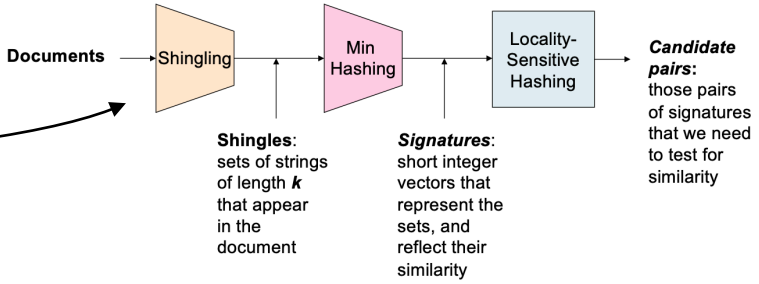
Data Analytics - Lecture 3

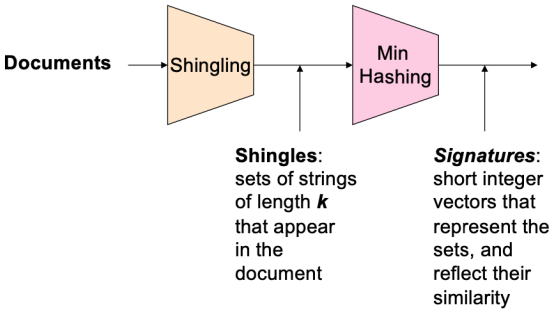
Similarity 2/3

The main advantage of shingles w.r.t. database similarity, is that shingles allow us to compute similarity even if there has been some small editing in the past. A database similarity is not robust to this. This means that even if there are minor edits or changes made to the data, the shingling approach can still identify similarities between documents accurately. With Database Similarity, instead, even minor edits or changes could significantly affect the similarity measure, making it less reliable (or even null) in such scenarios.
Recall that with this approach, we pass from a deterministic approach to a probabilistic one (i.e. we define what could be the most similar documents and we compare them).



Minhashing
(Recall) that:
- small k would mean not that much of a reduction from documents to shingles
- large k , good reduction but we won't be able to catch that much similarity

let's make an example on why working with integer vector is way more efficient:



Suppose we need to find near-duplicate documents among a million documents (=10^6 documents)
Naively, we would have to compute pairwise Jaccard similarities for (approx.) every pair of docs

Assu[illegible]ut 5-sh[illegible]nts
wit[illegible]g., k=5, then [illegible]10^{11}co[illegible]parisons
At 10[illegible]ecs/day and [illegible] comp[illegible]sons/sec, it takes mor[illegible]an a year [illegible] million d[illegible]ments
This is [illegible]ot of calc[illegible]ions; w[illegible] need to reduce this numbe[illegible]
The go[illegible] of *MinHash*[illegible] to es[illegible]nate Jaccard's quickly, *withou[illegible]xplicitly con[illegible]ting [illegible]and union*

Also[illegible]ote[illegible]at ou[illegible]s c[illegible]e the 1m[illegible]on d[illegible]umer[illegible]) is a **moving t[illegible]get:** it ch[illegible]nges c[illegible]er tim[illegible](with increasing d[illegible]creasin[illegible]and n[illegible]dific[illegible]locun[illegible]nts) so [illegible]e want t[illegible]perfo[illegible] this task in the f[illegible]tes[illegible]me poss[illegible]e.

Encoding Sets as Bit Vectors

Many similarity problems can be formalized as *finding subsets that have significant intersection*

- Encode sets using 0/1 (bit, boolean) vectors
  - One dimension per element in the universal set
- Interpret set *intersection* as bitwise-AND, and set *union* as bitwise-OR

Shingles \ Documents				
	1	1	1	0
	1	1	0	1
	0	1	0	1
	0	0	0	1
	1	0	0	1
	1	1	1	0
	1	0	1	0

The matrix is very sparse: a shingle would appear in few documents, in every small fraction of documents.

From Sets to Boolean Matrices

Rows = elements (shingles)
Columns = sets (documents)

- 1 in row ***e*** and column ***s*** if and only if ***e*** is a member of ***s***
- Column similarity is the Jaccard similarity of the corresponding sets

Each document is a column:

- **Example: sim(C_1 ,C_2) = ?**
  - Size of intersection = 3
  - size of union = 6
- Jaccard similarity = 3/6
- Jaccard distance = 1 – (Jaccard similarity) = 3/6

Documents

Shingles	C_1	C_2		
	1	1	1	0
	1	1	0	1
	0	1	0	1
	0	0	0	1
	1	0	0	1
	1	1	1	0
	1	0	1	0

If we find a 1 in the same row for both, that's a AND.
If we find a 1 in at least one of the two column, that's an OR. Jaccard is computed as AND/OR (i.e. intersection/union)

Documents

Shingles	C_1	C_2		
	1	1	1	0
	1	1	0	1
	0	1	0	1
	0	0	0	1
	1	0	0	1
	1	1	1	0
	1	0	1	0

Four Type of Rows

Given columns C1 and C2, rows may be classified as belonging to 3 different cases:

	C1	C2	
a	1	1	**AND = 1 OR = 1**
b	1	0	**AND =0 OR = 1**
c	0	1	**AND = 0 OR = 1**
d	0	0	(Notice: this row does not count!) **Not Counted**

Sim(C1, C2) = 1 / 3

Also, if a = # rows of type a, etc. = # rows of type b
Then: *Sim(C1, C2) = a / (a + b + c)*

Finding Similar Columns

For a computer comparing bits is way easier than working with texts. So, until now, we saw how to transform documents into a sets of shingles and how to represent sets of shingles as boolean vectors in a matrix. Despite this technique makes the computation faster, it is still a huge task given how sparse the matrix is. We nee dtp reduce the size of the matrix to make calculation even faster. To do this, instead of computing similarities of columns, we make the matrix less sparse by calculating **signatures**, i.e. we focus on columns with similar signatures. A signture of a column is somehow a "summary" of that column.

Next step: reduce the size of the job -> reduce the size of the matrix -> find similar columns while computing small signatures

- Similarity of columns == similarity of signatures

Goal: find similar columns using small signatures

Naïve approach:

1. Signatures of columns = small "summaries" of columns
2. Examine pairs of signatures to find similar columns
   - Essential: similarities of signatures and columns are related
3. Optional: check that columns with similar signatures are really similar

Warnings:

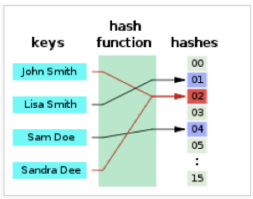
- These methods can produce *false negatives*, and even *false positives*

REMINDERS

Reminder: Hashing

A **hash function** is any function that can be used to map digital data of arbitrary size to digital data of fixed size

- The values returned by a hash function are called **hash values**



- Notice that sometimes hashes collide … is this a problem?

You can reduce a huge set of words (keys), the hash function reduces this words into hashing, which basically are integers. Is it possible that two keys will end up in the same hash. We cannot completely avoid this, but we can make it as less likely as possible.

Reminder: False Positive and False negative

False positive case = pair of documents that are found to be similar, but they are not

False negative case = pair of documents that are found not to be similar, but they are

Keep this is mind!

