Stage

Octave Jagora

octave.jagora@telecom-paris.fr

July 29, 2023

Contents

1 Tanimoto Index

The Tanimoto index measures similarity between finite sample sets A and B, and is defined as followed [1] :

$$T(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

We use this index over molecular fingerprints represented as binary vectors of size n

Let A and B define two such fingerprints :

$$A = (a_1, a_2, ..., a_n)$$
$$B = (b_1, b_2, ..., b_n)$$

Where a_i and b_i bits are either 0 or 1.

If we introduce the Hamming weight and think in terms of Boolean Algebra we then have the following definition :

$$\boxed{T(A,B) = \frac{w_h(A \cdot B)}{w_h(A + B)}}$$

1.1 GNPS Tanimoto and fingerprints

For the GNPS, the Tanimoto is called the Jaccard Index. Both refers to the same definition [2]. The index is calculated as shown above.
The fingerprints over which the index is calculated are determined via Sirius and CSI:FingerID [3].
Those fingerprints are calculated as follows :

(i) Retrieval of the MS/MS spectrum of the compound

(ii) Computation of the fragmentation tree

(iii) Computation of one-against-all kernels

(iv) Prediction of the fingerprint using SMVs

The fingerprint obtained is a vector of size n containing probabilities values in the interval $[0, 1]$.
Those fingerprints are converted to binary vectors by the GNPS by thresholding the values comparing to 0.5. The Tanimoto is then used on those binary fingerprint vectors.

1.2 Matchms Tanimoto and fingerprints (RDkit)

Matchms computes the Tanimoto index as defined above [4].
Usually when using Matchms, fingerprints are calculated using another library such as RDkit.
RDkit proposes different varieties of fingerprint calculation methods such as :

- Molecular subgraph hash
- Atom pairs
- Topological Torsions
- MACCS keys
- Morgan
- Pharmacophore
- ...

Usually, fingerprints are represented as binary vectors of size $n = 2048$. Morgan fingerprints of radius 2 are often used, but recent literature make references about the All Shortest Path fingerprint method, as being even more efficient in our context. Those two fingerprints are detailed in sections below.

1.3 Implementation Verification

Let us verify our implementation on a simple example. We take

$$a = \text{CCC(=0)N}, \qquad \text{SMILES of propanamide}$$

$$b = \text{C(CC)(O)=O}, \qquad \text{SMILES of propanoic acid}$$

If we calculate the Morgan fingerprint of those two molecules using RDKit, and code it as a 10 bits vector we have :

$$a_{MFP} = (1, 0, 1, 0, 1, 1, 0, 1, 1, 1)$$
$$b_{MFP} = (1, 1, 1, 0, 1, 1, 1, 1, 0, 1)$$

Thus we have

$$w_h(a_{MFP} \cdot b_{MFP}) = 6$$
$$w_h(a_{MFP} + b_{MFP}) = 9$$

And finally

$$T_{MFP}(a,b) = \frac{2}{3}$$

Using our code we find

$$T = 0.6666666666666666$$

Wich is the expected result.

2 Modified Cosine Index

Let A and B be arrays of size n, representing our spectres. We have :

$$A = (x_{a_i}, y_{a_i})_{1 \leq i \leq n} \ , \quad Y_A = (y_{a_i})_{1 \leq i \leq n}$$
$$B = (x_{b_i}, y_{b_i})_{1 \leq i \leq n} \ , \quad Y_B = (y_{a_i})_{1 \leq i \leq n}$$

Where x is the m/z ratio associated with the pick of intensity y.

2.1 Usual Cosine

The Cosine index is originally defined as follows :

$$S_{c_{\alpha,\beta}}(A,B) = \frac{\sum\limits_{i=1}^{n} SP_{A_i} SP_{B_i}}{\sqrt{\sum\limits_{i=1}^{n} {SP_{A_i}}^2}\sqrt{\sum\limits_{i=1}^{n} {SP_{B_i}}^2}}$$

With

$$SP_{X_i} = {x_{X_i}}^{\alpha} \times {y_{X_i}}^{\beta}$$

Where α and β are arbitrary parameters.
We usually choose $\alpha = 0$ and $\beta = 1$, which simplifies the index to the following formula :

$$S_c(A,B) = \frac{\sum\limits_{i=1}^{n} y_{A_i} y_{B_i}}{\sqrt{\sum\limits_{i=1}^{n} {y_{A_i}}^2}\sqrt{\sum\limits_{i=1}^{n} {y_{B_i}}^2}} = \frac{Y_A \cdot Y_B}{\|Y_A\| \, \|Y_B\|} \quad [1]$$

[1]We find the expression of the cosine of the angle between Y_A and Y_B

2.2 Modified Cosine

The modified cosine [5] allows for a less aligned way of comparing two spectres, by enabling the comparison of peaks not directly corresponding, within a certain tolerance margin.[2] An algorithm is used to find the best pairs of matching peaks within the tolerance margin. Let us define S, the set of index pairs of those matching peaks. We have $S \subset [\![1, n]\!] \times [\![1, n]\!]$. S is injective on each index of its pairs.
We therefore have the following definition of the modified cosine :

$$\boxed{S_{mc}(A, B) = \frac{\sum\limits_{(i,j)\in S} y_{A_i} y_{B_j}}{\sqrt{\sum\limits_{i=1}^{n} y_{A_i}^2}\sqrt{\sum\limits_{i=1}^{n} y_{B_i}^2}} \qquad \text{for } \alpha = 0 \text{ and } \beta = 1}$$

2.2.1 Matchms Modified Cosine

The Matchms library offers the possibility to choose α, β and t.
By default, those values are set respectively to 0, 1 and 0.1

2.2.2 GNPS Modified Cosine

By analyzing the GNPS code we find that :

$$S_{mc}(A, B) = \frac{\sum\limits_{(i,j)\in S} \sqrt{y_{A_i} y_{B_j}}}{\sqrt{\sum\limits_{i=1}^{n} y_{A_i}}\sqrt{\sum\limits_{i=1}^{n} y_{B_i}}}$$

GNPS Modified Cosine therefore uses $\alpha = 0$ and $\beta = 0.5$

According to [5], the square root transformation of the intensities can be applied to address the skewness and variance in the mass spectra data. The transformation reduces the influence of high intensity peaks and increases the influence of low intensity ones.

[2]it is important to note that the Modified Cosine is not obtained by a simple translation of the entire spectrum. Each individual peak is compared with another one in a margin defined by a tolerance factor t

2.3 Determination of S

Let M_A and M_B be the mass of our precursors corresponding to spectres A and B [3], and z_A and z_B be their charge. We can define $mz_A = \frac{M_A}{z_A}$ and $mz_B = \frac{M_B}{z_B}$
We have $\Delta = |M_A - M_B|$
Let t be a tolerance factor chose by the operator for discriminating the best matching pairs of peaks.
The best set S of matching peaks is determined by a 3 step algorithm that operates as follows :

(i) Mark $S_{i,j} = (A_i, B_j)$ as a matching pair if :

$$|xA_i + mz_{shift} - x_{B_j}| \leq t \quad (1)$$

With :

1. $mz_{shift} = 0$
2. $mz_{shift} = \Delta$

is verified.

(ii) Calculate the match score M associated to $S_{i,j}$:

$$M(S_{i,j}) = \frac{y_{A_i} \times y_{B_j}}{\|Y_A\| \, \|Y_B\|} \quad \text{, for } \alpha = 0 \text{ and } \beta = 1$$

(iii) To find S, we need to solve the bipartite matching problem of selecting the highest scoring subset of matching peaks $S_{i,j}$, considering that each peak can **at most** be associated with **one** other peak. (S can be seen as a matrix of $S_{i,j}$ with only one element in each row and each column, such that the sum of its element's match scores is maximized).
Both GNPS and Matchms use a Greedy Bipartite Matching algorithm to find S.

With the definition of the match score, the value of the Modified Cosine is simply the sum of the match scores of the elements of S [4].

$$S_{mc} = \sum_{(i,j)\in S} M(S_{i,j})$$

It is the value we want to maximize.

[3] A and B are defined as above
[4] Also true when $\alpha \neq 0$ and $\beta \neq 1$

2.3.1 Matchms determination of S

Matchms tests conditions (1) for both values of mz_{shift}

2.3.2 GNPS determination of S

- The GNPS always tests condition (1) for $mz_{shift} = 0$ but only test with $mz_{shift} = \Delta$ if $t < |\Delta|$
  Also, t is adjusted by a constant and thus becomes $t' = t + 0.000001$

- The GNPS also tests condition (1) with another definition for mz_{shift}

  $$mz_{shift} = |mz_A - \frac{z_B}{z} mz_B| \quad z \in [\![2, z_A]\!]$$

  Therefore GNPS tries (1) with

  1. $mz_{shift} = 0$
  2. $mz_{shift} = \Delta$
  3. $\boxed{mz_{shift} = \frac{\Delta}{z} \quad \textbf{for } z \in [\![2, z_A]\!]}$

  It is impossible to find any reference in literature of the third definition of mz_{shift}. Moreover, there are parts of code in GNPS that do not use that last definition. But it is difficult to see which version of the code is used depending on the utilisation of GNPS. This intriguing discovery calls for more research, to determine where and why this third definition is used[5]. One could understand that the third definition is logical when $z_A \neq z_B$, because a difference in charge of the two precursors implies a difference in mass and in the mz values. But this third definition is not implemented in another library. Strangely enough, the authors of GNPS are also the authors of the publication directly cited in the code of matchms.

  Git paths to GNPS definition 'spectrum_alignment' function :

  - GNPS modified cosine with third condition :
    - *GNPS_Workflows/msms-chooser /tools/msms-chooser/spectrum_alignment.py*
    - *GNPS_Workflows/metabolomics-snets /tools/metabolomicsnets/scripts/spectrum_alignment.py*

[5]The calculations take place in the score_alignment function defined in the spectrum_alignment python files

    - *GNPS_Workflows/metabolomics-snets-v2 /tools/metabolomicsnetsv2/scripts/spectrum_alignment.py*
    - *GNPS_Workflows/search_single_spectrum /tools/search_single_spectrum /spectrum_alignment.py*
    - *GNPS_Workflows/feature-based-molecular-networking /tools/feature-based-molecular-networking /scripts/spectrum_alignment.py*
  - GNPS modified cosine without third condition :
    - *GNPS_Workflows/ms2lda_motifdb /tools/ms2lda_motifdb/lda/code/spectrum_alignment.py*
    - *GNPS_Workflows/molecular-librarysearch-gc /tools/molecularsearch-gc/spectrum_alignment.py*

- Also, the Match score is defined as follows (because $\alpha = 0$ and $\beta = 0.5$) :

$$M(S_{i,j}) = \frac{\sqrt{y_{A_i} y_{B_j}}}{\sqrt{\sum_{i=1}^{n} y_{A_i}} \sqrt{\sum_{i=1}^{n} y_{B_i}}}$$

2.3.3 Implementation of the Greedy Algorithm

The principle of the greedy algorithm is to make locally optimal choices at each step in the hope of finding a global optimum. It follows a step-by-step approach, making the best choice available at each iteration without considering the potential consequences in future steps. The greedy algorithm aims to build a solution incrementally by selecting the most favorable option at each decision point.

The algorithm iterates through the list of matching pairs, which is sorted by their match score. The algorithm aims to find the maximum score by greedily selecting pairs with the highest match score and ensuring that each peak is paired only once.

