Collaborative Filtering on a Family of Biological Targets

Dumitru Erhan, Pierre-Jean L'Heureux, Shi Yi Yue, and Yoshua Bengio

J. Chem. Inf. Model., **2006**, 46 (2), 626-635• DOI: 10.1021/ci050367t • Publication Date (Web): 21 January 2006

Downloaded from http://pubs.acs.org on March 12, 2009

Collaborative Filtering on a Family of Biological Targets

Dumitru Erhan* and Pierre-Jean L'Heureux

Université de Montréal, Department IRO, CP 6128, Succ. Centre-Ville, Montréal, Québec, Canada H3C 3J7

Shi Yi Yue

AstraZeneca R&D Montréal, 7171 Frédérick Banting, St. Laurent, Québec, Canada H4S 1Z9

Yoshua Bengio

Université de Montréal, Department IRO, CP 6128, Succ. Centre-Ville, Montréal, Québec, Canada H3C 3J7

Received September 1, 2005

Building a QSAR model of a new biological target for which few screening data are available is a statistical challenge. However, the new target may be part of a bigger family, for which we have more screening data. Collaborative filtering or, more generally, multi-task learning, is a machine learning approach that improves the generalization performance of an algorithm by using information from related tasks as an inductive bias. We use collaborative filtering techniques for building predictive models that link multiple targets to multiple examples. The more commonalities between the targets, the better the multi-target model that can be built. We show an example of a multi-target neural network that can use family information to produce a predictive model of an undersampled target. We evaluate JRank, a kernel-based method designed for collaborative filtering. We show their performance on compound prioritization for an HTS campaign and the underlying shared representation between targets. JRank outperformed the neural network both in the single- and multi-target models.

1. INTRODUCTION

The developments of combinatorial chemistry and high throughput screening (HTS) provide the pharmaceutical industry with a great opportunity to filter millions of compounds in a short time for a given target. HTS data can be used to generate virtual screening models, which in turn, can be used to further virtually screen even more compounds.[1]

Many aspects of this process use machine learning techniques. Standard machine learning considers a single learning task at a time. For example, learning to predict Y from X using a set of pairs (x_i, y_i). Instead, humans learn a variety of tasks, and it is believed that there are interactions between these learning processes, which are fruitful because of shared underlying mechanisms. It has already been shown theoretically and practically[2−4] that taking into account multiple related tasks can be greatly beneficial to generalization, if the tasks are sufficiently related. [A necessary and sufficient condition for task relatedness is roughly the following: there exists a simpler—perhaps in the Kolmogorov complexity[5] sense—model that describes the joint distribution of inputs, outputs, and task than the separate models and inputs and outputs that one would obtain separeately for each task.] From a neural network perspective, this is due to a product increase of examples corresponding to a summation increase in targets. Of course, if the added targets are unrelated, the generalization power will decrease.

Interestingly, such multiple related tasks do exist in the pharmaceutical industry, where they are commonly called a *target class* (e.g., kinases, G-protein coupled receptors, and maybe ion channels). These target classes have some common features. First, they represent some significant portion of a therapeutic area. Some members of these target classes have been well studied. Second, targets within each of these target classes share a common structural frame. Members of each target class may have a similar binding site. Third, with the development of genomic projects, many new members of these target classes have been identified, though the biological roles of these newcomers (so-called orphans) are still unknown. The challenge we are facing here is how to transfer our knowledge from known targets to orphans. The traditional statistical approach considers a different machine learning task for each member of a given class. Our objective is to compare and evaluate methods to take advantage of the commonalities between the different tasks within a target class. In addition, we should also develop a solution that allows us to estimate QSAR models for orphans that have not yet been tested or for which there are very little available data.

We emphasize here that the goal of our approach is not to create the best global predictive model for a collection of accurately known targets. Here, we assume that we do not know the structure of the targets because we want to generalize to a new unknown target. We have thus developed a practical approach where very little prior knowledge of the target is needed. We acknowledge that a full multi-target model would need a much higher level of description, which

* Corresponding author e-mail: erhandum@iro.umontreal.ca; phone: (514)343-6111, ext. 1794.

would need many more target–ligand pairs than our data set. We also underline that we are less interested in building the best model for a single target than building a model for which we lack sufficient data.

For this paper, we have studied a set of compounds which have been used to screen a set of related targets in AstraZeneca's HTS campaigns. We first built up the models of single targets and then selected some of those most related targets to perform multiple target analysis. To improve prediction power, we also added descriptors of the target (i.e., the task) as side information to help in building the multi-task model. We compared the results from two different schemes, both seen as collaborative filtering models: a multi-task neural network and a kernel-based ranking algorithm called JRank. The analysis of the results allows to conclude on the performance of these two algorithms and on the contributions of the target descriptors.

2. METHODOLOGY

To test the efficiency of a multi-target scenario, we need a framework that would provide an estimate of target "relatedness". Such a framework can then be used to decide whether multi-target learning makes sense in the first place before proceeding to the actual HTS campaign for a new target.

The framework that we devised works as follows. Assume that we have a set of targets from the same family with enough screening data for each target. For each of them, we construct two data sets:

1. A training set that contains the screening data for all the targets except the current one *plus* a fixed small percentage of the screening data for the current target.

2. A testing set that contains the rest of the screening data for the current target.

By training an algorithm on the first data set and testing on the second one and then comparing the performance of this algorithm with the performance of some other algorithm that does standard, single-target, QSAR modeling (with the training set containing just the fixed small percentage of the screening data for the current target), we can see whether adding the screening data for the rest of the targets improves the results.

The reasoning behind choosing a small percentage is simple—we want an algorithm to generalize well given a *new* target, for which we have not enough screening data, and that is a quite realistic scenario. A standard single-target QSAR model that is trained on a small data set will most likely have a poor performance; ideally, an algorithm that does multi-target learning (using the above-mentioned training set) should perform no worse than such a single-target model.

We call the procedure of making the above data sets "undersampling". Our intention is to try to see the effects of undersampling on both multi-target and single-target data at several fixed percentages (which we sometimes call "undersampling fractions") of the screening data for the targets in our data set.

One of the assumptions behind our experiments is that the targets are related in some way that is encoded in our data sets. Our goal is to obtain multi-target learning procedures that will be at least as good as single-target learning and that will outperform single-target learning for small undersampling fractions. We want to test the hypothesis that such a procedure can be successful in the context of multi-target HTS data.

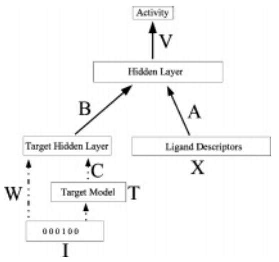

Figure 1. Multi-task neural network architecture.

We present two machine learning methods that we will use to test our hypotheses. The first is a custom-built neural network (which we refer to as a "Multi-Target Neural Network" or a "Multi-Task Neural Network") and the second is a kernel-based collaborative filtering algorithm called JRank. Both methods can deal with both single-target and multi-target learning and are, therefore, quite suitable for testing our hypotheses.

The methods take advantage of prior knowledge about the receptors. The idea is to choose a representation of each receptor and to train a model to predict a single scalar (e.g., probability of percentage inhibition) given both the representation of the compound and the representation of the target receptor protein. Because the representation of receptors is generic (and can accommodate receptors other than those for which assay results are available), this approach can in principle generalize to new receptors.

