Sequence Kernels for Predicting Protein Essentiality

Cyril Allauzen ALLAUZEN@GOOGLE.COM
Google Research, New York, NY

Mehryar Mohri MOHRI@CS.NYU.EDU
Courant Institute of Mathematical Sciences and Google Research, New York, NY

Ameet Talwalkar AMEET@CS.NYU.EDU
Courant Institute of Mathematical Sciences, New York, NY

Abstract

The problem of identifying the minimal gene set required to sustain life is of crucial importance in understanding cellular mechanisms and designing therapeutic drugs. This work describes several kernel-based solutions for predicting essential genes that outperform existing models while using less training data. Our first solution is based on a semi-manually designed kernel derived from the Pfam database, which includes several Pfam domains. We then present novel and general *domain-based* sequence kernels that capture sequence similarity with respect to several domains made of large sets of protein sequences. We show how to deal with the large size of the problem – several thousands of domains with individual domains sometimes containing thousands of sequences – by representing and efficiently computing these kernels using automata. We report results of extensive experiments demonstrating that they compare favorably with the Pfam kernel in predicting protein essentiality, while requiring no manual tuning.

1. Motivation

Identifying the minimal gene set required to sustain life is of crucial importance for understanding the fundamental requirements for cellular life and for selecting therapeutic drug targets. Gene knockout studies and RNA interference are experimental techniques for identifying an organism's "essential" genes, or the set of genes whose removal is lethal to the organism. However, these techniques are expensive and time-consuming. Recent work has attempted to extract from experimental knockout studies relevant features of essentiality, which aid in identifying essential genes in organisms lacking experimental results.

Several features have been proposed as indicators for essentiality, including evolutionary conservation, protein size, and number of paralogs (Chen & Xu, 2005). Using these basic features, Chen and Xu (2005) constructed a model of essentiality for *S. cerevisiae* (baker's yeast). Using Naive Bayes Classifiers (NBC), Gustafson et al. (2006) subsequently created a model of essentiality for *S. cerevisiae* and *E. coli* using an extended set of features generated from sequence data.

This work presents kernel methods to improve upon existing models. We first use several sequence kernels recently introduced by the computational biology community and show that the Pfam kernel (Ben-Hur & Noble, 2005) is most effective in selecting essential genes for *S. cerevisiae*. The Pfam kernel has recently been applied successfully in several biologically motivated learning tasks, and is generated from the Pfam database, the leading resource for storing protein family classification and protein domain data. However, the Pfam database is an ad-hoc solution relying on semi-manually tuned information.

In the second part of this work, we design general sequence kernels that produce effective similarity measures while bypassing the manual tuning of the Pfam database. We present two sequence kernels that are instances of rational kernels, a class of sequence kernels defined by weighted automata that are effective for analyzing variable-length sequences (Cortes et al., 2004). Using automata to represent and compute these ker-

Appearing in *Proceedings of the 25th International Conference on Machine Learning*, Helsinki, Finland, 2008.

nels is crucial in order to handle the large number of Pfam domains and the size of each of domain – we work with 6190 domains with the largest domain containing over 3000 protein sequences. These novel kernels are designed from the same domain-specific data used by the Pfam library, and we show how they compare favorably to the Pfam kernel at predicting protein essentiality. They are general *domain-based* kernels that can be used in many problems in bioinformatics or other applications where similarity needs to be defined in terms of proximity to several large sets of sequences.

The remainder of the paper is organized as follows. Section 2 describes the various sequence kernels and outlines the model used to improve prediction accuracy of protein essentiality in *S. cerevisiae*. Section 3 describes and analyzes the novel rational kernels we present as alternatives to the Pfam kernel. Section 4 presents the results of extensive experiments comparing these domain-based kernels to the Pfam kernel.

2. Pfam-Based Solution

Our first model uses Support Vector Machines (SVMs) (Cortes & Vapnik, 1995) to predict protein essentiality with choices of kernels including the RBF kernel as well as three sequence kernels. In the following subsections, we define the sequence kernels, outline the experimental design, and present our first results.

2.1. Sequence Kernels

Pfam Kernel

The Pfam database is a collection of multiple sequence alignments and Hidden Markov Models (HMMs) representing many common protein domains and families. Pfam version 10.0 contains 6190 domains, and for each domain an HMM is constructed from a set of proteins experimentally determined to be part of the domain ('seed' proteins). Each HMM is trained using a manually-tuned multiple alignment of the seed proteins with gaps inserted to normalize sequence length. Once constructed, the HMM is evaluated in an ad-hoc fashion and the entire process is repeated if the alignment is deemed 'unsatisfactory.' See (Sonnhammer et al., 1997) for further details.

