Drug repositioning
A Support Vector Machine based predictor for Drug-Target interactions

Alia Rahim

Submitted for the Degree of Master of Science in

Machine Learning Msc

Department of Computer Science
Royal Holloway University of London
Egham, Surrey TW20 0EX, UK

September 9, 2018

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count:

Student Name:

Date of Submission:

Signature:

Abstract

In this report I will be predicting drug-target interactions using a kernel SVM method, the idea behind this methodology is such that if two drugs are similar they are likely to share many of the same targets and if two targets are similar they are likely to be targeted by the same drug, so a similarity based approach is appropriate.

Kernels were computed w.r.t. the nature of the data i.e. the most fundamental piece of information relating to a drug is it's molecular structure therefore a tanimoto kernel was used to compute the similarity between meaningful patterns in a drugs molecular structure. The most fundamental aspect of a target is it's sequence therefore a mismatch kernel was used to find similarities between two targets. The mismatch kernel splits a given sequence into substrings of specified length m and finds both exact matches between substrings from different sequences as well as matches where there is at most k mismatches.

These two kernels were then combined using properties of valid kernels to formulate the basis of a pairwise classification task. The new kernels were then used in a support vector classifier in which we aim to find the optimal separating hyperplane, in order to classify whether a drug-target interaction exists or not.

The highest classification accuracy was achieved using the tensor product property which had a prediction accuracy of 63.1% whereas a kernel based on the direct sum property of kernels only managed to achieve an accuracy of 54.5%

Contents

1 Introduction

Drug re-positioning is the process of utilizing a drug that already exists as a treatment for a given disease to treat a different disease e.g. in the case of aspirin which was originally proposed as a treatment for inflammation and pain but has also proven to be useful for treating strokes and heart attacks due to its ability to prevent blood clots.

Due to the high cost associated with developing a new drug, often times rare diseases are left without a viable treatment, a logical way to stop this from being the case is drug re-positioning. By using drugs that already exist, drug re-positioning also has the added benefit of having already passed phase 1 trials which means the drug has already been deemed to safe to use. Exploiting computational power and advancements e.g. greater levels of data collection new methodologies

The discovery of the dual purpose of aspirin was made by chance but since then many advancements in technology and biology have been made. In 1979, around 1100 completed protein sequences in the human body had been identified, but today according to the human proteome map 30,057 proteins have been identified [21]. With greater levels of computational power and new methodologies, we can now exploit the wealth of biological data that has been discovered. So how do we formulate methods that will allow us to make similar discoveries.

In this paper I will be exploring the sub-problem of predicting drug-target interactions which is paramount in drug re-positioning. In order to do this I will collate information relating to existing drug-target interactions from drugbank and uniprot. I will then create negative data samples by randomly sampling drugs and targets as outlined in the paper, Interpretable Drug Target Prediction Using Deep Neural Representation [8]. In the paper Graph Kernels for Chemical Informatics [13], Ralaivola introduced the tanimoto kernel which given a set of molecular fingerprints(binary vector representation of chemical compounds) computes the similarity between two molecules. Leslie et al [2] introduced the mismatch kernel which is a string based kernel, given two protein sequences the mismatch kernel finds matching sub-strings of length k with a parameter p for acceptable amounts of mismatches in the two sub-strings, this kernel is used to compute sequence similarity. Jacob et al [9] then combined these kernels using a tensor product to create a tensor product kernel which computes a new kernel based upon contributions of two different kernels which are computed using different features from the data, using this and a few other kernel combinations such as the direct sum kernel, I will then train a support vector classifier using as a parameter the pre-computed grams of these kernels, this will allow us to output predictions for drug-target interactions. To check the performance of this method I will be using ROC curves and Precision Recall curves.

2 Background Research

2.1 Biology

A drug is a chemical that interacts with proteins(targets) in the body to change their physiological function this can often be noticed through a protein's phenotype(outward expression of protein). Each drug can have many targets some of which can produce unwanted changes(side effects) and each protein can be targeted by many different drugs(many to many relationship).

Proteins are used by the body to build and repair the body e.g. enzymes which are used to speed up chemical reactions such as insulin or antibodies which are used to fight off foreign particles in the body such as viruses. Proteins are made up of chained combinations of amino acids and do not have a fixed length, the amino acids come from the amino acid alphabet of length 20 but proteins can have any length e.g. 678, each individual protein has a unique 3D structure. Not all proteins will interact with drugs, a proteins affinity to binding with a drug is known as it's druggability, it is largely believed that if a protein is druggable then the proteins that belong to the same protein family as the protein are also druggable, it largely believed that only around 5% of all the proteins in our bodies are druggable, hence why there is currently an upward trend in biotech drugs and genome editing. Proteins always interact with other proteins so even though a protein may not be druggable, a protein that is druggable could have far reaching consequences by interacting with proteins that are not directly druggable.

Figure 1: Drug-target Interaction [10]

2.2 Biological data sets

2.2.1 DrugBank

Drugbank is a comprehensive freely accessible online database providing information about drugs and drug targets[5]. It provides an extensive set of data about approved, experimental and bio-tech drugs as well as their corresponding targets.

Drug-Target Statistics			
DRUG GROUP	DRUG TYPE	No DRUG TARGET ASSOCIATIONS	No UNIQUE TARGETS
Approved	Biotech	639	253
Approved	Small Molecule	7672	2408
Nutraceutical	Small Molecule	1065	844
Experimental	Small Molecule	7517	2577
Illicit	Small Molecule	492	94
Withdrawn	Biotech	40	37
Withdrawn	Small Molecule	408	219
Investigational	Biotech	374	201
Investigational	Small Molecule	3276	1402
		16959	4493

Table 1: Drugbank drug-target decomposition

The latest release (2018-07-03) contains 11,682 drug entries, with up to 200 associated fields for each drug, with half being devoted to chemical data and the other half devoted to protein data . Table 1 [6] shows a breakdown of drug-target interaction data available on the Drugbank website. DrugBank is consistently updated when new information is made available, and combines detailed drug (i.e. chemical, pharmacological and pharmaceutical) data with comprehensive drug target (i.e. sequence, structure, and pathway) information, it also provides links to other databases such as KEGG and Uniprot.

2.2.2 Uniprot

The Universal Protein Resource (UniProt) is a comprehensive resource for protein sequence and annotation data [1]. The data on Uniprot comprises of general annotation such as pathway, catalytic activity, cofactors and domains. The sequence annotation data consists of fields such as regions(motifs and topological domains), molecule processing(chains and peptides) and sites(active sites and metal binding). The Uniprot database also provides links to other datasets such as BindingDB, Pfam and KEGG.

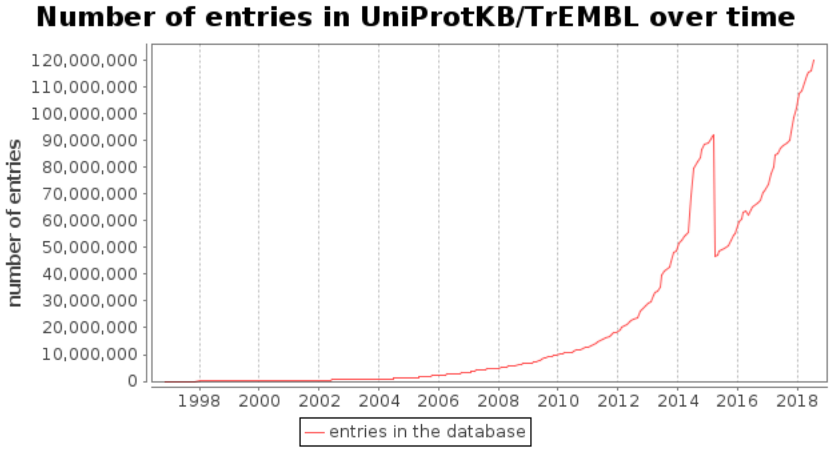


Figure 2: Uniprot entries, [1]

As you can see from figure 2, the number of entries on the Uniprot database have increased monotonically(almost exponentially) over the last 15 years. The current number of Uniprot entries as of 07-2018 is 120,243,849.