2.1. Multi-Task Neural Network. The approach of modeling more than one target at a time falls into a broad category of machine learning approaches, called multi-task learning.[6] Such techniques were first developed in the context of multi-layer neural networks, which were modified so as to allow the process of *inductive transfer* (from one modeling task to another) to take place. The simplest of such modifications would be increasing the number of output units to be equal to the number of tasks, but various other techniques exist that allow *biasing* the modeling process of one task by the other tasks.[4,6]

In our case, the basic architecture of the neural network model (Figure 1) has two hidden layers.[7,8] The first hidden layer is committed to biological descriptors, to discover a low-dimensional embedding for targets.

In one version (eq 1), we use in input of the first layer a one-hot variable (a vector full of zeros except for a 1 at position i for coding symbol i) indicating the target protein. The second layer receives the output of the first layer and the chemical descriptors of the compounds. This architecture will learn an individual predictive model for each target, but the first layer will contain information about the relatedness of targets in regard to their ligand activity:

$$P(_{\text{active}}|x, i) = {}_{\text{sigmoid}}(V \tanh(Ax + B \tanh(We_i))) \quad (1)$$

where e_i is the one-hot variable defined above. In another version (eq 2), we use threading-based target descriptors in the first layer. The target model is a set of 25 binding site pocket fingerprints. Here we learn an indirect predictive model for each target:

$$P(_{\text{active}}|x, i, T) = _{\text{sigmoid}}(V \tanh(Ax + B \tanh(CTe_i))) \quad (2)$$

The parameters of the neural network (V, A, B, W) or (V, A, B, C) are tuned by stochastic gradient ascent[9] on the average log-likelihood of the training set (average of the logarithm of the above probabilities). The number of hidden units (e.g., dimension of A) and optimization parameters are selected using a validation set disjoint from the test set.

2.2. JRank. In this section, we present a machine learning algorithm that was proposed for collaborative filtering applications. We describe the way it can be applied for the problem of multi-target QSAR and provide a pseudo-code as well as general intuitions behind the algorithm.

2.2.1. Collaborative Filtering and Multi-Task Learning. Collaborative filtering[10,11] has its roots in recommender systems applications, whereby automated recommendations are produced. Such recommendations are based on similarities between the preferences of different users of the system. Typical collaborative filtering data sets usually include some form of demographic data about the users of the system and/or some basic facts about the items (movies, songs, etc.) that are rated. Evidently, such data could be useful in improving the generalization performance of the algorithm, especially when for some user or item there is only a small number of ratings available. Systems that make use of such extra data have been termed as *content-based filtering*[12] algorithms.

The parallel with multi-target QSAR can be made almost immediately: the "users" are the biological targets, the "items" are the molecular compounds (or vice-versa), and the "ratings" are the levels of activity of the given compound for a given target. The descriptors or the features of the targets and of the compounds could be anything that might help us in uncovering relationships both between the targets and the items.

Ideally, one is interested in using all the data that is available—both ratings and user/item descriptors—such that the algorithm could exploit to the maximum the relationships between the users, items and the ratings. Such an algorithm would be a combination of collaborative and content-based filtering. Such a scenario can be described in a more general framework of *multi-task learning*, which was presented above. We consider learning the preferences of one user, given the user's past ratings, and the features of the items, as one "task". For a given number of users (and, possibly, some features associated with them), we could combine the "tasks" and, perhaps, profit from this combination.

Getting back to our QSAR setting, we can easily notice that a standard one-target QSAR can be seen as a (modeling) "task" and that multi-target QSAR is just a form of multi-task learning. The collaborative filtering technique that we tested in such a setting is the so-called JRank kernel perceptron algorithm.[13] It is an algorithm that performs ordinal regression (i.e., our "ratings" are ordinal values, and we are not modeling their absolute values but their order; see section 2.2.3) and that is an extension (in both the collaborative filtering sense and in the sense that uses kernels) of the PRank algorithm.[14]

2.2.2. Unifying Target and Compound Features. The algorithm makes use of kernels (similarity measures). These are used to encode similarities between items (compounds) and between users (protein targets). These two types of similarities need to be combined into a function that computes a "similarity measure" between user−item pairs. Such a measure can then be used to exploit efficiently the user and item features and to try to uncover a relationship between user−item (or target−compound) pairs and the activity rates (the ultimate goal of a QSAR technique).

The underlying structure of the algorithm is very similar to the original perceptron,[15] which means that it has several useful characteristics such as its simplicity and its online nature. It can also take advantage of Mercer kernels[16] (including custom-built kernels for collaborative filtering problems)—of the type used in Support Vector Machine algorithms[16]—in order to find a separating hyperplane in a high-dimensional space and can be used, therefore, for finding highly nonlinear solutions (such an extension of the perceptron algorithm is typically referred to as the *kernel perceptron*[17]).

Below we follow the notation and description of JRank from reference 13. The basic idea of the algorithm is to unify the target and compound features in a joint feature space in which distances (inner products) can be computed easily. We could then try to find a map Ψ that takes elements ($\mathbf{t}$, $\mathbf{c}$) into $\Psi(\mathbf{t}, \mathbf{c}) \in \mathbb{R}^D$, where $\mathbf{t}$ is the vector of target features and $\mathbf{c}$ is the vector of compound features for a given target−compound pair (with D being the—possibly infinite—dimension of the resulting combined space). Such a map would allow us to compute similarities between *pairs* of targets/compounds and would allow us to generalize across both target features and compound features at the same time.

Let $\mathcal{T}$ be the set of targets, $\mathcal{C}$ be the set of compounds, and the map be Ψ: $\mathcal{T} \times \mathcal{C} \rightarrow \mathbb{R}^D$, which gives a D-dimensional feature vector for each target−compound pair. Our goal is then to choose a function (which should be optimal in some sense) from the set of functions F, which are linear in Ψ, i.e.

$$F(\mathbf{t}, \mathbf{c}; \mathbf{w}) = \Psi(\mathbf{t}, \mathbf{c})^T \cdot \mathbf{w} \quad (3)$$

(where T is the transpose). This function would encode (in a linear fashion) the relationship between the features of pairs of targets and compounds and the respective activity levels.

The output of such a function is binned via a set of adaptive thresholds $\theta \in \mathbb{R}^k$, where k is the number of activity levels we are interested in ($\theta_k = +\infty$ for convenience). This is done in order to predict the activity level from a target−compound pair: by simply selecting the number of the bin where $F(\mathbf{t}, \mathbf{c}; \mathbf{w})$ falls into. The prediction function f ($\mathbf{t}$, $\mathbf{c}$; $\mathbf{w}$, θ) depends straightforwardly on θ: it outputs a level i associated with the interval $[\theta_i, \theta_{i+1})$ which contains $F(\mathbf{t}, \mathbf{c}; \mathbf{w})$.

Note that $\Psi(\mathbf{t}, \mathbf{c})$ from eq 3 is not computed directly (for reasons that will become clearer a bit later) and that our algorithm is only using dot products in the feature space defined by Ψ. The dot product between the application of Ψ on two pairs is referred to as a *kernel*. More precisely, for two given pairs ($\mathbf{t}$, $\mathbf{c}$) and ($\mathbf{t}'$, $\mathbf{c}'$), we define the kernel as

$$K((\mathbf{t}, \mathbf{c}),(\mathbf{t}', \mathbf{c}')) = \Psi(\mathbf{t}, \mathbf{c})^T \cdot \Psi(\mathbf{t}', \mathbf{c}') \quad (4)$$

As shown in refs 14 and 16, one can rewrite eq 3 via the kernel defined in eq 4 as follows, thanks to the Representer Theorem:

$$F(\mathbf{t}, \mathbf{c}; \alpha) = \sum_{(\mathbf{t}', \mathbf{c}')} \alpha_{(\mathbf{t}', \mathbf{c}')} K((\mathbf{t}, \mathbf{c}), (\mathbf{t}', \mathbf{c}')) \tag{5}$$

Thus if we can compute efficiently the dot product from eq 4, then we do not need to explicitly compute the feature vectors given by Ψ. This is important because the computation of Ψ may be impractical if we want this nonlinear transformation to be rich enough: in practice we choose not Ψ but the kernel K, and for many choices of interest for K, the corresponding Ψ is infinite-dimensional. The only constraint on the choice of K is that it must be positive semi-definite. It means that for any finite set $\mathscr{P}$ of pairs p_i, the Gram matrix G associated with $\mathscr{P}$ must not have any negative eigenvalues. The entry (i, j) of G is $G_{ij} = K(s_i, s_j)$ with $s_i \in \mathscr{P}$ and $s_j \in \mathscr{P}$.

