École Normale Supérieure de Lyon
Informatique fondamentale
Machine Learning

Machine learning project:

Predicting molecular activity with graph Kernels

Sergio Peignier

Abstract

The development of new drugs is a expensive and slow procedure. An important step in new molecules development is the "test" phase. This particularly slow and expensive step is significantly accelerated and cheapened if Structure-Activity Relationship Analysis (SAR) is applied to the tested chemical compounds. We studied here how to predict molecular activity with graph Kernels.

1 Introduction

The goal of Structure-Activity Relationship Analysis (SAR) is to find a relationship between the chemical compounds structures and a particular biological property (toxicity, medical benefits ...) in order to predict if a new molecule has or not the property. The molecules biological properties forecast reduces the "new drugs exploration space" through a virtual test: Only compounds labeled as "positive" (or "negative") for a particular biological property should been tested, thus many useless chemical compounds are not tested any more.

Constructing and studying systems that are able to learn from examples and use their "knowledge" to forecast is the paradigm of Machine Learning. Consequently many methods are available to predict an element properties (activity, class ...). We can mention for example neural networks, decision trees, the support vector machines... We studied here how to predict molecular activity with different graph Kernels.

1.1 Brief description of Kernel based methods

This versatile method is made of two well separated components.

The first component is the kernel, The kernel can be defined as a “comparison function” between elements (molecules in our case) : $K : X \times X \rightarrow \mathbf{R}$ where X is the elements space. The kernel performs the scalar product of two elements projected into a well-adapted reproducing Hilbert space. By comparing[1] the elements in this space it becomes possible to extract patterns, learn and predict the data properties. If the chosen comparison space is not well adapted to the problem the data separation in this spaces is very likely to be poor.

The second component [2] will use the kernel to make pairwise comparison between the training set elements and eventually put some constraints on the pairwise comparison function to classify the learning data elements. This step which is related to the minimization of a loss-function [3] uses the "representer theorem" to represent the minimization of the regularized loss-function defined over the kernel Hilbert space as a finite linear combination of the training elements kernel pairwise products. This second component works without making any hypothesis regarding the type of data [4] which leads to a general training algorithm.

The independence of the two algorithms elements provide us a total modularity between the learning algorithm and the kernel.

First-of-all we will briefly introduce the dataset used to test the kernels and the learning algorithm, then we will present the different graph kernels tested and finally we will compare the different results.

[1] Nevertheless it is important to note that the elements are never explicitly projected into the comparison space, the projection is implicitly done by the kernel function

[2] SVM or other learning algorithms ...

[3] penalization of misclassified data

[4] needs only a well defined kernel

2 The Dataset

The considered public Dataset contains 684 chemical compounds classified as "mutagens" (341 mutagens compounds) or "non mutagens" (343 non mutagens ones) according to a test known as the Salmonella/microsome assay. Both classes are well balanced [5] and noncongeneric [6]. According to the paper where the dataset was presented, toxicity is a very complex and multifactor mechanism and consequently a good dataset should present a very diverse set of chemical compounds.

To predict mutagenicity the model will have to detect small differences between homogeneous structures[7] and seek for structural and regular patterns[8].

3 The Soft Margin Classifier

3.1 Description

In this section we will give a very brief description of the Soft Margin Classifier SVM we used for this project.

The predicted class of an input vector $\mathbf{x}$ is decided by evaluating the sign of $f(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x})$. Where $\phi(\mathbf{x})$ is the representation of $\mathbf{x}$ in the feature-space. Lets call y the true class of the input vector.

The SVM algorithm must find the best hyperplane (defined by the vector $\mathbf{w}$) separating the positive and negative examples.

The SVM finds the hyperplane by solving the following equation:

$$\boxed{min_{\mathbf{w}}(\frac{1}{2}||\mathbf{w}||^2 + C\sum_i(y_i * \mathbf{w}^T\phi(\mathbf{x_i}) - 1))}$$

Lagrangian optimization theory shows that this problem is equivalent to the following dual problem [9]:

$$\boxed{max_{\alpha}(\sum_i \alpha_i - \frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j))}$$

subject to the constrains : $\sum_i \alpha_i y_i = 0$ and $0 \leq \alpha_i \leq C$, $\forall i$

The Soft Margin Classifier allows misclassification of the training examples. This is very useful to improve generalization when the distributions of the two classes are overlapping (avoid overfitting). C is the parameter which controls the trade-off between the penalty associated to misclassified data in the learning set and the margin L2 norm. When C goes to infinity, the SVM does not allow for misclassification of the training examples. This can lead to poor generalization if there is overlap between the distributions of the two classes. When C goes to zero, the SVM does not penalize the misclassified examples and minimizes only the L2 norm of the Lagrange multipliers vector, and consequently is less likely to learn from the examples.

3.2 Parameters impact

We have computed the 10-fold cross-validated prediction accuracy with different values of the parameter C (from 1 to 1000 with a step of 10) for the Markovian random walk kernel with a Morgan

[5] Both classes contain almost the same number of elements
[6] involving a diverse set of molecules
[7] separate very similar structures with different mutagenicity
[8] to distinguish between mutagen and non mutagen compounds
[9] This calculus can be achieved by standard Quadratic Problem solvers such as the *cvxopt* library in python

index equal to one and a stop probability equal to 0.1 (For further see the markovian random walk kernel section and the morgan index section).