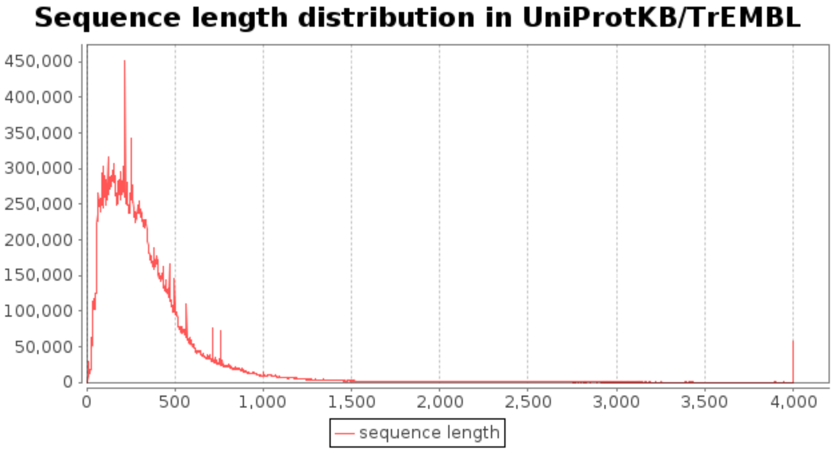


Figure 3: Sequence length statistics, [1]

As you can see from figure 3, the average sequence length of a protein entry on the Uniprot database is around 400 the distribution of sequence length is vaguely Poisson distributed although it can be seen that there is a spike for sequences that are of length 4000, there is around 50,000 proteins with a high sequence length.

2.3 Kernels

A kernel is a feature mapping $\phi : X \to \mathcal{H}$ from an object space $x, x' \in X$ to the Hilbert space $\phi(x)$, $\phi(x') \in \mathcal{H}$ such that $K(x,x'){=}\langle\phi(x), \phi(x')\rangle$ is the inner product of $\phi(x)$ and $\phi(x')$ in the Hilbert space. The kernel trick means that we do not need to use ϕ explicitly, for many kernels the feature spaces can be infinite-dimensional, we can instead use K directly. As you can see from Figure 4, by applying the kernel trick we can now find a separating hyper-plane. In order to deduce whether a potential kernel is a dot product in some space $K(x,x'){=}\phi(x) \cdot \phi(x')$ it must satisfy Mercer's conditions.

Theorem: A continuous function is a kernel $K : X^2 \to \mathbb{R}$ iff it is symmetric and non-negative definite.[20]

A kernel is symmetric if:

$$K(x,x') = \phi(x) \cdot \phi(x') = \phi(x') \cdot \phi(x) = K(x',x) \quad \text{where} \quad x, x' \in X$$

A kernel is positive definite (or non-negative definite) if for any n, any objects $x_1, \cdots, x_n$, and any numbers $a_1, \cdots, a_n$:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j K(x_i, x_j) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j \phi(x_i) \cdot \phi(x_j) =$$

$$\Big(\sum_{i=1}^{n} a_i \phi(x_i)\Big) \cdot \Big(\sum_{j=1}^{n} a_j \phi(x_j)\Big) =$$

$$\parallel \sum_{i=1}^{n} a_i \phi(x_i) \parallel \quad \geq \quad 0$$

For vectors the inner product in the Hilbert space is a measure of similarity between objects. Kernel methods are a way to turn linear methods to non-linear methods.

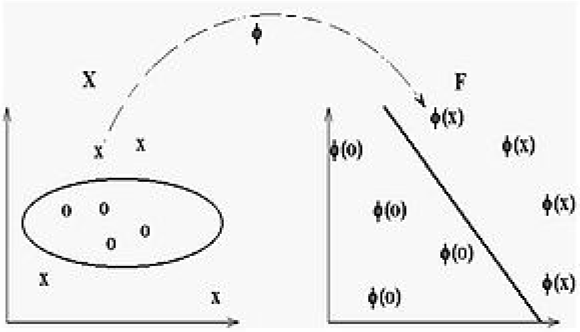


Figure 4: Left: Non-linearly separable input space, Right: Separable hyperplane once mapped to Hilbert space [3]

In order to normalize a kernel we use the following formula:

$$K_{norm}(x,x') = \frac{K(x,x')}{K(x,x) * K(x',x')}$$

$$K_{norm}(x,x) = \frac{K(x,x)}{K(x,x)^T K(x,x)}$$

$$K_{norm}(x',x') = \frac{K(x',x')}{K(x',x')^T K(x',x')}$$

where x denote training instances and x' denote test instances.

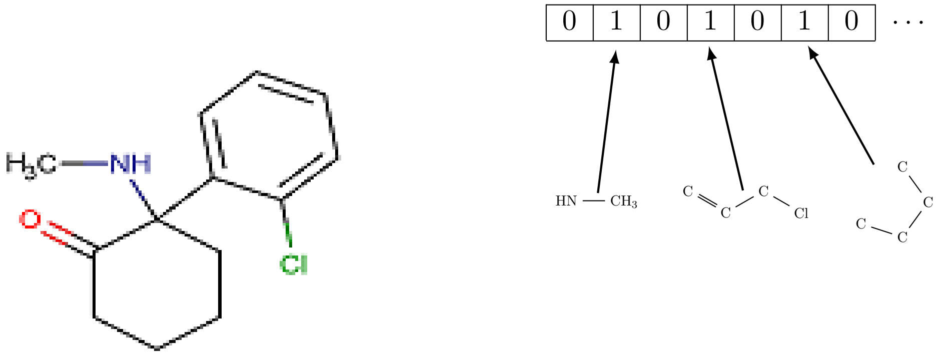
2.3.1 Tanimoto Kernel

The Tanimoto similarity(Jaccard index) is defined as:

$$J(x,x') = \frac{|x \cap x'|}{|x \cup x'|} = \frac{|x \cap x'|}{|x| + |x'| - |x \cap x'|}$$

2.3.1.1 Molecular fingerprinting

According to Ralaivola et al, 2005 [13], given a molecule in 2 dimensional form, a fingerprint is a binary vector representation of the molecule, which is typically of length 64, 512, 1024 or 2048. Consider the molecule in figure 5a, $C_{13}H_{16}ClNO$ the molecule has 32 atoms and 33 bonds. In order to create the molecules fingerprint [7] we first decide on a value for the depth e.g. d=4 and employ a depth-first search. We start from each atom(root) in the molecule and follow the bonds to the next atom along the path, we traverse all possible paths from the atom(root) through a maximum of d bonds, by doing this we get all of the molecules substructures, e.g. starting from N, we have N-C-C=O(a traversal through 3 bonds). For each substructure we compute a hash value v, the hash value v is then used as a seed number for a random number generator. The integer provided by the random number generator is then used as the position in the binary vector to be turned on(set to equal 1). If there is a clash and the position has already been set to 1 we leave the value as 1(1+1=1).



(a) 2D Molecule

(b) Molecular substructures of 5a corresponding to a Molecular Fingerprint

Figure 5: A molecule and its associated fingerprint

2.3.1.2 Tanimoto Kernel

Let x, x' denote two molecules and the feature map of x, x' be defined as $\phi_d(x), \phi_d(x')$ where $\phi_d(x), \phi_d(x')$ are extracted from a 2D molecule image using a graphical depth first search where depth=d, and as such are the molecular fingerprints described above. The tanimoto kernel is defined as:

$$K_d(x, x') = \frac{K_d(x, x')}{K_d(x, x) + K_d(x', x') - K_d(x, x')} \text{ where } K_d(x, x') = \phi_d(x) \cdot \phi_d(x')$$

Due to the fact that $\phi_d(x)$ and $\phi_d(x')$ are simply the molecular fingerprints, $K_d(x, x')$ is merely computing the ratio between $|\phi_d(x) \cap \phi_d(x')|$ and $|\phi_d(x) \cup$

$\phi_d(x')|$ where $|\phi_d(x) \cap \phi_d(x')|$ is the number of items in the intersection of $\phi_d(x)$, $\phi_d(x')$ and $|\phi_d(x) \cup \phi_d(x')|$ is the number of items in the union of $\phi_d(x)$, $\phi_d(x')$. If the feature map used is $\phi_{dl}(x)$ instead of $\phi_d(x)$ for a given $l \in \mathbb{N}$ then the corresponding kernel is the tanimoto similarity described above for vectors of fixed length l.
According to Ralaivola et al, 2005 [13], for any integer p and any set of n binary vectors $x_1, \cdots, x_n \in \mathbb{R}^p$ the corresponding similarity matrix is positive semi-definite and symmetric. The feature map associated with the tanimoto kernel are sets of binary vectors for depth d and therefore satisfies Mercer's condition meaning it is a valid kernel.