In our case, we must define a kernel over target–compound pairs. We take a bottom-up approach, by first defining similarity measures between pairs of targets, then between pairs of compounds, and then combining the two measures into a kernel function of the desired type. Thus, we use the following kernels:

1. an identity kernel $\mathscr{K}^{\mathrm{id}}_{\mathscr{T}}$, which returns one if the two targets have the same feature vector and zero otherwise,
2. a Gaussian kernel $\mathscr{K}^{\mathrm{ga}}_{\mathscr{T}}$ (e.g., see ref 16),
3. a correlation kernel $\mathscr{K}^{\mathrm{co}}_{\mathscr{T}}$, which computes the Pearson correlation coefficient, which is a dot-product between the normalized activity level vectors corresponding to each target (over the users for which activity is measured on both targets),
4. a quadratic kernel $\mathscr{K}^{\mathrm{qu}}_{\mathscr{T}}$, which is $\mathscr{K}^{\mathrm{co}}_{\mathscr{T}} \cdot \mathscr{K}^{\mathrm{co}}_{\mathscr{T}}$ (it has the necessary property of always being positive semi-definite).

So far, we have not mentioned a way of combining the kernels. If we were to deal with only target features (or only with the compound features), combining the kernels could be done by simple addition, since the sum of positive semi-definite matrices is also positive semi-definite:

$$\mathscr{K}_{\mathscr{T}} = \mathscr{K}^{\mathrm{id}}_{\mathscr{T}} + \mathscr{K}^{\mathrm{ga}}_{\mathscr{T}} + \mathscr{K}^{\mathrm{co}}_{\mathscr{T}} + \mathscr{K}^{\mathrm{qu}}_{\mathscr{T}} \tag{6}$$

We can do exactly the same for the kernel for the compound features:

$$\mathscr{K}_{\mathscr{C}} = \mathscr{K}^{\mathrm{id}}_{\mathscr{C}} + \mathscr{K}^{\mathrm{ga}}_{\mathscr{C}} + \mathscr{K}^{\mathrm{co}}_{\mathscr{C}} + \mathscr{K}^{\mathrm{qu}}_{\mathscr{C}} \tag{7}$$

If we are interested in combining $\mathscr{K}_{\mathscr{T}}$ and $\mathscr{K}_{\mathscr{C}}$, we could use the tensor product to get $K((\mathbf{t}, \mathbf{c}), (\mathbf{t}', \mathbf{c}'))$. However, we do not (and cannot, for any practical purpose, because of the infinite dimension vectors) compute this product in the spaces defined by the respective Ψ values. This is because it can be computed simply as

$$K((\mathbf{t}, \mathbf{c}), (\mathbf{t}', \mathbf{c}')) = \mathscr{K}_{\mathscr{T}}(\mathbf{t}, \mathbf{t}') \cdot \mathscr{K}_{\mathscr{C}}(\mathbf{c}, \mathbf{c}') \tag{8}$$

which is a handy shortcut that allows for great savings in the runtime of the algorithm.

Given this kernel and the definition of the F function that is to be learned (from eq 5), we can now define the kernel perceptron algorithm that will find both an optimal set of coefficients $\alpha_{(\mathbf{t},\mathbf{c})}$ $(\forall\ (\mathbf{t}, \mathbf{c}) \in \mathscr{T} \times \mathscr{C})$ *and* a matching set of adaptive thresholds θ (used for binning F).

2.2.3. The Algorithm. As previously mentioned, JRank works in the framework of *ordinal regression*. This means that we define an order among the activity levels and that we do not interpret their numerical value. This is appropriate for both binary classification problems—where we would interpret the two activity levels as "more active" and "less active"—and for multi-class/regression problems, where the transformation of the numerical values to an ordinal scale poses no problem.

Algorithm 1, proposed in ref 13, is a straightforward

Algorithm 1 JRank

```
{$\alpha$ is a sparse parameter matrix, with one element per experimental observation}
{$A_{(\mathbf{t},\mathbf{c})}$ is the activity level the compound with features $\mathbf{c}$ given target with features $\mathbf{t}$}
$\alpha_{(\mathbf{t},\mathbf{c})} = 0, \forall A_{(\mathbf{t},\mathbf{c})}$
{$\boldsymbol{\theta}$ is a vector of thresholds, defining the bins for the ordinal values}
$\boldsymbol{\theta}_i = 0, \forall i = 1, \ldots, k-1$ and $\boldsymbol{\theta}_k = +\infty$
{$N_{it}$ is the number of iterations}
for $n = 1$ to $N_{it}$ do
  for all $A_{(\mathbf{t},\mathbf{c})}$ do
    {The estimated activity level (equation 3)}
    $\hat{a} = f(\mathbf{t}, \mathbf{c}; \alpha, \boldsymbol{\theta})$
    {If the estimated activity level is incorrect, we update the learnt parameters}
    if $\hat{a} > A_{(\mathbf{t},\mathbf{c})}$ then
      {Following the gradient}
      $\alpha_{(\mathbf{t},\mathbf{c})} = \alpha_{(\mathbf{t},\mathbf{c})} + \hat{a} - A_{(\mathbf{t},\mathbf{c})}$
      {The value of the $F$ function becomes closer to the correct bin at the next iteration}
      for $i = A_{(\mathbf{t},\mathbf{c})}$ to $\hat{a} - 1$ do
        $\boldsymbol{\theta}_i = \boldsymbol{\theta}_i + 1$
      end for
    else if $\hat{a} < A_{(\mathbf{t},\mathbf{c})}$ then
      {Following the gradient}
      $\alpha_{(\mathbf{t},\mathbf{c})} = \alpha_{(\mathbf{t},\mathbf{c})} + \hat{a} - A_{(\mathbf{t},\mathbf{c})}$
      {The value of the $F$ function becomes closer to the correct bin at the next iteration}
      for $i = \hat{a}$ to $A_{(\mathbf{t},\mathbf{c})} - 1$ do
        $\boldsymbol{\theta}_i = \boldsymbol{\theta}_i - 1$
      end for
    end if
  end for
end for
```

extension to the kernel perceptron algorithm.[17] As in ref 14, JRank projects each instance from our data set onto the real line. Each ranking is then associated with a distinct sub-interval of the reals. During learning these sub-intervals are updated: if and when the current set of parameters predicts an incorrect sub-interval, the parameters are updated such that the new predicted rank is closer to the sub-interval (and vice-versa, by modifying the boundaries of the sub-intervals).

