AI for the Masses: Factor Tables a Possible Alternative to Neural Networks

5/2/2017 – Edition 1.0 – By Bryce Jacobs

Neural Networks have proven a powerful tool in artificial intelligence (AI). However, they are **difficult to relate to** in a common-sense or familiar way. They are esoteric and opaque in part because they don't sufficiently resemble common tools and idioms of the **typical work-place or office**, and therefore only specially-trained personnel can effectively work with them.

But, there is an alternative tool that *does* resemble common tools and techniques of the work-place: Factor Tables (**FT**'s). They can be analyzed and processed with common table-oriented tools like spreadsheets and relational databases, or at least tools that closely resemble those. They are the sort of things accountants and statisticians can relate to. They can make pattern recognition preparation and analysis more like "mundane" data processing and statistical analysis rather than an esoteric, highly-specialized field.

FT's can also be used to **break big problems into multiple smaller problems in a systematic way**, sometimes called the "divide and conquer" technique. This allows a specific team or worker to focus on narrower problem(s) to make them easier to understand and to manage from a personnel perspective. Thus, FT's may help "bring AI to the masses", if you will. It may *not* be a consumer-level of common, but potentially as common to commercial organizations as say accounting knowledge is. It could trigger an office AI revolution comparable to the introduction of the spreadsheet for personal computers in the early 1980's in terms of making financial analysis and planning mainstream

Perhaps these tables can also be called "weighted logic tables", "predicate tables", "fuzzy query-by-example", or a host of other names. I'll call the concept a Factor Table (FT) as a **working definition**, and don't claim it to be a formal or official name; nor do I claim to have invented them. They are similar enough to myriads of existing tools and conventions that tracing their pedigree is probably a lost cause. I'm not an AI professional, but merely an AI hobbyist. I like to "table-ize" things out of habit and thus played around with table-oriented AI. This presentation is a speculative and experimental idea to hopefully spark further research, analysis, and academic formalization.

Basic Example

Here's an example of a somewhat generic FT intentionally designed to resemble a spreadsheet for familiarity’s sake:

Observation ID	A	B	C	D	E	...	Notes
1	0.12	0.98	0.03	0.01	0.51	...	
2	0.85	0.29	0.94	0.33	0.54	...	
3	0.28	0.03	0.45	0.07	0.41	...	Corrected on Apr. 7 by Lisa
4	0.02	0.52	0.30	0.32	0.23	...	
5	0.72	0.77	0.21	0.84	0.04	...	
Etc...	...	...	...	...	...	...	

***Figure B.1** - A typical Factor Table*

The "Observation ID" is way to identify a particular observation or what may be called a training specimen instance in a typical AI project. The letters designate columns of scores or values from various metrics. These metrics may be simple measurements like height, weight, and color; or specific pattern-matching tests, such as the application of image processing templates like albedo (brightness) maps or edge maps, and perhaps measured levels for given frequencies and/or angles after Fourier analysis, etc. The selection of factors depends on the domain (project or object type) and related analysis to see what works best for the domain.

We'll call each metric a "factor". Each factor is referenced by the letters shown across the top. We'll assume all these factors are normalized to a value between 0 and 1, inclusive. This normalization is not necessary, and other normalization conventions are possible, such as -1 to 1, or even have no normalization. However, for consistency of illustration and explanation, we'll assume a 0 to 1 convention.

Normalization may also improve the mixing and matching of system components, parts, and tools to make the AI work-place of the future more efficient. For illustration simplicity, we'll also round most of our values to 2 decimal places. (It's assumed the reader is familiar with the general concepts of neural networks and the use of normalization of values.)

Although we used the alphabet to label our factors above to resemble a spreadsheet, it's also possible to use named columns, as found in a typical relational database (RDBMS). For our examples though, we'll usually use a single latter as the factor identifier. For instance, "W" will be used for "weight".

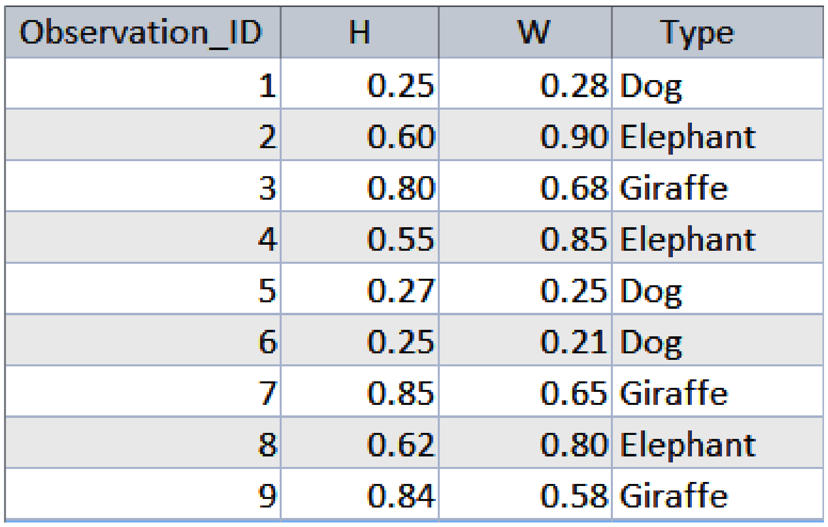
Some factors will be treated as inputs and some as outputs, depending on the task at hand. **FT's can be "chained" together in a way roughly analogous to levels of a neural network.** (In neural networks, these levels are sometimes called "hidden layers" or "hidden levels".)

We'll explore four kinds of Factor Tables:

1) Observation Table – actual observations and/or a "training set"
2) Match Table – a "test set" to test against the matching engine

3) Summarized Table – a kind of "compressed" table for abstraction and/or efficiency
4) Mask Table – a table used to filter factors of another table

Let's start with a simple example of an Observation Table:



Observation_ID	H	W	Type
1	0.25	0.28	Dog
2	0.60	0.90	Elephant
3	0.80	0.68	Giraffe
4	0.55	0.85	Elephant
5	0.27	0.25	Dog
6	0.25	0.21	Dog
7	0.85	0.65	Giraffe
8	0.62	0.80	Elephant
9	0.84	0.58	Giraffe

Figure B.2 *- Mammal Observation Table*
(H = height, W = weight)

These are 9 observations of 3 kinds of mammals: dogs, elephants, and giraffes. It uses 2 factors: height and weight. Typically there would be *many* more factors, but we are starting off simple. The "Type" column is a custom addition we've made for this particular project.

"H" stands for "height" and "W" for "weight". The values are normalized based on expected ranges of known mammals using approximate logarithmic scaling. On a normalized scale, the largest mammals, whales, would typically have a weight near 1, and the smallest mammals, perhaps shrews, would have a weight near 0. (We'll assume adult mammals here.)