When applied to a test sequence, a Pfam domain HMM generates an E-value statistic that measures the likelihood of the test sequence containing the domain. Given a dataset of protein sequences, the Pfam sequence kernel matrix is constructed by representing each protein in the dataset as a vector of 6190 log E-values and computing explicit dot products from these feature vectors (Ben-Hur & Noble, 2005). The

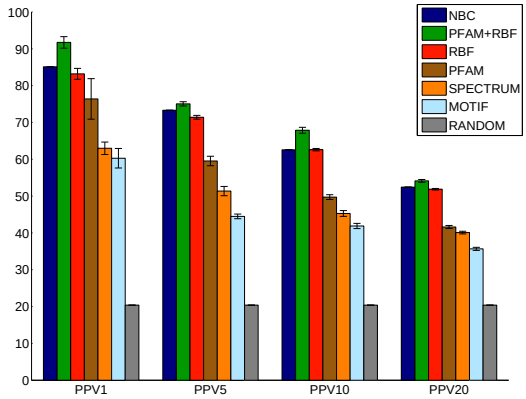

Figure 1. SVM's performance for RBF and Sequence Kernels using a reduced training set. Note that accuracy for NBC corresponds to a model trained on 50% of training data.

Pfam kernel has recently been applied successfully in learning tasks ranging from protein function (Lanckriet et al., 2004) to protein-protein interaction (Ben-Hur & Noble, 2005).

Spectrum and Motif Kernels

The Spectrum and Motif kernels are two recently proposed sequence kernels used in learning tasks to estimate protein similarity (Leslie & Kuang, 2004; Ben-Hur & Brutlag, 2003). Both kernels model a protein in a feature space of subsequences, with each feature measuring the extent to which the protein contains a specific subsequence. The Spectrum kernel models proteins in a feature space of all possible n-grams, representing each protein as a vector of n-gram counts (in our studies we set $n = 3$). Alternatively, the Motif kernel uses a feature space consisting of a set of discrete sequence motifs (we use a set of motifs extracted from the eMotif database (Ben-Hur & Noble, 2005)). For both kernels, the resulting kernel matrices are computed as an explicit dot product using these features.

2.2. Data

We used the dataset of *S. cerevisiae* proteins from Gustafson et al. (2006), consisting of 4728 yeast proteins of which 20.4% were essential. We constructed the RBF kernel matrix from a set of 16 features generated directly from protein sequences, corresponding to the 'easily attainable' features from Gustafson et al. (2006). We used data from Ben-Hur and Noble (2005) to construct the Pfam, Spectrum and Motif kernel matrices, each of which was constructed following the steps outlined in Section 2.1 and subsequently centered and normalized. In addition to the RBF and the three sequence kernels, we also used a combined Pfam/RBF

kernel, which we computed by additively combining the RBF kernel matrix with the normalized Pfam kernel matrix (RBF kernels are by definition normalized).

2.3. Experimental Design

We ran experiments with 100 trials. For each trial, we randomly chose 8.3% of the data as a training set and used the remaining points as a test set, subject to the constraint that an equal number of essential proteins were in each set.[1] We used the training set to train an SVM, and used the resulting model to make predictions on the test set in the form of probabilities of essentiality. We used libsvm's functionality (Chang & Lin, 2001) to estimate the outputs of an SVM as probabilities by fitting its results to a sigmoid function (Platt, 2000). To calculate the predicted probability of essentiality for each protein, we took the average over the predictions from each trial in which the protein appeared in the test set.

We measured the accuracy of the model for the proteins with the highest predicted probability of essentiality, using positive predictive value (PPV) as a performance indicator. Grid search was used to determine the optimal values for parameters C and gamma. Standard deviations were calculated from 10 'super-trials,' each corresponded to a 100-trial experiment described above. The Naive Bayes classifier (NBC) results were taken from Gustafson et al. (2006) and standard deviations were not reported.