$A_{(\mathbf{t},\mathbf{c})}$ is the activity level observed for pair $(\mathbf{t}, \mathbf{c})$. In the formulation it is also understood that we have access to the set of all the data triplets $(\mathbf{t}, \mathbf{c}, A_{(\mathbf{t},\mathbf{c})})$. The sparse parameter matrix α has nonzero entries $\alpha_{(\mathbf{t},\mathbf{c})}$ only for the observed pair $(\mathbf{t},\mathbf{c})$. It can be used for prediction via eq 5. A set of thresholds/bins θ_i (one per ordinal value) is also learned. The algorithm runs through the data set in N_{it} stages/iterations and updates $\alpha_{(\mathbf{t},\mathbf{c})}$ and updates the thresholds if it predicts an incorrect activity level. The algorithm assumes that the ranks are ordered from $i = 1$ to k, but it can be easily modified to accommodate other types of ranks.

The updates of $\alpha_{(\mathbf{t},\mathbf{c})}$ follow the prediction error (the difference between the predicted rank and the actual rank, also called the *ranking loss*), i.e., they follow the gradient, while the thresholds θ_i are updated so that the value of the F function becomes closer to the correct bin at the next iteration.

The algorithm has two hyper-parameters:

1. The width of the Gaussian kernel σ. Ideally, there should be one for each (target and compound) kernel.
2. The number of iterations N_{it}.

It is worth noting that the algorithm functions correctly and as expected when $k = 2$ (i.e., it learns to perform binary

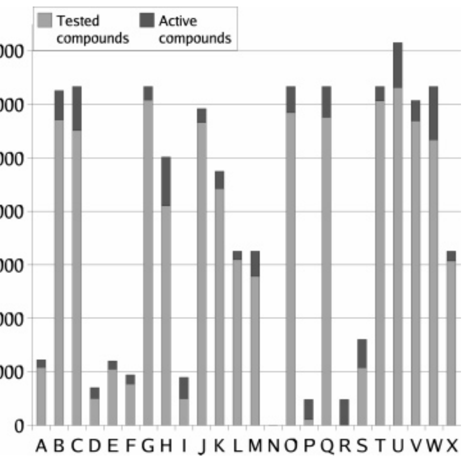

Figure 2. Number of compounds available for each target, classified as actives or inactives.

classification). The algorithm reduces to simple single-task learning if the data set has only one target (α becomes a vector)—which is quite handy because it allows us to compare directly single-task learning with multi-task learning.

2.3. Data Sets and Descriptors. All the ligands used in this study were made in AstraZeneca R&D Montreal. They all satisfy the Lipinski *rule of five*. Figure 2 shows the number of compounds for which the screening data was available for each target. We detailed the active and inactive compounds. Here, active compounds corresponds to any value of inhibition higher than 0.

We then computed descriptors with MOE (version 2004.03).[18] The set of 469 descriptors range from atom frequencies[19,20] and topological indices[21−24] to 3D surface area descriptors. We also computed MACCS,[25] Randic,[26] and EState[27] descriptors that are available in the MOE package. The numerical values were normalized.

Several receptor-binding fingerprints have been published. The binding pocket fingerprints we used in this work were based on our observations and some assumptions. The first assumption is that all the targets in our study share a similar binding position. The second assumption is that the amino acids of the targets in binding sites have three native interactions between their side chains: ionic, polar, and hydrophobic. When a ligand interacts in the binding site, it will break some of the native interactions and build up new, ligand involved (mediate) interactions. Based on the positions (at the binding site), the type of the interactions, and the variations amount the targets in this study, 14 bins were identified and used. Each bin represents a type of the interaction at the given position of the binding pocket. Adding, reducing, or changing the targets will alternate the binding pocket fingerprints.

Two preconditions should be satisfied for this type of studies. First, the ligands have not been designed to improve their selectivity, and/or second, the recognition region of the target is away from the binding region. To accommodate our study, we did not intend to further detail the differences between the targets, which would have been possible with other protein fingerprints.[28] As a final step, we selected the most varying receptor descriptors to match the small number of targets can studied.

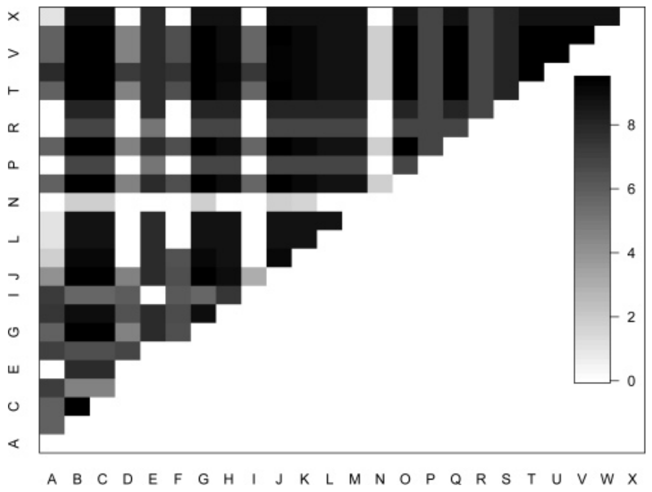

Figure 3. Number of compounds shared by each pair, in logarithmic scale.

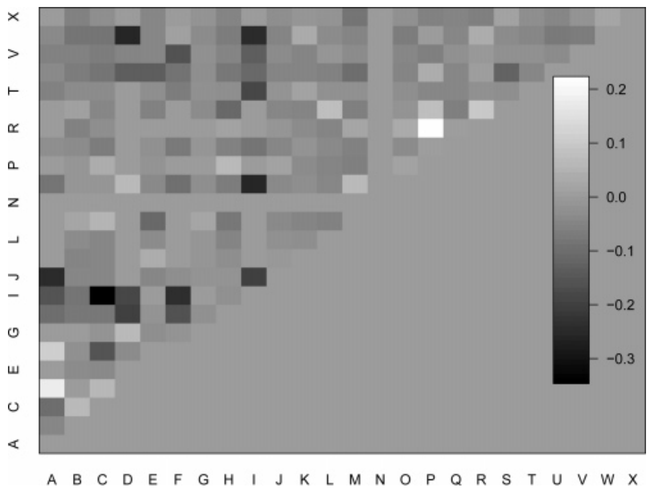

Figure 4. Pairwise correlation of biological activity.

3. RESULTS

In this section, we present suitable performance measures for the algorithms that we implemented as well as results that can be used for verifying our hypotheses.

3.1. Direct Activity Correlation. Before computing the relationships between different biological targets, we transform the biological activity in a bi-class target using a 20% inhibition cutoff. We can then compute the pairwise linear correlation of activity between each target for shared chemical compounds in the data set. The linear correlation will get higher when two targets have the same active and inactive compounds.

Figure 3 shows the number of compounds for which we have pairwise screening information. As can be seen, many pairs' shared number of compounds are small. This fact has many implications in the results shown afterward. It should be noted that the choices of compounds for the screening introduce some unnecessary biases in a multi-target scheme.

Figure 4 shows the actual pairwise linear correlation of biological activity. From this, we picked seven targets in the G1* family, for which cross-correlations were the strongest. These seven targets (A, D, F, H, I, S, and U) will constitute the main focus of interest for the undersampling experiments.

We also computed paired *t*-test directly on the percentage of inhibition of paired compounds. This is a more robust

Table 1. Paired *t*-Test of Biological Activity

pair of targets	score
D−F	0.954
F−O	0.899
Q−X	0.893
I−J	0.845
K−L	0.814
A−W	0.791
B−X	0.718
L−O	0.526
A−O	0.509

measure of relatedness between biological activity. Table 1 details the best results. Based on the score, we constructed three ensemble of targets for future experiments: D, F, O; D, F, I, J, O, Q, X; and A, B, D, F, I, J, K, L, O, Q, W, X.