A plot of the coordinates in this table will help illustrate the relationships:

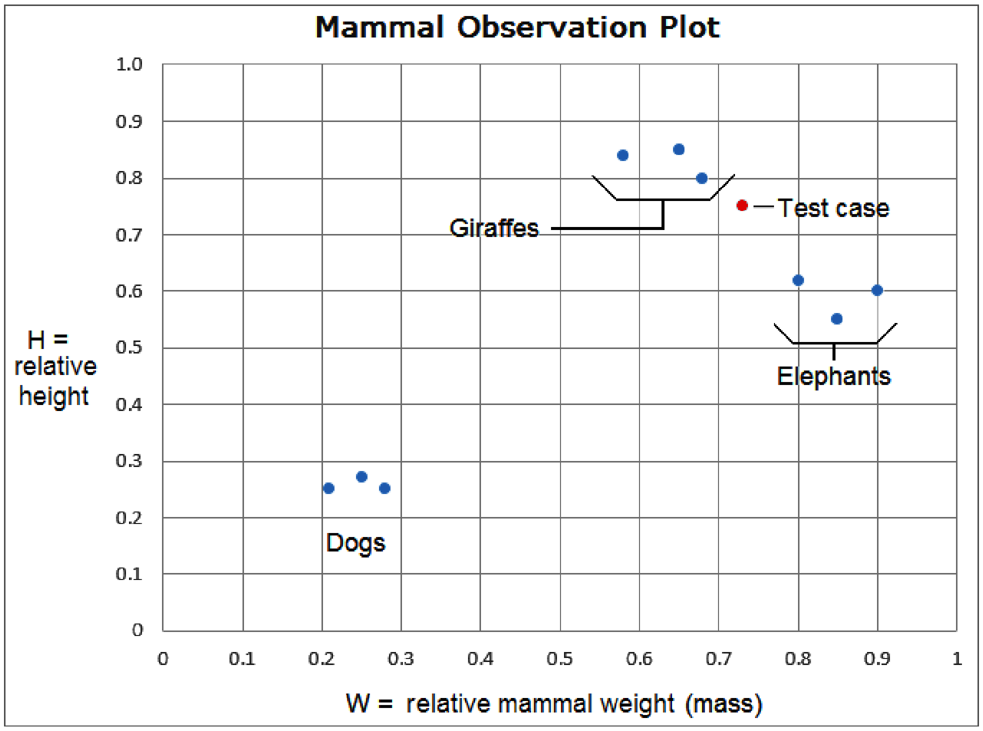


Figure B.3 *- Mammal observation plot*

Now that we have our observations, lets use a Matching Table to test a new sample (specimen). We want to see what mammal observation our test case is a best match for by comparing it to the observations or training set.

Test_ID	H	W
1	0.75	0.73

Figure B.4 *– A Match Table with one mammal match test case*

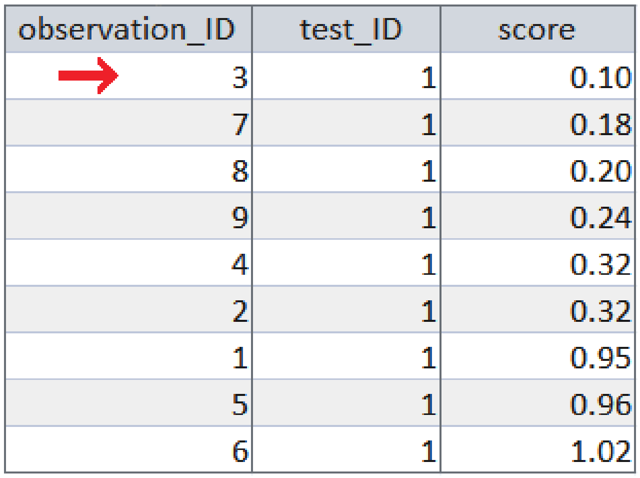
To keep things simple, we'll only test a *single* observation here. This single point is also shown plotted as "Test case" in Figure B.3 as a red dot.

For those of you familiar with SQL, here's some sample SQL to run such a test:

```
SELECT
   observation_ID,
   test_ID,
   abs(obs.H - mch.H) + abs(obs.W - mch.W) AS score
FROM observations AS obs, matches AS mch
ORDER BY score ASCENDING
```

SQL Notes: SQL knowledge is not assumed to generally understand the examples, but is given to help programmers relate, and help clarify the process. This example performs a "Cartesian join" to compare every test to every observation. It uses *no* key associations. Cartesian joins are rare in practice, but we've found a nice use for them here. Also, some dialects of SQL don't recognize column aliases in the ORDER BY clause. In such a case, you may have to define a view first, or a nested query, and then do the necessary ordering on the second or outer query. Remove "AS" in FROM clauses for Oracle SQL. Further, the SQL shown produces confusing output if one is comparing multiple tests. I'll leave the adjustments as a readers' exercise.

We'll go over the scoring math a bit later; let's first look at the result:



observation_ID	test_ID	score
→ 3	1	0.10
7	1	0.18
8	1	0.20
9	1	0.24
4	1	0.32
2	1	0.32
1	1	0.95
5	1	0.96
6	1	1.02

***Figure B.5** - Scoring result*

Here, the lowest score is the best match. It's giraffe number 3 (observation 3), which has the lowest match score of 0.10. We can visually tell it's the closest point to the "Test case" point shown in Figure B.3.

No normalization has been done with the scores shown. If we wanted to use the results with other FT tools and processes, it may be better to normalize for consistency. Whether to normalize based on a single result set (a results table), or based on prior tests is an engineering and/or business decision for a particular project.

The fact that we have *multiple* ranked results can come in handy in "back-tracking". If the top fit proves insufficient upon further analysis, we can use the second-best fit, and so on down the rank.

Back-tracking will be described in more detail later.

One may notice that outliers could potentially skew the results. The table probably should be cleaned up somehow to avoid such using the various outlier-removal techniques from the field of statistics. Averaging, one technique which will be described later under "Summarized Tables", is one crude approach to outlier removal. Removing outliers *before* averaging may prove even more effective, but is a situational judgment call, per project.

Because our simple example has only two factors (dimensions), we can visually verify the results by inspecting a graphical X/Y plot like Figure B.3, shown earlier. But in a production system there will usually be too many factors to plot, at least in a fashion that makes sense to most people. Various reports and statistical tests would instead be used to check the validity of tests and make adjustments. Examples of inspections will be given later.

To calculate the match score, we kind of cheated on our math. We used the sum of the absolute value of the differences. However, ideally we'd use a ***Euclidean distance***, an extension of the "Pythagorean distance formula", to find the closest match in multiple dimensions. But to simplify our math I used a rough-and-dirty **approximation** of the Euclidean distance via the sum of the absolute value of the differences. I'll leave the Euclidean version as a readers' exercise.

For larger data sets and more factors (more dimensions), it may be computationally advantageous to use the shorter approximation, especially if the matching engine is embedded in a portable device or there are several hundreds of factors. This accuracy-versus-computation-time is another engineering design trade-off we won't delve into further; this is just an introduction, not an optimization guide. The key point is that a FT system engineer has choices to select from based on intended usage.

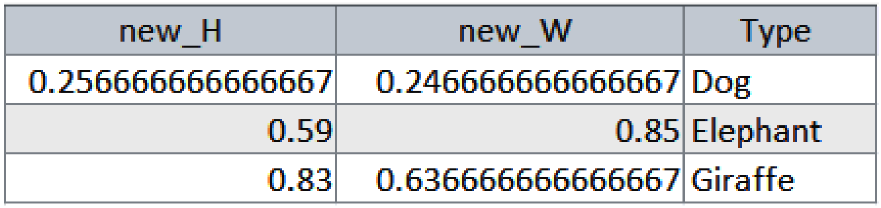
Summarized Tables

Rather than compare specimens to bunches of specific observations, we can simplify our tables and processing by creating a "summarized" version of an Observation Table by producing an average prototype record (row) for each mammal: one dog, one elephant, and one giraffe. We can "average" our table entries to achieve this:

```
SELECT Avg(H) AS new_H, Avg(W) AS new_W, type
FROM observations
GROUP BY type
```

SQL Notes: "Avg(...)" is the averaging function. "Type" is a reserved (special) word in some SQL dialects, and my trigger an error. Consider naming the column "mammal_type" or "m_type" if your dialect is confused by "type".