2.3.2 Mismatch Kernel

Introduced by Leslie et al, 2004 [2] the mismatch kernel is a string kernel in which we find matches between sub-strings whilst also allowing for a threshold m number of mismatches. Given two strings x, x', the mismatch kernel splits these strings into sub-strings of length p such that we have $|x| - p+1$ sub-strings, each sub-string is from x_i to x_{i+p} where $i = 1, \cdots, |x| - p+1$, this is done for both x and x'. Let Σ be the alphabet associated with the string i.e. the 26 letters of the alphabet for words or 20 letters of the amino acid alphabet for sequences. For the sub-string u associated with x the feature mapping is defined as:

$$\phi_{p,m}(u) = (\phi_v(u))_{v \in \Sigma^p}$$

where v is u with M mismatches such that $M \leq m$. For a string of length x this extends to

$$\phi_{p,m}(x) = \sum_{j=1}^{|x|-p+1} \phi_{p,m}(u_j) \text{ where } j \text{ is the } j^{th} \text{ sub-string}$$

For an alphabet of size 20 with sub-string length 5, the possible permutations are 20^5(feature space), which may result in a fairly sparse vector of length 20^5. Hence as stated above the kernel trick could not be more apt [15]. The resulting kernel is defined as:

$$K_d(x, x') =< \phi_{p,m}(x), \phi_{p,m}(x') >$$

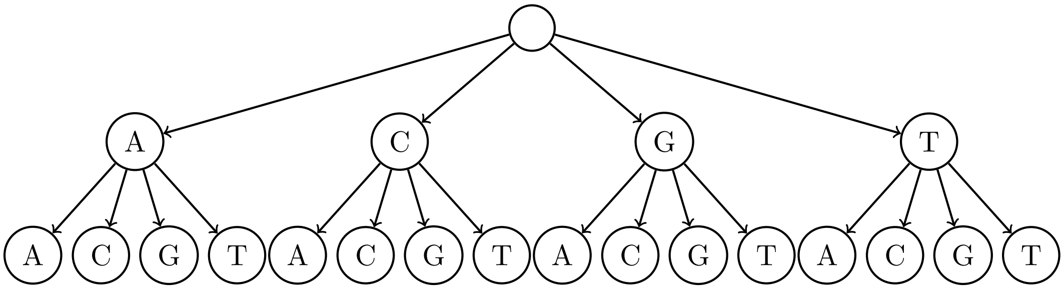


Figure 6: mismatch tree

Consider the DNA alphabet $\Sigma = \{A, C, T, G\}$ and the DNA sequence AGTA, the resulting contiguous(whereby if we add a prefix and or a suffix we can get back to the original sequence) sub-strings when k=2 are AG,GT,TA. If we decide on a mismatch value $m = 1$, we can then follow the tree denoted in figure 6.
At the root node we attach all sub-strings, we then employ a depth first search as shown in figure 7, where the values on top denote the number of mismatches. As you can see following the path AT provides us with 1 mismatch for both AG and GT where as the sub-string TA gives rise to 2 mismatches which is unacceptable therefore it provides no contribution.

$$\begin{array}{|c|c|c|}\hline 0&0&0\\\hline A&G&T\\ G&T&A\\\hline\end{array} \xrightarrow{A} \begin{array}{|c|c|c|}\hline 0&1&1\\\hline G&T&A\\\hline\end{array} \xrightarrow{T} \begin{array}{|c|c|c|}\hline 1&1&2\\\hline\end{array}$$

Figure 7: traversal from root node along the path AT

The mismatch kernel is a positive semi-definite rational kernel, a proof of this can be found in Cortes et al, 2003, [4].

2.3.3 Constructing kernels

2.3.3.1 Tensor products

According to Jacob et al, 2008, [9] the tensor product kernel is such that we can compute the joint similarity over two dimensions by evaluating the inner product between two kernels. Consider two objects x_1, x_1' defined on the space X_1 and x_2, x_2' defined on the space X_2, due to the properties of tensor products we can factorize the inner product of two tensor products as follows:

$$(\phi(x_1) \otimes \phi(x_2))^T(\phi(x_1') \otimes \phi(x_1)) = \phi(x_1')^T\phi(x_2) \times \phi(x_2)^T\phi(x_2')$$

By defining a kernel for the X_1 dimension and a kernel for the X_2 dimension:

$$K(x_1, x_1') = \phi(x_1) \cdot \phi(x_1')$$

defined on $X_1 \times X_1$

$$K(x_2, x_2') = \phi(x_2) \cdot \phi(x_2')$$

defined on $X_2 \times X_2$
We can define a tensor product kernel that takes into account similarities for two dimensions as:

$$K((x_1, x_2), (x_1', x_2')) = K(x_1, x_1') \times K(x_2, x_2')$$

defined on $(X_1 \times X_2) \times (X_1 \times X_2)$

2.3.3.2 Direct sums

The direct sum [14] is such that we can compute the joint similarity over to dimensions by evaluating the sum between two kernels. Consider two objects x_1, x_1' defined on the space X_1 and x_2, x_2' defined on the space X_2, due to the properties of direct sums we can factorize out the sum of two direct sums as follows:

$$(\phi(x_1) \oplus \phi(x_2))^T(\phi(x_1') \oplus \phi(x_2')) = \phi(x_1)^T\phi(x_1') + \phi(x_2)^T\phi(x_2')$$

By defining a kernel for the X_1 dimension and a kernel for the X_2 dimension:

$$K(x_1, x_1') = \phi(x_1) \cdot \phi(x_1')$$

defined on $X_1 \times X_1$

$$K(x_2, x_2') = \phi(x_2) \cdot \phi(x_2')$$

defined on $X_2 \times X_2$
We can define a direct sum kernel that takes into account similarities for two dimensions as:

$$K((x_1, x_2), (x_1', x_2')) = K(x_1, x_1') + K(x_2, x_2')$$

defined on $(X_1 \times X_2) \times (X_1 \times X_2)$

2.3.3.3 Polynomial combination kernel

The polynomial kernel is defined as:

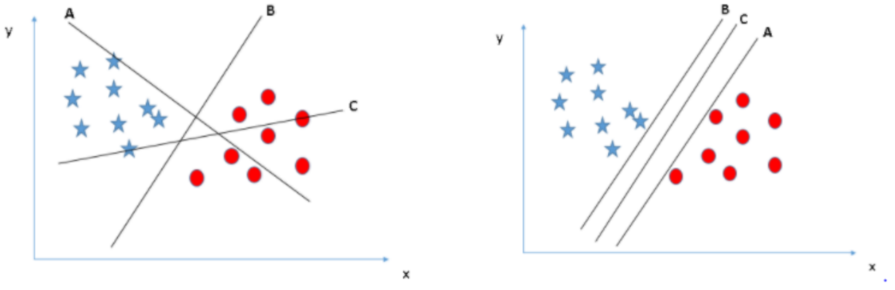
$$K(x, x') = (c + (x \cdot x'))^d$$

where c is an additive constant and d denotes the degree of the polynomial, a larger value of d increases the flexibility of the polynomial but may result in overfitting. As you can see from the above formula, we are computing the dot product between x and x' therefore we can apply the kernel trick and replace this with a kernel, therefore we can define a new kernel as:

$$K_2(x, x') = (c + K_1(x, x'))^d$$

2.4 Support Vector Machines

First introduced by Vapnik and Chervonenkis in 1963, the support vector machine method aims to find the optimal hyper-plane for linearly separable patterns, such that a test object x^* is assigned a label based on which side of the hyper-plane it is located. In the 2-D space we can separate the classes using a line and in higher dimensions we require a hyper-plane. For the set of all possible hyper-planes that can separate the data there exists a unique optimal hyper-plane which can be found by considering the maximum margin of separation between the training examples and the hyper-plane.