3.2. Undersampling Scheme. To recreate an environment where we lack sufficient number of examples to learn a predictive model, we artificially deplete a data set, focusing on a target, as explained in section 2. Following algorithm 2, we focused on the seven most correlated target in the G1 family. Algorithm 2 describes a simple method for testing the efficiency of a certain multi-task learning algorithm. By artificially depleting the data set used for training and validation of examples from a certain task and by varying the level of depletion, we can obtain a relatively complete picture of the performance of such an algorithm given different real-life scenarios. Such a scheme also allows for direct comparisons between multi-task learning and single-task learning.

Algorithm 2 A sample scheme of hypothesis testing

```
{$T$ is the total number of tasks}
{$R$ is a real number between 0 and 1 (the "undersampling fraction")}
{$K$ is the number of iterations}
for $t = 1$ to $T$ do
  for $R = 0$ to 1 (by increments $\Delta R$) do
    for $k = 1$ to $K$ do
      {Randomly subdivide the dataset into two parts (the parts in bold only apply to the multi-target case):}
      DTrain = {Fraction $R$ of data from task $t$ } + { Data from all tasks except $t$}
      DTest = {The rest of data from task $t$}
      {Perform training and validation (model selection) on DTest}
      LModel = Train({Single,Multi}TaskAlgo, DTrain)
      {Compute the error of the selected model on the test set}
      Error(T,R,k) = Test(LearnedModel,DTest)
    end for
  end for
end for
```

3.3. Hyper-Parameter Selection. To assess the generalization performance of the algorithms presented, we need to select a combination of hyper-parameters that gives an optimal performance on a validation set (which is independent from the training set). Given this combination of hyper-parameters, we can get an unbiased estimate of the generalization ability of the algorithms by testing the models on a testing set, that is independent from both the training and validation sets.

In the case of the neural network, such model selection procedures have been performed for parameters such as the number of hidden units, the weight decay, the learning rate, etc. Early stopping on the number of epochs (by computing the validation error at each step and stopping when it starts to increase) has also been performed.

In the case of JRank, the width of the Gaussian kernels (the Gaussian kernels for compounds and the one for the targets used the same σ) was computed by the above validation procedure, whereas early stopping was used for finding an optimal N_{it}.

3.4. Performance Measures. We use the lift to assess the performance of the models. The lift measures how much better than random we can order the compounds from active to inactive. We do this by testing a model on an independent test set and ordering (in decreasing order) the molecules by the scores that we obtain for each of them. We select a subset of this ordered list (from the highest ranking molecule downward) and compute ratio of actives to the total number of compounds in this subset. The higher this ratio, the better is our algorithm at predicting which compounds are active.

Let a/n be the average fraction of actives in our database, with a the total number of actives and n the total number of compounds. In the selected subset, a_s is the number of actives and n_s is the number of compounds in that subset. Then we compute the lift by

$$\text{lift} = \frac{a_s/n_s}{a/n} \tag{9}$$

In effect, it tells us how much better than chance our algorithm performs. The lift that we compute is a single point in an enrichment curve that corresponds roughly to an ROC curve.[29] The enrichment curve tracks the LIFT values across different sizes of the subsets and provides a comprehensive picture of the generalization capabilities of a learning algorithm; it can also be transformed straightforwardly into an ROC curve.[30] Here, the subset is 30% of the database, and we multiply the lift values by 100 to improve readability.

3.5. Multi-Task Neural Network. In this section, we present the results obtained with the multi-task neural network.

Table 2. Multi-Target NNet Learning without Target Descriptors[a]

target score	24 targets	12 targets	7 targets	3 targets
PLS	181	172	190	189
global	179	171	195	188
A	108	138		
B	117	120		
D	121	136	138	165
F	112	150	134	176
I	173	154	173	
J	125	133	172	
K	91	110		
L	125	146		
O	129	125	141	153
Q	127	125	153	
W	110	122		
X	123	136	141	

[a] Lift over 30% of data.

Table 3. Multi-Target NNet Learning with Target Descriptors[a]

target score	24 targets	12 targets	7 targets	3 targets
PLS	180	173	190	189
global	189	175	195	193
A	165	153		
B	128	126		
D	159	131	135	165
F	173	152	141	191
I	170	173	175	
J	145	133	178	
K	122	112		
L	168	152		
O	132	128	140	152
Q	127	129	153	
W	112	131		
X	137	129	143	

[a] Lift over 30% of data.

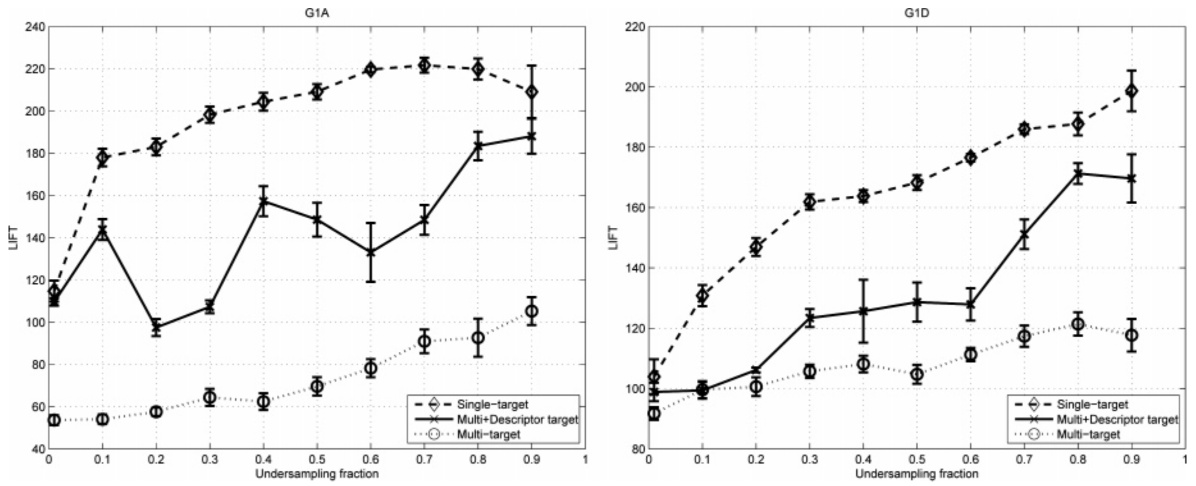

Figure 5. Effects of undersampling on NNet's performance. Measured on the G1A and G1D targets in three scenarios: multi-target learning with target descriptors, multi-target learning without target descriptors, and single-target learning.

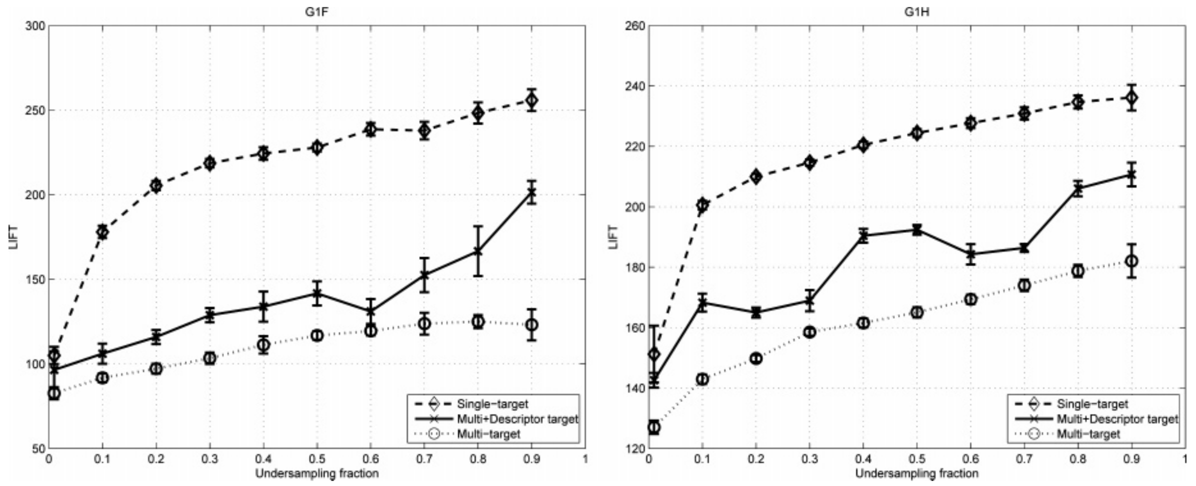

Figure 6. Effects of undersampling on NNet's performance. Measured on the G1F and G1H targets in three scenarios: multi-target learning with target descriptors, multi-target learning without target descriptors, and single-target learning.