The algorithm used by both GNPS and Matchms follows these steps:

(i) Iterate through the list of matching pairs in the order they are sorted (from highest to lowest match score).

(ii) For each pair, check if both peaks in the pair have not been used before. If they have not been used, it means they can be associated together.

(iii) If the peaks are available, update the score by adding the match score of the current pair to the total score.

(iv) Mark the peaks as used

(v) Increment a counter to keep track of the number of matching pairs used

(vi) Continue iterating through the list until all pairs have been considered

By making locally optimal choices (i.e., selecting pairs with the highest match score that haven't been used before), the algorithm aims to find a solution that maximizes the score. However, it's important to note that this greedy approach might not always yield the globally optimal solution for all problem instances. But the difference in results is marginal, especially with low values of t.

Matchms also proposes another approach, using the Hungarian Algorithm, which is more precise but is more time consuming.

2.3.3.1 Example of GNPS and Matchms code for determining S :

We do not redefine the different variables used. This is simply an illustration to show that both programs work in the same way.

```
// GNPS Greedy algorithm to find S
for match in all_possible_match_scores:
        if not match.peak1 in spec1_peak_used and
          not match.peak2 in spec2_peak_used:
            spec1_peak_used.add(match.peak1)
            spec2_peak_used.add(match.peak2)
            reported_alignments.append(
              Alignment(match.peak1,match.peak2))
            total_score += match.score
```

```
// Matchms Greedy algorithm to find S
for i in range(matching_pairs.shape[0]):
        if not matching_pairs[i, 0] in used1 and
          not matching_pairs[i, 1] in used2:
            used1.add(matching_pairs[i, 0])
            used2.add(matching_pairs[i, 1])
            used_matches += 1
            score += matching_pairs[i, 2]
```

2.4 Implementation Verification

Let us verify our implementation of matchms on a simple example. We take

$$A = ((100, 0.7), (150, 0.2), (200, 0.1)), \quad mz_A = 100$$
$$B = ((104.9, 0.4), (140, 0.2), (190, 0.1)), \quad mz_B = 105$$

Therefore $M = |mz_A - mz_B| = 5$
And we choose a tolerance of $t = 0.2$
We search for matching pairs by trying condition (1) :

- $|100 - 104.9| = 4.9 > t$
- $|150 - 140| = 10 > t$
- $|200 - 190| = 10 > t$

Condition (2) :

- $|100 + 5 - 104.9| = 0.1 < t$
- $|150 + 5 - 140| = 10 > t$
- $|200 + 5 - 190| = 10 > t$

Therefore $S_{1,1}$ is a matching pair and

$$M(S_{1,1}) = \frac{0.7 \times 0.4}{\sqrt{0.7^2 + 0.2^2 + 0.1^2} \times \sqrt{0.4^2 + 0.2^2 + 0.1^2}} = \frac{0.28}{\sqrt{0.1134}} \approx 0.83$$

Matchms gives us a score of 0.83 with a single matching pair, as expected.

3 Morgan Fingerprint

3.1 Definition

Morgan fingerprints are circular fingerprints. They are also known as ECFPs [6] [6].

The ECFP generation process can be summarized into three main sequential stages :

1. Initial Assignment Stage : Every atom is assigned an integer identifier
2. Iterative Updating Stage : Each atom's integer identifier is updated in relation to its neighbors identifiers
3. Duplicate Identifier Removal Stage : Multiple occurrences of features are reduced to a single one

The list of features obtained is then hashed to a binary vector of arbitrary size[7]. The hash function used has no importance according to the authors. This last point could appear as unexpected, and may require some investigation. The point of a Hash function is to maximize distance of hash value for two words that are close to each other, whereas the fingerprints are constructed for the exact opposite purpose : represent molecules in ways that we can find similarity between them (if two molecules are similar, we want their bitvector to be close such that the Tanimoto coefficient of the two is high).

3.2 Python Implementation

The Morgan Fingerprint is directly implemented in the RDKit package

```
//RDKit Morgan fingerprint calculation method

  rdkit.Chem.rdFingerprintGenerator.GetMorganGenerator(radius,
      countSimulation=False, includeChirality=False,
      countBounds=None, fpSize=2048, atomInvariantsGenerator=None)

    Get a morgan fingerprint generator

        ARGUMENTS:
```

[6]the two methods are not exactly the same. However, they are both based on Morgan's algorithm. Details of the differences are given in the publication cited.

[7]The vector still needs to be long enough to translate enough structural features

```
            radius: the number of iterations to grow the
               fingerprint

            countSimulation: if set, use count simulation while
               generating the fingerprint

            includeChirality: if set, chirality information will
               be added to the generated fingerprint

            useBondTypes: if set, bond types will be included as
               a part of the default bond invariants

            countBounds: boundaries for count simulation,
               corresponding bit will be set if the count is
               higher than the number provided for that spot

            fpSize: size of the generated fingerprint, does not
               affect the sparse versions

            atomInvariantsGenerator: atom invariants to be used
               during fingerprint generation

   This generator supports the following AdditionalOutput types:

            atomToBits: which bits each atom is the center of

            atomCounts: how many bits each atom sets

            bitInfoMap: map from bitId to (atomId1, radius) pairs

      RETURNS: FingerprintGenerator
```

3.2.0.1 Distribution We can graph the histogram of the distribution of the values of the Tanimoto for pairs of molecules represented by their fingerprint. Here we used a set of 5000 fungus origin molecules from the NPA [7] to plot the histogram.

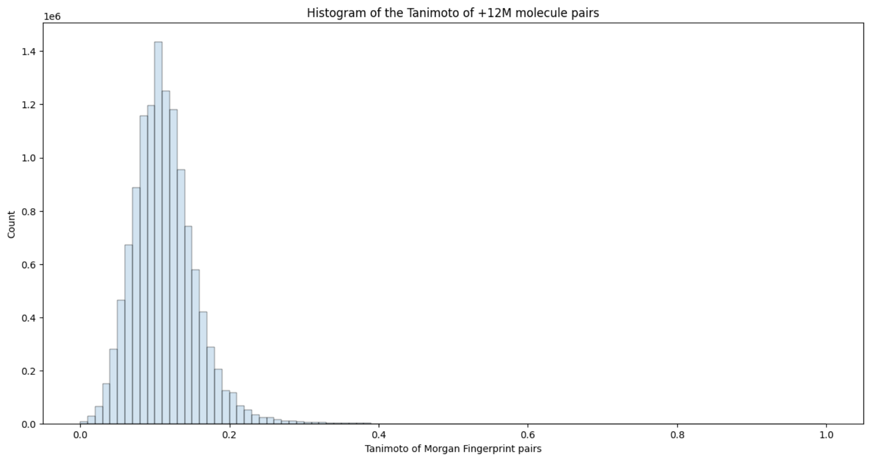


Figure 1: Morgan Fingerprint Tanimoto's distribution - linear scale

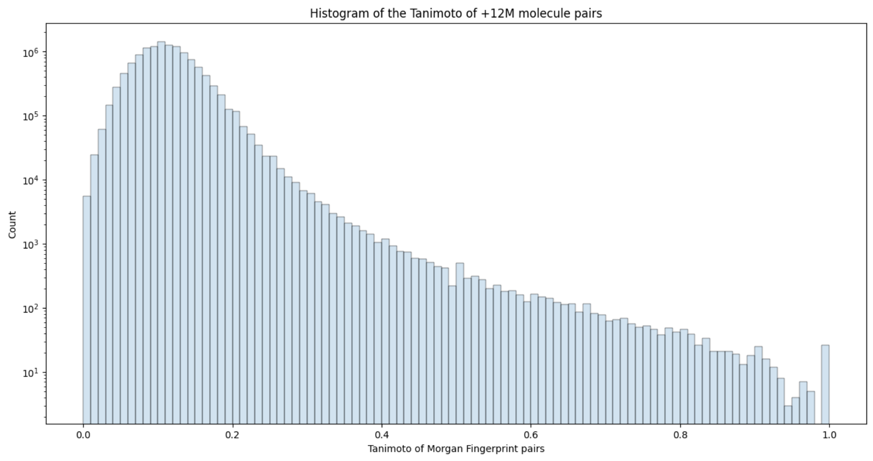


Figure 2: Morgan Fingerprint Tanimoto's distribution logarithmic scale

We can see that the Morgan Fingerprints almost behaves as a normal distribution

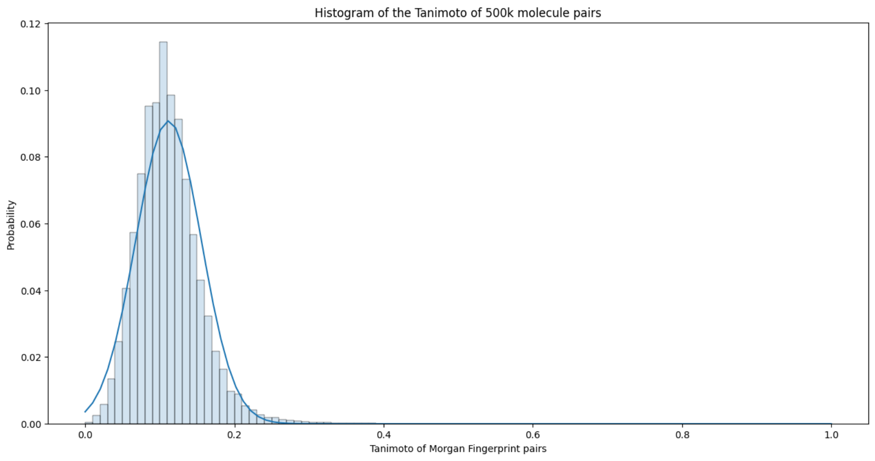


Figure 3: Morgan Fingerprint Tanimoto's distribution compared to a scaled Normal distribution ($\sigma = 0.04,\ \mu = 0.11$)

3.3 Similarity Maps

Similarity Maps is a tool used to visualize molecular fingerprints methods [8]. Similarity Maps are constructed by calculating the "weight" of each atom in the molecule. The weight of an atom is the similarity obtained when the bits in the fingerprint corresponding to the atom are removed. Those weights are used to color the atoms in the molecule. A **green color indicates a positive difference** in similarity, and **pink a negative difference**.