2.4. First Results

Figure 1 shows SVM's performance using the set of kernels outlined above. The results show that the Pfam kernel is the most effective of the three sequence kernels at predicting essentiality. They also clearly show that the combined Pfam/RBF kernel outperforms all other models. The importance of the phyletic retention feature is a possible reason for the superior performance of the combined kernel compared with Pfam alone. As shown by Gustafson et al. (2006) and verified in our work, phyletic retention (a measure of gene conservation across species) is a powerful predictor of essentiality. This feature is used by RBF but not by Pfam (or by the domain-based kernels defined in Section 3) because it requires comparing sequences across organisms.

These results improve upon the leading model for prediction of protein essentiality while reducing the size of the training set more than five fold. Further, this is the first result showing the effectiveness of the Pfam kernel for this task, a fact that motivates the following sections of this paper, in which we seek a more general alternative to the Pfam kernel.

[1] Gustafson et al. (2006) used 50% of the data for training, but otherwise, our experimental designs are identical.

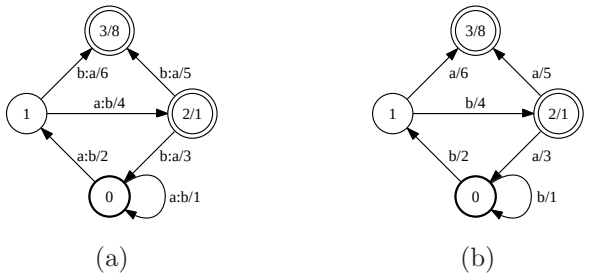

Figure 2. (a) Example of weighted transducer T. (b) Example of weighted automaton A. A can be obtained from T by projection on the output and $T(aab, bba) = A(bba) = 1 \times 2 \times 6 \times 8 + 2 \times 4 \times 5 \times 8$.

3. Domain-Based Sequence Kernels

In the previous section, we tested various sequence kernels, all introduced precisely to compute the similarity between protein sequences. Our results showed that the Pfam kernel was the most effective of these kernels, and we now aim to find a more general solution free of the manual tuning associated with the Pfam database.

Specifically, we wish to determine a method to extract similarities between protein sequences based on their similarities to several domains, each represented by a set of sequences, i.e., Pfam domains. Although several sequence kernels have been recently introduced in the machine learning community, e.g., mismatch, gappy, substitution and homology kernels (Leslie & Kuang, 2004; Eskin & Snir, 2005), none of these kernels provides a solution to our domain-based learning problem. Indeed, these kernels are not designed to efficiently compute the similarity between a string and a large set of strings, which in our case consists of 6190 Pfam domains each containing tens to thousands of sequences.

Alternatively, large sets of strings, such as the Pfam domains, can be efficiently represented by minimal deterministic automata. Hence, an efficient way to define a similarity measure between such sets is to use automata-based kernels. This leads us to consider the framework for automata-based kernels proposed by Cortes et al. (2004). An additional benefit of this framework is that most commonly used string kernels are special instances of this scheme.

3.1. Representation and Algorithms

We will follow the definitions and terminology given by Cortes et al. (2004). The representation and computation of the Domain-based kernels are based on *finite-state transducers*, which are finite automata in which each transition is augmented with an output label in addition to the familiar input label (Salomaa & Soittola, 1978). Input (output) labels are concatenated along a path to form an input (output) sequence. *Weighted transducers* are finite-state transducers in which each transition carries some weight in addition to the input and output labels. The weights of the transducers considered in this paper are real values.

Figure 2(a) shows an example of a weighted finite-state transducer. In this figure, the input and output labels of a transition are separated by a colon delimiter, and the weight is indicated after the slash separator. A weighted transducer has a set of initial states represented in the figure by a bold circle, and a set of final states, represented by double circles. A path from an initial state to a final state is an accepting path.

The weight of an accepting path is obtained by first multiplying the weights of its constituent transitions and multiplying this product by the weight of the initial state of the path (which equals one in our work) and the weight of the final state of the path (displayed after the slash in the figure). The weight associated by a weighted transducer T to a pair of strings $(x, y) \in \Sigma^* \times \Sigma^*$ is denoted by $T(x, y)$ and is obtained by summing the weights of all accepting paths with input label x and output label y. For example, the transducer of Figure 2(a) associates the weight 416 to the pair (aab, bba) since there are two accepting paths labeled with input aab and output bba: one with weight 96 and another one with weight 320.