Giving us:



new_H	new_W	Type
0.256666666666667	0.246666666666667	Dog
0.59	0.85	Elephant
0.83	0.636666666666667	Giraffe

***Figure S.1** – A Summarized Table - results of averaging. (Un-rounded)*

It won't always give the same match results as the original (pre-averaged) table if we re-ran test cases, but it does save space and time. It's **an abstraction** or generalization; a form of AI "learning" where generalizations are formed based on repeated encounters (samples).

In a typical business you have high-level reports for executives that may sum up product or service categories by sales regions. If the executives have a question about one of the entries in the summary report, the staff usually keeps detail reports or data that allow the organization to trace the source and reason for each value on the summary report back to a specific product at a specific store on a specific day if need be. (In the newer "web" report way of doing things, executives often can "drill down" to specifics themselves by progressively clicking on the summary categories.)

We can have something similar going on here: one can readily know where the abstraction or summary data came from as long as we keep a copy of our original data. There is no mystery about how we got our numbers: accounting meets AI (or AI meets Accounting).

We used basic averaging to keep our example simple, but there are more sophisticated approaches to provide potentially better statistical accuracy, such as weighted averages, density maps, and cluster analysis as found in advanced statistical courses and literature. Some of these may lead to dealing with multi-dimensional polygons, "fuzzy" polygons, and other "fancy" techniques, which we *won't* delve into here. Whether to stick with points or use advanced techniques is again an engineering/business tradeoff decision.

If you over-complicate it, then your staffing requirements go up. I suspect in many cases one can create more connected layers (steps) of simple FT's to compensate for not using the fancier techniques, and/or have a higher quantity of simpler factors rather than fewer complex factors. In other words, you can choose to have a system with lots of simpler parts instead of a system with fewer complicated parts with roughly the same performance. Being a new field, we'll have to wait and see to what degree my hunch is correct.

If the distribution of a factor is a fairly typical "bell-shaped curve", sometimes called a Gaussian distribution, then averaging points may be sufficient. Plots and/or statistical analysis can be done to see if the curve is typical. If it's not, then further investigation may be warranted.

For example, there are smaller "pigmy" elephants (Loxodonta pumilio) in addition to regular elephants. If pygmy elephants got into our data, you may find a **two**-humped distribution curve for factors such as height. In such a case, consider splitting your "elephant" category into two categories: pigmy elephants and regular elephants, or removing the pigmy data altogether if they are not likely to be encountered in production data. It's generally not a good practice to average diverse sub-sets into a single point; Studying unusual distributions will also help you learn about the domain (subject matter) so as to better tune the system.

The process designer may even decide to suppress usage of height and weight info altogether for elephants if they don't want to bother tracking these two kinds separately (assuming we have other factors to use beyond those in our example). Masking Tables, which we'll explore later, are one possible way to *conditionally* dampen or remove factors that prove unhelpful or uneconomical to tune for.

Summarized Tables (**ST**'s) are best used for **classification and grouping**-related tasks, such as determining if a given image is a certain *kind* of animal, or if a piece of furniture is a "couch". One probably would *not* use ST's to identify individual objects or persons. Observation Tables (**OT**), mentioned earlier, are typically a better choice for finding (matching) individuals.

However, some stages in processing may work better with ST's and others OT's. For example, in an image classification system, ST's may be used to determine if an image has one or more human faces in it. Once it's determined a given photo has faces, it can then be fed into facial recognition system to identify the identity if given individual(s) using specific specimen matching in an OT, similar to the mammal observation example related to Figure B.2, seen earlier. Thus, "contains faces" may be computed via averaged or summarized data, and therefore ST's. But, identification of faces of individual people would more likely be done via OT's.

Mask Tables For Selective Filtering

We just explored the reduction of rows to simplify our data or model. But at times it may also make sense to reduce the columns (factors) used in processing.

One way to reduce columns would be testing the accuracy of draft FT's to figure out which columns don't matter enough to keep for production. For example, suppose we have factors for general color split into red, blue, and green levels (which digital images typically use to represent the full spectrum of colors visible to people).

Tests and/or statistical analysis may indicate that the factor for green is not very useful for identifying mammals. Most mammals are various shades of gray, brown, or tan such that green-ness doesn't come into play often. Thus, we can remove the column for green with minimal loss in accuracy.

It may also be the case that certain factors matter for certain targets but not others. One can say they "**conditionally matter**".

For example, in a mammal identifier system, an animal's coat-texture-related factors (columns) may make a big difference for identifying tigers and zebras, but not for dogs or cats because their coats vary widely (due to domestic breeding of a variety of coats). If *other* factors tell the system a specific observation is not looking at a tiger or zebra, then maybe we should suppress coat-texture-related factors for this specimen to avoid biasing the scores and/or wasting processing time. If we know certain factors are useless under some conditions, then we'd want to switch them off, or at least down-play them.

> Warning about similar terms: "Weight" (mass) of mammals is not to be confused with factor weighting. Factor weightings are used to emphasize or de-emphasize a given value via multiplication. Similarly, be careful not to confuse "image mask" and a "Mask Table". These are different concepts.
>
> With albedo (brightness) image masks, you typically subtract the candidate image pixel values from the mask image and add the absolute value of the differences to get a difference score, perhaps after brightness and/or alignment normalization. Transparency maps or other techniques may also be applied to ignore backgrounds. Edge maps may be multiplied instead of subtracted to ignore background features. I used the negative of a typical edge map for illustration clarity in later examples, but one would probably would flip the values during processing such that white is zero. Such a trick may also help staff work with photos, being the "dark edge" version is easier to relate to. Albedo and edge-detection masks are common kinds of image masks, but not the only. This document is *not* intended as an introduction to image masking and scoring at the image and pixel level. Please consult other resources for such.

Here is a conditional Mask Table for our mammal example:

Condition	M1 (height mask)	M2 (weight mask)
➡ 1	0.00	1.00
2	1.00	0.00
3	0.50	1.00

Figure M.1 *- Simple Mask Table*

Let's add masking weights to our SQL expression originally seen just below Figure B.4:

```
SELECT
  mask.M1 * abs(obs.H - mch.H) +
  mask.M2 * abs(obs.W - mch.W) AS score
FROM observations AS obs, matches AS mch, mask
WHERE mask.condition = 1
ORDER BY score
```

For those new to SQL, here's a guide to the expression parts on the second line:

`*`	= multiplication (an SQL convention)
`mask.M1`	= Mask Value 1 (mammal height multiplier)
`mask.M2`	= Mask Value 2 (mammal weight multiplier)
`abs()`	= absolute value function. Thus abs(x) is |x|
`obs.H`	= Observation (or training summary) of mammal height
`obs.W`	= Observation (or training summary) of mammal weight
`mch.H`	= Match case (test) of mammal height
`mch.W`	= Match case (test) of mammal weight

So, because **Condition 1** is active, per our SQL "WHERE" clause, mammal height (H-related results) is suppressed because multiplying it by M1, which is zero, produces zero. If all rows tested result in zero for the H factor computation, then those values will not be a difference maker in the final score because adding zero to a given value doesn't change it.

If **Condition 2** were active, then weight would be similarly suppressed instead of height. If **Condition 3** were active, then we get a ***partial*** *dampening* of height's effect on the final scores because the multiplier on height (M1) is 0.5, and thus height would be "semi-masked".

There are many ways to compute which condition(s) are active, largely dependent on the project and subject matter (domain). In some cases, programming code will need to be written to make such determinations. In others, standardized prepackaged techniques can be used so that a data analyst can fill in parameters or weights to guide the automated system(s) in determining which mask table row or mask table is applied. Since FT's are relatively new in AI, what kind of prepackaged computation kits or features work best will require practice and experience. A future AI work-bench software tool may have tool-bar icons to launch or set up common and prepackaged FT operations, similar to what's found in spreadsheet software.