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

If two **hash** collide they form a false positive. A false negative is when two strings are the exact same but the hash they produce are different (this is extremely rare). (look at the table)False positive: predicted positive but is negative. False Negative: predicted Negative but is Positive.
Understanding false positive and false negative is fundamental when modelling a task. The "weight" of false and true positives really depends by the context of your application/analysis. There are cases in which you cannot afford to have false positives (and thus you accept a higher number of false negatives in order to reduce the number of false positives) and vice versa.

Evaluation Measures

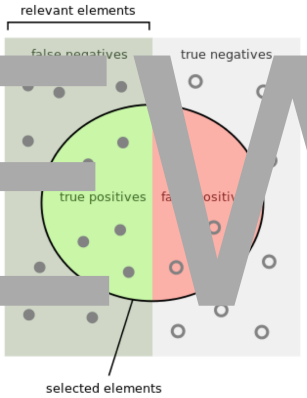
False positive – doc ent fo
o be milar, but it ot

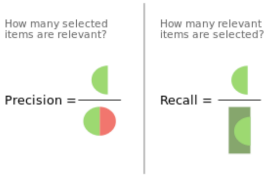
False n - docum
found ot to be sim r, but s

We can derive two evaluation measures from this:

- Precision
- Recall

ns all poss e ca s.
Precision is e ratio of e tru ositi s over the entire ize of t picke set (t e + false ositiv . It tells e ho many selected ms e worth o be picked.
Recall: size of the e positive over t entire set of true positives + false negatives. It tells me how many relevant items I have picked (among all the one I should have picked).





Precision measures the accuracy of positive predictions, meaning it evaluates the proportion of true positive results in all the positive cases identified by the model. High precision indicates that most of the selected items are correct, while low precision means many false positives are included in the results. Recall, on the other hand, evaluates how many of the actual positive cases the model successfully identifies. High recall signifies that most of the relevant items are captured, while low recall suggests that many true positives are missed.

Hashing Columns (Signatures)

In order to reduce the size of the matrix we need to *"hash"* each column C to a small signature h(C), such that:

1. *h(C)* is small enough that the signature fits in memory (RAM)
2. *sim(C_1, C_2)* approx the "similarity" of signatures *h(C_1)* and *h(C_2), so sim(C_1, C_2) = sim(h(C_1) , h(C_2))*

point 2. means that when you hash the data from each column into a smaller form, you want to make sure that this compressed version still reflects the key characteristics of the original data. This way, when you compare the hash signatures of two columns, the result should closely mirror the comparison you'd get if you were comparing the full, original columns. The goal is to reduce data size without losing the ability to accurately assess the relationship between different pieces of data

LSH Involves Some Tradeoff

To control the performance of LSH we need to pick:

- The number of Min-Hashes (rows of *M*)
- The number of bands *b*, and
- The number of rows *r* per band

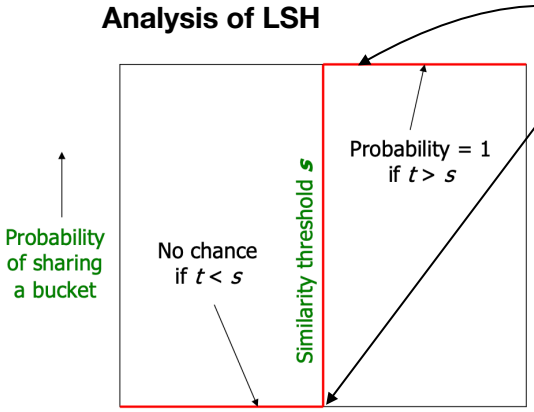
to balance false positives/negatives

Remember b * r = M

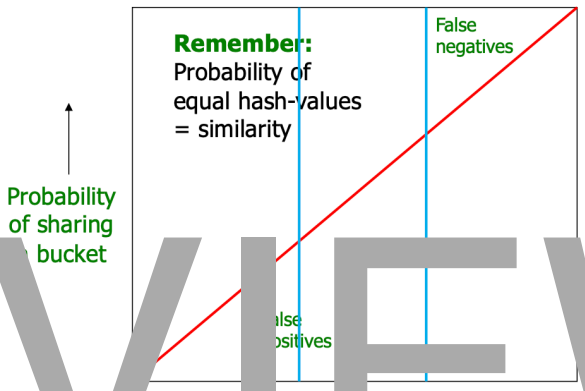
- There is a relationship between these values

Example: If we had only 15 bands, instead of 20, of 5 rows each, the number of false positives would go down, but the number of false negatives would go up (so less recall!)

Analysis of LSH



So, above threshold is document we should look at, below document we should avoid. This in the image is our ideal function/ideal case.



Calculation with *b* bands, *r* rows/band

Columns C_1 and C_2 have similarity *s*

Pick any band (with *r* rows)

- Prob. that all rows in band are equal = s^r
- Prob. that some rows in band are unequal = 1 - s^r

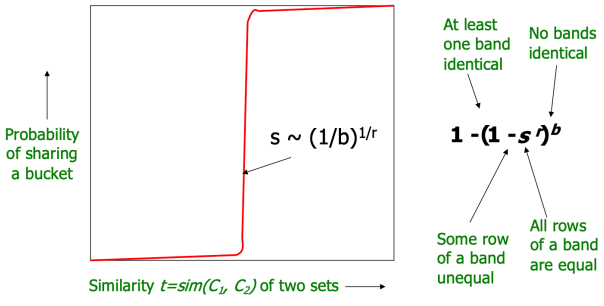
Prob. no band is identical = $(1 - s^r)^b$

Prob. at least one band is identical = $1 - (1 - s^r)^b$

So, we need to choose *s*, *r* and *b* wisely!

What *b* Bands of *r* Rows Gives You

It can be proved that the threshold s can be approximated by $(1/b)^{1/r}$ *



* see proof in the book!

Sampling from Data Streams: Sampling a fixed-size sample

Maintaining a fixed-size sample

We want to optimise/make sure that we use all the resource that are available to properly monitor samples. We may have a particular amount of memory and we want to use that all to sample, to keep the storage of the sam[ples that I am going to monitor. Of course that depends very much on the sample size. Why is this difficult? Mainly because of the lack of stability of the streams, i.e.e we do not know how much information is coming at a particular point of that string. It could be stable or it could be that at one particular point the stream become ver very big. Is a aismilar problem when you try to monitor and put barriers on a steam of water. With a stream of water you more or less know that the water is going to be of a particular amount and you put barriers on the side of the water channel. But if the stream become very unstable (and at some point can be a lot more or less w.r.t. the barrier you put), the stream of water could overflow.

Problem 2: Fixed-size sample

Suppose we need to maintain a random sample *S* of size exactly *s* tuples

- E.g., main memory size constraint

Why is this difficult?

- We do not know the length of the stream in advance

Suppose at time *n* we have seen *n* items

- Each item is in the sample *S* with equal prob. *s/n*

How to think about the problem: say s = 2
Stream: a x c y z k c d e g…

At n= 5, each of the first 5 tuples should be included in the sample S with equal prob.
At n= 7, each of the first 7 tuples should be included in the sample S with equal prob.

- Impractical solution would be to store all the *n* tuples seen so far and out of them pick *s* at random; why?