The standard operations of sum $+$, product or concatenation $\cdot$, and Kleene-closure * can be defined for weighted transducers (Salomaa & Soittola, 1978). For any pair of strings (x, y),

$$\begin{aligned}(T_1 + T_2)(x, y) &= T_1(x, y) + T_2(x, y)\\ (T_1 \cdot T_2)(x, y) &= \sum_{\substack{x_1x_2=x\\ y_1y_2=y}} T_1(x_1, y_1) \cdot T_2(x_2, y_2). \end{aligned} \quad (1)$$

For any transducer T, T^{-1} denotes its *inverse*, that is the transducer obtained from T by swapping the input and output labels of each transition. For all $x, y \in \Sigma^*$, we have $T^{-1}(x, y) = T(y, x)$.

The *composition* of two weighted transducers T_1 and T_2 with matching input and output alphabets Σ, is a weighted transducer denoted by $T_1 \circ T_2$ when the sum:

$$(T_1 \circ T_2)(x, y) = \sum_{z \in \Sigma^*} T_1(x, z) \cdot T_2(z, y) \quad (2)$$

is well-defined and in $\mathbb{R}$ for all x, y (Salomaa & Soittola, 1978).

Weighted automata can be defined as weighted transducers A with identical input and output labels, for any transition. Since only pairs of the form (x, x) can have a non-zero weight associated to them by A, we denote the weight associated by A to (x, x) by $A(x)$ and call it the *weight associated by A to x*. Similarly, in the graph representation of weighted automata, the output (or input) label is omitted. Figure 2(b) shows an example of a weighted automaton. When A and B are weighted automata, $A \circ B$ is called the *intersection* of A and B. Omitting the input labels of a weighted transducer T results in a weighted automaton which is said to be the *output projection of T*.

3.2. Automata-Based Kernels

A string kernel $K : \Sigma^* \times \Sigma^* \to \mathbb{R}$ is *rational* if it coincides with the function defined by a weighted transducer U, that is for all $x, y \in \Sigma^*$, $K(x, y) = U(x, y)$. Not all rational kernels are *positive definite and symmetric* (PDS), or equivalently verify the Mercer condition, which is crucial for the convergence of training for discriminant classification algorithms such as SVMs. But, for any weighted transducer T, $U = T \circ T^{-1}$ is guaranteed to define a PDS kernel (Cortes et al., 2004).

Furthermore, most rational kernels used in computational biology and natural language processing are of this form (Haussler, 1999; Leslie & Kuang, 2004; Lodhi et al., 2002; Zien et al., 2000; Collins & Duffy, 2001; Cortes & Mohri, 2005). For instance, the n-gram kernel is a rational kernel. The n-gram kernel K_n is defined as

$$K_n(x, y) = \sum_{|z|=n} c_x(z)c_y(z), \quad (3)$$

where $c_x(z)$ is the number of occurrences of z in x. K_n is a PDS rational kernel since it corresponds to the weighted transducer $T_n \circ T_n^{-1}$ where the transducer T_n is defined such that $T_n(x, z) = c_x(z)$ for all $x, z \in \Sigma^*$ with $|z| = n$. The transducer T_2 for $\Sigma = \{a, b\}$ is shown in Figure 3.

We will now extend this formalism to measure the similarity between domains, or sets of strings represented by an automaton. Let us define the count of a string z in a weighted automaton A as:

$$c_A(z) = \sum_{u \in \Sigma^*} c_u(z)A(u). \quad (4)$$

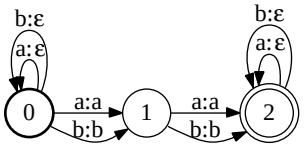

Figure 3. Counting transducer T_2 for $\Sigma = \{a, b\}$.

The similarity between two sets of strings represented by the weighted automata A and B according to n-gram kernel K_n can then be defined by:

$$\begin{aligned} K_n(A, B) &= \sum_{x,y \in \Sigma^*} (A \circ T_n \circ T_n^{-1} \circ B)(x, y) \\ &= \sum_{|z|=n} c_A(z) c_B(z). \end{aligned} \tag{5}$$

Other rational kernels can be extended into a similarity measure between sets of strings in the same way. We will now define two families of kernels that can be used in a variety of applications where similarity with respect to domains is needed.