Identification processing may be easier to test and manage if there are chains of such operations, or perhaps hierarchies. For example, analysts may split mammal identification into a tree of potential steps resembling the following:

- Bipedal (walks on 2 legs):
    - Snout detected = Kangaroo
    - No snout = Human
- Quadruped (walks on 4 legs):
    - No or insignificant tail:
        - Low hind-quarters-level = Ape
        - Medium hind-quarters-level = Bear
    - Significant Tail detected:
        - Dog...
        - Cat...
        - Monkey...
        - etc...

***Figure M.2** – Mammal identification tree (simplified)*

This approach can be used to split up the analysts and other staff members into sub-groups to focus on specific topics, such as having one group working on just bi-pedal mammals, another on quadrupeds without tails, another on quadrupeds with tails, and so forth. The processing flowchart may end up looking something like Figure T.2 (discussed later), and the staff organizational structure may roughly mirror the tree also.

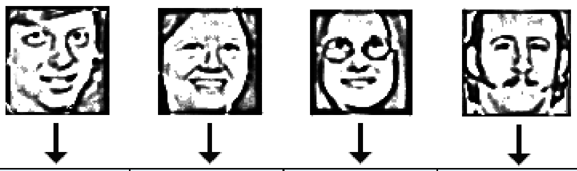
Note that a single factor such as tail-size is probably too limiting in practice. Real-world decision criteria would likely involve more factors and/or computations. Also, there may be other ways to partition the processing besides just biological classification, such as identifying the orientation of a mammal in an image. A standing animal may be routed differently than a sitting mammal, for example. Further, the processing division may not be a "pure" tree because given tasks or components may be used for multiple purposes or animal types.

A key goal is to make AI more like regular business operations were projects are split up into teams and sub-teams as needed instead of relying on "lone geniuses" with their esoteric opaque neural networks or what-not. If we use tables; then factors, masks, and processes can be divided up as needed to break big projects into bunches of smaller tasks that are easier to understand and manage. We don't try to do too much in any given step (processing branch).

Interpolation with Mask Tables

Above we assumed one condition would be active at a time, such as having enough evidence to conclude we are we looking at a quadruped mammal in order to simplify processing and/or division of labor. But it **doesn't have to be all-or-nothing**.

Let's consider a hypothetical facial recognition application where a large security-conscious organization wants to be able to identify employees in security camera photos (stills) using the employee security badge photos. An FT from such a system may resemble the following:



Employee Number	Template A	Template B	Template C	Template D	Etc...
091837	0.12	0.98	0.03	0.01	...
138921	0.85	0.29	0.94	0.33	...
488250	0.28	0.03	0.45	0.07	...
590493	0.02	0.52	0.30	0.32	...
Etc...	...	...	...	...	...

Figure I.1 *- Employee badge photo match scores*

Here we have the matching results for multiple image templates based on the employee badge photos. Assume both the factor templates and badge photos are edge-map-processed. (The later Figure F.1 illustrates how an edge-map relates to a "regular" photo. Edge-maps are often used in AI in part because they are less sensitive to lighting conditions than albedo [regular] photos.)

Let's say we are using 100 facial templates, which would be 100 factors, or 101 columns if we include the employee number. The templates themselves are *not* necessarily based on employee photos; they could be extracted from stock photos selected by a facial recognition specialist. If there are ten-thousand employees, then there would be ten-thousand rows, assuming the preparation staff is able to keep it up-to-date. (The organization may even keep ex-employee photos to protect against breaches by former employees.)

Each and every employee badge photo is pre-processed for normalization, edge detection, etc., and then matched against the 100 templates (factors). The match result scores are stored in the FT using our normalization convention of being between zero and one, giving us something like Figure I.1, above. This would be an Observation Table. There may be more than one Observation Table for different facial angles and/or feature emphasis, as we'll discuss later.

When a security camera photo is analyzed for a badge photo match, the security camera photo is processed the same way as badge photos to produce a Match Table of one row, comparable to figure B.4 of our earlier mammal example, but with 100 factor columns. (One could also do batch processing on multiple security camera photos, but for brevity we are describing only one.) We'd get a result comparable to Figure B.5 of the mammal example.

When an employee of our hypothetical organization is photographed for their security badge, photos are taken at **three facial angles**: frontal (0 degrees), a 45 degree angle, and profile (90 degrees). Only the frontal photo is used on an employee badge, but the other two angles are kept for the facial recognition system. Figure I.2 illustrates our labeling and normalization conventions.

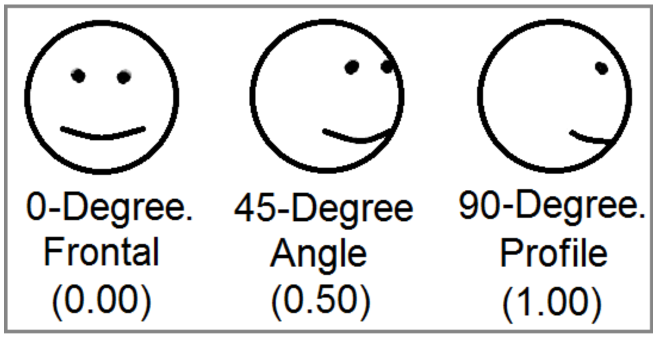


Figure I.2 *- Facial angle conventions*

(No, that's *not* the Jack-In-The-Box mascot. No jokes about the high-budget graphics, please.)

To follow our value normalization conventions, the angle will be represented as a value between 0 and 1, inclusive. Zero (0) degrees (frontal) would be represented as 0, 45 degrees would be 0.5, and 90 degrees (profile) would be 1. An angle of 22.5 degrees would thus produce a 0.25 when normalized. (If the math for this doesn't make sense yet, be patient, for we'll revisit that issue.)

Let's assume all faces are normalized such that all angled shots face to the right. A given image is mirrored (reversed) to normalize to the right if necessary. While we can control the side used for the badge photo sessions, we cannot control it in security camera photos.

> *Side Note*: Significantly uneven faces will be considered too rare to bother with for this application. This assumption can reduce storage space and processing to about half of what it would otherwise be. We'll also ignore from-above, from-below, from-behind shots, and smiling-versus-serious to keep the example simpler. Also, some kind of sizing and positioning normalization will be assumed so that we don't have to save and process image templates for different relative sizes. These could be additional pre-processing steps in a production system, where each pre-processing step also uses FT's or other AI tools.

An image being examined (tested) is unlikely to be at one of our standard observation angles of 0, 45, and 90 degrees. A test case (security camera photo) may have an angle of 23 degrees, for example. We don't have a close match for that in our observation set. Thus, we have to somehow **make do with what we have**: three angles.

Let's assume we have an FT-based pre-processing system that estimates the angle of the test case face, perhaps also using image templates, but the technique doesn't matter here. Keep in mind this pre-processing step's job is *not* to identify the face in terms of who it is, but only estimate the angle (between 0 and 90 degrees with our mirroring assumption). We will normalize the angle value between 0 and 1 as described earlier.

Those people building and tuning the angle detector only have to focus on angles, and not facial matching itself. Matching to individual people could be another group's job, once the data from the pre-processing steps is ready. (Possible cross-sharing to simplify the processing load will be discussed later.)

