =>

ctm2(featIndex)(featValues) + count

} ~~this~~ This should be 0 by default

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

ctm

}

def initContingencyTablesMatrix() = { from Vector to Tuple }

val c = (0 to nFeats).combinations(2).map { case Vector(i, j) => (i, j) }

val maps = Vector.fill(c.size)((new Map[(Int, Int), Int]).withDefaultValue(0))

c.zip(maps).toMap, withDefault

}

featIndex featValue count
 val featValuesCounts: Map[(Int, Int), Int]

= new Map[].withDefaultValue(0)