3.3.1 Python Implementation

We simply use the built in RDKit tool to generate the similarity map for our Morgan fingerprints :

```
fig, maxweight =
    SimilarityMaps.GetSimilarityMapForFingerprint(refmol,
                mol, lambda m,idx:
                    SimilarityMaps.GetMorganFingerprint(m,
                    atomId=idx, radius=2),
                metric=DataStructs.TanimotoSimilarity)
```

3.3.2 Examples

3.3.2.1 Example 1

We have $T_{MG}(a, b) = 0.16$

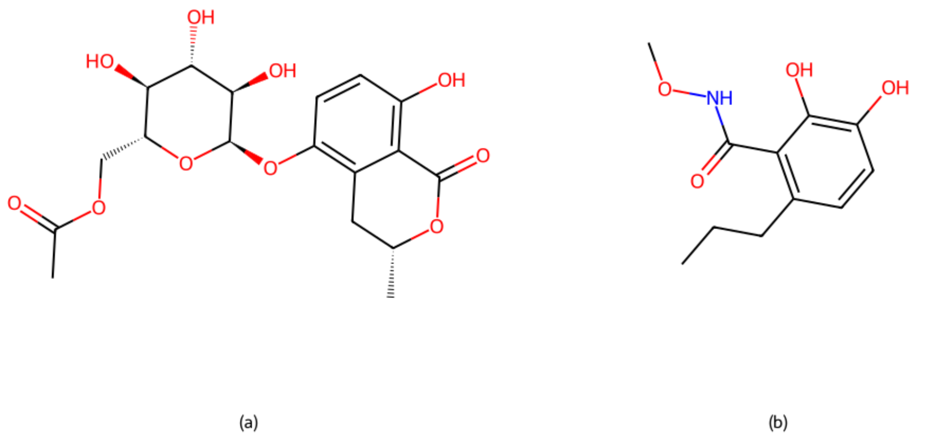
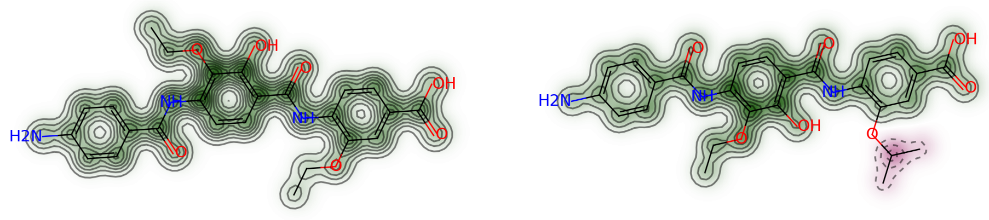


Figure 4: Example 1 - Molecules



(a) Similarity Map of (a) with (b)

(b) Similarity Map of (b) with (a)

Figure 5: Example 1 - Similarity Maps

3.3.2.2 Example 2

We have $T_{MG}(a, b) = 0.36$

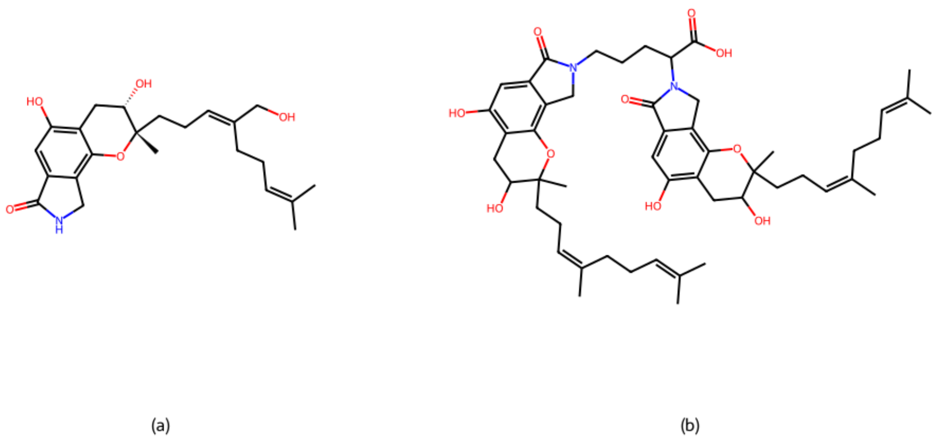
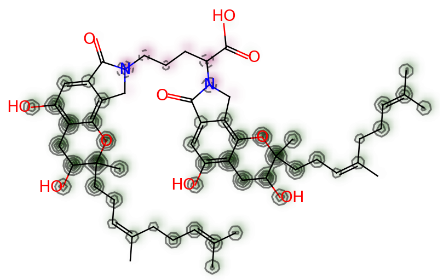
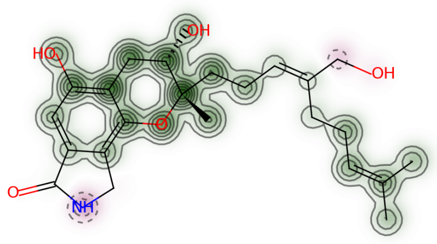


Figure 6: Example 2 - Molecules



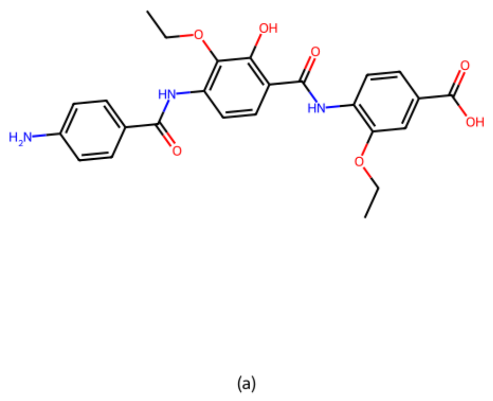
(a) Similarity Map of (a) with (b)



(b) Similarity Map of (b) with (a)

Figure 7: Example 2 - Similarity Maps

3.3.2.3 Example 3 We have $T_{MG}(a, b) = 0.83$



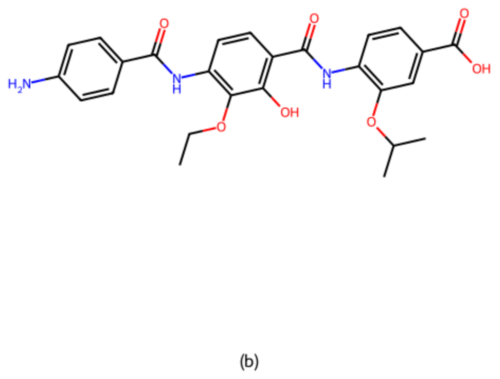
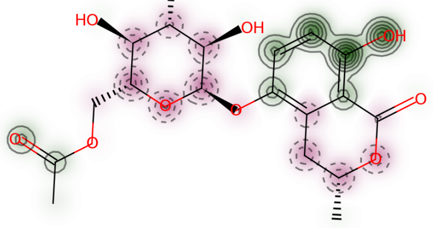
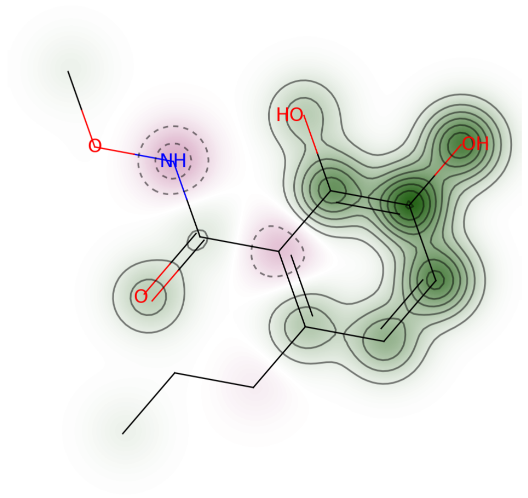


Figure 8: Example 3 - Molecules



(a) Similarity Map of (a) with (b)



(b) Similarity Map of (b) with (a)

Figure 9: Example 3 - Similarity Maps

4 All Shortest Paths - ASP Fingerprint

Morgan fingerprints were introduced in 1965, and have since been used everywhere by everyone. But recent articles suggest that ASP fingerprints may be more relevant than Morgan fingerprints in many applications of Cheminformatics [9]. Knowing that the Tanimoto coefficient is very sensible to the type of fingerprint used, therefore, it can be interesting to investigate on whereas to use Morgan or ASP fingerprints.

4.1 Mathematical definition

The ASP [10] is calculated by first obtaining the shortest path between every atom pair in a molecule C, hence the appellation of the fingerprint. Those shortest path are considered as features and are obtained with

$$F(C) = \bigcup_{i}^{n} \{DFS(\hat{a}_i, d),\ |p_{ij}| = t_{ij}\}$$

Where

- $\hat{a}_i$ is the label of an atom in C.
- d is the depth for topological patterns (i.e the maximum number of bonds in a feature)
- p_{ij} is the path, and therefore $|p_{ij}|$ is its length
- t_{ij} is the length of the shortest path possible between atoms in position i and j
- DFS refers to the Depth First Search algorithm. It returns two alphanumerical strings representing the original and the reversed path. We only keep the string of higher lexicographical order.

We then generate binary vectors of size n by hashing those features with a hash function H [8] :

$$H \circ F(C) \to \mathbb{Z}_2^n$$

[8]For example, we can use the standard SHA-256 hash function

4.2 Python Implementation

```
def getASPfinger(mol, atom = -1, n = 1024) :
#calculates mol fingerprint without atom of index atom and in a
    bit vector of size n
paths = []
num_atoms = mol.GetNumAtoms()
for i in range(num_atoms):
    for j in range(i + 1, num_atoms):
        path = Chem.rdmolops.GetShortestPath(mol, i, j)
        #get shortest path between atom of index i and j
        if path and not(atom in list(path)):
            atom_indices = [i] + list(path) + [j]
            path_atoms = [mol.GetAtomWithIdx(idx) for idx in
                atom_indices]
            path_smiles = Chem.MolFragmentToSmiles(mol,
                atom_indices, atomSymbols=None, bondSymbols=None)
            path_smiles_bis = path_smiles[::-1] #backward string
            paths.append(max(path_smiles_bis,path_smiles))

binary_vector = [0] * n

for path in paths:
    path_hash = hashlib.sha256(path.encode()).hexdigest() #
        Hashing the path using SHA-256
    path_index = int(path_hash, 16) % n # Convert hash to integer
        and modulo by n to get the index
    binary_vector[path_index] = 1

return CreateFromBitString("".join(str(bit) for bit in
    binary_vector))
```

4.2.0.1 Distribution

We can graph the histogram of the distribution of the values of the Tanimoto for pairs of molecules represented by their fingerprint. Here we used a set of 1000 fungus origin molecules from the NPA to plot the histogram.