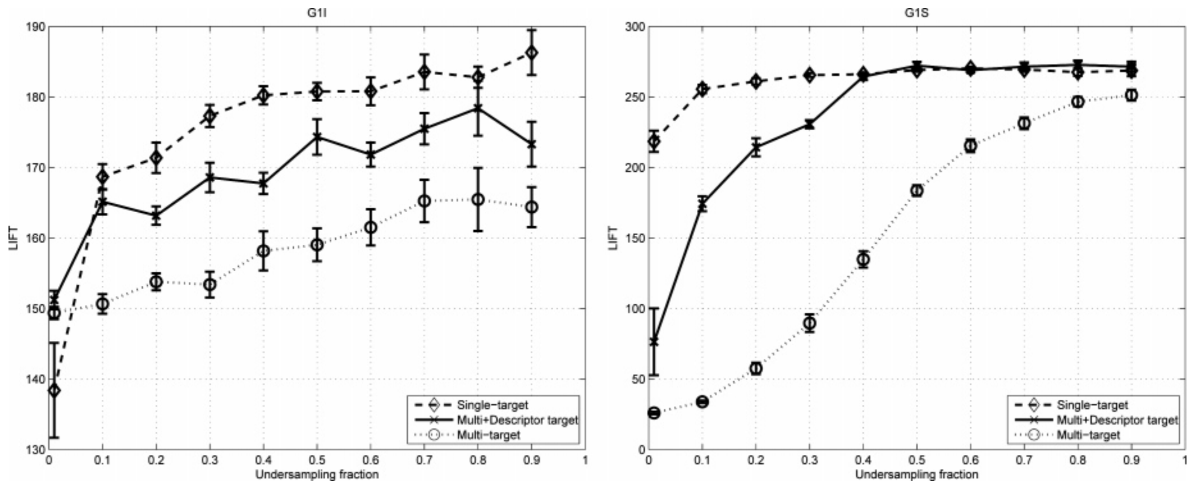

Figure 7. Effects of undersampling on NNet's performance. Measured on the G1I and G1S targets in three scenarios: multi-target learning with target descriptors, multi-target learning without target descriptors, and single-target learning.

3.5.1. Learning without Target Descriptors. Table 2 shows specific target performance of multi-target learning on either all the 24 G1* targets or the three ensemble of correlated targets, using no target descriptors. In this case, a double cross-validation scheme was used for performance measurement. We divide the entire data set into two sets: one that will be used for training and validation and another for testing. We select an optimal set of hyper-parameters by performing (a usually 10-fold) cross-validation on the first set and an estimate of the generalization error is computed

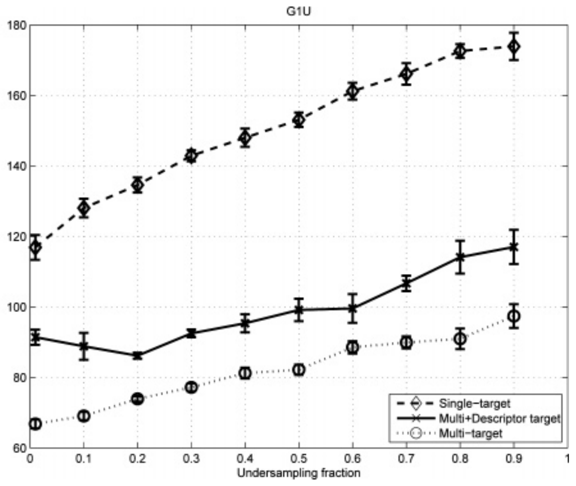

Figure 8. Effects of undersampling on NNet's performance. Measured on the G1U target in three scenarios: multi-target learning with target descriptors, multi-target learning without target descriptors, and single-target learning.

on the second set using these optimal hyper-parameters. This procedure is repeated several times to provide variances of the generalization error.

The PLS[31] and global lift are computed by regrouping all the actives together in a single active class. When checking individual target performance, it deteriorates going from 3 to 24 targets. Without target descriptors, the model cannot find a shared representation and solve conflicting examples. It is also harder to model the remaining 12 targets not in the subgroups, because they are farther from any other targets (see Table 1).

3.5.2. Learning with Target Descriptors. Table 3 shows specific target performance of multi-target learning with biological target descriptors on the 24 targets or 3 subgroups of targets (the same cross-validation scheme as in Section 3.5.1 is used).

By comparing Tables 2 and 3, we notice that the global lift rises substantially when using target descriptors. The neural network learns a shared representation from the group of tasks that helps the undersampled task.

Table 3 also shows a well-shaped performance as we add new targets. There are two factors that affect the performance in this case. First, the individual choices of targets, when small, has a direct impact on generalization. For instance, the choice of three other targets would yield a very different individual performance. Second, as we add more targets, the shared representation theory picks up. As such, a majority of individual targets get better prediction as we add more to the 12 targets. Having 8 out of 12 targets showing this behavior, despite the crude target descriptors we use, is a

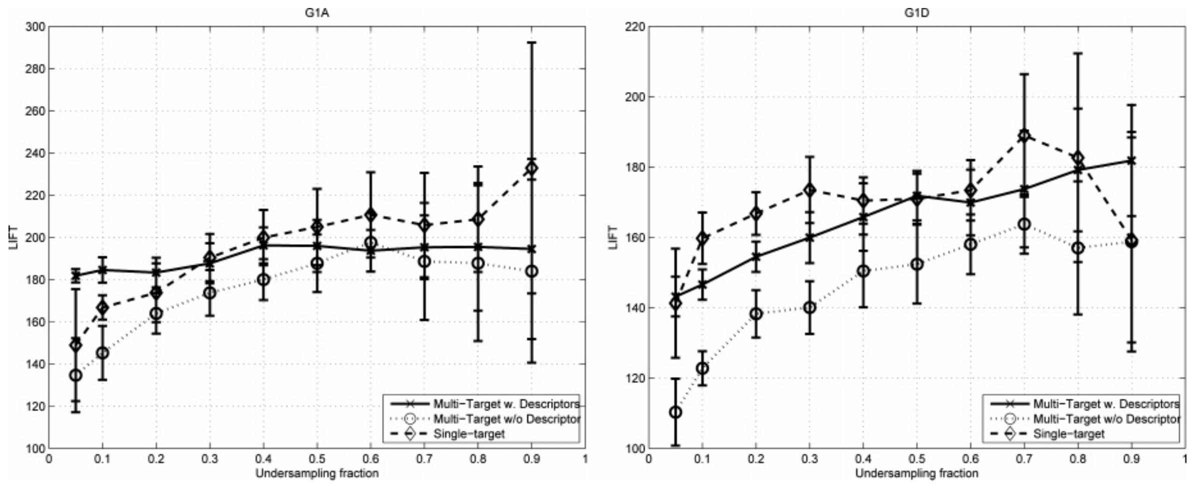

Figure 9. Effects of undersampling on JRank's performance. Measured on the G1A and G1D targets in three scenarios: multi-target learning with target descriptors, multi-target learning without target descriptors, and single-target learning.