3.3. Independent Domain Kernel

The Independent Domain kernel (IDK) measures the similarity between two sequences in our dataset $\mathcal{D}$ by comparing their similarities to each domain, e.g., Pfam domains.[2] For the i-th Pfam domain (with $1 \leq i \leq P = 6190$), let $\mathcal{P}_i$ be the set of all seed protein sequences for that domain. Each sequence in $\mathcal{P}_i$ is represented as a string in an alphabet, Σ, consisting of $|\Sigma| = 21$ characters, 20 for different amino acids plus an optional gap character used to represent gaps in the seed alignment. We denote by D_i the minimal deterministic automaton representing the set of strings $\mathcal{P}_i$. For a given sequence x in our dataset, we can use the n-gram kernel K_n to compute the similarity between x and the i-th Pfam domain $\mathcal{P}_i$: $K_n(x, D_i)$. This leads to an overall similarity measure vector $s(x) \in \mathbb{R}^P$ between x and the set of domains: $s(x) = (K_n(x, D_1), \ldots, K_n(x, D_P))$. We now define the IDK K_I as, for all x, y in Σ^*:

$$\begin{aligned} K_I(x, y) &= \sum_{i=1}^{P} K_n(x, D_i) K_n(y, D_i) \\ &= \sum_{i=1}^{P} \Big(\sum_{|z|=n} c_x(z) c_{D_i}(z) \Big) \Big(\sum_{|z|=n} c_y(z) c_{D_i}(z) \Big). \end{aligned} \tag{6}$$

K_I is PDS since it is constructed via an explicit dot-product. Any PDS kernel K with positive eigenvalues can be normalized to take values between 0 and 1 by defining K' as

$$K'(x, y) = \frac{K(x, y)}{\sqrt{K(x, x) K(y, y)}}. \tag{7}$$

We apply this normalization to K_I to account for the varying lengths of proteins in our dataset, since longer proteins will contain more n-grams and will thus tend to have more n-gram similarity with every domain.

The kernel K_I can be efficiently computed by computing $K_n(x, D_i)$ for all $1 \leq i \leq P$ as follows:

1. Compute D_i for each $\mathcal{P}_i$ by representing each sequence in $\mathcal{P}_i$ by an automaton and determinizing and minimizing the union of these automata.
2. For all $1 \leq i \leq P$ compute $T_n \circ D_i$, and for each sequence x in the dataset compute $T_n \circ X$, where X is the automaton representing x. Optimize the results by projecting onto outputs, applying epsilon-removal, determinizing and minimizing to obtain weighted automata A_i and Y_x.
3. Compute $K_n(x, D_i)$ by intersecting A_i and Y_x and using a generic single-source shortest-distance algorithm (Cortes et al., 2004) to compute the sum of all the paths in the resulting automaton.

The complexity of computing $K_n(x, D_i)$ for a fixed set of domains grows linearly in the length of x, hence the complexity of computing $K_I(x, y)$ grows linearly in the sum of the length of x and y, *i.e.* in $O(|x| + |y|)$. Thus, this kernel is efficient to compute. However, it does not capture the similarity of two sequences in as fine a way as the next kernel we present.

3.4. Joint Domain Kernel

Let us consider two sequences x and y and a given domain $\mathcal{P}_i$. Let $\mathcal{X}$ be the set of n-grams in common between x and $\mathcal{P}_i$, and $\mathcal{Y}$ the set of n-grams in common between y and $\mathcal{P}_i$. When computing the similarity between x and y according to $\mathcal{P}_i$, the IDK K_I takes into account all n-grams in common between x and $\mathcal{P}_i$ and between y and $\mathcal{P}_i$, i.e., all the n-grams in $\mathcal{X} \cup \mathcal{Y}$, regardless of whether these n-grams appear in both x and y. Thus, K_I may indicate that x and y are similar according to $\mathcal{P}_i$ even if $\mathcal{X}$ and $\mathcal{Y}$ differ significantly, or in other words, even if x and y are similar to $\mathcal{P}_i$ for different reasons. In contrast, the Joint Domain kernel (JDK) only takes into consideration the n-grams in common to x, y and $\mathcal{P}_i$, that is the n-grams in $\mathcal{X} \cap \mathcal{Y}$, when determining the similarity between x and y. It will thus declare x and y similar according to $\mathcal{P}_i$ iff x and y are similar to $\mathcal{P}_i$ for the same reason.

[2]Both the IDK and spectrum kernels represent sequences as vectors of n-gram counts but only the IDK accounts for the n-gram spectrums of the domains of interest.

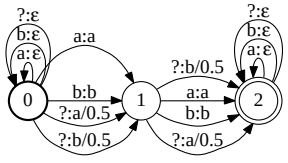

Figure 4. Counting transducer $\overline{T}_2$ with $\Sigma = \{a, b\}$, '?' representing the gap symbol and an expansion penalty weight of 0.5.