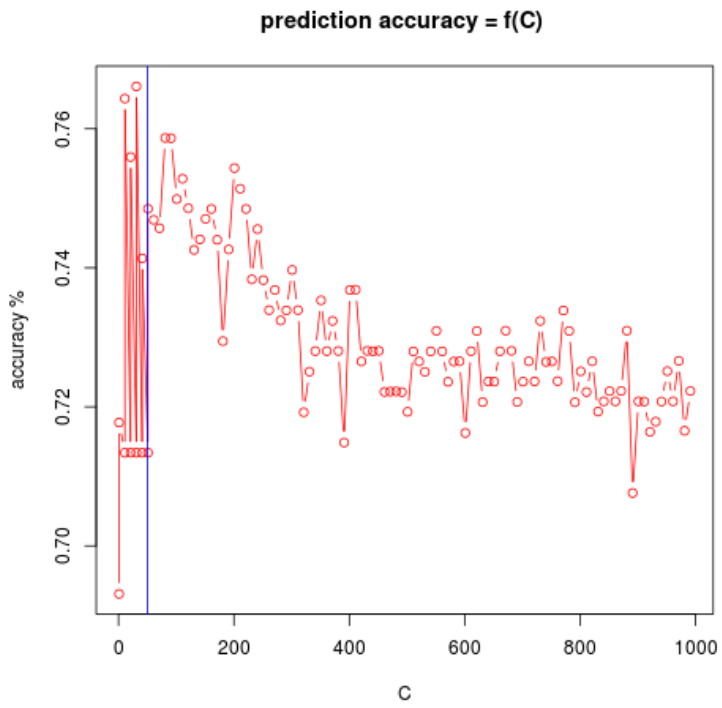

Figure 1: *10-fold cross-validated average accuracy for the Markovian random walk kernel with* $p_q = 0.1$ *and Morgan index* $= 1$ *using different values of the the soft margin classifier* C*. When the parameter* C *is too small, the prediction accuracy is good but it presents a higher variability. When* C *increases (greater than 50) the variability seems to decrease but also does the accuracy. Finally the final accuracy seems to converge around 0.72 when* C *increases.*

These results confirm the hypothesis we had on the dataset, the classes are overlapping. The accuracy decreases when C (misclassification penalization) increases, this is a typical result obtained for overlapping classes. Consequently the use of a soft-margin classifier is necessary. Furthermore we will set the value of C to 100, for this value the results has a low variability and a good accuracy.

4 kernels for chemical compounds classification

A chemical compound can easily be converted in a labeled graph, the atomes being represented as nodes and the chemical bonds as edges. Consequently it is natural to represent a molecule as a graph and apply kernels for graphs to represent each graph by a vector [10] either explicitly [11] or implicitly [12] and use a linear method for classification in this new space.

4.1 fragments kernels: explicit representation

A first approach would be to represent explicitly each graph by a vector of fixed dimension. This vector is filled up with the graph descriptors. This descriptors should retain as much information as

[10] in a reproducing Hilbert space
[11] fragment kernels
[12] random walks kernel, geometric kernel ...

possible and they should be fast to compute. The main challenge with this representation is to choose the best descriptors. As we are considering molecules we can note that the presence of some chemical groups (substructures) are important as predictive patterns. Consequently the graph descriptor vector could indicate the number of particular substructures in the chemical compound. However some particular substructures are computationally hard to detect and computing all the subgraph occurrences is NP-hard. Consequently only some substructures can be searched in a graph[13].

I have not implemented a program performing this kind of representation. I considered more interesting to implement the other functions.

4.2 Implicit representation

4.2.1 Walk kernel: Definition

- Let S represent the set of all possible label sequences of walks W of variable length ($length > 1$). If S_n denote the set of all possible label sequences of walks of length n then $S = \cup_{n \geq 1} S_n$.
- Let $\lambda_G(w)$ be a weight associated to each walk $w \in W(G)$.
- $K(G_1, G_2)$ is the walk kernel product of G_1 and G_2 (two graphs).
- $\phi_s(G)$ the projection of the graph G in a reproducing Hilbert space is defined as:

$$\boxed{\phi_s(G) = \sum_{w \in W(G)} \lambda_G(w) \mathbf{1}(s \textit{ is the label sequence of } w)}$$

- Finally the walk kernel of two graphs is defined as:

$$\boxed{K_{walk}(G_1, G_2) = \sum_{s \in S} \phi_s(G_1)\phi_s(G_2)}$$

4.2.2 Product graph

4.2.3 Definition

- Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs with labeled vertices
- The product graph $G = G_1 \times G_2$ is a graph G = (V , E) with:

1.- $V = \{(v_1, v_2) \in V_1 \times V_2\}$ and both v_1 and v_2 have the same label

2.- $E = \{((v_1, v_2), (v_1', v_2')) \in V \times V$ with $(v_1', v_1) \in E_1$ and $(v_2, v_2') \in E_2\}$

4.2.4 Lemma

There is a bijection between the walks on the product graph $w \in W_n(G1 \times G2)$ and the pairs of walks $w_1 \in W_n(G_1)$ and $w_2 \in W_n(G_2)$ with the same label sequences.

This important result allow us to compute the graph kernel walking in the graph product, avoiding many useless calculus. Only the walks with the same label sequences will be used (otherwise the result is equal to zero and do not need to be explicitly computed):

$$\boxed{K_{walk}(G_1, G_2) = \sum_{w \in W(G_1 \times G_2)} \lambda_{G_1 \times G_2}(w)}$$

The following kernels are computed using this important result. The main differences between the kernels is the way to compute the weights ($\lambda_{G_1 \times G_2}(.)$) and the walks lengths.

[13] choosing the better descriptors could be particularly hard and arbitrary

4.3 nth order walk kernel

4.3.1 Description

The nth-order walk kernel compares two graphs through their common walks of length n. For this walk kernel we have:

- $\lambda_G(w) = 1$ if the length of w is n, 0 otherwise.

To compute the nth-order walk kernel we can use n-th power of A, the adjacency matrix of $G_1 \times G_2$:

$$\boxed{K_{walk}(G_1, G_2) = \sum_{i,j} [A^n]_{i,j}}$$

We found in the literature that the computation of this kernel is in $O(n|G_1||G_2|d_1d_2)$, where d_i is the maximum degree of G_i.