(a) hyper-planes

(b) separating hyper-planes

Figure 8: Hyper-planes [19]

Consider figure 8a, we need to find the hyper-plane that is able to separate the classes from the possible options B is the only one that can. Now consider figure 8b all three of the hyper-planes separate the data so which one do we choose.

2.4.1 Maximal Margin Classifier

By assuming that the binary classification problem is linearly separable, we can define the maximal margin classifier for a training set consisting of features x_i and labels $y_i \in \{-1, 1\}$ where $i = 1, \cdots, m$, in order to construct the optimal hyper-plane that divides the classes:

$$(x_i * w) + b = 0$$

As seen in figure 8b there are many separating hyper-planes but we do not want to pick one that is closer to one class and further from another as that would generalize badly and we would have less confidence in our predictions. In order to find the optimal hyper-plane we compute the perpendicular distance from each training object to a given separating hyper-plane, the smallest of these distances is called the margin. The maximal margin hyper-plane is the separating hyper-plane for which the margin is the largest.

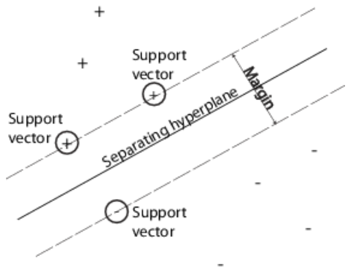


Figure 9: maximal margin hyper-plane [11]

We calculate distance from a data point x_i to the separating hyper-plane using:

$$r = \frac{w * x_i + b}{\parallel w \parallel}$$

The points equidistant from the maximal margin hyper-plane on the dashed lines are called the support vectors x if any of these points were moved the maximal margin hyper-plane would also move, therefore it can be seen that maximal margin hyper-plane is only dependent on these points if any other points were moved unless they crossed the boundary the maximal margin hyper-plane would not be affected. As you can see from figure 9 the margin is the distance between the support vectors of different classes. By normalizing the equation of the support vectors $x * w + b = 1$ or $x * w + b = -1$ we can define the optimal margin as:

$$m = \frac{2}{\parallel w \parallel}$$

therefore maximizing the margin is equivalent to minimizing the norm. There exists a weight vector w and threshold b such that the following constraints are satisfied:

$$(x_i * w) + b \geq 1 \text{ if } y_i = 1 \tag{1a}$$

$$(x_i * w) + b \leq -1 \text{ if } y_i = -1 \tag{1b}$$

and the vector w has the smallest norm

$$\parallel w \parallel = w * w$$

therefore to construct the optimal hyper-plane we must solve:

$$\text{minimize } \frac{1}{2} \parallel w \parallel^2$$

subject to:

$$y_i((x_i * w) + b) \geq 1 \; i = 1, \cdots, m$$

which is equivalent to constraints 1a and 1b.
It can be seen that we want to minimize the quadratic form subject to linear constraints. This can be solved in the primal space (space of parameters w and b) or in the dual space (space of Lagrange multipliers). In order to solve the quadratic optimization problem in the dual space, we need to find the saddle point of the Lagrange function:

$$L(w, b, \alpha) = \frac{1}{2} \parallel w \parallel^2 - \sum_{i=1}^{m} \alpha_i (y_i[(x_i * w) + b] - 1)$$

where $\alpha_i \geq 0$ are the Lagrange multipliers. The Lagrangian L must be minimized w.r.t the primal variables w and b and maximized w.r.t the non-negative dual variables α_i. At the saddle point the derivatives w.r.t the primal variables must vanish according to the Fermat theorem:

$$\frac{\partial L(w, b, \alpha)}{\partial w} = w - \sum_{i=1}^{m} y_i \alpha_i x_i = 0$$

$$\frac{\partial L(w, b, \alpha)}{\partial b} = \sum_{i=1}^{m} y_i \alpha_i = 0$$

which leads to:

$$w = \sum_{i=1}^{m} y_i \alpha_i x_i$$

$$\sum_{i=1}^{m} y_i \alpha_i = 0$$

we can substitute the above equation into the Lagrangian as follows:

$$L(w, b, \alpha) = \frac{1}{2} \sum_{i=1}^{m} y_i \alpha_i x_i \sum_{j=1}^{m} y_j \alpha_j x_j + \sum_{i=1}^{m} \alpha_i$$

$$- \sum_{i=1}^{m} \alpha_i y_i \sum_{j=1}^{m} y_j \alpha_j x_i * x_j - b \sum_{i=1}^{m} y_i \alpha_i$$

given that $\sum_{i=1}^{m} y_i \alpha_i = 0$

$$-b \sum_{i=1}^{m} y_i \alpha_i = 0$$

and by re-arranging the terms we get:

$$W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j (x_i * x_j)$$

we must now maximize the above equation subject to $\alpha_i \geq 0$ for all $i = 1, \cdots, m$ and $\sum_{i=1}^{m} y_i \alpha_i = 0$, we therefore obtain the solution:

$$w = \sum_{i=1}^{m} y_i \alpha_i x_i$$

The value of b is chosen to maximize the margin, it can be obtained from a positive support vector $w * x + b = 1$. The optimal solution w and b must satisfy the Karush-Kuhn-Tucker(KKT) conditions:

$$\alpha_i (y_i ((x_i * w) + b) - 1) = 0$$

from this we know that many of the α_i are zero and w is a linear combination of just a few examples, the examples with non-zero α_i are the support vectors x, this means we obtain a sparse representation of the data.
The hyper-plane decision function for a new object x_n is therefore:

$$f(x, \alpha) = \sum_{i} y_i \alpha_i x_n * x_i + b \text{ where } i \in \text{ support vectors}$$

2.4.2 Soft Margin Classifier(SVC)

The above maximal margin classifier(hard margin classifier) is very sensitive to noise and outliers, what happens if the data is not separable or if a maximal margin hyper-plane does exist but due to bad generalization we don't want use it, in 1995 Cortes and Vapnik [17] introduced the soft margin classifier.

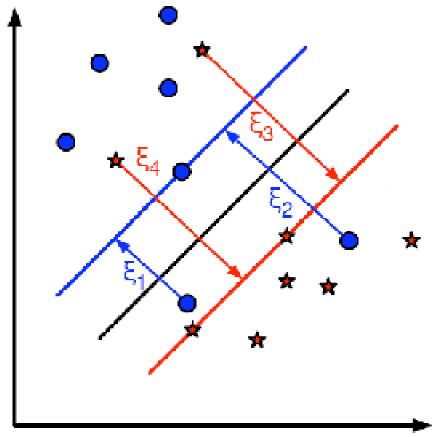


Figure 10: linearly non-separable hyper-plane [16]

If we look at figure 10 the data is non-separable i.e. there is no hyper-plane that can perfectly separate the two classes, we would therefore not be able to find a maximal margin hyper-plane but we can instead use the soft margin classifier. In order to use the soft margin classifier we want to relax the conditions of the hard margin classifier so that we are allowed to make some classification errors which overall may lead to better generalization. We do this by introducing the slack variable($\zeta_i \geq 0$) which represent the deviation of the examples from the margin i.e. in figure 10 it is the distance from the data point to support vectors where it would be correctly classified and the function:

$$F(\zeta) = \sum_{i=1}^{m} \zeta_i$$

In order to minimize the error, we can minimize $\sum_i \zeta_i$ subject to the following constraints:

$$(x_i * w) + b \geq 1 - \zeta_i \text{ if } y_i = 1 \tag{2a}$$

$$(x_i * w) + b \leq -1 + \zeta_i \text{ if } y_i = -1 \tag{2b}$$

$$\zeta_i \geq 0 \tag{2c}$$

The number of support vectors and slack variables give an upper bound of the leave one out error. In order to introduce the slack variable to the original optimization problem, we need to add an additional parameter C to control the trade off between the error and the margin. The optimization problem in the primal form then becomes:

$$\text{minimize } \frac{1}{2} \| w \|^2 + C \sum_{i=1}^{m} \zeta_i$$

subject to:

$$y_i((x_i * w) + b) \geq 1 - \zeta_i \text{ for all } i = 1, \cdots, m \text{ and } \zeta_i \geq 0$$

Transforming this to the dual problem we obtain:

$$W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j (x_i * x_j)$$

we must now maximize the above equation subject to $C \geq \alpha_i \geq 0$ for all $i = 1, \cdots, m$ and $\sum_{i=1}^{m} y_i \alpha_i = 0$, we therefore obtain the solution:

$$w = \sum_i y_i \alpha_i x_i \text{ where } i \in \text{ support vectors}$$

as you can see the only difference between the two solutions is that α_i now has an upper bound C. We can alter the parameter C such that a large value corresponds to a harder margin and a smaller value of C tolerates more training errors.