Suppose that at time n we have seen n items. If we want to sample s, out of that stream, we can sample with a. particular probability s/n. For instance, one method would involve assigning a unique random number to each of the **n** items as they appear in the stream. Then, we could select **s** of these random numbers and choose the corresponding items from the stream to form our sample. If the stream changes at a particular time the probability will also change: the number n, which is the size of the information passing trough my streams, will be higher. So elements will have a lower chance to be picked and kept in the storage.

How can we thus perform a correct fixed-size sampling? We can use the reservoir sampling algorithm

Solution: Fixed Size Sample

Basically:

- initially, you fill your “box” (i.e. our memory) with all the n elements that are arriving in the stream, until your box is full, is filled with the initial elements who came. We assume that a sample is infinite (we don’t know the beginning neither the end, but still we start somewhere, we have a starting point)
- once I fill the box, the next element/item that comes, I decide if stirring it or not. First of all I generate a random number which tells me if I should store it (i pick that element if the assigned random number is within a pre-detrimined threshold). Now I have to free a space to store this new item. To do so, I generate another random number to decide which one I throw away.

Reservoir Sampling algorithm

- Store all the first *s* elements of the stream to *S*
- Suppose we have seen *n-1* elements, and now the n^{th} element arrives ($n > s$)
  - With probability *s/n*, keep the n^{th} element, else discard it
  - If we picked the n^{th} element, then it replaces one of the *s* elements in the sample *S*, picked uniformly at random

Claim: This algorithm maintains a sample *S* with the desired property, i.e., after *n* elements, the sample contains each element seen so far with probability *s/n*

- Proof is by induction, see book

Proof skipped (i.e. slides 24-25)

Filtering Data Streams

How do we analyse the tuple which comes in the stream? Given a list of keys S determine which tuples

The problem:

- Each element of data stream is a tuple
- Given a list of keys S determine which tuples of stream are in *S*

Obvious solution: Hash table

- But suppose we do not have enough memory to store all of *S* in a hash table
  - E.g., we might be processing millions of filters on the same stream
- How do we do it?

of stream are in S (i.e. those that we need to keep). We want to pick out specific elements. We need to be sure that we pick out elements which are noise/useless.

Applications

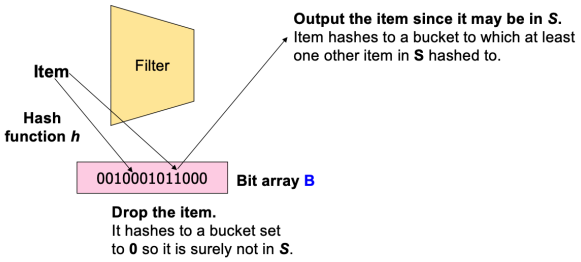
Example: Email spam filtering

- We need to remove spam coming to some email address
- The IR approach requires looking at the content of the email (but this is quite complex for obvious reasons)
- Different approach: we know 1 billion "good" email addresses; if an email comes from one of these, it is NOT spam

Example: Publish-Subscribe systems

- You are collecting lots of messages (news articles)
- People express interest in certain sets of keywords
- Determine whether each message matches user's interest

To filter, we can follow two approaches:
-analyse the content of the tuple/item in the stream (Recommender System)
-I do not look at the content: e.g. for email spam, instead of checking the content of the mail, you can check the address, i.e. the origin of that item.
One important approach is the so-called **Publish-Subscribe System** —> suppose I want to filter emails. My filtering systems, using an hash function, generates an hash function and match it into/hashes a bit-array. Thus, in this bit array I can encode all the email addresses of everybody who is not spam (suppose we have 8 million "good" addresses, that is only 1 GigaByte bit-array). We are surely going to create false positives: sometimes a spam will pass trough, but we have no false-negatives. We want to make sure that false negatives are 0 and false positive are as small as possible.



First Cut Solution

Given a set of keys *S*, we want to:

1. Create a bit array *B* of *n* bits, initially all *0*s
2. Choose a hash function *h* with range [*0,n*)
3. Hash each member of *s* in *S* to one of *n* buckets, and set that bit to 1, i.e., *B[h(s)]=1*
4. Hash each element *a* of the stream and output only those th… was se…
   - O… t *a* if B[h(a)] ==

It creates false positives, but no false negatives
the it… is in *S* … surely output it, if n… e may still … ut it

Exam…

|S| = 1 l… on email addresses
Space a…ilable: |B|= …B = 1…lion bytes = 8 billion b…

If the er… l address is …*S*, th… …es to a buc… that has been set to 1, so it always gets through (*no false negatives*)

Approximately 1/8 of the bits are set to 1, so about 1/8th of the addresses not in *S* get through to the output (*false positives*)

- Actually, less than 1/8th, because more than one address might hash to the same bit
- We are still getting rid of 7/8th true negatives (~87.5%)

so 1/8 are false positives, but we get rid of 87% (7/8) of the true negatives.

Analysis: Throwing Darts

This is a generalisation of the above problem. Imagine that you have **m** darts and you have a number **n** of targets, which are way more than the darts. The targets are the buckets and the darts are the hash values of items. What is the probability that a target gets at least one dart? We can calculate this as follows:

Analysis: we need a more accurate estimate of the number of false positives

Consider: if we throw *m* darts into *n* equally likely targets (with n>>m), what is the probability that a target gets at least one dart?

In our case:

- Targets = bits/buckets
- Darts = hash values of items

The probability that one target will get a dart is 1/n, the probability that some target are not hit by that is 1-1/n.
We have basically a function that we can put into a graph. The fraction of 1s in the array B, is the probability of false positives. This can be generalised into hat is called a bloom filter (we'll see).

Approaches to picking Seed Set
We need to pick a good seed set of k pages. How do we do that? We can rely on two approaches:

1. Use PageRank:
   - Pick the top *k* pages by PageRank
   - Theory is that you cannot get a bad page's rank really high
2. Use trusted domains whose membership is controlled,
   - Examples: .edu, .mil, .gov, .eu, …

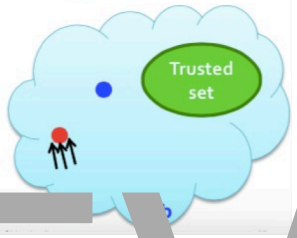
Spam Mass
In the TrustRank model, we start with good pages and propagate trust. **NW.** When a teleport is made to a trust set, then the random walker is going to navigate within the trust set, i.e. following links within trusted/valid pages.
What fraction of a page's PageRank comes from spam pages? In practice, we do not know all the spam pages, so we need to estimate. To estimate it, we can calculate the so called **spam mass**.

Spam Mass Estimation

- r_p = PageRank of page p
- r_p^+ = PageRank of p with teleport into **trusted** pages only
- **Then:** What fraction of a page's PageRank comes from **spam** pages?

$$r_p^- = r_p - r_p^+$$

- **Spam mass of p** = $\frac{r_p^-}{r_p}$



The higher the spam mass of a page the less trustworthy it is.