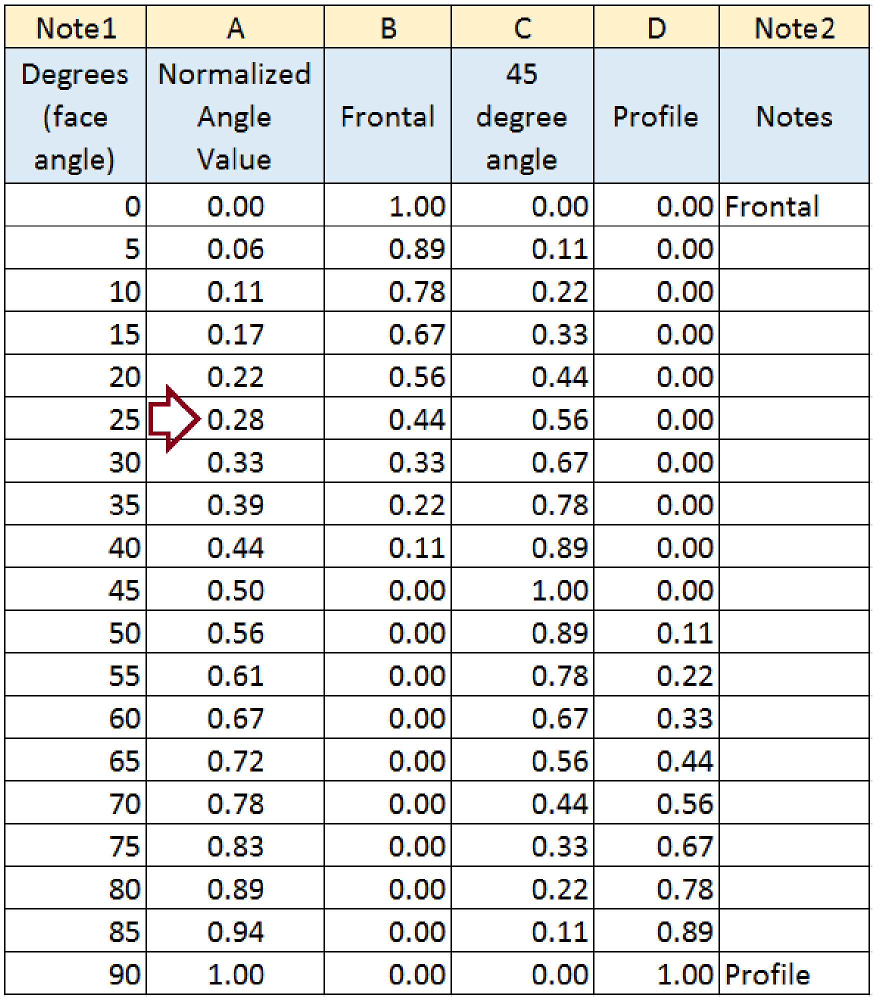
> *Side Note:* In constructing the angle detector, keep in mind we can use facial image masks (templates) that are at a variety of angles, unlike our observation set (employee photos). This is because we are comparing to actual photos, such as security camera stills, that are at various angles.
>
> One possible implementation of an angle detector is to use image templates with sample faces at 5-degree angle increments, giving us 19 rows in an FT to match against. I would guesstimate we'd need *at least* 20 templates (factors) per angle for decent results. But, we *won't* go into that idea further here, and assume the angle detector supplies us an estimated angle value for a test case (specimen) to use for further processing.

It's likely that certain detection factors will only be applicable to certain angles or ranges of angles. A profile-detection image template probably won't work very well on a frontal image, for example, and thus we'd **want to suppress** most or all profile-related factors for small angles (nearly frontal).

If we use the "condition" selection approach described earlier near Figure M.1 in our mammal example, then we'd have to determine discrete categories to use. But angles are not discrete. If we had observation (training) samples from a wide variety of angles, say in increments of 5 degrees, then rounding to the nearest 5-degree "condition" may be sufficient. But, our particular observation set is not that thorough.

Obviously it would take more time to photograph employees from 19 different angles in order to get 5-degree increments. An organization likely would want to avoid that to save time and money if possible. Mug shots (arrest ID photos) are typically taken only at two angles: frontal and profile. Detectives hunting for matches among mug shots will have to live with just two angles in the observation set. We'll assume we are slightly more lucky and have *three*.

In order to handle the in-between angles, an organization may use **interpolation**, similar to the following:



Note1	A	B	C	D	Note2
Degrees (face angle)	Normalized Angle Value	Frontal	45 degree angle	Profile	Notes
0	0.00	1.00	0.00	0.00	Frontal
5	0.06	0.89	0.11	0.00	
10	0.11	0.78	0.22	0.00	
15	0.17	0.67	0.33	0.00	
20	0.22	0.56	0.44	0.00	
25	⇨ 0.28	0.44	0.56	0.00	
30	0.33	0.33	0.67	0.00	
35	0.39	0.22	0.78	0.00	
40	0.44	0.11	0.89	0.00	
45	0.50	0.00	1.00	0.00	
50	0.56	0.00	0.89	0.11	
55	0.61	0.00	0.78	0.22	
60	0.67	0.00	0.67	0.33	
65	0.72	0.00	0.56	0.44	
70	0.78	0.00	0.44	0.56	
75	0.83	0.00	0.33	0.67	
80	0.89	0.00	0.22	0.78	
85	0.94	0.00	0.11	0.89	
90	1.00	0.00	0.00	1.00	Profile

Figure I.3 *- Interpolation for masking*

This example Mask Table has 4 factor columns: A, B, C, and D; and two "note" or comment columns for reference info. The angle detector step described earlier would supply the normalized estimated angle factor to be used as the "look-up" factor here, per column "A".

If the estimated angle of the test facial photo were 22.5 degrees (on the usual 360-degree scale), this would translate to a normalized angle of 0.25 on our project's 0-to-1 scale. Because we'll round up if a tie, the nearest match in our interpolation table is 0.28, the 6th data row of figure I.3.

Looking at columns B, C, and D for the 6th data row, we know to apply the frontal templates with a weighting of 44% (0.44) for the frontal templates, a weighting of 56% (0.56) for the 45-degree templates, and zero (none) for the profile factors (profile image templates).

Note that 0.44 plus 0.56 adds up to 1.0 (one). This is typical when we are "distributing" the contributions of multiple rows, FT's, or processes; and keeps us compliant with our normalization convention. We could say we are giving the frontal factors 44% of the votes, the 45-degree factors 56% of the votes, and no votes for profile templates in our test specimen.

We could have instead programmed in mathematical formulas to calculate the interpolation weights, but if a programmer is not available or deemed too expensive, a table or sheet such as this can be computed by hand or via spreadsheet formulas so that we can use a simple existing find-nearest-match operation. We can ask our local math whiz to create the interpolation data in a spreadsheet, and we then import it into our FT system(s).

> *Side Notes*: Our find-nearest-match operation perhaps could produce multiple ranked results, similar to Figure B.5 to be more general-purpose, but for our simplified example we'll just grab the first and best match (angle) only. It's basically the same operation as the Euclidean Distance (or it's approximation) approach described earlier, but done on just one factor. Thus, we don't have to create a new kind of operation for this if we were making a generic FT tool kit. It's one we already used (perhaps along with a get-top-X operation). If we keep our values normalized, I suspect a lot of powerful processing can be done with just a handful of "generic" FT operations tuned via parameters, reducing the need for custom programming.
>
> In addition, if we are using typical RDBMS's, then most of the SQL would also likely be generated for us, once we have a way to systematically identify which factors (columns) to process for a given step. Spreadsheet-like ranges and/or column group names, reminiscent of email group reference names, such as "departmentX" as found in a typical medium or large company, are some possibilities to simplify column selection on a larger scale. Some examples of why such may be warranted will be given later.