4.3.2 Parameters impact

The kernel order (n) is the only parameter. We have computed the 10-fold cross-validated accuracy using different values for the parameter n (the kernel order). The best results were obtained with $n = 1$ and $n = 2$. The accuracy decreases when the kernel order increases. This result means (for this dataset) that the comparison of two graphs through their common walks of length n is more accurate when the length of the considered walks is small (one or two). When the walks are too longs, the kernel is not able to classify accurately the molecules. Moreover when $n > 3$ the specificities is very close to 0 and sensitivities are very close to 1. This shows that the kernel does not detect any pattern, but classify almost every molecule as a mutagen.

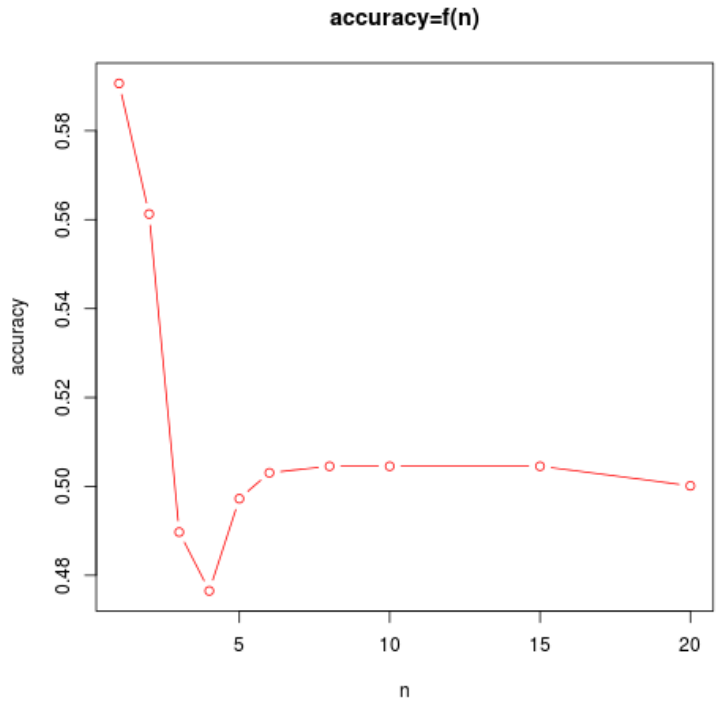

Figure 2: *10-fold cross-validated average accuracy for the nth order kernel using different values for the order n. the best results were obtained with $n = 1$: Long walks are penalized and the comparison focus on short walks.*

4.4 Geometric walks kernel

4.4.1 Description

The geometric walk kernel is characterized by a feature space of infinite dimension (all the walks with all the possible lengths). The geometric walk kernel is obtained (when it converges) with $\lambda_G(w) = \beta^{length(w)}$, for $\beta > 0$. For $w = (v_1, \ ... \ , v_n)$ a walk in the product graph ($w \in W(G_1 \times G_2)$).

To compute efficiently the geometric kernel we decompose the weight calculus as follows:

$$\boxed{\lambda_G(v_1, \ ... \ , v_n) = \lambda^i(v_1) * \prod_{j=2}^{n} \lambda^t(v_{j-1}, v_j)}$$

- $\lambda^i(x)$ is the weight associated to the observation of a vertex x in the graph. This weight will be called the "emission weight"
- $\lambda^t(x, y)$ is the weight associated to the observation of a vertex x followed by a vertex y in the graph. This weight will be called the "transition weight"

Therefore the kernel calculus is given by:

$$\boxed{K_{walk}(G_1, G_2) = \sum_{n=1}^{\infty} \sum_{w \in W(G_1 \times G_2)} \lambda^i(v_1) * \prod_{j=2}^{n} \lambda^t(v_{j-1}, v_j)}$$

Putting this calculus in a matricial form we get:

$$\boxed{K_{walk}(G_1, G_2) = \sum_{n=1}^{\infty} \Delta_i \Delta_t^n \mathbf{1}}$$

With Δ_i the vector of emission weights and Δ_t the matrix of transition weights. And identifying in this expression the limit development of $(I - \Delta_t)^{-1}$ we finally obtain:

$$\boxed{K_{walk}(G_1, G_2) = \Delta_i^T (I - \Delta_t)^{-1} 1}$$

The transition weights matrix can be calculated by multiplying the adjacency matrix of the product graph by the parameter β. The emission weights vector can be calculated by multiplying the identity vector of its length equal to the number of vertex in the product graph by β. This kernel can be computed in $O(|G_1|^3 |G_2|^3)$.

4.4.2 parameters impact

The geometric weight (β) is the only parameter. We have computed the 10-fold cross-validated prediction accuracy with different values of the parameter β. The best results were obtained with $\beta = 0.05$. The comparison of two graphs using the geometric kernel is better achieved when the geometric parameter is low. This result means that the long walks are penalized and the comparison focus on short walks.

4.5 Markovian random walk kernel

4.5.1 Description

The random walk kernel is obtained with $\lambda_G(w) = P_G(w)$, where P_G is a Markov random walk on G. In that case we have, $K(G1, G2) = P(label(W1) = label(W2))$, where W1 and W2 are two independent random walks on G_1 and G_2 , respectively.