T st P pagation
ll "tr ed pag the su pages at ar dentifie as g a ve low Spam ss). We set a reshold so that ba call f we no alize e maximum pag Ran to be 1, 1 is a thres ld th we can use to labe p ge as sp : if th value is close to w are almost re that the age spam. NW. Now pag nk can be een a the sum of trusted) and no rusted (r_p^-) pager
There are only two problems with this kind of solution:

1. **Trust Attenuation:** as we propagate the page rank around the web, the trust gets lower and lower. This because is very likely that we navigate trough pages that have much less trust. In short, *the degree of trust conferred by a trusted page decreases with the distance in the graph.*
2. **Trust Splitting:** The larger the number of out-links from a page, the less scrutiny the page author gives each out-link. If we have a large number of out links (which is the case of many hub - pages) than the trust get splitter among all these links and tests lost.

Solutions have been proposed, but details are unknown (they are commercial secrets, and we can only make hypothesis)

To clarify:

The correlation is as follows: r_p^+ quantifies the trust derived from the trusted set for a specific page p. A high value of r_p^+ indicates that page p is frequently reached from the trusted set and is therefore considered trustworthy itself. If a trusted page links to page p, then a portion of that trusted page's score is passed on to page p, weighted by the number of outbound links the trusted page has. The more direct links from trusted pages to page p, the higher its trust score r_p^+ becomes. Over time, as this random walk continues, the trust values reach a steady state where they don't change much from one iteration to the next. This steady state is the trust distribution we are seeking. In this distribution, pages that have strong connections to the trusted set — either directly or through a short chain of links — will have higher trust scores, indicating that they are likely to be reputable. Then, now that we have a good measure of "trustness" for a page (i.e. r_p^+), we can define the PageRank value of a page as being composed of $r_p^+ + r_p^-$. Furthemore, we can define a treshold, computed as $1 - r_p^+$, which tells us how probably is a page to be spam.

Data Analytics - Part 2
Lecture 1 - 11/04/24

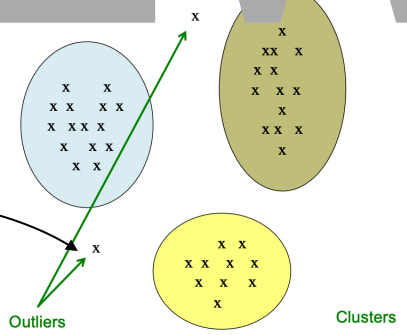
Clustering 1

When dealing with high-dimensional data (i.e. each datapoint is in a high-dimensional space), we want to find ways such that, given a cloud of data points, we want to understand its structure. Recall that clustering is fundamental in non-supervised ML and aims to investigate and find unseen relationship between data in high dimensional spaces (i.e. spaces otherwise intractable by humans). (e.g. image) each one of these points here is a letter.
Clustering wants to group these elements in groups that are made in such a way that all the elements within each group are much more similar to each other and are different from the one in other groups. We want to maximise **intra-cluster similarity** and minimise **inter-cluster similarity**.
Instead of treating all of these elements in the group as individuals, we can replace them with an element/something that represents all the things that thy have in common. In doing this we have to be very careful: we have to find a proper grouping, a proper clustering. (e.g. image) If this was a 2-Dimensional space, we can in some way define three clusters among the datapoints, but if this was a 2-D projection of a 3-Dimensional feature space, it becomes extremely complicated to visually group elements and assess their similarity —> despite it looks like so (i.e. it looks easy to cluster visually) when projected in 2-D, in reality, if we come form a N-Dimensional space (where N>2), this is not true anymore. We need to get the proper tool to identify how these elements are actually built.

The problem of Clustering

Given a **set of points**, with a **notion of distance** between points, **group the points** into some
- Members of a cluster are close/similar to each other (intra-cluster sim.)
- Members of different clusters are dissimilar (inter-cluster sim
usually
- Points are in a high-dimensional space
- Similarity is defined using a distance measure (e.g. Euclidean, Cosine, Jaccard, edit distance, …).

Example- Clusters and Outliers



What do we do with the two elements left out from the three clusters? We can include each outlier in the closest cluster, and we impose assignment when the distance is the same (e.g. we impose this to go to the yellow)
But, by doing so, the created clusters become less cohesive. The intra cluster similarity decreases, or, in other words, it increases the average inter-distance between clusters. The best thing to do, to avoid such, is to recognise these elements as **outliers/ anomalies.**
E.g. (professor's example) If you have a computer vision system which detects the number of wheels of vehicles, given that it usually detects 2-3-4-6 wheeled vehicles, if your system detects some 7-wheeled vehicles, you treat this reading as a system fault/anomaly/ remove this outlier from the dataset.
The **outliers** are vary rare and create anomalies when analysing/ clustering the dataset.
Even when the outliers are removed, clustering is a hard problem. Some clusters are very very tight/cohesive (e.g. the black), others are much more sparse instead (e.g. the blue, if you look closely some blue points are very far from the cluster center)

Why clustering is hard?
Clustering in two dimensions looks easy, also clustering small amounts of data looks easy and in most cases, looks are not deceiving, but:

- Many applications involve not 2 or 3, but 10 or 10,000 dimensions. For example, in a dataset of customer information, dimensions could include age, income, spending habits, etc. While visualizing two or three dimensions (like plotting data on a 2D or 3D graph) is relatively straightforward, many real-world applications involve datasets with far more dimensions. When you have a dataset with a large number of dimensions (sometimes referred to as "high-dimensional" data), traditional visualization and even some computational techniques become impractical or less effective. This is because the increase in dimensions can lead to complexities in understanding how data points are distributed across these dimensions.
- High-dimensional spaces look very different: almost all pairs of points are at about the same distance. This point refers to a phenomenon known as the "curse of dimensionality": as the number of dimensions increases, the geometric intuition we have from three-dimensional or lower-dimensional spaces does not hold up. Specifically, in high-dimensional spaces, the concept of distance between points becomes less intuitive. In these spaces, the distance between almost any pair of points tends to be similar. This means distinguishing between "close" and "far" points becomes harder.

NW. Always recall that clustering is very different from classification: clustering is, given a bunch of elements, applying an algorithm which groups the data in clusters/classes which I had no idea before (as stated above, clustering is a non-supervised ML task, it aims to discover unseen data patterns). Classification instead is the opposite, you define some classes, and you want an algorithm which groups data in these classes.

Clustering: taking the right perspective

In some cases, finding a measure of similarity / distance is not easy; e.g., music!

Intuitively: music divides into categories, and customers prefer a few categories

- But what are these categories?
- Most likely too many and too complex to "describe"

Take a *different perspective*: represent a CD by the set of customers who bought it

- Similar CDs have similar sets of customers, and vice-versa
- Easier problem!

How do you cluster music?
Clustering music requires an algorithm to find similarities and differences among musical elements, and this process heavily depends on the dataset's composition. For example, if the dataset includes only classical music, the algorithm will distinguish between various classical pieces. In contrast, a diverse dataset containing genres like rock, punk, jazz, and classical will prompt the algorithm to cluster based on these broader genre categories.