Factor Generation and Pruning

One potential weakness of FT's is that they don't automatically generate or compute factors. For example, with NN's (neural networks) one can take the bytes of an image and directly map those to the input layer of a NN. The starting NN can be filled with random node weights, and one can then start the training process.

FT's as defined here cannot practically do something equivalent. The factors have to be manually selected before-hand, or at least generated by a technology besides FT. A skilled AI practitioner would probably have to be involved in the creation of factors, although NN's and genetic algorithms can perhaps assist in the generation and/or tuning and selection of factors.

> *Side Note:* If one wanted to make a generic image classifier using FT's, one experimental possibility is to use a wide variety of edge detection masks and albedo (brightness) thumbnails as factors, perhaps with fairly wide rotation, size, and position comparing when scoring against each cell. However, it would probably need an awful lot of templates (factor columns) to be useful. Both Observation Tables and Summarized Tables can be tried. If a "diversity detector" senses a picture is complex, then splitting it into overlapping tiles for sub-analysis may be in order.

I'll use our on-going facial recognition example to speculate on how this process may proceed. Rather than having one big network or component to do everything, FT's are probably best used when one wants to split tasks up to distribute the staff, system design, and/or processing work-load, similar to our mammal sub-categories related to Figure M.2.

We already discussed using preliminary detection and normalization of face angle as a separate step from actual person detection. We'd may also want to normalize face size and position in a given image to simplify later sub-processes. For example, you don't need as many edge-detection image mask templates if the faces are all scaled to be approximately the same size, say using the vertical distance between the mouth and eyes as the reference standard. One still may want some localized "micro-alignment" matching by testing various small-scale template offsets and using only the best score. But normalization (pre-processing) of position and size greatly reduces the number or range of such offsets needed.

Keep in mind that factors and/or their results used in one stage of processing may also be reused for others, if applicable. Thus, we can save some processing and/or storage resources. Some image masks used for facial angle determination may also be found useful for face size normalization, for example.

Since we are talking about a chain or tree of several processing steps, it may seem we are overwhelming the machine. But if we can re-use some factors and/or their results between these processes, we can cut down the load. If we play our cards right, we only have to "reinvent" *some* of the computation wheels for each step.

While statistics and trials can help narrow down which existing factors are most useful, we still need factors (measurements and templates) to **start with** to test and study. Some experience in AI specialties helps here. Those who have studied image recognition in general will probably agree that edge detection masks (sometimes called delta masks), and direct albedo (brightness) masking can both potentially be useful, although are not the only known techniques.

If we want to downplay hair and clothes, then focusing on eyes, nose, and mouth may be warranted such that we may want to wash out the edges of our image templates to only have those three features. If we are not sure, we can try both full-face templates and eyes/nose/mouth-only templates and then use analysis to prune factors.

What we focus also depends on the intended use of the final recognition tool, or even intent of a *specific* search case. For example, if we want our face identifier to also detect people with disguises, such as fugitives, then we'd probably want templates that downplay hair and beards, perhaps focusing on mostly just eyes and nose. But for "every day" identification, hair style and color are probably useful clues.

People also tend to get chubbier as they get older such that facial boundaries tend to spread out over time. Thus, if we are using old photos to identify a specimen, we perhaps would want to filter out facial boundaries and use templates with only the middle of the face, similar to template "**(e)**" in Figure F.1, below.

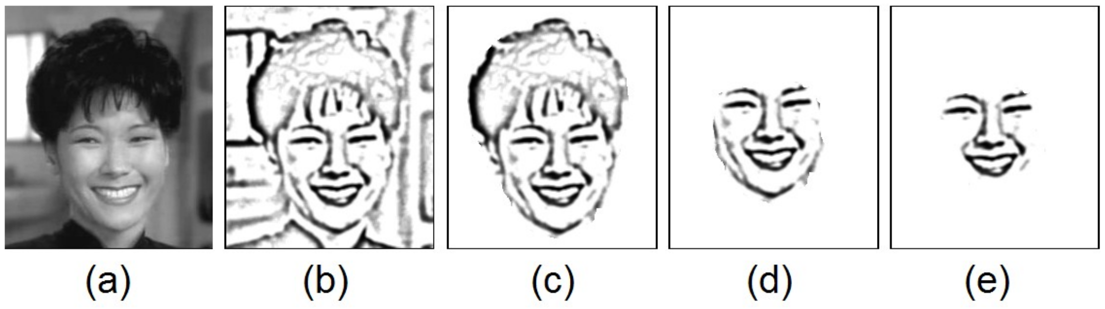


Figure F.1 *- Image templates emphasizing different aspects*

Explanation of each panel of Figure F.1:

(a) Original image, an albedo (brightness) map.
(b) Raw edge-masked version (areas of high albedo delta's are darker).
(c) "Cleaned" edge-mask with background manually removed.
(d) Hair manually removed to make it more hair-style agnostic.
(e) Face edges manually removed to make it more age or weight agnostic.
Note: Images centered for illustration, not necessarily for production runs.

Using Mask Tables, described earlier, we can control which factors or group of factors we want playing a role in the matching score for a *particular* search. If we wanted to diminish the impact of hair-related factors, factors tied to image templates or masks having hair, such as template "(c)" in Figure F.1, would then be multiplied by zero or low values to diminish their weights in the match scoring process.

Thus, the factors we choose can relate to what features we want to target, and we can adjust how features or feature groups are weighted during actual field match sessions. We can switch various groups on or off, reminiscent of a traditional church pipe-organ player using the "stop" knobs to change the organ's sound by switching on and off banks of air pipes (typically grouped around octaves.)

Using our employee identification example, our original description assumed 100 frontal face templates. (Let's ignore the "angle" issue for now.) These would be comparable to template "**(c)**" in Figure F.1. If we wanted another set that de-emphasized (ignored) hair and facial boundaries like "**(e)**", then somebody could use a graphics editor to remove hair and face edges from copies of the first set of templates.

Thus, we'd have 200 templates (factors) now: 100 with hair and face edges, and another 100 without. Whether to store these in one big FT with 200 factors or split them into two FT's with 100 factors in each is another one of those engineering tradeoff decisions based on usage and system design.

But before we make such variations, we still have to find factors (templates) to start with. One technique is to <u>use actual specimens</u> as image masks/templates. We could get photos of 50 people, for example, create an edge mask and/or an albedo mask for each one, and then have 100 factors. We may have to repeat this for different facial angles or feature emphasis as described earlier, but since those may be a different processing path, we'll just concentrate on one node of process tree here.

Experience suggests it's often useful to use somewhat extreme actual specimens as masks or templates. We could find photos of actual faces with somewhat extreme features. For example, we can find people with wide-apart eyes, narrow eyes, wide mouths, narrow mouths, long noses, short noses, etc. Semi-extreme masks and templates tend to save processing time compared to similar-looking templates, although we probably want an average specimen also. For example, we may want to include an edge mask with at least one wide mouth, one medium mouth, and one narrow mouth. A graphic artist may want to fudge some of the templates if we cannot find actual specimens covering a sufficiently wide gamut.

The granularity of our templates may also need trials and statistical analysis to see which are most effective. For example, do we want masks with individual facial features such as just eyes and just noses? Or perhaps combinations of two features such as eyes-and-noses together and nose-and-mouth together? Do we want high resolution or low resolution masks? We don't want to keep ineffective factors in our production system, for they slow processing and could even create unwanted side-effects, like inadvertently matching an eyebrow to a mouth.