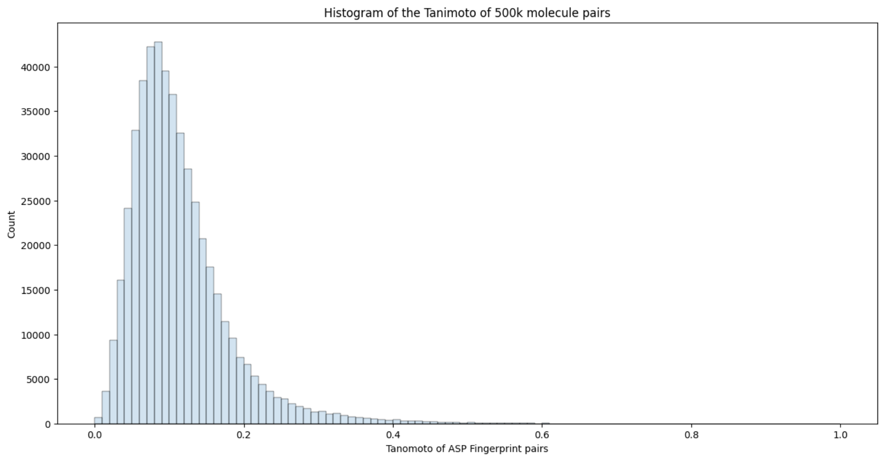


Figure 10: ASP Fingerprint Tanimoto's distribution - linear scale

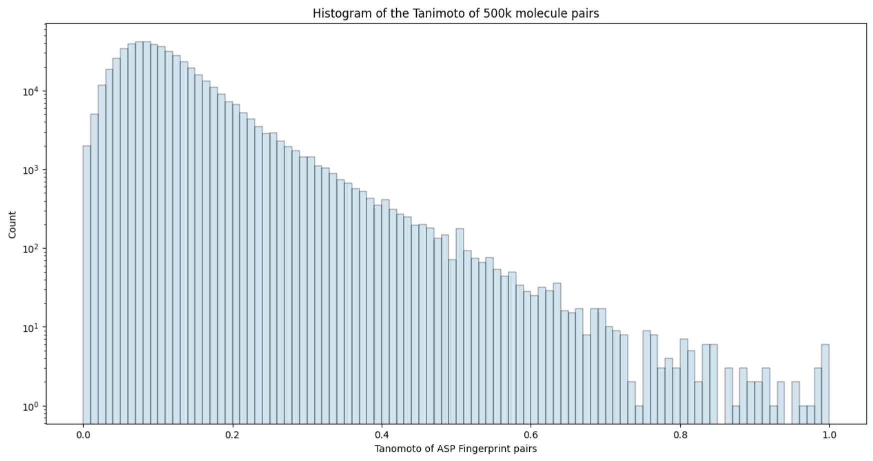


Figure 11: ASP Fingerprint Tanimoto's distribution - logarithmic scale

We can see that the ASP Fingerprints also resembles a normal distribution

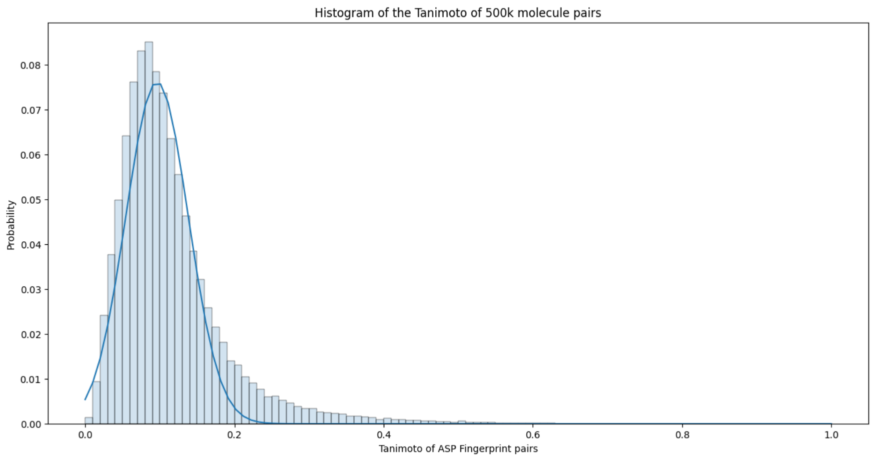


Figure 12: ASP Fingerprint Tanimoto's distribution compared to a scaled Normal distribution ($\sigma = 0.04,\ \mu = 0.10$)

4.3 Similarity Maps

4.3.1 Python Implementation

We simply use the RDKit GetSimilarityMapForFingerprint function, specifying our fingerprint calculation method

```
    fig, maxweight =
        SimilarityMaps.GetSimilarityMapForFingerprint(refmol,
                    mol, lambda m,idx: getASPfinger(m, atom=idx),
                    metric=DataStructs.TanimotoSimilarity)
```

4.3.2 Examples

4.3.2.1 Example 1

We have $T_{ASP}(a, b) = 0.17$

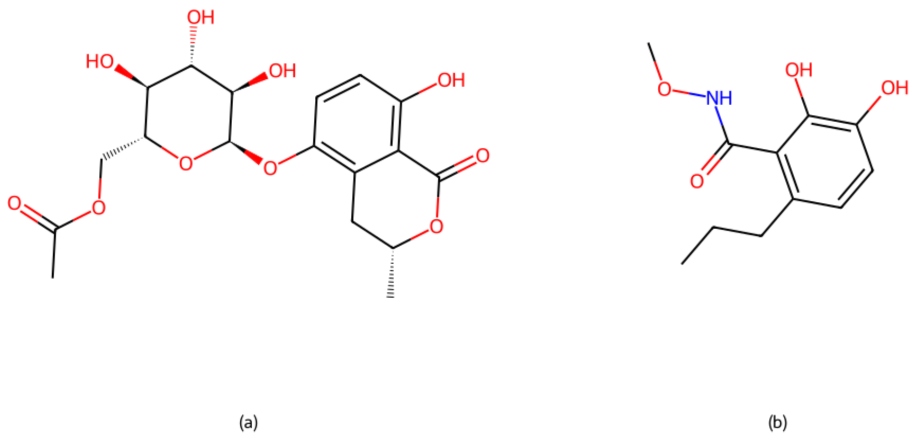
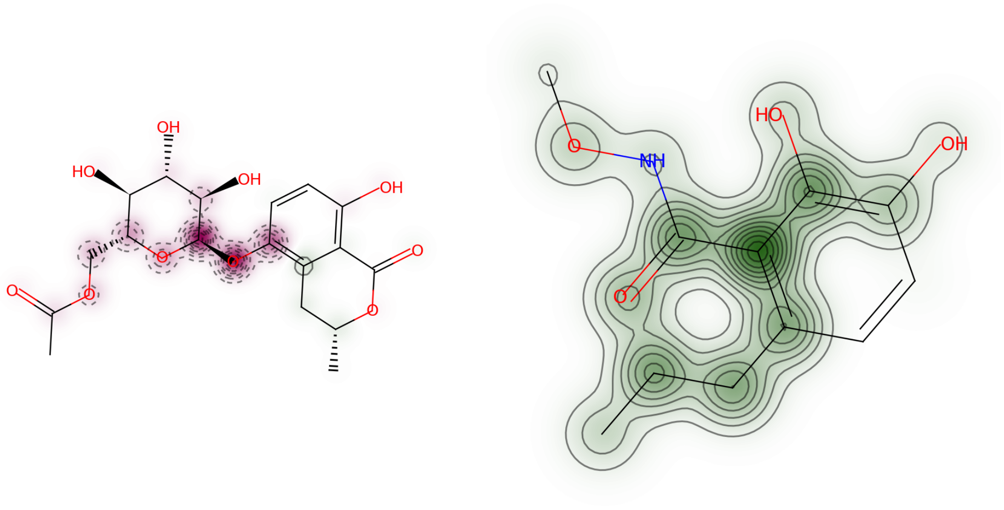


Figure 13: Example 1 - Molecules



(a) Similarity Map of (a) with (b) (b) Similarity Map of (b) with (a)

Figure 14: Example 1 - Similarity Maps

4.3.2.2 Example 2

We have $T_{MG}(a, b) = 0.27$

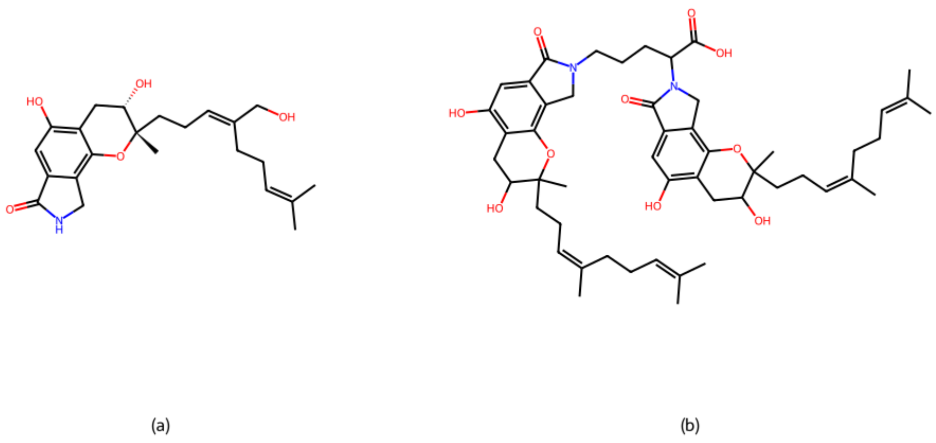
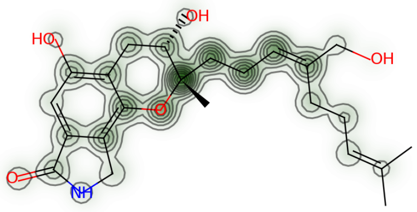
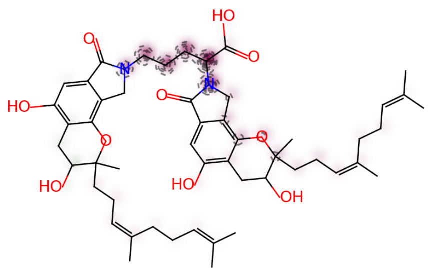


Figure 15: Example 2 - Molecules



(a) Similarity Map of (a) with (b)



(b) Similarity Map of (b) with (a)

Figure 16: Example 2 - Similarity Maps

4.3.2.3 Example 3

We have $T_{MG}(a, b) = 0.99$

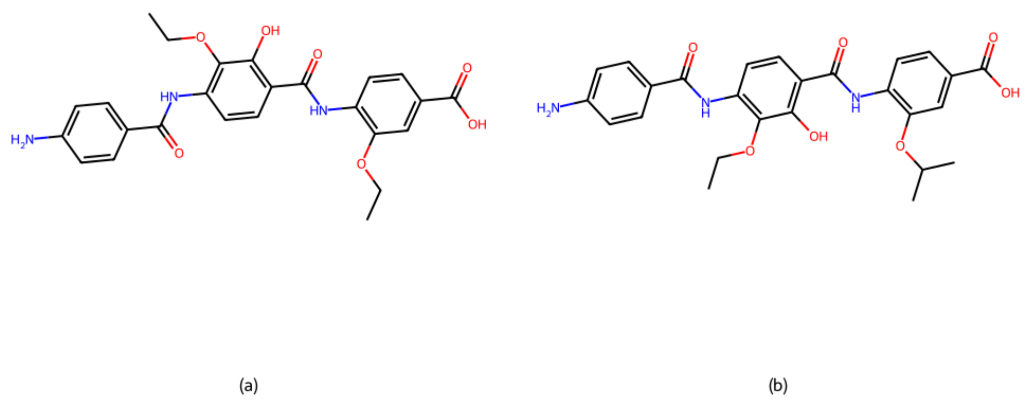
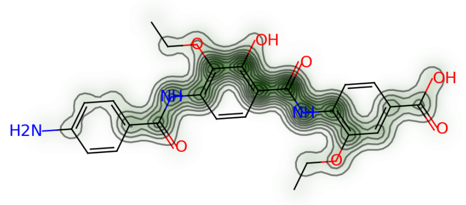
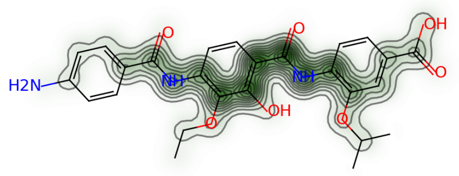


Figure 17: Example 3 - Molecules



(a) Similarity Map of (a) with (b)



(b) Similarity Map of (b) with (a)

Figure 18: Example 3 - Similarity Maps

5 Comparison of fingerprints

Despite having the same behavior[9], ASP and Morgan fingerprint do not always lead to the same Tanimoto values.