This illustrates how the "perspective" in clustering is crucial and varies with the data provided. The term "perspective" here refers to how an algorithm interprets and processes the data to determine similarities. Taking the right perspective means tailoring the clustering approach to effectively match and leverage the specific characteristics and diversity of the dataset, ensuring that the algorithm produces meaningful and actionable insights.Clustering fundamentally alters our perspective because it operates based on the specific characteristics of the data we provide. 'Changing perspective' means to tackle a clustering problem from another point of view, e.g., instead of clustering CDs looking at its musical content and features, we can directly cluster CDs by grouping similar sets of costumers (and this is an easier and more effective task).

Clustering Problem: Music CDs

Space of all CDs: think of a space with one dimension for each customer (e.g., own or not the CD)

- Values in a dimension will be 0 or 1 only
- A CD is a point in this space (x_1, x_2,…, x_k), where x_i = 1 iff the i^{th} customer bought the CD, 0 otherwise
- The space is multidimensional
- The number of dimensions could be tens of millions!

Task: find clusters of similar CDs

A different perspective and the same problem whatever the objects we try to cluster (apart from small differences)

If we think as every customer as 1-Dimension in my space, I have a multidimensional space. The best approach is to gather a dataset as big as possible, with data features that may seem useless, and let the cluster find grouping that may come from unexpected way of tackling the problems, i.e. let the cluster algorithm find perspective of analysing the data you couldn't think of.

The BFR Algorithm begins by selecting k points, using one of the methods discussed in Section 7.3.2. Then, the points of the data file are read in chunks. These might be chunks from a distributed file system or a conventional file might be partitioned into chunks of the appropriate size. Each chunk must consist of few enough points that they can be processed in main memory. Also stored in main memory are summaries of the k clusters and some other data, so the entire memory is not available to store a chunk. The main-memory data other than the chunk from the input consists of three types of objects:

1. *rd Se* e s ple sumr ries f th er elves. Ve shall ldre the form of uster s mma atic shortly. No that he clust sur aries are not "disca ued"; ney e in fact ess tial lowever he p nts that the sur mar repres ts *a* discarded and av o repres tati in main memor er thar hrou h this summary

2. *The Compressed Set*: These are summaries, similar to the cluster summaries, but for sets of points that have been found close to one another, but not close to any cluster. The points represented by the compressed set are also discarded, in the sense that they do not appear explicitly in main memory. We call the represented sets of points *miniclusters*.

3. *The Retained Set*: Certain points can neither be assigned to a cluster nor are they sufficiently close to any other points that we can represent them by a compressed set. These points are held in main memory exactly as they appear in the input file.

The picture in Fig. 7.11 suggests how the points processed so far are represented.

The discard and compressed sets are represented by $2d + 1$ values, if the data is d-dimensional. These numbers are:

(a) The number of points represented, N.

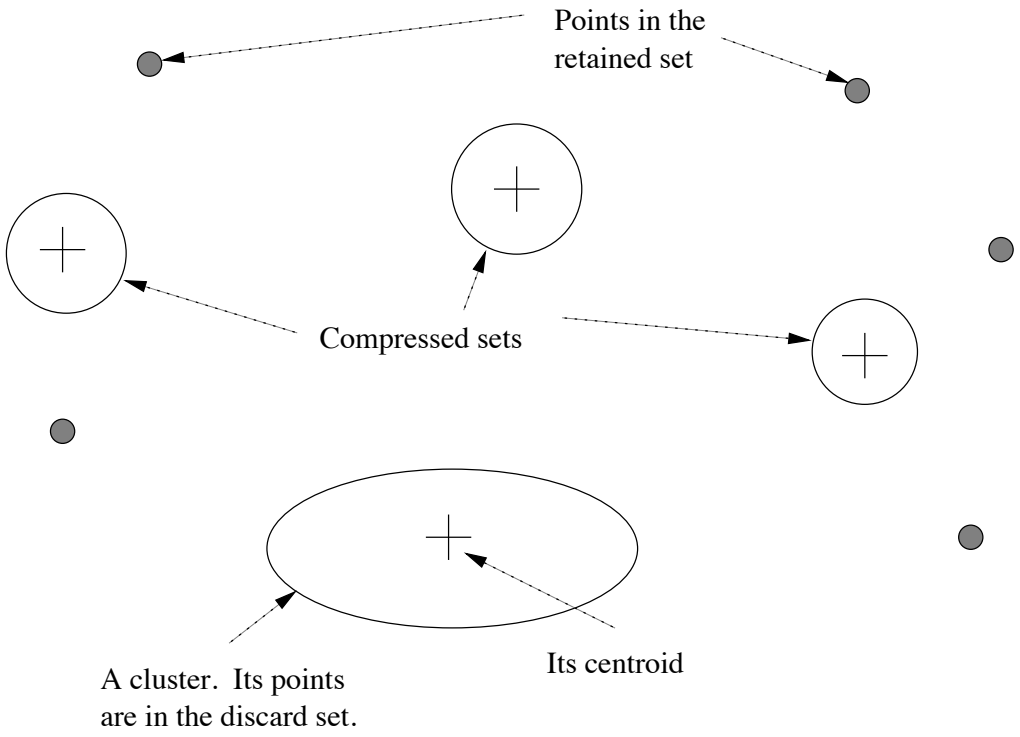


Figure 7.11: Points in the discard, compressed, and retained sets

b) The sum of the components of all the points in each cluster. This data is a vector SUM of length d, and the ith component is the ith dimension is SUM_i.

c) The sum of the squares of the components of all the points in each dimension. This data is a vector SUMSQ of length d, and its component for the ith dimension is SUMSQ_i.

Our real goal is to represent a set of points by their count, their centroid and the standard deviation in each dimension. However, these $2d+1$ values give us those statistics. N is the count. The centroid's coordinate in the ith dimension is the SUM_i/N, that is the sum in that dimension divided by the number of points. The variance in the ith dimension is $\text{SUMSQ}_i/N - (\text{SUM}_i/N)^2$. We can compute the standard deviation in each dimension, since it is the square root of the variance.

Example 7.9: Suppose a cluster consists of the points $(5,1)$, $(6,-2)$, and $(7,0)$. Then $N = 3$, SUM $= [18,-1]$, and SUMSQ $= [110,5]$. The centroid is SUM$/N$, or $[6,-1/3]$. The variance in the first dimension is $110/3 - (18/3)^2 = 0.667$, so the standard deviation is $\sqrt{0.667} = 0.816$. In the second dimension, the variance is $5/3 - (-1/3)^2 = 1.56$, so the standard deviation is 1.25. □

7.3.5 Processing Data in the BFR Algorithm

We shall now outline what happens when we process a chunk of points.

Latent Factors Intuition

Assume that both movies and users live in some **low-dimensional space** describing their properties

Recommend a movie based on its **proximity** to the user in the latent space

We will not explore it further than this!

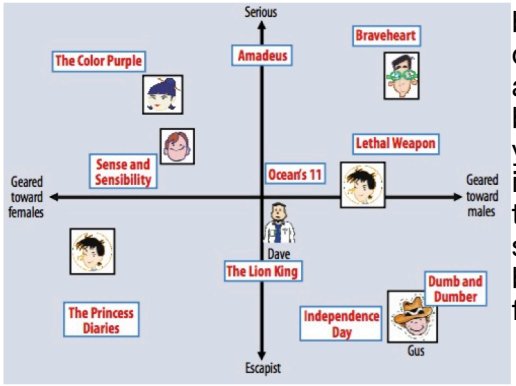


Figure from Koren et al. (2009)

The intuition is that the axis brings me to a lower dimensional space where I am able to infer relationship between items that I have to yet discover. (e.g. image) if items are put in the space like this, I can create/reduce the space to a new axis that could be how serious/escapist is a female as opposed to a male.