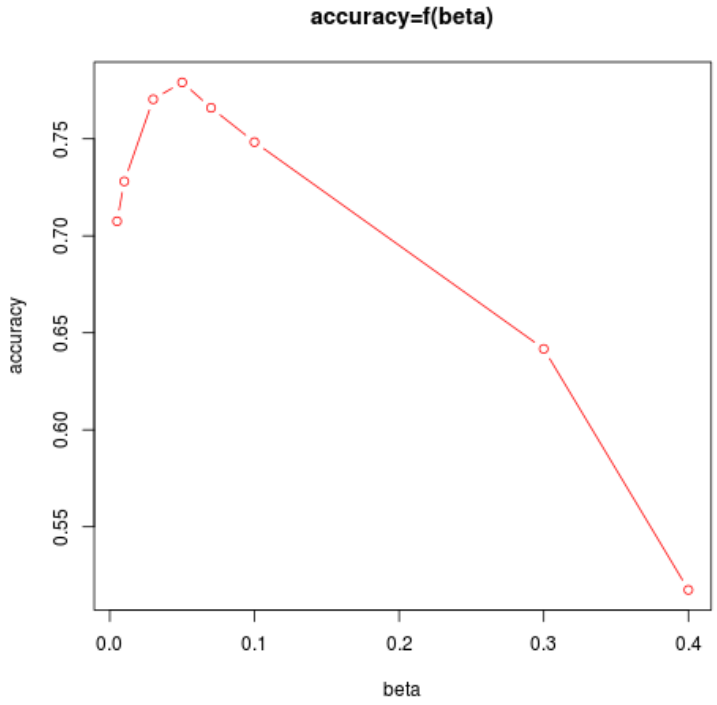

Figure 3: *10-fold cross-validated average accuracy for the geometric walk kernel using different values of weight β. the best results were obtained with $\beta = 0.05$: Long walks are penalized and the comparison focus on short walks.*

To compute the random walk kernel we use the same decomposition used in the geometric kernel calculus replacing "emission weight" vector by the "emission probabilities" in the graph product and replacing the "transition weight" by the "transition probabilities" in the graph product.

For a Markov random walk, the probability of a walk w in a graph $G_1 = (V, E)$ is defined as:

$$\boxed{p_G(v_1 ... v_n) = p_s(v_1) \prod p_t(v_i|v_{i-1})}$$

Where $p_s(x)$ is the emission probability (probability to observe a vertex x) and $p_t(y|x)$ is the transition probability (probability to observe a vertex y knowing that we have observed a vertex x at the previous position). To ensure that this equation defines a probability distribution ($\sum_{w \in W} = 1$) some constraints are applied on the emission and transition probabilities p_s and p_t:

$$p_s(v) = p_0(v) p_q(v)$$

$$p_t(u|v) = \frac{1 - p_q(v)}{p_q(v)} p_a(u|v) p_q(u)$$

Where p_0 is initial distribution ($\sum_v p_0(v) = 1$ and $\forall v,\ p_0(v) > 0$. In our case $p_0(v) = 1/|V|$), p_a the transition probability ($\sum_{v,u} p_a(v|u) = 1,\ \forall v, u,\ p_a(v|u) > 0$. In our case $p_a(u|v) = 1/d(v)$ if $(v, u) \in E$, 0 otherwise and $d(x)$ being the density of the node v), and p_q the stopping probability ($0 < p_q < 1$).

To compute the random walk kernel, it is necessary to apply this definitions to the graph product:

Lets define the functional π on the set of paths W in the graph kernel as:

$$\boxed{\pi((u_1, v_1), ..., (u_n, v_n))) = \pi_s((u_1, v_1)) \prod_{i=2}^{n} \pi_t((u_i, v_i)|(u_{i-1}, v_{i-1}))}$$

with:

$$\pi_s(u_1, u_2) = p_s^{(1)}(u_1)p_s^{(2)}(u_2)$$

$$\pi_t((v_1, v_2)|(u_1, u_2)) = p_t^{(1)}(v_1|u_1)p_t^{(2)}(v_2|u_2)$$

Where $p_t^{(i)}$ is the transition probability function for the graph i and $\pi_s^{(i)}$ is the emission probability function for the graph i. Using this functional we are able to define the transition probabilities matrix and the emission probabilities vectors just as in the geometric kernel case. Consequently the kernel computation has the same formula.

4.5.2 Parameters impact

The stop probability (p_q) is the only free parameter. We have computed the 10-fold cross-validated prediction accuracy with different values of the parameter p_q. The best results were obtained with $p_q = 0.3$. The accuracy seems to increase when p_q increases util $p_q = 0.3$, then p_q remains almost constant (or even decreases a little). This result seems to be opposite to the result found in the paper that presents this method. But in fact in the paper the authors do not use exactly the same kernel, but a "radialized" version of it (see the preprocessing section for further details).

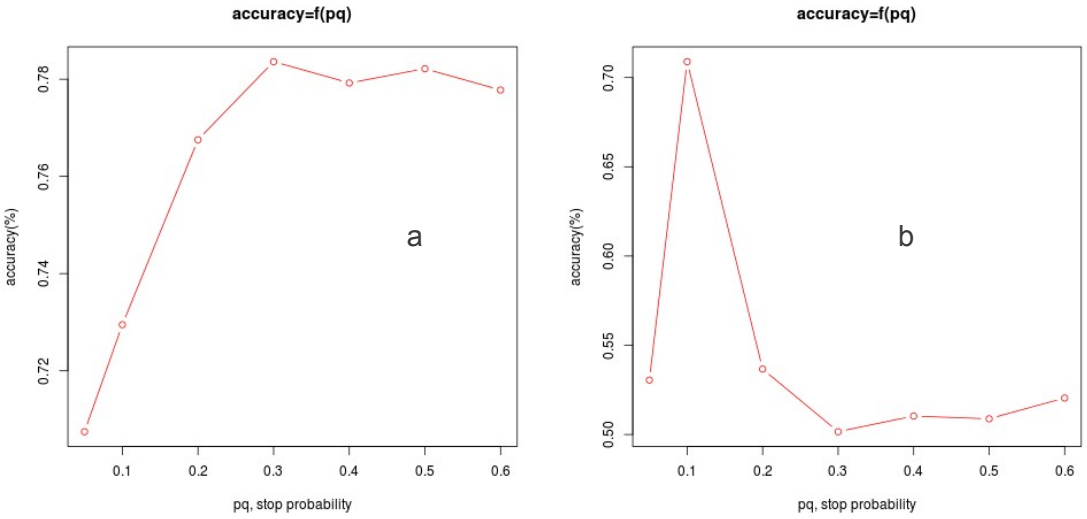

Figure 4: *panel a: 10-fold cross-validated average accuracy for the Markovian random walk kernel using different values of stop probability p_q. the best results were obtained with $p_q = 0.3$. Panel b: 10-fold cross-validated average accuracy for the radialized Markovian random walk kernel using different values of stop probability p_q using a standard deviation parameter $\sigma = 2$ for the radialization. In this case, the best results are obtained when $p_q = 0.1$.*