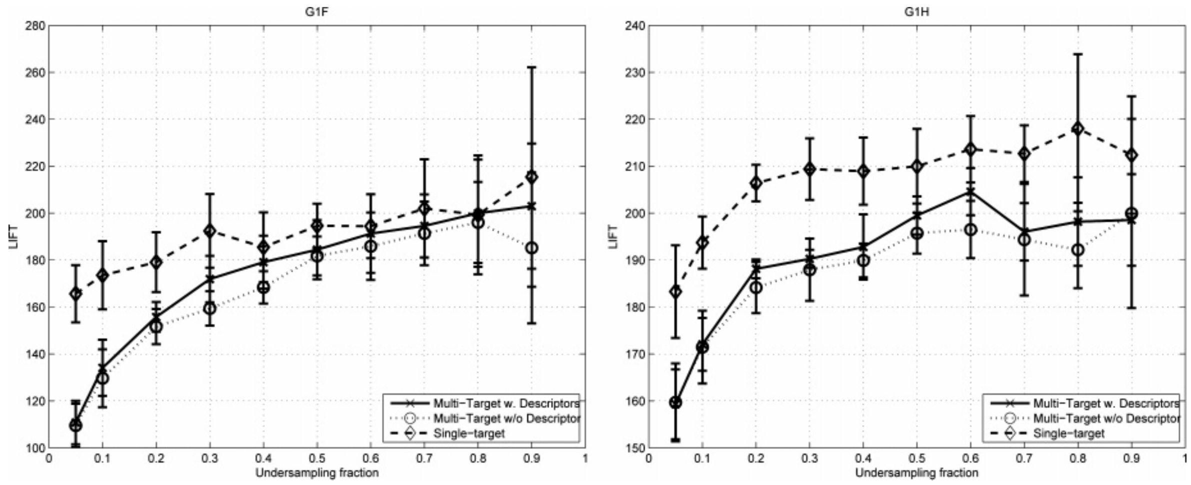

Figure 10. Effects of undersampling on JRank's performance. Measured on the G1F and G1H targets in three scenarios: multi-target learning with target descriptors, multi-target learning without target descriptors, and single-target learning.

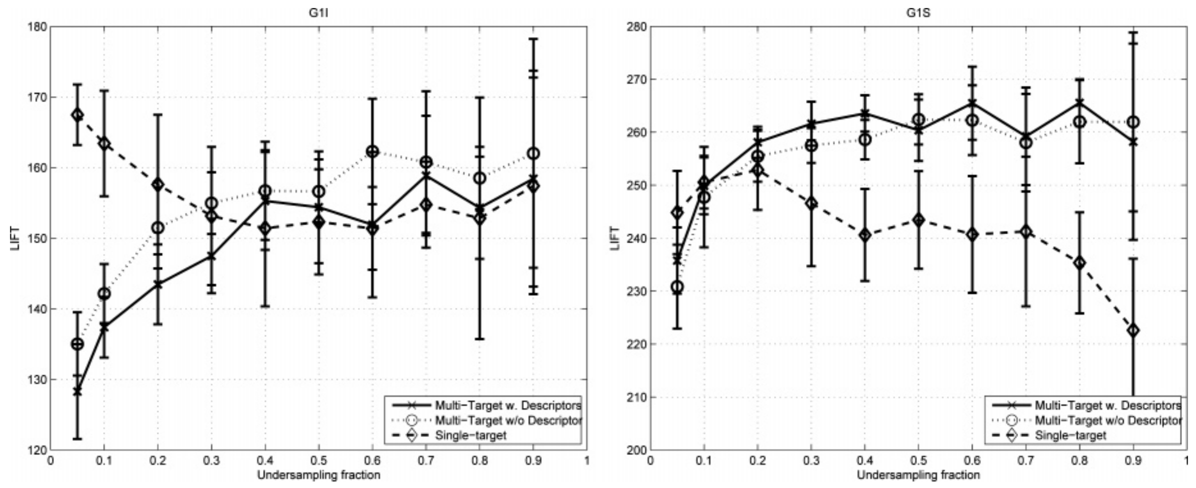

Figure 11. Effects of undersampling on JRank's performance. Measured on the G1I and G1S targets in three scenarios: multi-target learning with target descriptors, multi-target learning without target descriptors, and single-target learning.

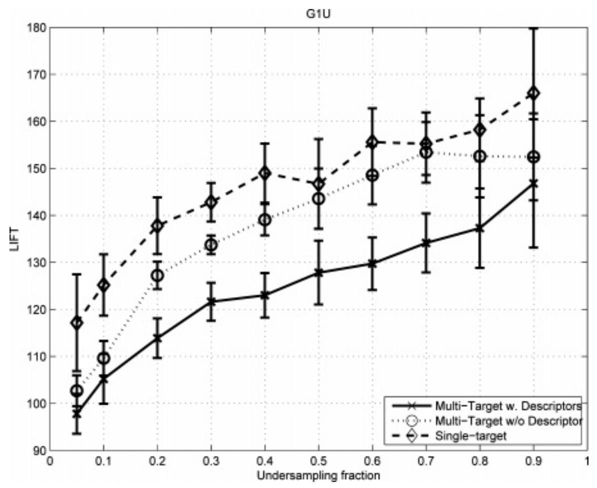

Figure 12. Effects of undersampling on JRank's performance. Measured on the G1U target in three scenarios: multi-target learning with target descriptors, multi-target learning without target descriptors, and single-target learning.

remarkable observation. This result is in line with collaborative filtering expectations. It has supported our research on multi-target learning.

3.5.3. Undersampled Learning with Target Descriptors. Given the positive results with target descriptors, we tried to measure the performance of the network with the undersampling method. Figures 5−8 show the details of undersampling the seven most correlated targets with the neural network. We test our algorithm with and without target descriptors and compare with single target learning. We see that the lift rises quickly when doing single target learning and that multi-target learning without target descriptors lags far behind. Depending on the target, multi-target learning with target descriptors falls in between. For G1I, we even see a slight range of undersampling where multi-target learning beats single target learning.

3.6. JRank. Encouraged by the results obtained with the multi-target neural network, we performed the same type of experiments with JRank. We found that using a combination of identity and Gaussian kernels produced the best results. The correlation kernel did not seem to capture too well the similarities between pairs of targets or pairs of compounds (a possible reason is that the simple linear correlation

Table 4. Comparison of Multi-Target NNet Learning with Target Descriptors and JRank[a]

target score	multi-target NNet	JRank
A	183	**196**
D	159	**179**
F	173	**200**
H	200	**201**
I	**170**	154
S	264	**265**
U	79	**137**

[a] Lift over 30% of data.

coefficients between either targets or compounds are not sufficient to capture any similarity measure between them).

Figures 9−12 contain the results obtained with this algorithm. This time it seems that in most of the cases multi-target learning with target descriptors is at least as good as single-target learning or multi-target learning without descriptors and that, in one case (Figure 9), it seems to perform better than either of them.

By comparing Figures 5−8 with Figures 9−12 and by analyzing Table 4 (which contains lift values at the 0.8 undersampling fraction for each algorithm, corresponding roughly to a 5-fold cross-validation result), we clearly see that JRank scores much higher than the neural network on the smaller fraction of undersampling. Clearly, JRank is to be preferred in a multi-target setting. We also notice that JRank performs quite well even in a simple single-target scenario and therefore can be used in a stand-alone fashion.