Content-based RS

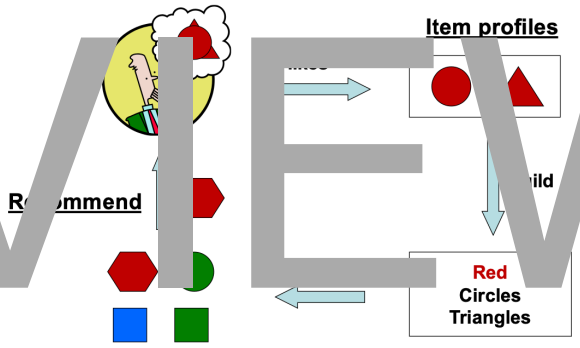
The idea of content based is that you recommend to a user x, something similar to what it bought before, by just looking at previous product informations and looking for similar ones in the proposed products.

Main idea: Recommend items to customer ***x*** similar to previous items rated highly by ***x***

Examples:

- Movie recommendations
  - Recommend movies with same actor(s), director, genre, …
- Websites, blogs, news
  - Recommend other sites with "similar" content

(e.g. im… …w that …rou… …d tri…gular red s…pes …d we build a pr…le k…owin… what user …es. I… …is case the colo… …is t… most im… …tant f… …ture for this user. …o the use… …e four …ems proposed, I recom… …nd the rec… …ne. T… …s is of course base… on extr… …ting featur… …from …e user's items.



For each item, create an item profile

Profile is a set (vector) of features

- Movies: author, title, actor, director,…
- Text: set of "important" words in document

How to pick important features?

- Usual heuristic from text mining is TF-IDF weight (Term Frequency * Inverse Doc Frequency)
  - Term … Feature
  - Document … Item

f_{ij} = frequency of term (feature) ***i*** in doc (item) ***j***

$$TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}}$$

Note: we normalize TF to discount for "longer" documents

n_i = number of docs that mention term ***i***
N = total number of docs

$$IDF_i = \log \frac{N}{n_i}$$

TF-IDF score: $w_{ij} = TF_{ij} \times IDF_i$

Doc profile = set of words with highest TF-IDF scores, together with their scores

This model can be used also for features that are different from words in documents

- Colors in a drawing, melodies in music, etc.

this part was quickly done by professor, especially the mathematical part was fully skipped

Next Step: Consider the Whole Context
If we want to include the context in our system, we have to take into account all of these contextual factors and the constraints which emerge from them.

Geographical:
- Having lunch in **Milan**

Temporal:
- Going for lunch on Friday at **13:00**

Weather:
- Is it going to be **raining**?

Purpose of trip:
- It is **business event.** Dress accordingly!

Accompanying people:
- Could I take my **kids**? If now, what could they do at that time?

TREC Contextual Suggestion Track: TREC is an evaluation exercise who's being around for quite some time (30 years), where some task are created and are advertised to users. The user perform these tasks and sends the results by a certain date. Then the results are evaluated and the user knows who did best. Whoever did best will be invited to do a presentation. There where different tracks and one was the **Contextual Suggestion Track:** (e.g.) the task consist, given information about a person (where he lives, was born, tastes, work and so on), and assuming she is in Paris, to suggest her what to do in specific days of the year. They evaluated the various algorithm and choose the best one.

Contextual Factors Different Importance
Contextual Factors have different importance of course.

POI Recommendation

The system needs to keep all these information into account and suggest a list of options to the user

The systems needs also to record the choice of the user, gather the user feedback, and learn from this for next recommendation
- AI algorithms have a great influence on this!

Friendship:
- Should I trust my **friend's suggestion**?

Rating:
- Should I risk going to a **low-rated** place just because it is highly rated by friends?

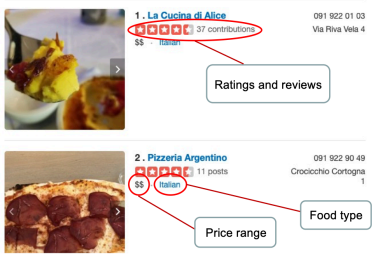
Reviews:
- **Someone said** the restaurant's staff is rude. Should I care?

Taste:
- There is a vegan person with us. Could I go to a **steakhouse**?

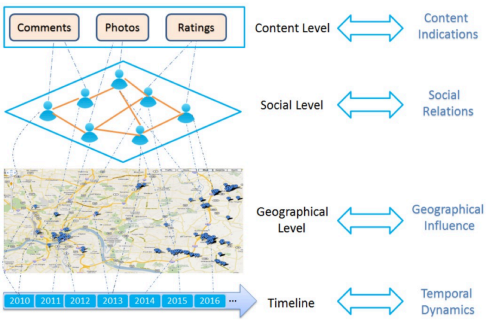
Price:
- The place is said to be cheap; **how much** do I like to spend?

Availab Informa tion
POI RS are particularly us ul/works better ith device hat ve a c sens s. This bec use P I RSs require kn ving the ontext You w nt t sys m to mak se o least be aware all the ailable informa

Availab Informat n: Th Location
(this sli s seems p tty u says son nformation you get from certain locations and, in case of location with similar reviews and cuisine, which do we suggest? So the system should be able to infer a decision also in these cases).



Available Information: the Context
Imagine you have a place and there is no picture/very little information about it, but it has lots of reviews. Are you going to trust it? Of the comments of the users who reviewed it some are friends, other aren't. (e.g.) if a bunch of chinese rates good a chinese restaurant, it thus should be a valid suggestion instead of an italian restaurant reviewed by a bunch of chinese. **Temporal Dynamics:** is a good judgement in 2010 as valuable as one in 2015?



The main idea of these slides is the following: *Point of Interest (POI) recommendation systems are most effective when integrated with devices that have numerous sensors, as they rely heavily on contextual information. This includes data about the user's location, time, social environment, and weather conditions. The system uses this data to*

make informed decisions, distinguishing between locations with similar attributes and providing relevant recommendations. User-generated content, such as reviews and ratings, is crucial. The system must evaluate the reliability of reviewers and consider cultural relevance, such as prioritizing reviews of a Chinese restaurant by Chinese users. Temporal dynamics are also important; recent reviews are more valuable than older ones, reflecting current quality and relevance. Effective POI recommendation systems integrate data from various levels: content (comments, photos, ratings), social relations, geographical information, and temporal changes. Utilizing this comprehensive data set ensures accurate and relevant recommendations. Understanding how to leverage and integrate this information is essential for developing functional and efficient POI RSs.