2.4.3 Non-linear Hyper-planes

We have previously only discussed data that is linearly separable but what about data that isn't. As shown in figure 4 by using the kernel trick we can project the original data into a higher dimensional feature space using a mapping ϕ so that even though the data may be non-linear in the input space it is linear in the feature space. As the above explanation of the dual form optimization problem only involves dot products we can replace the dot products with any valid kernel such that we now have the decision function:

$$f(x, \alpha) = \sum_i y_i \alpha_i \phi(x_n) * \phi(x_i) + b = \sum_i y_i \alpha_i K(x_n, x_i) + b$$

and the optimization problem becomes:

$$W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j K(x_i, x_j)$$

subject to $C \geq \alpha_i \geq 0$ for all $i = 1, \cdots, m$ and $\sum_{i=1}^{m} y_i \alpha_i = 0$
It should also be noted that the SVM method is computationally expensive in the training phase but once we know what the support vectors are, we can make predictions efficiently.

2.5 Performance Measures

We will be predicting whether a drug interacts with a target or not, by re-defining the problem as a binary classification, an appropriate way to measure the success of the method is to use a ROC curve and a precision recall curve which will allow us to visualize the performance of our method.

2.5.1 Confusion Matrix

A confusion matrix is a good way to visualize the performance of a classifier as it allows us to represent successes and failures of the classifier with respect to the real label. The rows in the confusion matrix denote the predicted classes and the columns denote the actual class.

A binary classifier can make two types of errors, it can either predict a label as positive when the real label is negative, meaning a relationship does exist between a drug and target when in fact this relationship does not exist, or it can predict a label as negative when the real label is positive, meaning that a relationship between drug and target does not exist when in fact it does. From table [6] below, the first type of error is denoted by the letters FP(false positive) and the second type of error is denoted by FN(false negative).

		True Label		
		Positive	Negative	Total
Predicted Label	Positive	TP	FP	$TP + FP$
	Negative	FN	TN	$FN + TN$
	Total	$TP + FN$	$FP + TN$	$Total$

Table 2: Confusion Matrix

Sensitivity(true positive rate or recall) is defined as the proportion of instances correctly identified as positive(true positive) out of all instances that actually belong to the negative class(Positive column total):

$$TPR = \frac{TP}{TP + FN}$$

Specificity (true negative rate) is defined as the proportion of instances correctly identified as negative(true negative) out of all instances that actually belong to the negative class(negative column total):

$$TNR = \frac{TN}{FP + TN}$$

Precision(positive predictive value) is defined as the proportion of instances correctly identified as positive(true positive) out of all instances that were predicted as positive(positive row total):

$$PPV = \frac{TP}{TP + FP}$$

Accuracy is defined as the proportion of instances that were correctly identified by the classifier(trues) out of all instances(N):

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} = \frac{TP + TN}{Total}$$

2.5.2 ROC Curves

A ROC curve allows us to see how well our model can differentiate between positive and negative instances. It is a way to graphically represent the trade-off between the specificity(TNR) and the sensitivity(TPR) for a given classification threshold.
By lowering the threshold more instances are classified as positive, which consequently increases the sensitivity(TPR) and decreases the specificity(TNR). By increasing the threshold more instances are classified as negative, which consequently increases the specificity(TNR) and decreases the sensitivity(TPR).
When plotting the ROC curve, instead of plotting the specificity we plot 1-specificity which can be shown to be the false positive rate(number of instances incorrectly identified as positive):

$$1 - TNR = 1 - \frac{TN}{FP + TN} = \frac{FP + TN}{FP + TN} - \frac{TN}{FP + TN} = \frac{FP}{FP + TN} = FPR$$

we do this because it allows us to simply look at the positives, when increasing the threshold both FPR and TPR increase, when decreasing the threshold both TPR and FPR decrease. The ROC curve is created by plotting the true positive rate against the false positive rate for different threshold values e.g. for a threshold value of 0.5, the true positive rate = 0.67 and the false positive rate=0.22, we therefore plot the point (0.22,0.67). This is done for all possible threshold values as seen in figure 11

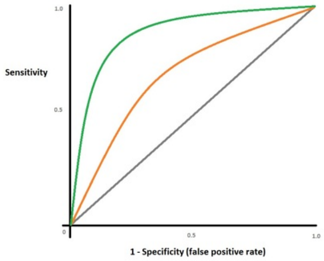


Figure 11: ROC curve [12]

We are looking for points where the sensitivity is high and 1-specificity is low i.e. the closer the classifier is to the upper left quadrant the better it is, a perfect classifier would be a vertical line from the origin up to 1 (0,0) to (0,1) followed by a horizontal line (0,1) to (1,1). Presuming the orange line is classifier 1(C1) and the green line is classifier 2(C2) we can see that for all possible thresholds C2 outperforms C1. We can also use the figure to find the optimal threshold which again would be the point closest to the upper left quadrant. Additionally the line through the origin is known as the no information classifier in which the classification is chosen at random instead of using a classification model, if your model is below this line then it is severely under performing and it is better to assign items randomly.
The area under the ROC curve (auROC) is a performance measure which allows us to quantify how well our model differentiates between the two classes, it enables us to measure performance across all possible thresholds. The higher the auROC the better the model, it can easily be seen that the area under to ROC curve for the perfect classifier is 1 and that of the no information classifier is 0.5.

2.5.3 Precision-Recall Curves

The precision-recall curve is a plot of precision(PPV) against recall(TPR) where precision and recall are defined above, it is a way to graphically represent the trade-off between the true positive rate and the positive predictive value for a given classification threshold. Instead of just looking at a ROC curve we also need to look at the precision recall curve in order to get a more complete picture as to how our model is performing. In many cases there may be a class imbalance and we may care more about a particular class, in this case it is very important to look at the precision recall curve. Whereas the ROC curve is monotonically increasing the precision recall curve is monotonically decreasing as seen in figure . The no information classifier would yield a horizontal straight line such that precision does not change as recall increases. A perfect classifier is such that we have a horizontal line from (0,1) to (1,1) followed by a vertical line from (1,1) to (1,no information line). The precision-recall curve allows us to see the relationship between, the proportion of instances of the class we care most that were successfully detected versus all the instances that we classified as the class we care most about for different possible thresholds. We can also calculate the average precision which is the weighted mean of the precision's achieved at each threshold where the weight is the increase in recall from the previous threshold. A large area under the curve indicates both high precision and high recall, this means that we are correctly classifying most instances of the class we care about and we are not incorrectly classifying the instances of the class we care about.

3 Method

The main idea behind this project is similarities, if two proteins are similar they may be targeted by the same drug and if two drugs are similar they may target the same protein.

3.1 Data

In order to solve any problem associated with drug re-positioning, we must first deduce what data to obtain and to what end we would like to use this data. Obtaining the appropriate data and having a good understanding of the data is paramount to any exercise connected to machine learning. As outlined above data was taken from Drugbank and Uniprot wherein an abundance of data can be found.

3.1.1 Drugbank

In order to take any data from Drugbank you must first create an account, the data available is for non-commercial use only, Drugbank does in fact have a web API but in order to use it you must have a Drugbank plus account.

Initially due to being unsure of what data to use, I downloaded all of the data available on the Drugbank website and extracted data relating to drugs, drug interactions, pathways, targets and enzymes. I used the xml tree implementation in python to extract information from this data set which involved traversing through different elements and their child elements to extract data.