When we set the standard deviation parameter of the radial kernel (*sigma*) to 2, and we compute the 10-fold cross-validated prediction accuracy with different values for the parameter p_q for the radialized kernel, we find that the accuracy reach a maximum for $p_q = 0.1$. Which is quite close to the paper result. Nevertheless the overall profile is quite different, this may be related to the choose of the standard deviation parameter *sigma* (the value used in the paper is not cited).

4.6 Subtree kernel

4.6.1 Description

The subtree kernel is very similar to the walk kernel. This kernel computes the weighted number of subtrees in the product graph. The only difference is the way to move within the graph, while the walk kernel considerate walks, the subtree kernel considerate subtrees. Since this kernel considerates subtrees, it is efficiently computed in a recursive way. $T(v, n)$ denotes the weighted number of subtrees of depth n rooted at the vertex v (in the product graph):

$$\boxed{T(v, n+1) = \sum_{R \in N(v)} \prod_{v' \in R} \lambda_t(v, v') T(v', n)}$$

Where $N(v)$ is the set of neighbors of v, and each $R \in N(v)$ consists of one or several pair(s) of neighbors of v that are identically labeled and connected to v by edges of the same label.

To compute the subtree kernel it is necessary to repeat this calculus for each different vertex as the tree's root.

This kernel has two parameters, the subtree depth (n) and the transition weight (λ). λ has the effect of rewarding tree-patterns depending on their degree of complexity. A value of λ greater than one favors the impact of tree-patterns of increasing complexity over the trivial linear tree-patterns and a value of λ smaller than one penalize their impact.

4.6.2 Parameters impact

The subtree depth (n) and the transition weight (λ) are the subtree kernel parameters. We have computed the 10-fold cross-validated prediction accuracy with different values for the parameter n (the kernel order) for a fixed value of $\lambda = 0.1$. We also computed the 10-fold cross-validated prediction accuracy with different values for the parameter λ for a constant depth $n = 2$.

In the first case the best results were obtained with $n = 3$. This result means (for this dataset) that the comparison of two graphs through their common subtrees of depth n is better achieved when the length of the considered subtrees is small (one or two or even three). When the subtrees are too longs, the kernel is not able to classify accurately the compounds.

In the second case, the best results were obtained with $\lambda = 0.1$. We get a higher accuracy when the parameter λ is low. When two graphs are been compared, complex tree-patterns are strongly penalized. The main differences (and similarities) between two graphs (chemical compounds) are those described with a simple tree-pattern complexity.

4.7 Morgan index

When many vertex have identical labels, the product graph has many vertex too. Thus the computation of the graph kernel can be time-consuming [14]. Moreover the detection of interesting patterns could be difficult since this search is to be too naive [15].

To solve both issues it is possible to increase the local specificity of labels [16]. Increasing the specificity, the number of common label paths between graphs decreases so does computation time. Including information about the vertex is likely to increase the relevance of the features used to compare graphs, allowing the detection of interesting patterns.

[14] and consequently difficult to use with large chemical data banks
[15] interesting patterns are "hidden" by generality ...
[16] adding more information about the vertex in their own labels