Back to the User Context:
Different level of context accessibility:

- **Fully observable:** contextual factors relevant to the application, as well as their structure and their values at the time when recommendations are made, if known explicitly. For example, knowing the user's exact location, the current weather, the time of day, and their dining preferences allows for highly tailored restaurant recommendations.
- **Partially observable:** Only some of the information about the contextual factors, as described above, is known explicitly. For instance, knowing the user's location and the time of day but not their current mood or who they are with can still lead to reasonably accurate recommendations.
- **Unobservable:** No information about contextual factors is explicitly available to the recommender system, which makes recommendations utilizing only the latent knowledge of context in an implicit manner. An example would be a system recommending a coffee shop based on the user's historical preferences without current contextual data.
- **Static:** contextual factors and their structure remain the same (stable) over time. For example, a user's home address or long-term preferences that do not change frequently.
- **Dynamic:** when contextual factors change in some way over time. For instance, the user's current location, real-time weather conditions, or recent activities can change, affecting the relevance of recommendations.

Context Awareness
When we put all these contextual factors together we get the following **levels** of **context awareness**:

How Contextual Factors Change	Knowledge of the RS about the Contextual Factors		
	Fully Observable	Partially Observable	Unobservable
Static	Everything Known about Context	Partial and Static Context Knowledge	Latent Knowledge of Context
Dynamic	Context Relevance Is Dynamic	Partial and Dynamic Context Knowledge	Nothing Is Known about Context

The Recommendation Space
This multiplication results in a large, sparse hypermatrix, with each cell representing a unique combination of a user, an item, a context, and a relevance score. In other words, each cell contains the relevance assessment of a specific item for a specific user in a specific context. Most cells in this matrix are empty because not all users have interacted with all items in all possible contexts. The task of the recommendation system is to populate this sparse hyperspace with data in the most promising areas. High values within this matrix indicate strong user preference for a specific item in a particular context. These "most promising" areas are the combinations of user, item, and context where the system predicts a high likelihood of user satisfaction based on available data. When the system identifies high values, either from actual user ratings or estimated preferences, it uses this information to make recommendations.

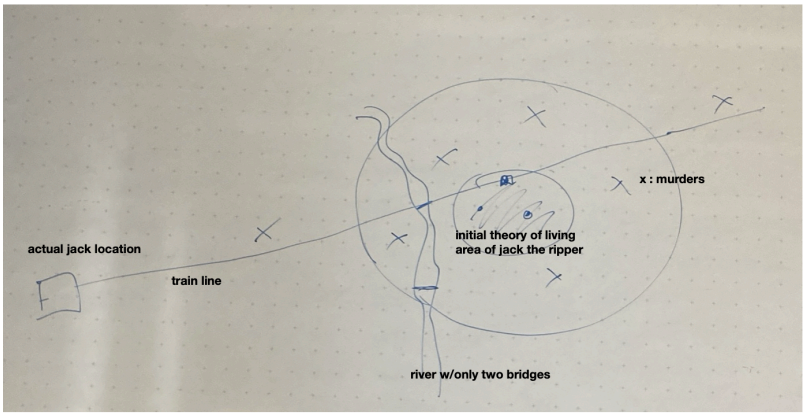
The space representing all the POI rec. combinations is multidimensional:

Data = U x I v C x R

where:

- U = users
- I = items/locations/etc we could recommend
- C = contextual factors
- R = relevance assessments (i.e. ratings provided by users)

Example: Criminal Geographical Profiling



Geographical profiling is a powerful tool used in criminology to analyze the movements and actions of a person based on location data. One historical example is the case of Jack the Ripper in 1888, who terrorized the streets of Whitechapel in London. The map shows the murder locations marked with 'X', the train line, and the river with two bridges. Jack the Ripper's killings were distributed across a particular area, and by analyzing these locations, investigators aimed to determine his base of operations. The assumption was that he lived within the area circumscribed by his crimes. This inference was based on the idea that a killer would avoid committing crimes too close to home to prevent recognition but would also not travel excessively far to ensure a quick escape after the crime. Initially, the theory suggested that Jack the Ripper lived within the central area marked on the map. However, despite investigations, no conclusive suspect was found in that area. Later hypotheses proposed that Jack the Ripper might have lived on the outskirts of London, using the train line to travel to and from Whitechapel to commit his crimes. This theory was supported by the occurrence of a few murders outside the initial area but near the train line. This example demonstrates the importance of geographical profiling in criminal investigations. By mapping crime locations and analyzing their distribution, investigators can form hypotheses about a criminal's base of operations and movement patterns. This process involves examining spatial data, recognizing patterns, and considering additional factors such as transportation routes and temporal sequences of the crimes.
The case of Jack the Ripper illustrates how geographical profiling can guide investigations, even if the exact culprit is not identified. It also highlights the need to adapt profiling techniques to account for new variables, such as transportation options, which may influence a criminal's behavior. Visualization tools play a crucial role in this process by converting abstract data into visual representations, providing insights that support the development and testing of investigative hypotheses.

Why [illegible]?

Use the eye for pattern recognition; people are good at
- [illegible]nning
- [illegible]ognizing
- [illegible]membering images

Graphical elements facilitate comparisons via:
- length
- shape
- orientation
- texture

Animation shows changes across time

Color helps make distinctions

Aesthetics make the process appealing

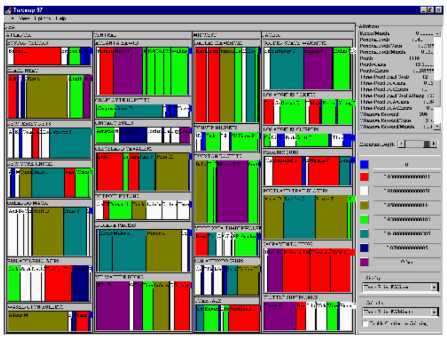
A Key Question

How do we **convert** abstract information into a **visual representation** while **preserving the underlying meaning** and [illegible] **providing new insight**?

Many creative ideas often fail in practice
- The hard part: how to apply ideas judiciously
- Inventors usually do not accurately predict how their invention will be used

Case Study: TreeMap

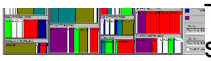
The TreeMap Idea
- Represent a large set of objects that are related to one another (e.g., one is part of another)
- Show a hierarchy as a 2D layout
- Fill up the space with rectangles representing objects
- Size on screen indicates relative size of underlying objects



A TreeMap is a visualization technique that represents a large set of hierarchical data using nested rectangles. This approach was introduced by Johnson and Shneiderman in 1991. The key idea is to show a hierarchy in a two-dimensional layout, where each rectangle represents an object. The size of the rectangle corresponds to a particular quantitative attribute, such as the size of the underlying object. By filling the space with these

rectangles, TreeMaps can effectively convey the relative sizes and relationships among the objects within the hierarchy. Early applications of TreeMaps, such as those applied to file systems, revealed some limitations. The TreeMap shown in the slide is difficult to understand and navigate. The visual representation is cluttered, and it is hard to interpret the adjacency of rectangles. Moreover, the aspect ratios of the rectangles are uncontrolled, leading to many skinny boxes that further contribute to the visual clutter. The use of color is not meaningful and does not convey additional useful information. This initial application of TreeMaps highlights the importance of thoughtful design in data visualization to avoid confusion and misinterpretation.