4. DISCUSSION

Building a virtual screening model for a new target is a difficult task. We developed a special kind of neural network that used a collaborative filtering approach to address the problem. We were disappointed by the poor results. We nevertheless developed the current target descriptors, which need a minimal knowledge of the 3D structure of the target. These target descriptors helped to improve predictive performance and suggested that adding new targets helped the learning.

On the other hand, the receptor descriptors that we used throughout our experiments are in a preliminary form and we did not verify or optimize them extensively. Such optimizations should take into account the ratio between the number of features and the number of targets.

We then implemented and evaluated a kernel-based algorithm, JRank, to address our multi-task problem. We presented evidence that JRank is better than our neural network architecture in both single and multi-task settings. We also found that multi-task learning with JRank never performs significantly worse than single task learning. This result is markedly better than for our neural network architecture.

It is quite clear that our work is not finished here: the results of JRank (and, more frequently, the neural network results) do not always clearly outperform single-target learning and that might be a dis-incentive for using it.

5. CONCLUSIONS

In this paper, we have validated our hypotheses concerning multi-target learning on a family of biological targets. We have shown that, for a certain ensemble of targets, adding more targets can markedly improve the generalization performance of our algorithms. We have also shown a multi-task learning algorithm whose generalization performance is, on average, at least as good as the one of standard single-target learning. Moreover, it seems that the target (receptor) descriptors are quite important in improving the generalization abilities of a multi-target algorithm.

We demonstrated the performance of JRank, a kernel perceptron algorithm, in a scenario in which activity rates have to be predicted for a target for which there is insufficient screening data. JRank's predictions in a multi-target setting were at least as good as its predictions in a single-target setting, and in some cases, they outperformed the single-target results. This is a very encouraging result that validates our general approach and calls for future work on refining it. The goal here would be for JRank to beat single-target models for a longer range of undersampling for all targets.

ACKNOWLEDGMENT

We express our appreciations for the support and discussions with Dr. Christopher Walpole and other AstraZeneca R&D Montreal colleagues. We also thank Olivier Delalleau for technical support. This work was supported financially in part by Valorisation Recherche Québec and the National Science and Engineering Research Council of Canada, and was made possible thanks to the RQCHP (Quebec Network for High-Performance Computing).

REFERENCES AND NOTES

(1) Bhm, H., Schneider, G., Eds. *Virtual Screening for Bioactive Molecules*; *Methods and Principles in Medicinal Chemistry*, Vol. 10; Wiley−VCH: Weinheim, Germany, 2000.

(2) Baxter, J. Learning model bias. In *Advances in Neural Information Processing Systems*, Vol. 8; Mozer, M., Touretzky, D., Perrone, M., Eds.; MIT Press: Cambridge, MA, 1996.

(3) Baxter, J. A Bayesian/Information theoretic model of learning to learn via multiple task sampling. *Mach. Learn.* **1997**, *28*, 7−40.

(4) Ghosn, J.; Bengio, Y. Bias learning, knowledge sharing. *IEEE T. Neural Network* **2003**, *14*, 748−765.

(5) Li, M.; Vitanyi, P. *An Introduction to Kolmogorov Complexity and Its Applications*; Springer-Verlag: Heidelberg, 1997.

(6) Caruana, R. Multitask learning. *Mach. Learn.* **1997**, *28*, 41−75.

(7) Bishop, C. *Neural Networks for Pattern Recognition*; Oxford University Press: London, UK, 1995.

(8) Zupan, J.; Gasteiger, J. *Neural Networks in Chemistry and Drug Design*, 2nd ed.; Wiley-VCH: Weinheim, Germany, 1999.

(9) Bishop, C. M. *Neural Networks for Pattern Recognition*; Oxford University Press: Oxford, UK, 1996.

(10) Goldberg, D.; Nichols, D.; Oki, B. M.; Terry, D. Using collaborative filtering to weave an information tapestry. *Commun. ACM* **1992**, *35*, 61−70.

(11) Resnick, P.; Iacovou, N.; Suchak, M.; Bergstrom, P.; Riedl, J. GroupLens: an open architecture for collaborative filtering of netnews. In *CSCW '94: Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*; ACM Press: New York, USA, 1994.

(12) Balabanovic, M.; Shoham, Y. Fab: content-based, collaborative recommendation. *Commun. ACM* **1997**, *40*, 66−72.

(13) Basilico, J.; Hofmann, T. Unifying collaborative and content-based filtering. In *ICML '04: Proceedings of the Twenty-First International Conference on Machine Learning*; ACM Press: New York, 2004.

(14) Crammer, K.; Singer, Y. Pranking with ranking. In *Advances in Neural Information Processing Systems Vol. 14*; Dietterich, T. G., Becker, S., Ghahramani, Z., Eds.; MIT Press: Cambridge, MA, 2002.

(15) Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychol. Rev.* **1958**, *65*, 386-408.

(16) Schölkopf, B.; Smola, A. J. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*; MIT Press: Cambridge, MA, 2001.

(17) Freund, Y.; Schapire, R. E. Large margin classification using the perceptron algorithm. In *COLT '98: Proceedings of the Eleventh Annual Conference on Computational Learning Theory*; ACM Press: New York, 1998.

(18) Chemical Computing Group. MOE, www.chemcomp.com,.

(19) Bicerano, J. *Prediction of Polymer Properties*, 2nd ed.; Marcel Dekker: New York, 1996.

(20) Oprea, T. Property distribution of drug-related chemical databases. *J. Comput.-Aided Mol. Des.* **2000**, *14*, 251−264.

(21) Kier, L. Shape indexes of orders one and three from molecular graphs. *Quant. Struct.-Act. Relat.* **1986**, *5*, 1−7.

(22) Hall, L.; Kier, L. The molecular connectivity chi indices and kappa shape indices in structure−property modeling. In *Reviews of Computational Chemistry*, Vol. 2; VCH Publishers: Weinheim, Germany, 1991; Chapter 9, pages 367−422.

(23) Balaban, A. Highly discriminating distance-based topological index. *Chem. Phys. Lett.* **1982**, *89*, 399−404.

(24) Petitjean, M. Applications of the radius-diameter diagram to the classification of topological and geometrical shapes of chemical compounds. *J. Chem. Inf. Comput. Sci.* **1992**, *32*, 331−337.

(25) Durant, J.; Leland, B.; Henry, D.; Nourse, J. Reoptimization of MDL keys for use in drug discovery. *J. Chem. Inf. Comput. Sci.* **2002**, *42*, 1273−1280.

(26) Randic, M. On molecular identification numbers. *J. Chem. Inf. Comput. Sci.* **1984**, *24*, 164−175.

(27) Hall, L.; Kier, L. Electrotopological state indices for atom types: a novel combination of electronic, topological, and valence state information. *J. Chem. Inf. Comput. Sci.* **1995**, *35*, 1039−1045.

(28) Deng, Z.; Chuaqui, C.; Singh, J. Structural interaction fingerprint (SIFt): A novel method for analyzing three-dimensional protein−ligand binding interactions. *J. Med. Chem.* **2004**, *47*, 337−344.

(29) Hanley, J. A.; McNeil, B. J. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology* **1982**, *143*, 29−36.

(30) Ling, C. X.; Li, C. Data mining for direct marketing: problems and solutions. In *KDD*; AAAI Press: New York, 1998.

(31) Malinowski, E. *Factor Analysis in Chemistry*, 3rd ed.; John Wiley & Sons: New York, 2002.

CI050367T