Once I had more of an understanding of the problem and how to proceed, I simply downloaded CSV files from Drugbank, one of which contained drug ids, drug names and simplified molecular-input line-entry system (SMILES) representations of drugs. The other contained the targets associated with each drug and the Uniprot ID's which can be used to obtain additional information about targets from the Uniprot database.

The SMILES representation of a molecule is a line notation of ASCII characters that allow efficient storage of molecular structure, the SMILES representation is obtained through a depth first traversal of a chemical graph. An example of the SMILES representation of molecule is as follows: [H][C@@]12CC3=CNC4=CC=CC(=C34)C1=C[C@H](CN2C)C(=O)N(CC) where the brackets [ ] enclose atoms other than common atoms, { -, =, #, \, /} and a few more not shown in the representation above denote bonds and () denote branches in the molecule. I initially decided to take the FASTA format sequences from Drugbank but as it was no longer appropriate for what I planned on doing I decided against using it. This data was stored in a pandas dataframe which allows storage of labelled data of different types. I used the pandas join method to create a single table from all the data in which the joining key was the drug id's.

3.1.2 Uniprot

The Uniprot database provides a Web API for efficient data collection, the programmatic access pages on Uniprot provide in depth information on how to use the Web API. The Uniprot database is well managed in the sense that if you have any difficulties taking the data that you want they are very easily reachable by email.

The data obtained from the Uniprot database consisted of just protein

sequences. Originally I had taken data relating to Pfam ID in order to group proteins by the family that they were a part of, I also took the GO notation intending to create a tree of the GO notation to find unions and intersections between the GO notations of different targets. Once I'd opted for a kernel based method this was no longer a logical way to approach the problem as using a kernel in this way would've required me to prove that it was symmetric and positive semi definite. An example of a short protein sequence:
TDHCWFGWYWYRVAAGPAMKWLRKGNWMSDDEFYYAWYGISY
As you can see applying a kernel to sequence data, involves looking at kernels for structured data as protein sequences have more of a pattern than words but are still strings and therefore any kernel for similarity of sequences must be appropriate for the task. The data for protein sequences was concatenated with data from Drugbank

3.2 Data Generation

In order to formulate the problem as that of a binary classification task, negative data must be generated as negative data does not already exist in the data sets. Many of the research papers I read surrounding this task had used the simple method of randomly sampling drugs and randomly sampling targets to create new drug-target interactions. If the data didn't exist already this data was issued a negative label. This method may be slightly problematic as the main purpose of this task is to find relations that we don't already know exist. The true label for some of the drug targets may in fact be positive but by already assigning them as negative we introduce some bias in our data.
I did think about using a graphical approach to accurately deduce whether drug-target interactions would exist or not, using the idea that if to drug-targets were far away from each other in the graph then they would never interact i.e. in terms of social media analytics, if two people have nothing in common they most likely never interact. I did try to find information relating to this approach of generating negative data, but I could not find any relevant documentation. I then realized, that using data that was either very clearly positive or very clearly negative may lead to bad generalization in the testing phase so I opted for a more traditional approach for data generation using random sampling.

3.3 Tanimoto Kernel

For chemical similarity I used the tanimoto kernel, in order to use the tanimoto kernel we first need to generate molecular fingerprints, the python package RDKit provides a great implementation for reading in SMILES data and generating molecular graphs to create a molecule object type. RDKit has a method called Pandas Tools which allows you to append your dataframe with a new column for molecule objects, which was perfect as my data was already in a dataframe but initially the versions of pandas and RDKit were incompatible, this required going into the source code of the Pandas Tools method to change the older methods of Pandas being used to the current methods of Pandas. Once I changed from Python 2.7 to Python 3.6 this was no longer the case and I could use the method without a problem.

The package also provides many different types of fingerprinting techniques such as RDKit fingerprints, Avalon fingerprints and Morgan fingerprints, through reading a lot of material I found that the majority of the fingerprint techniques produce similar outputs therefore the choice of fingerprinting technique can be made arbitrarily, I opted for the Morgan fingerprint technique. The Morgan fingerprint technique unlike the method explained above, uses the neighbourhood of an atom instead of a depth first search. It goes through each atom of the molecule and produces all paths through the atom for a specific upper bound radius, I decided to choose a radius of 2.

As stated above the tanimoto kernel is valid and the same as tanimoto similarity when the feature vectors are of the same length, the RDKit implementation allows us to set the number of bits we would like, having a low number of bits such as 32 may result in low discriminative power therefore I chose to use 1024 bits, in order to not lose too much information. RDKit also has an implementation of tanimoto similarity, which takes in two molecular fingerprints and outputs the similarity, this method was used to compute the tanimoto kernel, the kernel does not require centering.

3.4 Mismatch Kernel

For sequence similarity, the main method used in the biological community is sequence alignment, there are two different types of sequence alignment, global and local. For global alignment(Needleman-Wunsch algorithm) you take the entirety of the two sequences into consideration, when the two sequences have different lengths this can be very inappropriate, there would be many gaps in the alignment and even if 1 sequence contained the entirety of the other sequence due to the difference in lengths we would not have a good alignment score. For local alignment(Smith-Waterman algorithm) we are looking for local regions that are similar, this is a more appropriate procedure for my use of similarities.

The Smith-Waterman algorithm is not a valid kernel in most cases but vert et al, 2004, [18] created a local alignment kernel which I initially tried to implement, the kernel does have some issues with diagonal dominance but I couldn't seem to get the code to work, after reading Jacob et al, 2008,[9] and seeing that in this setting the kernel performed the worst, I started to look into other kernels.

I looked at both the fisher kernel and the mismatch/spectrum kernel, the fisher kernel assumes a prior distribution and is based on a HMM probabilistic model, for applications such as protein classification, the fisher kernel is very good, but for my application it didn't seem appropriate.

The mismatch kernel as described above was the kernel I finally settled on, naively I assumed implementing this kernel would be fairly straightforward, with not nearly enough documentation relating to implementations, it was hard to understand. For fast implementation a trie or suffix tree approach are best, but the python packages for tries did not provide the relevant methods for application to the mismatch kernel, I then used and rewrote portions of a basic online implementation of a trie in order to use it for this application, but this again didn't work.

Dr Watkins informed me that the method can be implemented using set intersections and due to the fact that pythons set implementations are fairly quick this is what I used. As I wasn't entirely sure of my implementation I again spoke to Dr Watkins who told me to check the eigenvalues and perform singular value decomposition to check it was a valid kernel. All the eigenvalues were positive and the singular value decomposition did not return negative values therefore it was in fact a valid kernel.

3.5 Constructing Kernels

To construct kernels based on the maths explained above I used pythons numpy package to perform simple matrix operations, an important thing to note is kernel normalization and the introduction of an intercept addend to avoid failures when values are 0.

3.6 SVM

There are many different packages in python to perform SVM classification, my requirements for the package involved being able to either define my own kernel function or the ability to use a pre-computed gram matrix, both Sci-kit learn and LIBSVM provide functionality to do this. As I needed to centre my kernels the main functionality required was inputting a pre-computed gram. The LIBSVM package requires adding ID's as the first element for each instance. Initially the LIBSVM package was chosen as I wanted to apply conformal prediction to my method and hence the support vectors would be required, sci-kit learn on the other hand does not output support vectors for pre-computed grams, but as time went on I realized that I would not have time to implement conformal prediction, so I instead opted to use sci-kit learn, due to it being easier to use and it also had great documentation explaining the maths behind the methods being used.
Sci-kit learn also provides options to output predicted probabilities which are invaluable when generating ROC and precision-recall curves.

3.7 Performance Measures

Sci-kit learn has a method called metrics, which has classes for computing confusion matrices, ROC curves and precision recall curves.

4 Results

4.1 Tensor Product Kernel

The prediction accuracy achieved by the tensor product kernel is 63.1% when altering the cost parameter from the default value of 1, either by increasing or decreasing the value of C the accuracy reduces.