Problem of Tree Maps



1. Too disorderly
   - What does adjacency mean?
   - Aspect ratios uncontrolled leads to lots of skinny boxes that clutter
2. Color not used appropriately
   - In fact, is meaningless here
3. Wrong application
   - Do not need all of this to just see the largest files in the OS

The early TreeMap implementation had several problems. Firstly, it was too disorderly; the adjacency of rectangles did not convey any meaningful relationships. Secondly, the aspect ratios of the rectangles were uncontrolled, resulting in many awkwardly shaped boxes that cluttered the visualization. Thirdly, the use of color was inappropriate; it was either meaningless or not used to its full potential to differentiate between important aspects of the data. Lastly, the application itself was misguided; using TreeMaps to visualize file system content did not provide the necessary clarity or insights that one might expect from a well-designed visualization tool.

Successful Applications of TreeMaps

Think more about the use you are going to do with it
- Break into meaningful groups
- Fix these into a useful aspect ratio

Use visual properties properly
- Us olor to dis uish m aningfully
  - e only two col (i.e., j distinguish one thing n another)
  - act numbers a not ve important

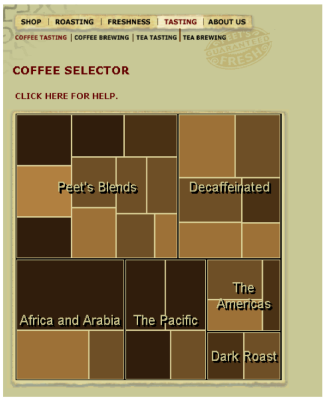
Provid nteractiv
- Ac s to the rea ata
- Ma s it into a us l too

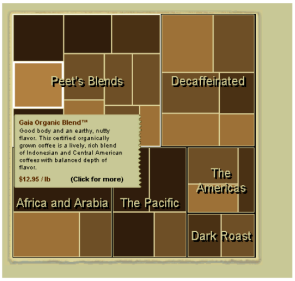
For TreeMaps to be effective, their application must be carefully considered. It is crucial to think about how the TreeMap will be used and to break the data into meaningful groups. Fixing aspect r os i o a u lp in tter ualiz the data. Usi visual roper s like olor meaningful can greatly nhan the interpretabi of t TreeMa For stance, using col to disting h be een different cate or s or age ra es ca make the visualiza more i rmat interac ty, such as allowing users to click on rectangles to access more detailed data, turns the TreeMap into a more useful tool. Proper application and thoughtful design ensure that TreeMaps can successfully represent complex hierarchical data, making it easier to understand and navigate.

Example:

TreeMaps are versatile visualization tools that can be applied to various contexts. A notable example is their use on a website dedicated to selling coffee, such as Peet's. In this case, TreeMaps enable users to navigate through a complex assortment of coffee products efficiently. When a user visits the Peet's coffee site, they are greeted with a TreeMap that categorizes coffee into different sections such as Peet's Blends, Decaffeinated, Africa and Arabia, The Pacific, The Americas, and Dark Roast. Each category is represented by a rectangle, the size of which indicates the relative quantity or importance of that category within the overall assortment. This interactive TreeMap allows users to delve deeper into their specific interests. For instance, if a user is interested in decaffeinated coffee, they can click on the "Decaffeinated" rectangle. This action will reveal a detailed view of various decaffeinated coffee

Treemaps of Peets site





MarketMap's use of TreeMaps allows for sophisticated analysis

Peets' use of TreeMaps is more for presentation and communication

TreeMaps are useful for different purposes!

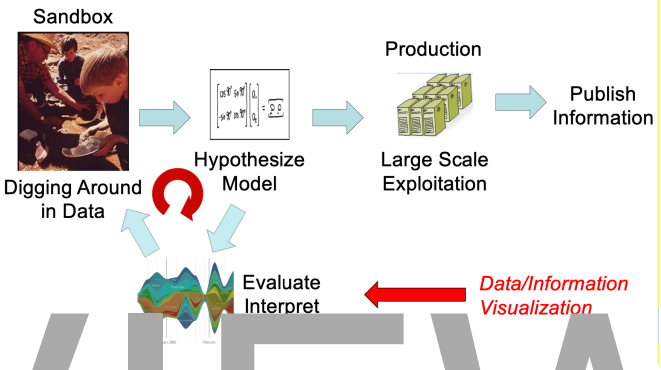
over time, and how these patterns interact. This layered approach enables a multifaceted analysis, making it possible to compare not just within the same period but across different periods and factors.

Important Lesson

Visualization or analysis?
- Visualization tools are helpful for exploring hunches and presenting results
- Examples: scatterplots
- They are the WRONG primary tool when the goal is to find a good classifier model in a complex situation
- In this case we need an analysis tool!
- Also need:
  - Solid insight into the domain and problem
  - Tools that visualize several alternative models.
  - Emphasize “model visualization” rather than “data visualization”

The final slide emphasizes the distinction between visualization and analysis. Visualization tools, such as scatterplots, are excellent for exploring hypotheses and presenting results visually. However, they might not be the best primary tools when the goal is to develop a classifier model in complex situations. In such cases, analysis tools are more appropriate as they provide deeper insights into the domain and problem. Effective analysis often requires visualizing several alternative models and focusing on model visualization rather than just data visualization. This approach ensures a thorough understanding and accurate interpretation of complex data.

Conclusions – The Data Scientist Workflow



Throughout this course, we have explored the typical workflow of a data scientist, culminating in the critical step of data visualization. However, visualization is not necessarily the final step in data analysis. The process begins with creating a "sandbox" environment, where initial exploration and experimentation with the data take place. This sandbox, a safe and controlled space, allows data scientists to dig around in the data, testing various hypoth odels w of using mag

In he sandbox phas dat scientists engag n "diggi arou in da ." This involves xplori the dat set, clean g it, a trying out variou nalyti techn ues t understand its s ctur and un rns. T stage is crucial f fami rizing c self n the data and i ntif ng any init insights o ssue hat need addressi

After th sandbox p ase, t next step is to "hy t size mod s." Th involves develop theoret l models l sed s and rela nships ob rved ial exp tion. These models represent the hypotheses that the data scientist believes might explain the underlying processes generating the data. This step requires a good understanding of the domain and the ability to translate observed patterns into formal models.

Once a model has been hypothesized, the next step is "Evaluate/Interpret." This involves testing the model against the data to see how well it fits and whether it can accurately predict or explain the data. Visualization is a critical tool at this stage because it helps to compare the model's predictions with the actual data visually, making it easier to spot discrepancies or confirm alignments. If the model does not perform well, the data scientist might need to return to the sandbox phase to refine the model or explore new hypotheses, creating a recursive loop between these two steps.

When a model is sufficiently robust and performs well during the evaluation phase, it moves to the "production" stage. In this phase, the model is deployed on a larger scale, often involving the full dataset or even larger, more comprehensive datasets. The transition to production involves ensuring that the model can handle real-world data complexities and operate efficiently at scale. This stage may include optimizing the model's performance, integrating it with existing systems, and ensuring it can be maintained and updated as needed.

The final step in the workflow is "publication," where the findings and insights derived from the model are shared with a broader audience. This could involve writing reports, creating dashboards, or publishing academic papers. The goal is to communicate the results clearly and effectively, ensuring that stakeholders understand the implications and can use the information to make informed decisions. Effective visualization plays a crucial role in this step, as it helps convey complex data and model results in an accessible and comprehensible manner.