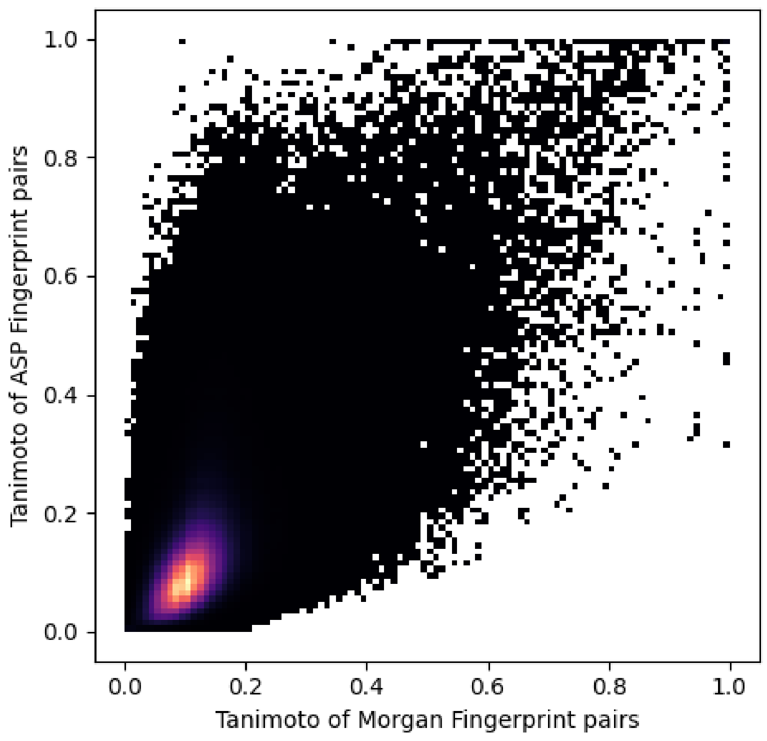


Figure 19: 2D-Histogram of Morgan Fingerprint Tanimoto's vs ASP Fingerprint Tanimoto's for 5000 molecules from the NPA

We have a Pearson's coefficient of $r \approx 0.66$ [10]

6 Sensibility analysis of the g function

In this section we will try to quantify the sensibility of the function g regarding the four weights it depends on. The weights are the following :

$$(w_1, w_2, w_3) \in [0,1]^3 \quad \text{with} \quad \sum_i w_i = 1$$

[9]Similar distributions

[10]This was calculated for 1000 molecules because of computation time

Representing the weight of the aggregating function, and

$$w^t \in [0,1]$$

the ratio weight between the Tanimoto and the Cosine index.

Let $w = (w_1, w_2, w_3, w^t)$
we can define

$$\zeta_T = w^t c_T + (1 - w^t) a_T$$

where c_T is the cosine given by tool T and a_T is the average Tanimoto.
We thus have

$$g(w,\zeta) = OWA(w,\zeta) = \sum_i w_i \zeta_i$$

where ζ_j is the $j^t h$ largest element of all ζ_T

6.1 Variance-based sensitivity analysis

To determine the sensibility, we can do a variance-based sensitivity analysis by describing our different weights and variables as probability distributions [11] [12]:

6.1.1 Weights inputs construction

We can construct a multivariate random variable to simulate our weight vector (w_1, w_2, w_3) with

$$X_1 \sim \mathcal{U}([0,1])$$
$$X_2 \sim \mathcal{U}([0,1])$$
$$X_3 \sim \mathcal{U}([0,1])$$
$$X_4 \sim \mathcal{U}([0,1])$$

With $\hat{X}_1 = \frac{X_1}{\sum_i^3 X_i}$, $\hat{X}_2 = \frac{X_1}{\sum_i^3 X_i}$, $\hat{X}_3 = \frac{X_1}{\sum_i^3 X_i}$ we have $\sum_i^3 \hat{X}_i = 1$

We therefore have the weight input vector :

$$(\hat{X}_1, \hat{X}_2, \hat{X}_3, X_4)$$

6.1.2 Variables inputs construction

We also need to describe our other inputs : values of the different indexes calculated by the different tools. For this purpose, we analyse the distribution of indexes for each tool and we get :

- For GNPS : $X_5 \sim \mathcal{N}(0.1, 0.05) + \mathcal{N}(0.56, 0.05)$ and $X_6 \sim \mathcal{U}([0, 1])$
- For Sirius : $X_7 \sim \mathcal{N}(0.11, 0.05) + \mathcal{N}(0.56, 0.05)$ and $X_8 \sim \mathcal{U}([0, 1])$
- For ISDB : $X_9 \sim \mathcal{N}(0.1, 0.05)$ and $X_{10} \sim \mathcal{N}(0.55, 0.1)$

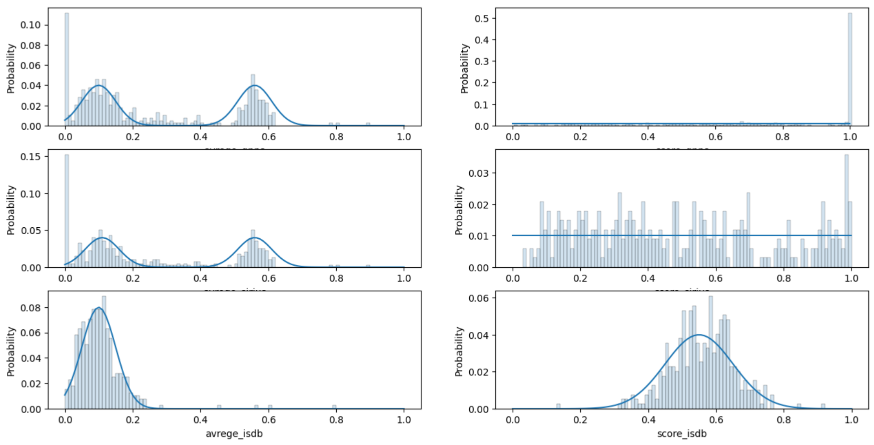


Figure 20: Distribution histogram of score and average value from our database

The values of the indexes given by the different tools differs but still are correlated. We need to simulate the values given by a tool taking into account each value given by the other tools.
For this, we can uniformly choose a random tool for each sample, that will act as a guiding distribution. We then simulate the other variables by a normal distribution centered on the value.

Let X_0^i and X_1^i be the Cosine and Tanimoto coefficient of tool i. i is chosen randomly and uniformly, and X_0^i and X_1^i are calculated accordingly to the distribution indicated above. We can then simulate the other coefficient with :

$$\widetilde{X_j^{i'}} \sim \mathcal{N}(X_j^i(w), Var(X_j^{i'}, X_j^i))$$

We can determine the different variance by analysing our data :

- $Var(X^{GNPS}_{Tanimoto}, X^{SIRIUS}_{tanimoto}) = 0.021$
- $Var(X^{GNPS}_{Tanimoto}, X^{ISDB}_{tanimoto}) = 0.16$
- $Var(X^{ISDB}_{Tanimoto}, X^{SIRIUS}_{tanimoto}) = 0.163$
- $Var(X^{GNPS}_{Cosine}, X^{SIRIUS}_{Cosine}) = 0.323$
- $Var(X^{GNPS}_{Cosine}, X^{ISDB}_{Cosine}) = 0.331$
- $Var(X^{ISDB}_{Cosine}, X^{SIRIUS}_{Cosine}) = 0.253$

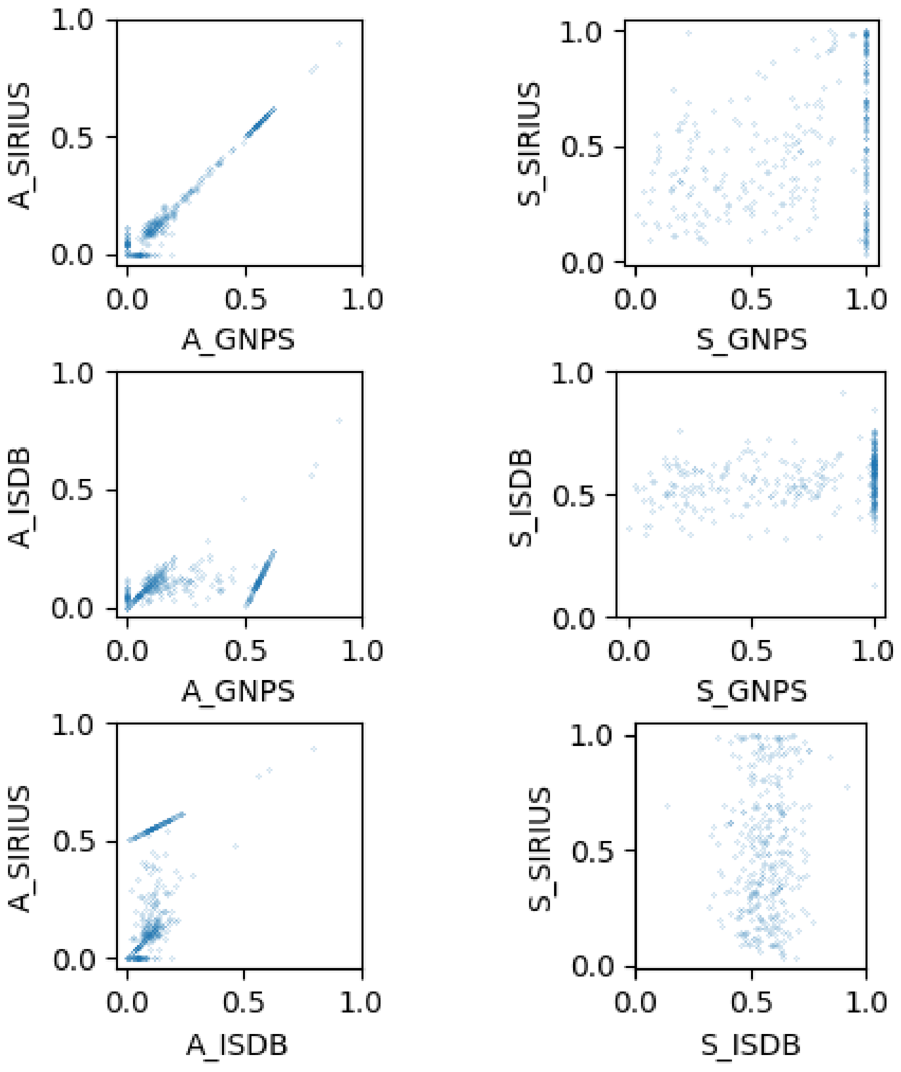


Figure 21: Distribution of X^i_j in our database

We obtain the following distributions for our simulated variable inputs :