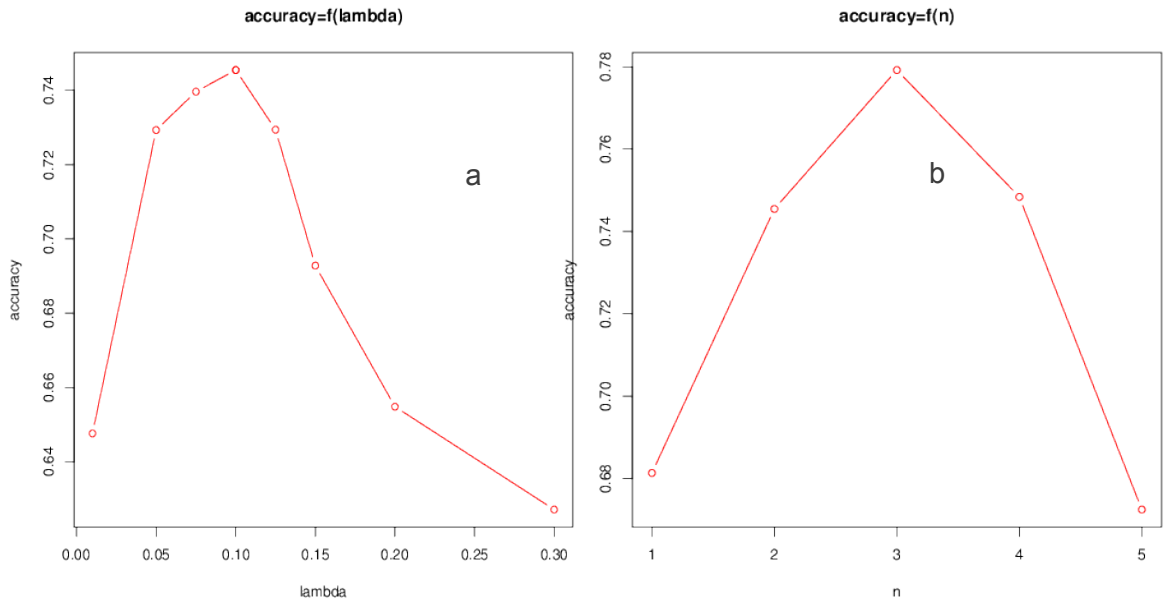

Figure 5: *panel a: 10-fold cross-validated average accuracy for the subtrees kernel with a constant depth* $n = 2$ *and using different values of weight* λ. *the best results were obtained with* $\lambda = 0.1$, *complex tree-patterns are thus strongly penalized. The main differences (and similarities) between two graphs (chemical compounds) are those described with a simple tree-pattern complexity. panel b: 10-fold cross-validated average accuracy for the subtrees kernel with a constant weight* $\lambda = 0.1$ *and using different values of depth. The best results were obtained with* $n = 3$, *the comparison of two graphs through their common subtrees of depth n is better achieved when the depth of the considered subtree is small.*

A way to add information about the vertex it is possible to compute the Morgan index for each vertex. The Morgan index is computed iteratively: initially, the Morgan index are equal to 1. Then, at each iteration, the Morgan index of each vertex is defined as the sum of the Morgan index of its adjacent vertex.

We have tested the effect of the Morgan index computing a 10-fold cross-validated prediction accuracy with different values of Morgan index using a random walk kernel with a fixed parameter of $p_q = 0.1$ (see the random walk kernel section for further details). The best results were obtained using a Morgan index equal to 1.

For a fixed value of $p_q = 0.1$ and for the current dataset, classification seems to be improved for the first iteration of the process and then decreases. This means that although the first step of the Morgan process improve the expressive of the kernel (and its predictive power), however for further iterations this information seems to becomes too specific. This result is consistent with the results presented in the article that describes the random walk kernel methods. Since this information seems to improve results we have decided to use a Morgan index of 1 while computing all the kernels. For time-computation reasons we were not able to compute the impact of the Morgan index on the other kernels learning.

4.8 Centering, normalizing and "radializing" the original kernel

Sometimes it is useful to preprocess the learning dataset values (in the features space). Some possibilities are to center, normalize and "radialize" the original kernel. However, these transformations do not always guarantee better results.

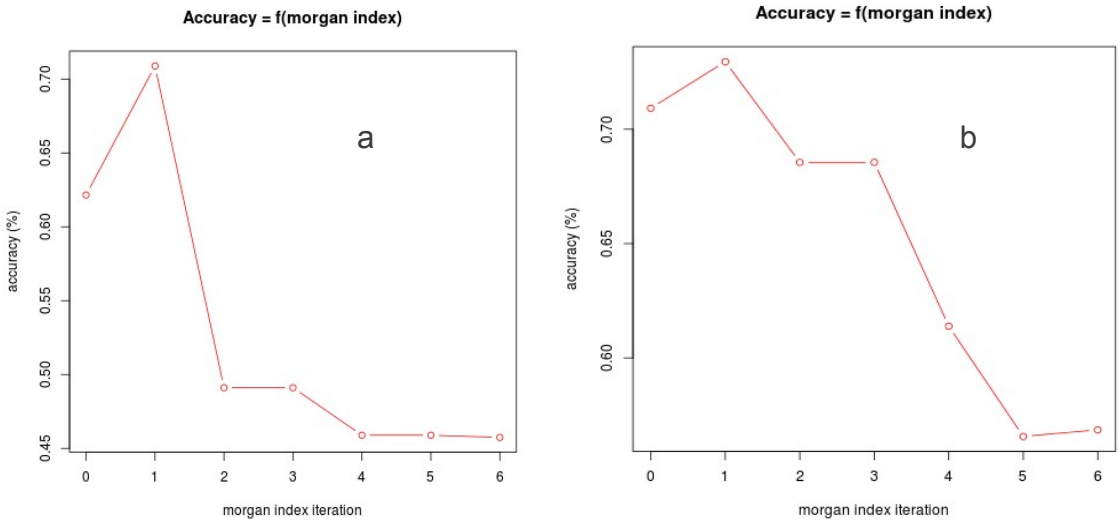

Figure 6: *10-fold cross-validated average accuracy for the Markovian random walk kernel ($p_q = 0.1$) using different values for the morgan index. As we can see in the graphic, the first iteration of the Morgan index improves the accuracy, while more iterations make the accuracy decrease. This result is also true when we use a radialized Markovian random walk kernel. The results for the radialized kernel are plotted in the panel a, and those for the original kernel are plotted in the panel b.*

4.8.1 Centering

The learning dataset can be centered in the feature space by applying the following operation to the gram matrix K [17]:

$$\boxed{(I - \frac{1}{n}\mathbf{1} \times \mathbf{1}^T)K(I - \frac{1}{n}\mathbf{1} \times \mathbf{1}^T)}$$

4.8.2 Normalizing

The learning dataset can be normalized in the feature space by applying the following operation to the gram matrix K:

$$\boxed{k_{norm}(x, y) = \frac{k(x, y)}{\sqrt{k(x)k(y)}}}$$

4.8.3 Radializing

The learning dataset can be radialized in the feature space by applying the following operation to the gram matrix K:

$$\boxed{k_{radial}(x, y) = exp(\frac{k(x, x) + k(y, x) - 2k(x, y)}{2\sigma^2})}$$

Where σ is the standard deviation of the Gaussian low used to "radialize" the original kernel [18].

4.8.4 Impact of normalizing, centering and radializing the kernel

We have computed the 10-fold cross-validated prediction accuracy for the different kernels with constant parameter values, centering, normalizing and radializing the gram matrix.