We can choose how deep to explore these questions per practical constraints. The more factors and specimens (training set) we test, the better "pruned" our tables and processes will be. How long we spend depends on a combination of up-front pruning labor, machine efficiency (storage and/or speed) of the final application, accuracy of the results, and ability of the staff to trouble-shoot problems or poor matches. How these are are balanced is, you guessed it, a business and engineering decision.

For initial testing we probably want a wide variety of kinds of factors but not too many of each kind because it's premature to invest such resources. When tests show a certain kind of factor is more useful than others, we can then create more factors of the same or similar kind. Thus, if edge masks prove more useful than albedo (brightness) masks in trials, we create more edge masks for the production system, and maybe trim or skip albedo masks.

Genetic algorithms could even be used to generate factors to try, such as generating and/or cross-mating edge map facial templates. One could potentially "breed" better templates/factors this way. FT's could be part of a multi-tool AI kit.

Back-Tracking

If we are using the multi-step approaches described above, it's possible certain matches will "derail" the processing path a matching run takes, producing poor results. For example, a match (test) specimen may be a profile view of a head, and the ear is mistaken for a nose by the system such that it's classified as a frontal view of a face. All **further matches down the process road will probably be useless** because the system would be applying frontal-intended templates and/or algorithms to a profile image. We will likely get poor match scores and a poor final result.

We want to do something about this problem when detected. If a bad match is made early in the process, the rest of matches down the chain will often *also* have poor match scores. We can use this fact to **attempt a "redo"**. Low matching scores from later steps could trigger a flag in the system that indicates one of the earlier steps may have gone wrong. The trigger levels could be determined by examining test runs, logs, and/or statistical averages from production processing. The method selection would be an engineering tradeoff decision. We'll start off with a rough guess using a rule of thumb.

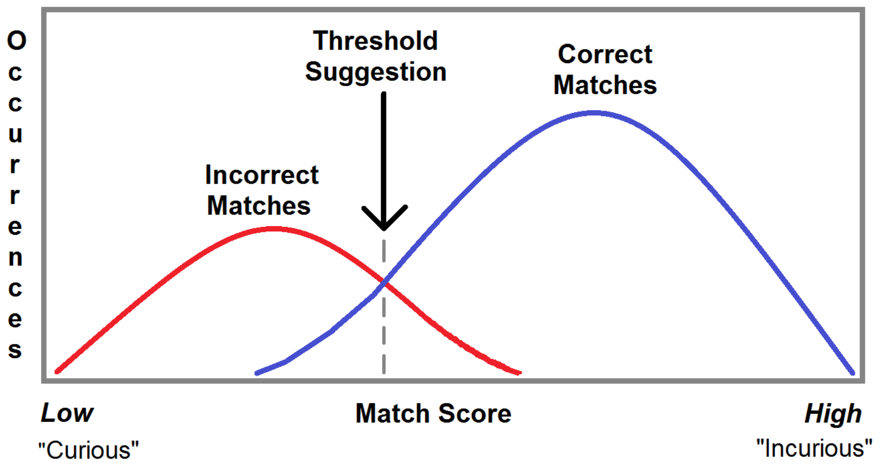


Figure T.1 *- Estimating SCF threshold*

Figure T.1 shows a roughly typical plot of the final match score of successful matches versus unsuccessful ones based on trials or log data studies. Here we are eyeballing a line to guess at a level to trigger a suspicious match condition. There are trade-offs to setting it too far in either direction, which we will discuss later.

If such a **suspicious condition flag (SCF)** is set (triggered), then the system can know to "backtrack" up beyond the point where the last poor score is found to try one or more alternatives. It's not unlike getting lost while driving, and returning back a prior road that we had more confidence in to serve as a base from which to explore alternative paths. One could call such the "last known good road". (Before in-car GPS, this was an important skill for drivers.)

Further, we can log SCF's in testing or production systems so that later analysis can be done to better understand the failures. The specimen (image) and the scores from the score tree (process steps) can all be saved in logs when SCF's are encountered. Ideally we save everything so we can study it later, but that's not always practical for resource or privacy reasons. The SCF can help us limit the log size.

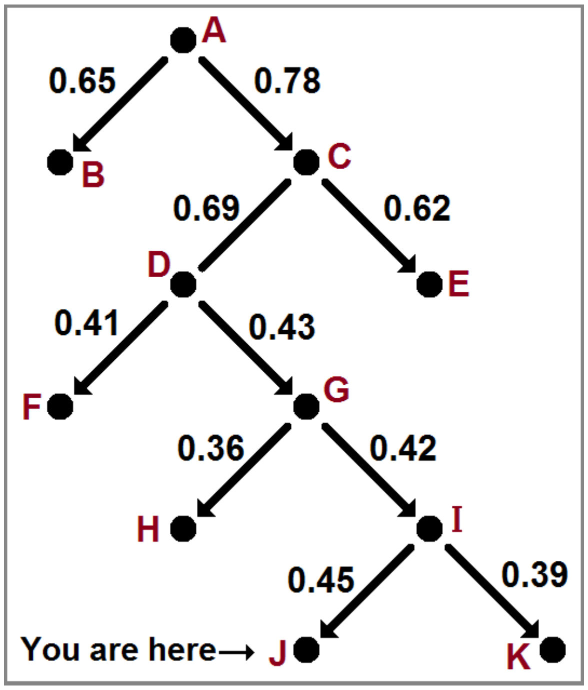


Figure T.2 *- Processing step tree. A higher score indicates a better match in this case.*

Here in Figure T.2, we have a hypothetical processing tree with normalized scores from 0 for lowest match score to 1 for a highest match score. This is a different convention than our scores in the earlier Figure B.5, which wasn't shown in a normalized form to keep the math easier to follow. Score representation conventions may vary for various reasons, but in *this* case, a high value means "good match".

In practice there may be many more than two branches per node, depending on the kind of test or application. I'm using a loose definition of "branch" here in that any process or step that returns or can return more than one candidate result is considered a potential "split" in our branching. All the rows in Figure B.5 from the earlier mammal example can be potential branches, for example. (Queries are often limited to the top-most matches or scores in practice so as not to overwhelm the system. The selected limit level is another one of those engineering decisions.)

And, we won't necessarily know the scores of *all* possible branches because we usually don't want to investigate unpromising branches in order to save time and resources. Figure T.2 merely shows enough branch scores to illustrate a point.

In our sample, we'll assume the processing is currently on the path A-C-D-G-I-J of Figure T.2, which so far has produced the highest known scores. If we keep track of the average path score, we'll see that our average is rather low, or at least trending low.

If we study the scores on our path, we see that things looked fine *until* node D's results (0.41 and 0.43). If we wanted to backtrack to try a different route, then we could could pursue E or F. Node E looks more promising because 0.62 is sufficiently higher than the 0.41 of node F. If node E also turns out to be a dud, we could come back and try node F later.

The backtracking techniques are similar enough to backtracking techniques found in game theory and general AI decision-tree traversal such that we probably don't need to delve deep into that here beyond rough examples, for most of the tree-traversal algorithm development work has already been done for us in gaming AI and similar fields.

Backtracking in gaming AI would be used for example when a bot computes the utility of various chess piece moves. The bot would simulate a potential move of its own, and then simulate the opponent's likely counter move(s), and then the best bot move after that, and so on. This creates a candidate "game tree", or what may better be called a "candidate game-play branch". If after simulating several branch levels the benefits appear to be too low, such as a high loss of important chess pieces, the bot may conclude it's poor choice, abandon the branch, and instead explore (emulate) different candidate routes.

If the threshold for the SCF (suspicious condition flag) is low such that backtracking is triggered often, it could waste system resources and/or processing time re-checking what didn't need to be re-checked. A low SCF is "over-curious". It's comparable to taking a walk with a toddler: the toddler will stop to explore everything they encounter because everything is new to them. If you let them dictate the pace, you'd never get anywhere.