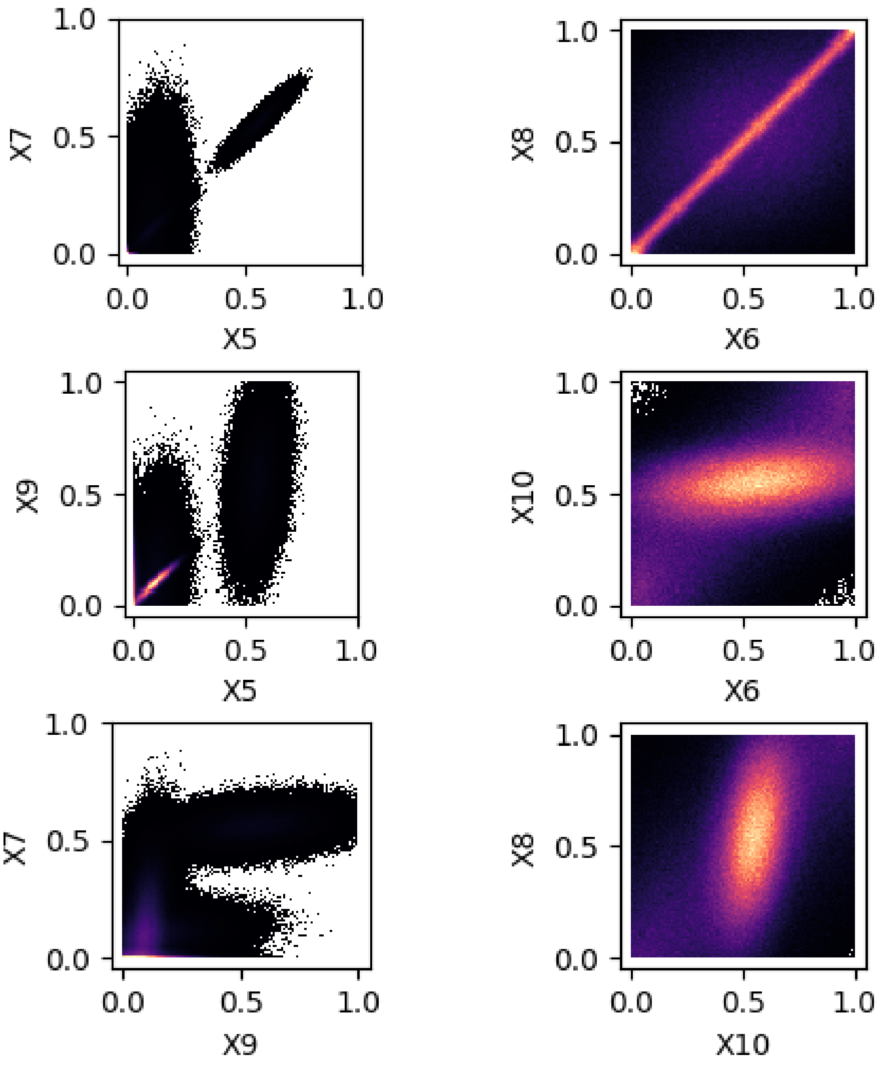


Figure 22: Distribution variable inputs for 1 million samples

6.1.3 Monte Carlo Simulation

Our total input vector is therefore :

$$X = (\hat{X}_1, \hat{X}_2, \hat{X}_3, X_4, \widetilde{X}_5, \widetilde{X}_6, \widetilde{X}_7, \widetilde{X}_8, \widetilde{X}_9, \widetilde{X}_{10})$$

With

$$Y = g(X)$$

we can calculate the first-order sensitivity index :

$$S_i = \frac{V_i}{\sum\limits_i V_i}$$

where

$$V_i = Var_{X_i}(E_{X_{\sim i}}(Y|X_i))$$

For those calculations we use the (quasi) Monte Carlo approach. The following estimator is used to calculate V_i :

$$V_i = \frac{1}{N}\sum_{j=1}^{N} f(B)_j(f(A_B^i)_j - f(A)_j)$$

where A is a sample matrix of our inputs of size $N \times k$, with N the number of samples and k the number of inputs. B is constructed in the same way. A_B^i is a duplicate of A with its i^{th} column swapped with B i^{th} column.

6.1.4 Results

If the value of g is not very sensible to the values of the weights, we expect S_i associated to weights to be consistently lower than for the other inputs.

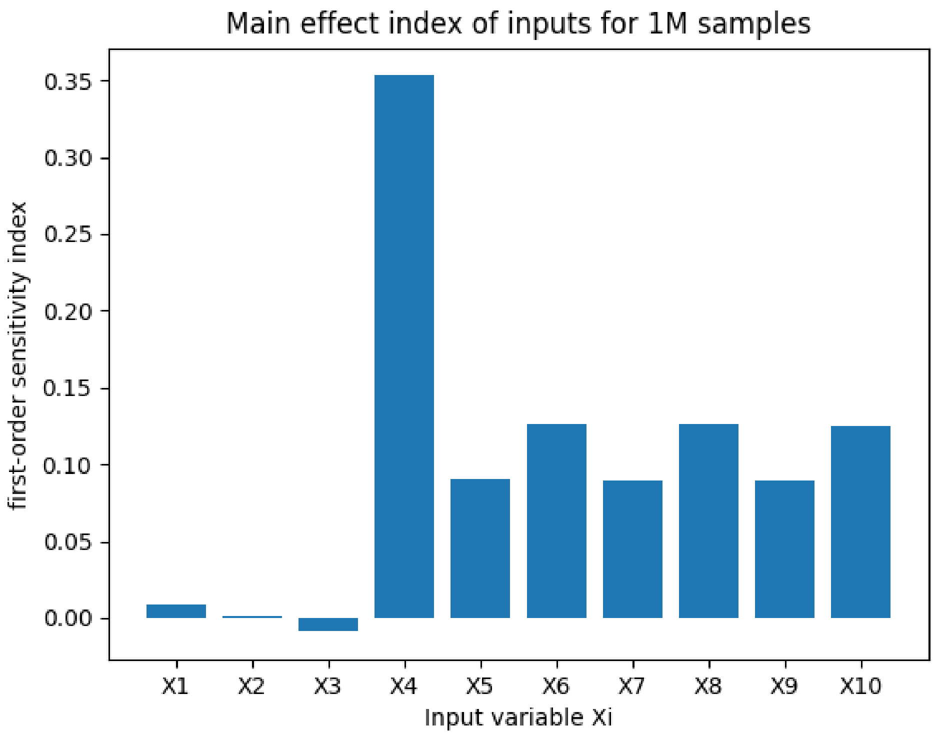


Figure 23: Barplot of estimated values of S_i for 1 million samples

We can see that the weights (w_1, w_2, w_3) do not seem to have a noticeable impact on the total variance of the g function. However, the w^t seems to play the most important role in the variance of g. Values for (w_1, w_2, w_3) can be arbitrarily chosen, but specific attention must be given to the w^t weight. For this purpose, we will analyse in a following section the real data we're using.[11]

6.2 Database analysis

6.2.1 Analysis of the impact of the ratio weight

Let us fix $(w_1, w_2, w_3) = (0.7, 0.2, 0.1)$
We can plot the values of g for our dataset, with values of w_t ranging from 0 to 1. Values of g are sorted such as g evaluated for $w^t = 0.5$ is decreasing.

[11]Moreover, those sobol indices are quite difficult to interpret, and the calculation methods we used is certainly too weak to make conclusions at this point. The analysis of our data is therefore the best option

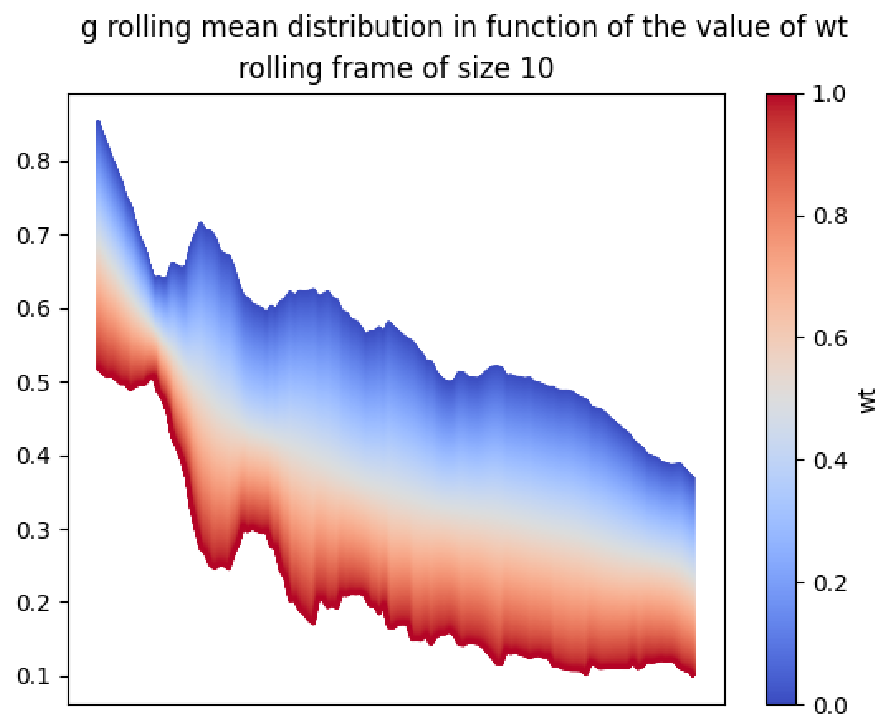


Figure 24: Distribution of g for varying w^t

We can see that 0.5 is the best value for the w_t coefficient if we want values of g to be evenly spread, facilitating the comparison of value and maximizing the information contained by g.
We introduce an index δ to measure the disparity of our values. δ is the surface of the envelope of our distribution and i calculated as follows :

$$\delta_w = \frac{1}{n}\sum_{i=0}^{n}(g_{w=0}(x_i) - g_{w=1}(x_i))$$

We have $\delta_{w^t} = 0.34$

6.2.2 Analysis of the impact of the OWA weights

If we fix w^t to 0.5, we can visualize the distribution of g values for one of the w_i fixed to a value, and the other $w_{j\neq i}$ verifying $\sum_i^3 w_i = 1$

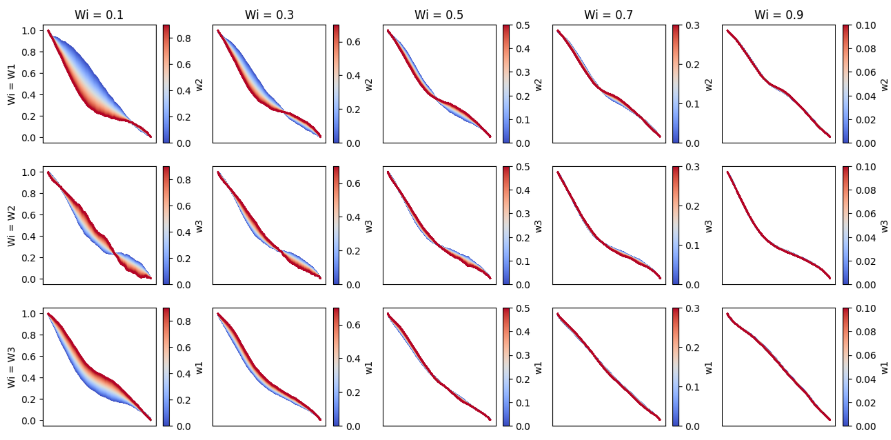


Figure 25: Distribution of g for different fixed values for each w_i

We can also plot heatmaps of δ values when two parameters are fixed. We get :

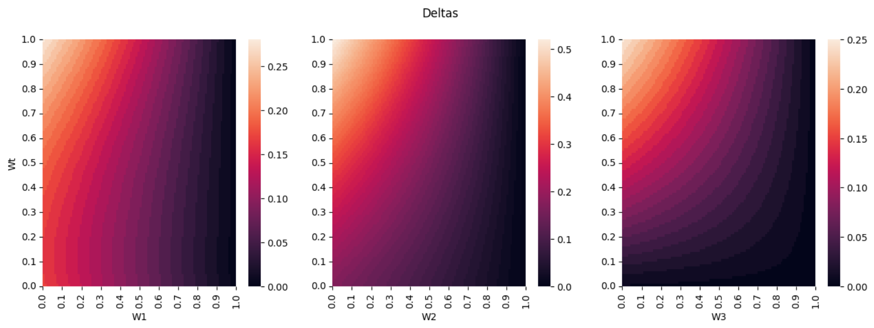


Figure 26: Heatmaps of δ values for each w_i and w_t fixed

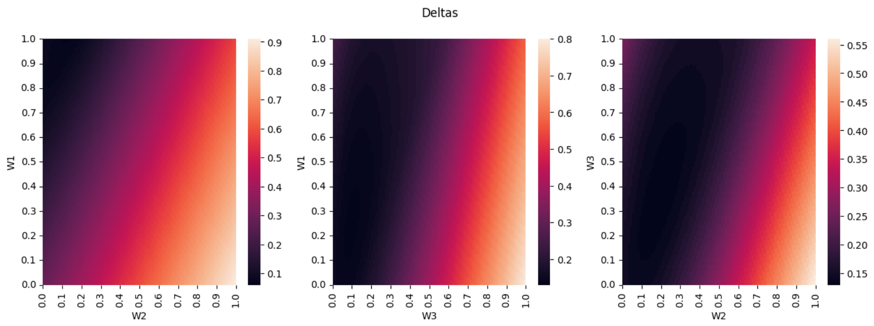


Figure 27: Heatmaps of δ values for each w_i pair fixed

6.2.3 Results

If we choose $w = (0.7, 0.2, 0.1)$ and $w^t = 0.5$ we get the following values for g over our database :