[17] matrix of pairwise kernel product of the training set elements

[18] It is interesting to note that $k(x, x) + k(y, x) - 2k(x, y)$ is the distance between x and y in the feature space.

kernel	centering	normalizing	radializing ($\sigma = 2$)	not modified kernel
geometric($\beta = 0.05$)	0.784976124751	0.779208663517	0.752798196234	0.779123660404
nth order($n = 1$)	0.59365324255	0.636088921757	0.703105376234	0.590652632653
Markov ($p_q = 0.1$)	0.74394512199	0.745689810889	0.708894088246	0.729494592739
subtree ($\lambda = 0.05$, $n = 2$)	0.723450871388	0.62702333973	0.650539451007	0.729324586513

10-fold cross-validated accuracy the different kernels, centered, normalized, radialized and not pretreated kernel with constant parameters.

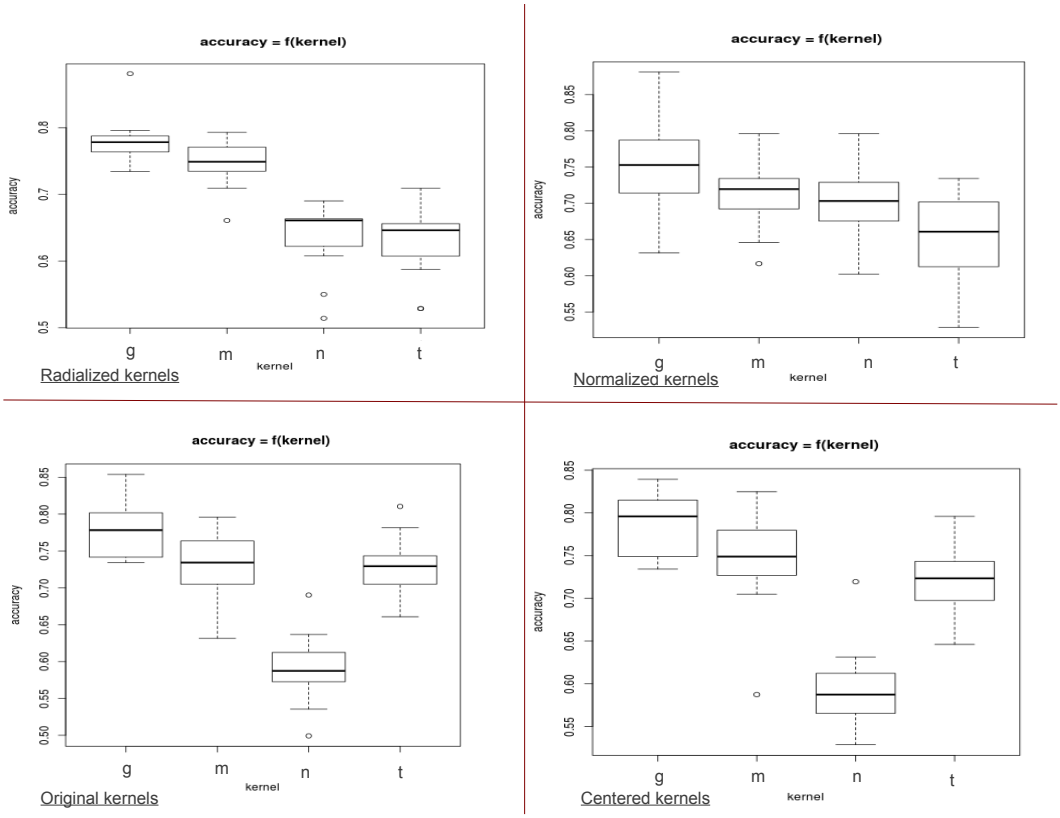

Figure 7: *10-fold cross-validated prediction accuracy for the different kernels with constant parameters and having radialized, normalized or centered (or having applied none transformation to) the gram matrix. t: subtree kernel, m: Markovian random walk kernel, g: geometric kernel and n: n-th order kernel*

Centering the gram matrix seems to have a good impact on the accuracy of the geometric, the nth order and the Markovian random walk kernels. For the subtree kernel, centering the gram matrix seems to reduce the accuracy. Nevertheless the impact of centering the gram matrix is almost null (the variation of the accuracy related to this operations are near to 1%).
Normalizing the gram matrix has a low positive impact on accuracy of the geometric and the Markovian random walk kernels. This operation has a more significant good impact on the nth order kernel (increase of almost 3%). But this operation has a significant bad impact on the subtree kernel (decrease of almost 10%).

Finally radializing the gram matrix with a standard deviation parameter $\sigma = 2$ has a bad impact on the accuracy of the geometric kernel (decrease of almost 3%), of the Markovian random walk kernel (decrease of almost 3%) and of the subtree kernel (decrease of almost 8%). Nevertheless this operation has a very good impact on the accuracy of the nth order kernel (increase of almost 10%).

4.9 Comparing the different kernels

We have computed the 10-fold cross-validated prediction accuracy for the different kernels with their best parameter values[19]. With this parameters the Markovian random walk has the best accuracy, then the geometric kernel, the subtree kernels and finally the nth order kernel. It is necessary to note that the subtree parameters space was not totally explored, we only evaluated one parameter impact at a time, then we took the parameters that maximized the accuracy at a constant level of the other parameter, so we have not considerate the possible interactions between both parameters. Consequently we never compared the maximum of the subtree kernel to the other kernels.