And if the threshold is too high, we will probably have more poor final matches because we don't explore enough of the tree. A high SCF is "under-curious". One can say the SCF level is an inverse "curiosity level". The best level is, you guessed it, a business and/or engineering decision based on goals and available resources. Time and/or resource limit thresholds may also be set to control how many alternative branches are explored. Again, there are existing tree exploration algorithms in AI literature that use such factors.

Keep in mind the processing tree doesn't have to be made up of *just* FT operations. Different techniques, both AI and traditional, may be mixed and matched in practice. When the FT and AI market matures, you may be able to purchase plug-in modules that do specific things, like normalize size and position of faces so that your group can focus on its specialty, typically on the final step(s) in the tree. Whether the modules use Neural Nets, FT's, Genetic Algorithms, or something else under the hood may not even be knowable by your team (due to say compiled code). As long as they work as advertized, such as normalizing face size and position correctly, you don't have to know or care. It would be a sign that AI "parts" are becoming a commodity.

Trouble-Shooting Poor Matches

So what do we do when even backtracking doesn't help find better matches? Assuming we logged poor matches or non-matches as described earlier, then we can study or recreate the matching process in the lab to understand what when wrong.

For example, we might eventually find that one of our face matching templates has an ear that looks too much like a nose, and thus frontal views are often mistaken for profile views during the early stages of face processing.

How exactly would we track down such bad template(s)? Let's assume we have access to the test case, which is the photo of the person who either got manually reported as poorly matched to the actual person, and/or had a Suspicious Condition Flag (described earlier) along the processing tree.

The first thing to do would be to **check each branch** of the processing tree to make sure it got routed to the correct paths. The log or lab recreation should give us the paths, comparable to the letters shown in Figure T.2. The log may store the path "A-C-D-G-I-J" along with the scores of each process node, for example.

Using our angle-based facial recognition example from earlier, if the tree shows an assumed 35 degree tilt, that is branching on the 35-degree row per Figure I.3 (8th data row down), but the test photo is nearly frontal, let's say 5 degrees, then we know the angle detection (at least) went wrong. We would *also* want to check all the pre-processing steps on the path such as size and centering normalization to make sure they are as expected to verify the test case image didn't get "mis-framed".

Document the result assumptions of these normalization and preparation steps somewhere. For example, if the documentation says that all photos are to be normalized so that nose is centered in the middle of the image and the mouth at 75% the way down from the image, we'd check (or re-run) the problem specimen to make sure it got normalized this way. If the mouth is say 95% the way down, then we know the sizing step went wrong (at least).

Assuming the image normalization steps check out fine and only angle detection went wrong (outside of expectations), the next step is to see which template or templates (factors) put us into the wrong angle row. Let's assume we are using spreadsheet software to inspect the template computations. In a mature system there may already be tools in our "FT kit" to prepare most of the following for us; but since we are the glorious and brave pioneers, we are going to use regular spreadsheets. (A basic familiarity with spreadsheets and spreadsheet formulas is assumed here.)

So we get a copy of the angle match scores for the 35 degree angle row and the 5 degree row for our problem specimen (test). We we want *both* because we want to see why the expected angle row (5 degree) didn't score well and why the wrong one (35 degree) *did* score so well. (It may be good to check adjacent rows also, but we'll skip those to simplify the example.)

With common spreadsheet software, one can even add color coding to each cell and/or plot bar graphs to make it easier to visually spot unexpectedly high and low scores. If you are dealing with hundreds of factors, visual cues can boost productivity over reading raw numbers. But this is not a spreadsheet lesson, so we'll skip such instructions.

If we use matching templates that are fairly easy to relate to, such as albedo and edge-detection image maps, then we can check the actual image templates corresponding to the highest or lowest scores. For example, we would *expect* mostly poor matches for the 35 degree row generated for our problem specimen because it's a low-angle (frontal) image. Thus, select say the best three template matches for the 35 degree row and visually compare the templates to the problem specimen. There's a good chance you will spot the reason for the unexpected "good" match score under the wrong angle, such as a template(s) that has an ear that looks too much like a nose.

We can take the problem template(s) and either remove that factor (template) from our system, blur out or remove the ear detail with a graphics editor, or replace the ear with a more typical looking one using another image. Of course, we'd probably want to run some tests with the new template or removed template to make sure we didn't somehow create new problems.

In some cases the reason(s) for poor matches may may need to be analyzed by an expert in the actual scoring technique used, such as an experienced expert in image processing and image masking algorithms, which is below the level that FT processing is dealing with. More straight-forward scoring algorithms and techniques, such as simple image templates, will typically make it easier for non-specialists to do much of the diagnostics themselves. But more obscure metrics could produce better final matches, at the expense of general staff familiarity. Balancing approachability versus power is yet another business/engineering decision.

Further, in some cases the organization may just agree to live with some poor matches. If somebody's face at profile happens to look a lot like a frontal face out of sheer coincidence, it may not be worth re-tuning everything for such rare cases. (Reminds me of the old joke about somebody having a face that resembles the south end of a north-walking baboon.)

Side Notes: The SQL shown earlier doesn't "keep" the difference scores for each cell (factor). A separate tool or hand calculations would be needed to obtain per-cell values for detailed inspection. A "mature" FT system would probably have built-in gizmos to automate or simplify such reports and related inspections.

Also, in the SQL-based example, low value results meant "high" scores, i.e. a "good match". But the normalization process may reverse this so that good matches have a higher score (close to 1). Arguments can be made on each side for which direction is "good". Perhaps a standard filter (transformation operation) would be available in an FT kit to reverse and/or scale it as needed for different sub-operations.

As far as analyzing image template matches, a mature image processing kit could have tools and guides to assist in identifying why and where specific matches are high or low. For example, "heat map" color coding of an image mask next to a specimen image can show high-match areas of the image and/or template in reddish colors, and low-match areas in blueish colors. (If shift-matching is used, then typically we'd analyze the top-scoring positions.)

Theory and Practice

Neural networks (NN's) have a lot of more-or-less verified math and theory behind them because they have been used in both academia and practice for a relatively long period. FT's have not, at least not as described here. (There may be closely related research topics and techniques that can potentially be reworked a bit and re-applied as FT's.)

But that's not a reason to ignore FT's. I'm reminded of the parable of the man who looks for his missing watch only under the street light because that's the easiest area to see. The missing watch may very well be in some dark area. Sure, check the lighted areas first, but there still may be some gems in the darker or less-explored areas.

NN's themselves have popped in and out of favor over the years. NN's, genetic algorithms, rule-based systems, symbol processing systems, and others fight for and sometimes trade dominance and interest over time. It may be that different tools are better for different projects, and future AI systems will use a combination of them rather than one-size-fits-all.

As described in this document, FT's are assumed to be for tasks where an AI expert(s) has actual domain experience, such as facial recognition, loan approval, marketing, text search, etc. so that the expert can usefully partition the steps into hierarchical teams and sub-systems. Such experts become process and system architects who partition the system so as to leave most of the details and tuning to task-specific teams or specialists. Such AI would more closely resemble traditional system design and management where experienced domain and technology experts supply the forest-level design and partitioning; and then programmers, configuration experts, and power users fill in the tree-level details.

Neural network's may still prove better for new domains, though. I suspect future AI systems and kits will use a combination of techniques, FT's among them. Trace-ability, modularity, and similarity to existing work-place tools and techniques makes Factor Tables handy, and perhaps gives them an edge over the others as AI matures and scales from research labs into an industrial endeavor.