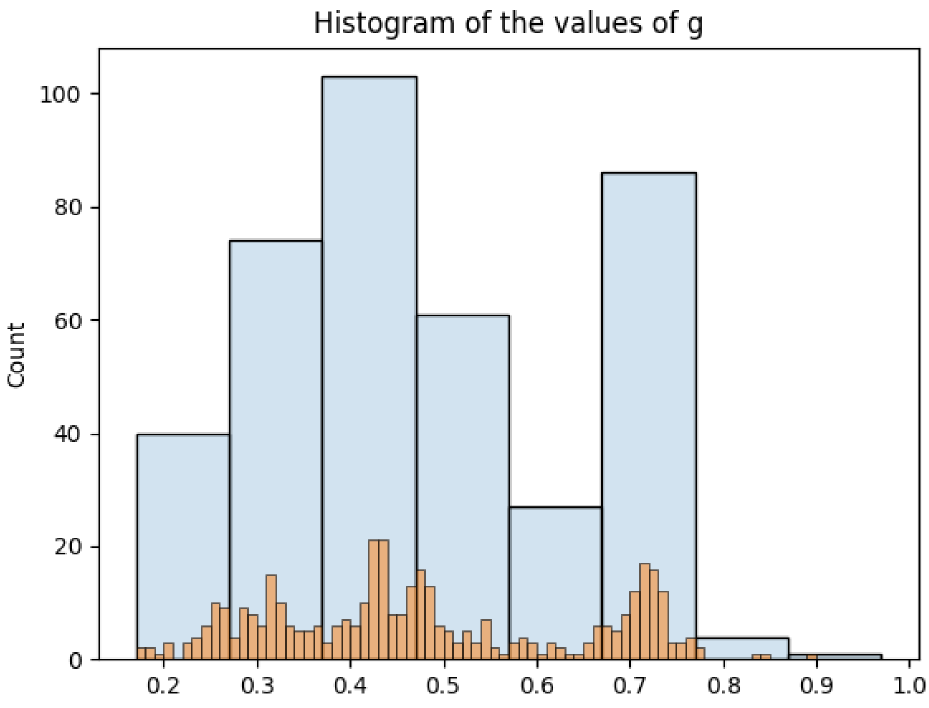


Figure 28: Histogram of g for our database

If simulate values of g with the random variables defined in 6.1.2 we find [12]

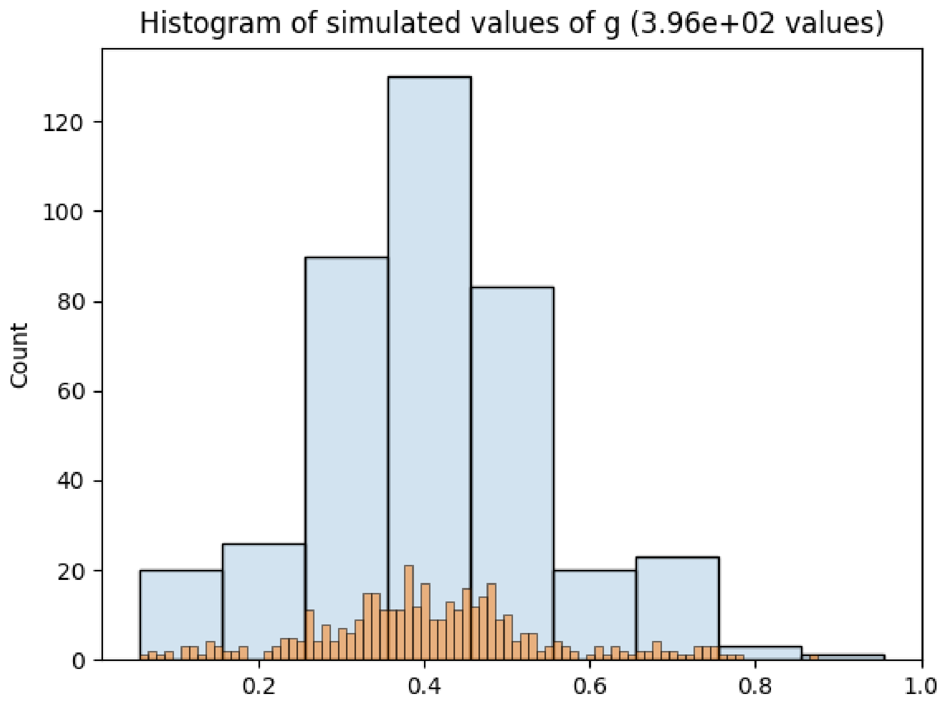


Figure 29: Simulated histogram of g for 396 values

[12]The size of our database is 396

7 Proposition for f function

7.1 Analysis and Definition

If we analyse the pertinence of the first f function proposed with this graph :

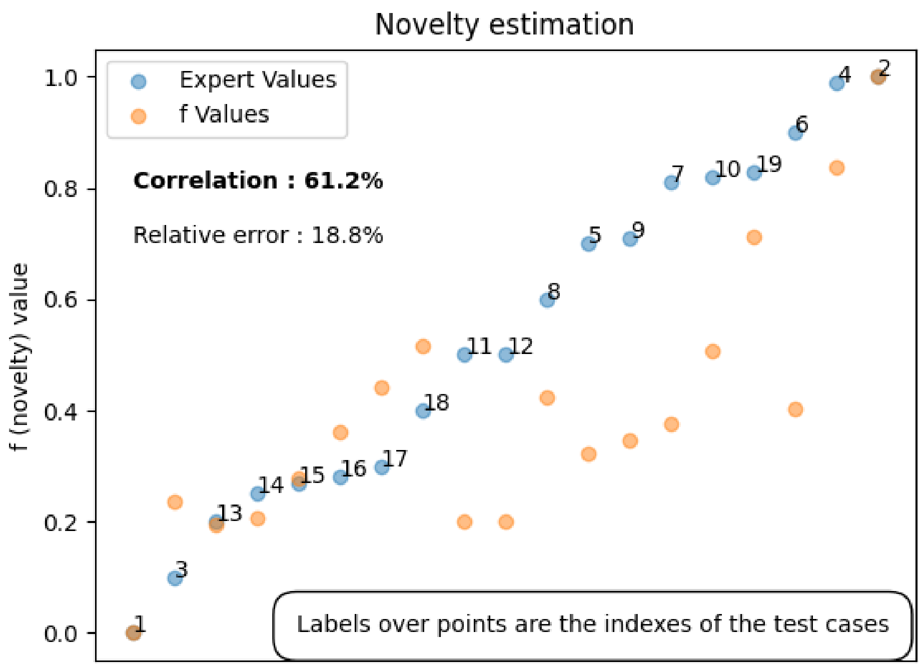


Figure 30: Comparison of expert and f predictions

We can see that this function struggles to simulate novelty values according to those of our expert (Kendall's Tau Coefficient of 0.612).

We analysed our expert novelty evaluation on 19 cases and we get :

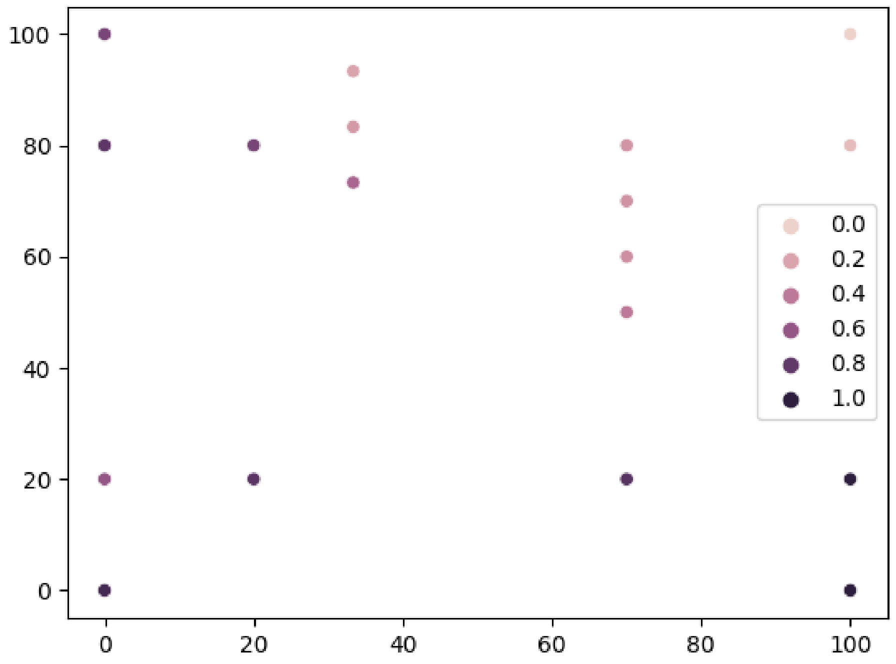


Figure 31: Expert Evaluation of Novelty

The x-axis is the mean value of our average tool's tanimotos, and the y-axis is the mean value of our tool's cosines. We can propose a new approach to simulate mathematically the reasoning of our expert.