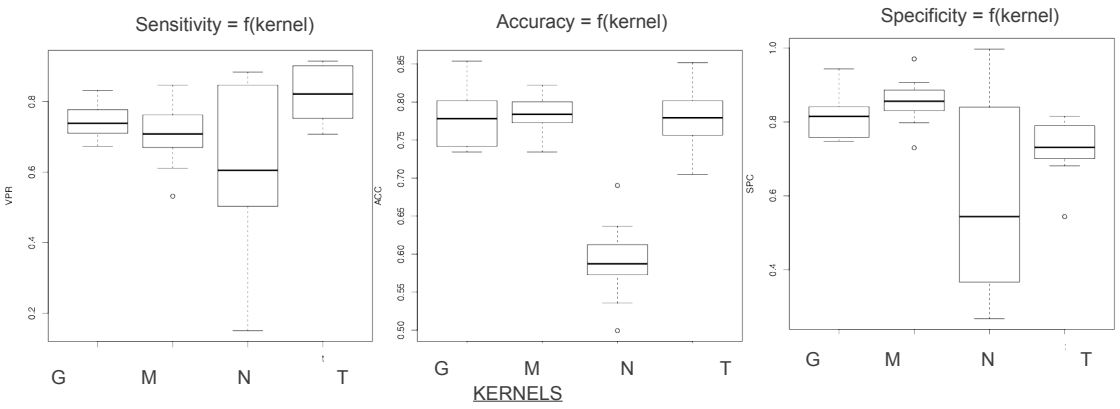

Figure 8: *10-fold cross-validated accuracy, specificity and sensibility for the different kernels with their best parameters values. t: subtree kernel, m: markovian random walk kernel, g: geometric kernel and n: n-th order kernel*

If we compare the specificity of the different kernels, the geometric, the subtree and the Markovian random walk kernels have good specificities, while the nth order kernel has the lowest specificity (gap at least of 10% between the nth order kernel and the other kernels). The geometric, the subtree kernels and also the Markovian random walk are quite accurate predicting mutagenicity.

If we compare the sensitivity of the different kernels, The subtree kernel, the markovian random walk kernel and the geometric kernel have the highest sensibility, the nth order kernel has the lowest sensitivity. The subtree kernel, the Markovian random walk kernel and the geometric kernel are better predicting non mutagenicity than the nth order kernel.

The geometric kernel, the Markovian random walk and the subtree kernel have a high specificity and a high sensitivity, these kernels can be used to predict both mutagenicity or nonmutagenicity, with a similar (good) degree of confidence. The nth order kernel has a low specificity and a low sensibility, mutagenicity and nonmutagenicity can be predicted with a similar (low) degree of confidence.

kernel	sensitivity	specificity	accuracy
geometric ($\beta = 0.05$)	0.742620038606	0.814401460957	0.779123660404
nth order ($n = 1$)	0.605031844703	0.593630446857	0.590652632653
markov ($p_q = 0.3$)	0.70778370744	0.85605016564	0.783613949861
subtree ($\lambda = 0.1$, $n = 3$)	0.821075946521	0.731250252736	0.779208663517

10-fold cross-validated accuracy, specificity and sensibility for the different kernels with their best parameters values.

[19] found during the parameter impact studies

5 Discusion and conclusions

In this project we have deal with different formats of chemical compounds input files (we have produced a little wrapper for importing such files in python). We have also implemented and tested different kernels for graphs with a chemical compounds database, to predict their mutagenicity. We have also tested the impact of centering, normalization and radialization on the accuracy of the original kernels. A little "basic statistic" script with some useful functions (cross-validation, ROC ...) was also implemented. We have implemented a SVM program (which calls the *cvxopt* Quadratic Problem Solver python library) to check the impact of the learning step (SVM) on the accuracy, nevertheless it would be interesting to compare our SVM with other implemented SVM programs (libsvm ...). The program we have implemented can also work with the libSVM library SVM. Unfortunately we were not able to compare both methods because of time issues. Finally we must note the complexity of the parameters interaction. This interaction make impossible an extensive parameter analysis (high time-computation cost of the kernel functions).

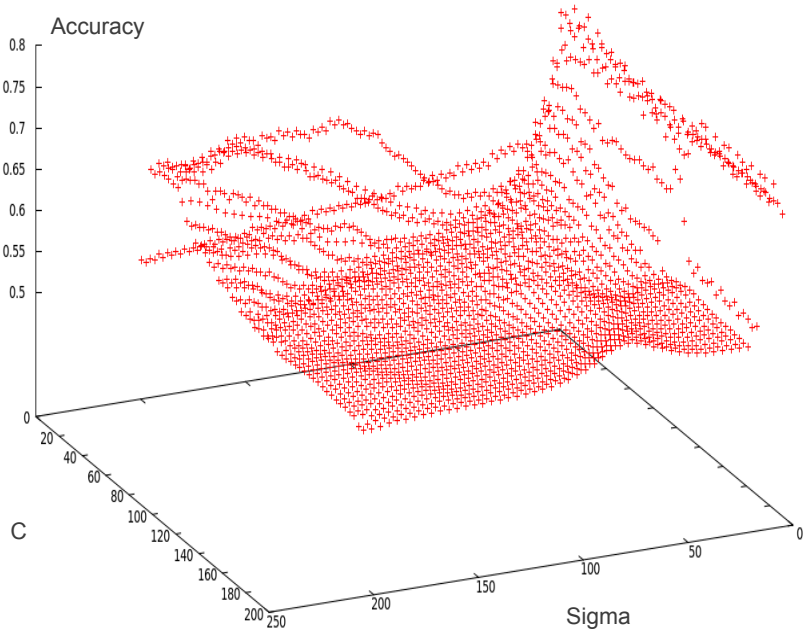

Figure 9: *The accuracy of the radialiazed Markovian random walk kernel (p_q = 1 and Morgan index = 1) as a function of the SVM soft margin parameter (C) and the standard deviation parameter of the radialization. This image is a good illustration of the important complexity of the relationship between the different parameters. As we can see accuracy is improved when σ is low or when C is quite low too (but not 0 nor close to 0), also accuracy decreases when C and σ are both high. However different behaviors appear when both parameters are different from the extreme values. We are not always able to handle this issues because of the computational cost of an extensive analysis.*