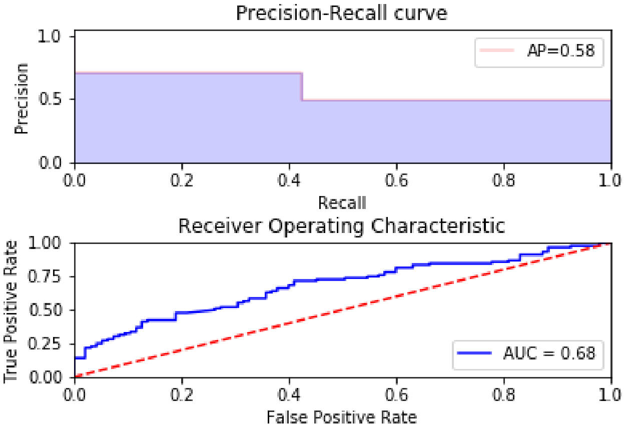


Figure 12: Tensor Product Kernel

The area under the ROC curve shows the performance across all thresholds, the AUC was calculated to be 0.68, we can see that the model does perform better than the no-information classifier, but the precision recall curve shows that it often incorrectly classifies instances where there isn't a drug-target interaction as having an interaction.

		True Label		
		Positive	Negative	Total
Predicted Label	Positive	39	16	$39 + 16$
	Negative	53	79	$53 + 79$
	Total	$39 + 53$	$16 + 79$	187

Table 3: Tensor Product Kernel Confusion Matrix

4.2 Polynomial Tensor Kernel

Parameter search was employed by performing a grid-wise search for different possible parameters and finding out what the prediction accuracy was. A few examples are given below to indicate the trends that existed in the parameter search, for values greater than c=1 the accuracy became progressively worse. Between $1 \geq c \geq 0.5$ the accuracy when keeping d constant resulted in an increase in accuracy and between $0.5 \geq c \geq 1$ the accuracy again started to worsen . Changing the value of d whilst keeping c constant

c	**d**	Accuracy
1	2	58.8%
4	2	50.8%
0.8	2	60.4%
0.6	2	61%
0.4	2	60.4%
0.6	4	61%
0.6	5	59.9%
0.6	1	58.3%

Table 4: Polynomial Tensor Kernel parameter search

gave no clear indication of a pattern in the change of accuracy. As you can see from table 4 the best accuracy was achieved by:

$$K(x, x') = (0.6 + K_{tanimoto}(x_1, x'_1)) * K_{mismatch}(x_2, x'_2))^4$$

but this was exceeded by the Tensor product kernel.

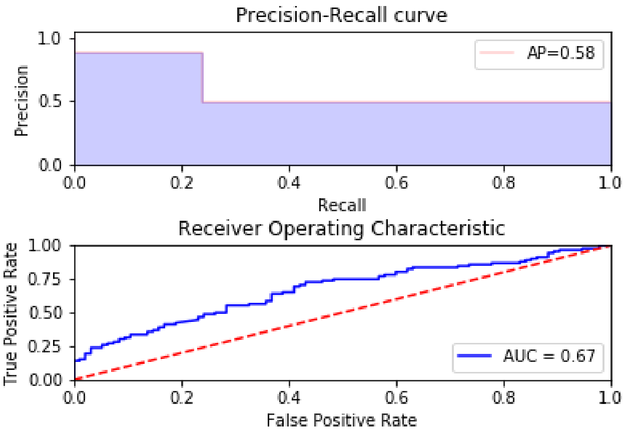


Figure 13: Polynomial Tensor Kernel

The area under the ROC curve shows the performance across all thresholds, the AUC was calculated to be 0.67, we can see that the model does perform better than the no-information classifier but it also behaves very similarly if not slightly worse than the tensor product kernel.the precision recall curve shows that it often incorrectly classifies instances where there isn't a drug-target interaction as having an interaction, the precision recall plot proves that both methods produce similar outputs, with the average precision value across all thresholds being the same as that of the tensor product kernel. This implies that the discriminating power is not increased when applying a polynomial transformation to the tensor product kernel.

		True Label		
		Positive	Negative	Total
Predicted Label	Positive	22	3	$22 + 3$
	Negative	70	92	$70 + 92$
	Total	$22 + 70$	$3 + 92$	187

Table 5: polynomial tensor kernel Confusion Matrix

4.3 Direct Sum Kernel

The accuracy produced by this method was 51.3% with the cost parameter c=1 whereas for the cost parameter c=0.5 the accuracy increases to 54.5%. The following plot is for the cost parameter c=1:

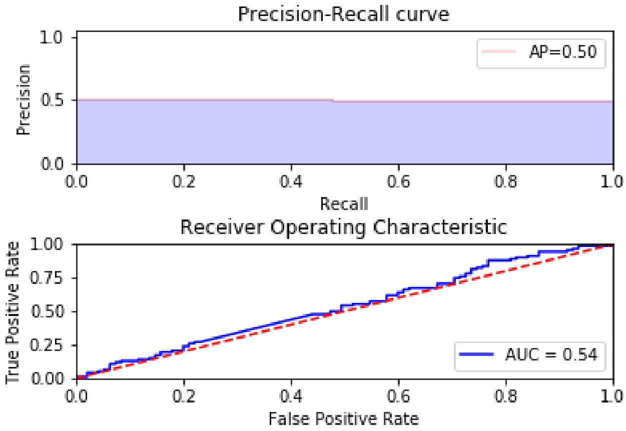


Figure 14: Direct Sum Kernel

As you can see the direct sum kernel provides none of the discriminating power that is required for this methods, the precision recall curve has an average precision of 0.5 and the area under the ROC curve is 0.54 the direct sum method does not seem to work well at all, but what about the case where we add in an additional kernel namely the polynomial kernel, will this improve our results.

		True Label		
		Positive	Negative	Total
Predicted Label	Positive	44	43	44 + 43
	Negative	48	52	48 + 52
	Total	44 + 48	43 + 52	187

Table 6: Direct Sum Kernel Confusion Matrix

4.4 Polynomial Direct Kernel

c	**d**	**Accuracy**
1	2	53.5%
4	2	51.8%
6	2	53.5%
0.8	2	52.9%
0.6	2	53.5%
3	2	54%
3	3	55.1%
3	5	56.1%
3	7	55.6%
3	1	54%

Table 7: Polynomial Direct Kernel parameter search

Parameter search was employed by performing a grid-wise search for different possible parameters and finding out what the prediction accuracy was. A few examples are given above to indicate the trends or lack there of, that existed in the parameter search. As you can see from table 4 the best accuracy was achieved by:

$$K(x, x') = (3 + K_{tanimoto}(x_1, x'_1)) + K_{mismatch}(x_2, x'_2))^5$$

which is markedly better than the direct sum kernel but far worse than the tensor product kernel, the increase in accuracy could be due to overfitting as the degree of the polynomial is fairly high.

		True Label		
		Positive	Negative	Total
Predicted Label	Positive	42	32	$42 + 32$
	Negative	50	63	$50 + 63$
	Total	$42 + 50$	$32 + 63$	187

Table 8: Polynomial Direct kernel Confusion Matrix

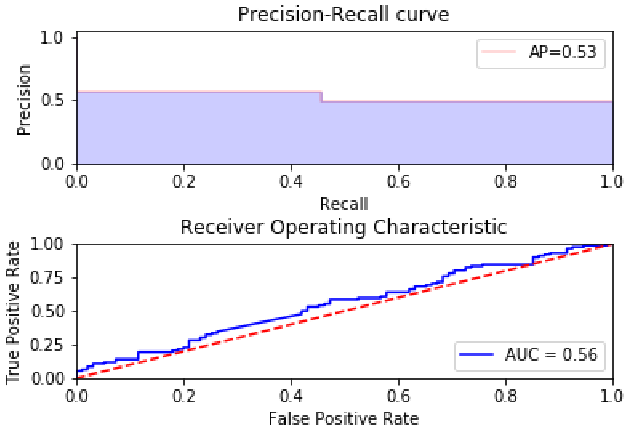


Figure 15: Polynomial Direct Kernel

For the tensor product kernel and the polynomial tensor kernel changing the cost parameter always resulted in poorer discriminating power, whereas for the direct sum kernel and polynomial direct kernel decreasing the cost parameter to 0.5 resulted in better performance but the same could be achieved by altering the parameters of the polynomial.