Corners of our graph representing reference limit cases, we can define 4 intermediate functions :

- $z_1(x, y) = sin(x)sin(y)$
- $z_2(x, y) = sin(x + 2)sin(y + 2)$
- $z_3(x, y) = sin(x + 2)sin(y)$
- $z_4(x, y) = sin(x)sin(y + 2)$

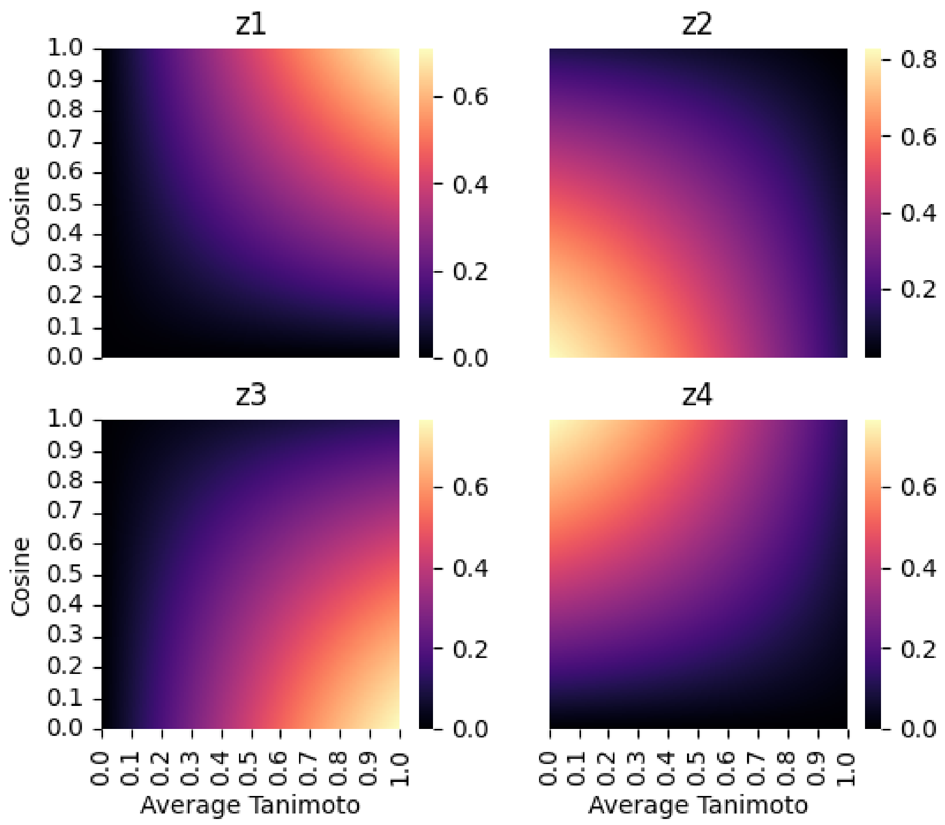


Figure 32: z_i distributions

and we can try to approach our expert's predictions by adding our weighted z_i in a single function :

$$z_{w_1,w_2,w_3}(x,y) = z_1(x,y) + w_1 z_2(x,y) + w_2 z_3(x,y) + w_3 z_4(x,y)$$

And therefore we propose :

$$f_{w_1,w_2,w_3}(x,y) = 1 - \frac{z_{w_1,w_2,w_3}(x,y) - \alpha}{\beta - \alpha}$$

Where

$$\alpha = min(z_{w_1,w_2,w_3}(x,y))$$

and

$$\beta = max(z_{w_1,w_2,w_3}(x,y))$$

By experimenting we find that $w_1 = -0.172, w_2 = 0.192, w_3 = -0.434$ are the best parameters to approach our expert reasoning. With those values we obtain a Kendall's Tau Coefficient of $\tau = 0.882$

We can graph f distribution for those weights :

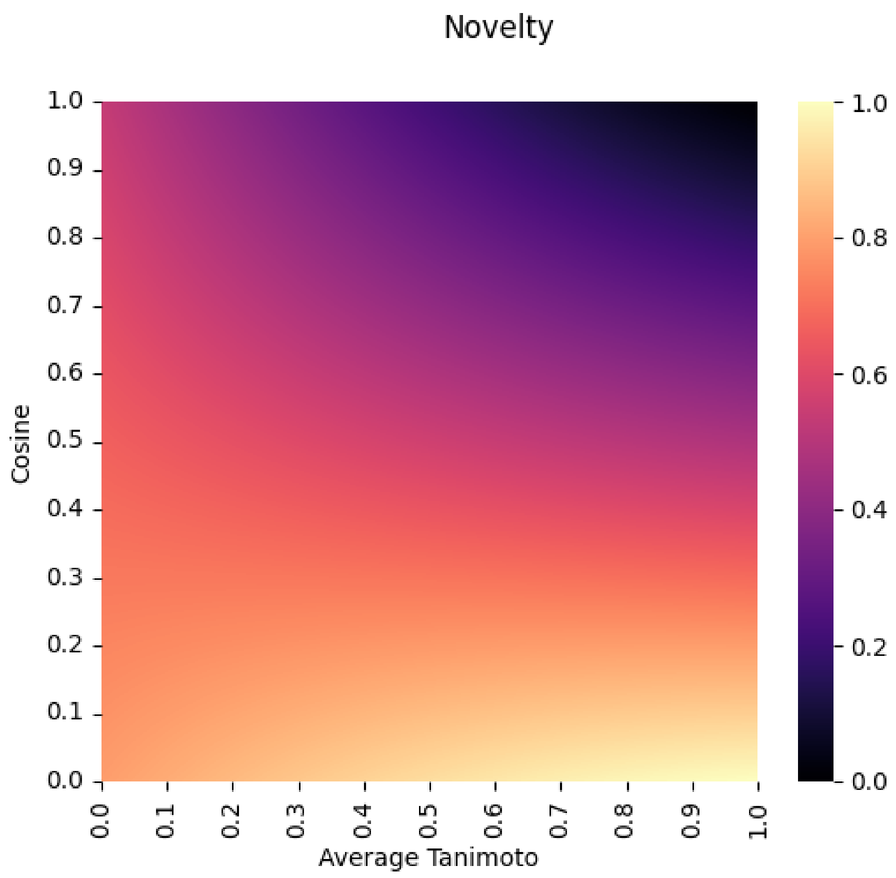


Figure 33: f distribution

7.2 Results

If we compare our predictions with our expert values we get :

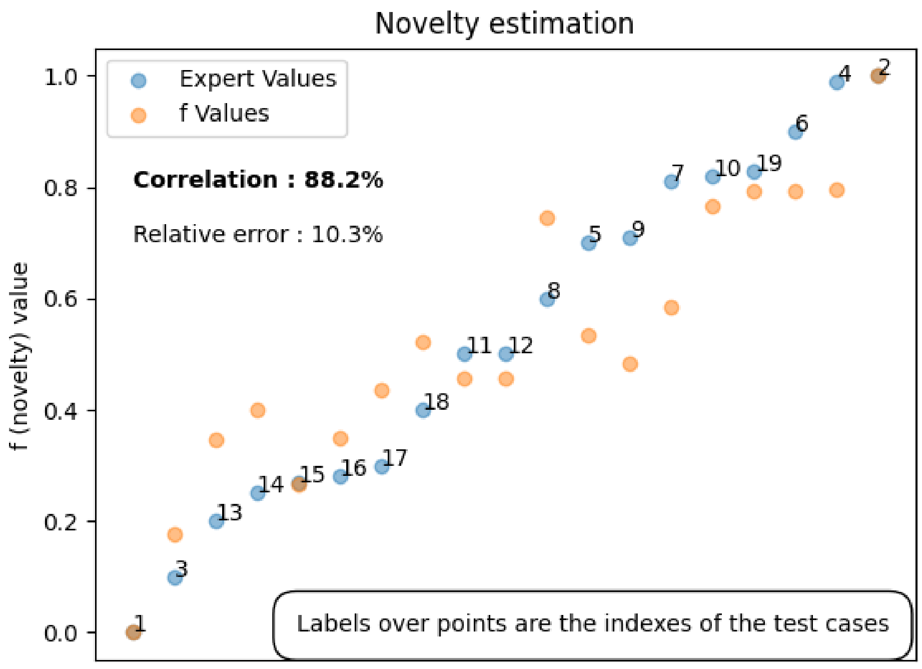


Figure 34: Comparison of expert and f predictions

Our new correlation factor is much more higher (27 pts increase) and our relative error marginally smaller (8.5 pts lower).

References

[1] Paul Jaccard. THE DISTRIBUTION OF THE FLORA IN THE ALPINE ZONE.1. *New Phytologist*, 11(2):37–50, February 1912.

[2] Mingxun Wang, Jeremy J. Carver, Vanessa V. Phelan, Laura M. Sanchez, Neha Garg, Yao Peng, and Don Duy Nguyen et al. Sharing and community curation of mass spectrometry data with Global Natural Products Social Molecular Networking. *Nature biotechnology*, 34(8), 2016.

[3] Kai Dührkop, Huibin Shen, Marvin Meusel, Juho Rousu, and Sebastian Böcker. Searching molecular structure databases with tandem mass spectra using CSI:FingerID. *Proceedings of the National Academy of Sciences*, 112(41):12580–12585, October 2015.

[4] Florian Huber, Justin J. J. van der Hooft, Jurriaan H. Spaaks, Faruk Diblen, Stefan Verhoeven, Cunliang Geng, Christiaan Meijer, Simon Rogers, Adam Belloum, Hanno Spreeuw, Efrain Manuel Villanueva Castilla, Kianoosh Ashouritaklimi, Niek de Jonge, Helge Hecht, Maksym Skoryk, and Zargham Ahmad. Matchms GitHub.

[5] Jeramie Watrous, Patrick Roach, Theodore Alexandrov, Brandi S. Heath, Jane Y. Yang, Roland D. Kersten, Menno van der Voort, Kit Pogliano, Harald Gross, Jos M. Raaijmakers, Bradley S. Moore, Julia Laskin, Nuno Bandeira, and Pieter C. Dorrestein. Mass spectral molecular networking of living microbial colonies. *Proceedings of the National Academy of Sciences*, 109(26), June 2012.

[6] David Rogers and Mathew Hahn. Extended-Connectivity Fingerprints. *Journal of Chemical Information and Modeling*, 50(5):742–754, May 2010.

[7] Jeffrey A. Van Santen, Grégoire Jacob, Amrit Leen Singh, Victor Aniebok, Marcy J. Balunas, Derek Bunsko, Fausto Carnevale Neto, Laia Castaño-Espriu, Chen Chang, Trevor N. Clark, Jessica L. Cleary Little, David A. Delgadillo, Pieter C. Dorrestein, Katherine R. Duncan, Joseph M. Egan, Melissa M. Galey, F.P. Jake Haeckl, Alex Hua, Alison H. Hughes, Dasha Iskakova, Aswad Khadilkar, Jung-Ho Lee, Sanghoon Lee, Nicole LeGrow, Dennis Y. Liu, Jocelyn M. Macho, Catherine S. McCaughey, Marnix H. Medema, Ram P. Neupane, Timothy J. O'Donnell, Jasmine S. Paula, Laura M. Sanchez, Anam F. Shaikh, Sylvia Soldatou, Barbara R. Terlouw, Tuan Anh Tran, Mercia Valentine, Justin J. J. Van Der Hooft, Duy A. Vo, Mingxun Wang, Darryl Wilson, Katherine E. Zink, and Roger G.

Linington. The Natural Products Atlas: An Open Access Knowledge Base for Microbial Natural Products Discovery. *ACS Central Science*, 5(11):1824–1833, November 2019.

[8] Sereina Riniker and Gregory A Landrum. Similarity maps - a visualization strategy for molecular fingerprints and machine-learning methods. *Journal of Cheminformatics*, 5(1):43, December 2013.

[9] Hamid Safizadeh, Scott W. Simpkins, Justin Nelson, Sheena C. Li, Jeff S. Piotrowski, Mami Yoshimura, Yoko Yashiroda, Hiroyuki Hirano, Hiroyuki Osada, Minoru Yoshida, Charles Boone, and Chad L. Myers. Improving Measures of Chemical Structural Similarity Using Machine Learning on Chemical–Genetic Interactions. *Journal of Chemical Information and Modeling*, 61(9):4156–4172, September 2021.

[10] Georg Hinselmann, Lars Rosenbaum, Andreas Jahn, Nikolas Fechner, and Andreas Zell. jCompoundMapper: An open source Java library and command-line tool for chemical fingerprints. *Journal of Cheminformatics*, 3(1):3, December 2011.

[11] I.M Sobol. Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates. *Mathematics and Computers in Simulation*, 55(1-3):271–280, February 2001.

[12] Andrea Saltelli, Paola Annoni, Ivano Azzini, Francesca Campolongo, Marco Ratto, and Stefano Tarantola. Variance based sensitivity analysis of model output. Design and estimator for the total sensitivity index. *Computer Physics Communications*, 181(2):259–270, February 2010.