5 Conclusion

Overall it is clear to see that the direct sum kernel applied to the field of drug-target predictions has approximately no ability to discern between the two classes, whereas the tensor product kernel has slightly more ability but it is nowhere near the state-of-the-art. For a solution that would yield better results please look towards deep learning as even though this methodology has a more solid foundation in terms of the mathematics than deep learning

does it clearly is not appropriate.
The results suggest that this methodology is not appropriate for the task but it could be for any number of reasons, are the kernels for structured data not performing as well as they should, has applying many kernels on top of each other convoluted the data so much that we were unable to make good predictions or was it simply due to the way in which I generated the data.

6 Professional Issues

Machine Learning as applied to the biological sector creates many situations in which we may inadvertently cause harm to others, in the case of the drug thalidomide, which was originally meant to deal with anxiety and insomnia but later it was re-purposed to treat morning sickness in pregnant women. Many of the pregnant women gave birth to children with malformed limbs, around 5000 children died. Not taking into account risk factors and solely relying on computational efforts that may not be taking into account things like molecular positioning(racemic mixtures as in the case of thalidomide) as is the case with the tanimoto kernel, therefore this must be something that we are constantly weary of, and we should work towards minimizing adverse risks.
We should never try to misrepresent our results and pretend that they are more favourable than they actually are, we should aim to care about more than just the speed and accuracy of our results and instead try to look at the far reaching implications of what we are doing. More importantly data can come from many sources, i.e. patient data in clinical trials and as professionals we should never knowingly violate the privacy of others and we should aim to advise patients on how their data may be used.
Plagiarism is when you pass off someone else's ideas or work as your own, when writing a report it is very important not to take someone else's work without giving them their due credit.
Most of the data I used in this project come with disclaimers, they can not be used for commercial gain and many of them state that if you are using them for research purposes, they need to be appropriately cited failing to do so may result in legal action. During my project I wanted access to the OMIM disease database, but in order to gain access you must register for an account and once your request has been approved you are granted access, they sent me an email letting me know my request was being processed but I am still waiting for a response 2 and a half months later. I could have

very easily obtained the data set from the supplementary files associated with the work of others but I didn't because that would have been a violation of OMIM's terms and conditions, it is not a publicly available data set, so until I was granted access it wouldn't have been my data to work with.

7 Self-Assessment

I really enjoyed using latex especially the tikz package because a lot of the times I couldn't find a picture that accurately illustrated what I wanted to show, therefore creating my own images i.e. figure 5b and figure 6, to aid my explanations was really helpful. Creating your own images also has the added benefit of not violating any copyrights on images and allows you to avoid plagiarism issues.

Some of the main methods I used such as valid kernels and SVM's have a solid mathematical basis around them, it was interesting to go more in depth into material I had previously only touched upon.

I really struggled to understand how to solve my problem, not having a biology background I struggled to understand papers relating to the field as many of them used complex biological terms, and googling them just resulted in more biological terms I didn't understand. Doing an end to end project in a field I knew nothing about resulted in a lot of wrong turns. I tried to learn as much biology as possible in a short space of time but it didn't really help so I instead thought about the problem in terms of machine learning, instead of searching for papers in the domain of biology and drug re-positioning, I started searching for machine learning methods that have applications to biology.

Dealing with structured data was another area I hadn't previously had much experience with apart from in a deep learning setting where there exists some very impressive methods for dealing with it. Most of the techniques I previously knew about were not appropriate for structured data, so I feel I've learnt a lot from this project i.e. graph kernels. I also had no knowledge whatsoever about the existence of multiple kernel learning.

Time management really hindered my progress, I often go on tangents whether they are small or large, whether they help to improve my methodology or not, for an end to end project there are so many avenues you can take, I found it really hard to stick to one methodology and run with it, especially when confronted with some issues in the methodology and having the knowledge that there were other better ways that wouldn't have the

same issues.
I'd be excited to try different kernel approaches and maybe create some kernels of my own and see how they perform in the biological domain, I'd also like to try to combine more features once I have more of an understanding of there effect when applied to this problem.
As you may be able to tell none of the methods used are computationally efficient and therefore may be difficult to scale, I would love to shift to an online setting and use parallel computations to truly see the performance of the different models I create. More data often leads to more accurate predictions, I was only able to use 5% of the data available, the data was randomly sampled therefore it wasn't really indicative of the larger set.
I would like to also employ conformal predictors, having already done all the relevant research to find appropriate non-conformity measures for algorithms such as SVM's and kernel KNN's, I'd really like to see whether having confidence in the predictions improves the results.
Most importantly I've learnt a lot from this project and from that standpoint I couldn't be happier, I strongly believe machine learning applications to biological problems can lead to great advances and I'd love to participate in this field more.

8 How to use my project

Step 1) Create an account on Drugbank to download the relevant datasets, which are as follows:
Step 1a) External links− >Target Drug-UniProt Links− >approved
Step 1b) Structures− >Structure External Links− >approved
Step 1c)Save data to current working directory
Step 2)Run scripts in the following order:
dataprocessing.py
datageneration.py
mismatchkernel.py
constructing_kernels_1.py
SVM.py
constructing_kernels_2.py

References

[1] Chuming Chen, Hongzhan Huang, and Cathy H. Wu. *Protein Bioinformatics Databases and Resources*, pages 3–39. Springer New York,

New York, NY, 2017.

[2] Adiel Cohen Jason Weston William Stafford Noble Christina S. Leslie, Eleazar Eskin. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20:467–476, 2004.

[3] Wikimedia Commons. File:svm 8 polinomial.jpg — wikimedia commons, the free media repository, 2015. [Online; accessed 9-September-2018].

[4] Corinna Cortes, Patrick Haffner, and Mehryar Mohri. Positive definite rational kernels. In Bernhard Schölkopf and Manfred K. Warmuth, editors, *Learning Theory and Kernel Machines*, pages 41–56, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[5] Drugbank. About drugbank, 2018.

[6] Drugbank. Drugbank statistics, 2018.

[7] Darren R. Flower. On the properties of bit string-based measures of chemical similarity. *Journal of Chemical Information and Computer Sciences*, 38(3):379–386, 1998.

[8] Kyle Yingkai Gao, Achille Fokoue, Heng Luo, Arun Iyengar, Sanjoy Dey, and Ping Zhang. Interpretable drug target prediction using deep neural representation. In *IJCAI*, 2018.

[9] Laurent Jacob and Jean-Philippe Vert. Protein-ligand interaction prediction: an improved chemogenomics approach. *Bioinformatics*, 24:2149–2156, 2008.

[10] BellBrook Labs. File:drugtarget_interaction.png — bellbrook labs, taking up residence: Using the transcreener adp assay to measure the kinetics of kinase inhibitor interactions, 2017. [Online; accessed 2018].

[11] Mathworks. File:svmhyperplane.png — mathworks, support vector machines for binary classification, 2015. [Online; accessed 2018].

[12] Matt A. Morgan. File:roc_curve.png — radiopaedia, receiver operating characteristic (roc) curve, 2015. [Online; accessed 2018].

[13] L. Ralaivola, JS. Swamidass, H. Saigo, and P. Baldi. Graph kernels for chemical informatics. *Neural Networks*, 18(8):1093–1110, 2005.

[14] Bernhard Schlkopf and Alexander J. Smola. *Learning with Kernel*. The MIT Press, Cambridge,US, first edition, 2002.

[15] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for pattern analysis*. Cambridge University Press, Cambridge,UK, first edition, 2004.

[16] Ajay Unagar. File:svm.png — medium, support vector machines unwinded, 2017. [Online; accessed 2018].

[17] Vladimir Vapnik. *Statistical Learning Theory*. John Wiley Sons, New York, first edition, 1998.

[18] J-P. Vert, H. Saigo, and T. Akutsu. *Local Alignment Kernels for Biological Sequences*, pages 131–153. MIT Press, Cambridge, MA,, 2004.

[19] Analytics Vidhya. File:svm_21.png, analytics vidhya,, 2017. [Online; accessed 8-September-2018].

[20] Vladimir Vovk. Lecture notes in machine learning, February 2018.

[21] K A Walsh, L H Ericsson, D C Parmelee, and K Titani. Advances in protein sequencing. *Annual Review of Biochemistry*, 50(1):261–284, 1981.