For each domain $\mathcal{P}_i$, the JDK defines a kernel K_i that measures the similarity between two sequences x and y according to $\mathcal{P}_i$, using as a similarity measure the count of the n-grams in common among x, y and $\mathcal{P}_i$. More precisely, we define $K_i : \Sigma^* \times \Sigma^* \to \mathbb{R}$ as follows:

$$K_i(x, y) = \sum_{|z|=n} c_x(z) c^2_{D_i}(z) c_y(z). \tag{8}$$

Each K_i is normalized as shown in Equation 7. We then combine these P kernels to obtain the kernel $K_J : \Sigma^* \times \Sigma^* \to \mathbb{R}$ defined as follows:

$$\begin{aligned} K_J(x, y) &= \sum_{i=1}^{P} K_i(x, y) \\ &= \sum_{i=1} \sum_{|z|=n} c_x(z) c^2_{D_i}(z) c_y(z). \end{aligned} \tag{9}$$

We will now show that each K_i and thus K_J is a PDS rational kernel. Let A_i be the weighted automata obtained by composing D_i with T_n and projecting the result onto its output: $A_i = \pi_2(D_i \circ T_n)$. From the definition of T_n, it follows that $A_i(z) = c_{D_i}(z)$ and $c_x(z) = T_n(x, z)$ for all $|z| = n$. Thus, for all (x, y), $K_i(x, y)$ can be rewritten as:

$$\begin{aligned} K_i(x, y) &= \sum_{|z|=n} T_n(x, z) A_i(z) A_i(z) T_n(y, z) \\ &= (T_n \circ A_i \circ A_i \circ (T_n)^{-1})(x, y). \end{aligned} \tag{10}$$

Observe that $(T_n \circ A_i)^{-1} = A_i^{-1} \circ T_n^{-1} = A_i \circ T_n^{-1}$ since for an automaton the inverse A_i^{-1} coincides with A_i. Thus, $K_i(x, y) = \big((T_n \circ A_i) \circ (T_n \circ A_i)^{-1}\big)(x, y)$, which is of the form $T \circ T^{-1}$ and thus K_i is PDS. Since PDS kernels are closed under sum, K_J is also PDS.

The computation of the kernel K_J is more costly than that of K_I since a full $D \times D$ kernel matrix needs to be computed for each Pfam domain. This leads to $D^2 \times P$ rational kernel computations to compute K_J, compared to only $D \times P$ rational kernel computations for K_I. This is significant when $P = 6190$. Thus, it is important to determine an efficient way to compute the kernels K_i. The following is an efficient method for computing K_J that we adopt in our experiments, in which the complexity of computing $K_J(x, y)$ for a fixed set of domains linearly depends on the product of the length of x and y, *i.e.* in $O(|x||y|)$:

1. Compute each A_i and optimize using epsilon-removal, determinization and minimization.
2. For each sequence x in the dataset, compute $Y_x = \pi_2(T_n \circ X)$ where X is the automaton representing x and optimize Y_x using epsilon-removal, determinization and minimization.
3. $K_i(x, y)$ is obtained by computing $A_i \circ Y_x$ and $A_i \circ Y_y$, computing the intersection of the resulting automata and using a generic single-source shortest-distance algorithm (Cortes et al., 2004) to compute the sum of all paths in the resulting automaton.

Gap Symbol Handling

The sequence alignments in the Pfam domain ($\mathcal{P}_i$) contain a gap symbol used to pad the alignments. In the previous two sections, we either ignored the gap symbol (when dealing with raw sequence data) or treated it as a regular symbol (when dealing with aligned sequences). In the latter case, since this symbol does not appear in the sequences in the dataset, the result is that all n-grams containing the gap symbol are ignored during the computation of K_I and K_J.

Alternatively, we can treat the gap symbol as a wildcard, allowing it to match any regular symbol. This can be achieved by modifying the transducer T_n to match any gap symbol on its input to any regular symbol on its output (these transitions can also be assigned a weight to penalize gap expansion). We denote by $\overline{T}_n$ the resulting transducer and replace T_n by $\overline{T}_n$ when composing with D_i. Figure 4 shows $\overline{T}_2$ for $|\Sigma| = \{a, b\}$ with the symbol '?' representing the gap symbol and an expansion penalty weight of 0.5.

3.5. Domain Kernels Based on Moments of Counts

Although counting common n-grams leads to informative kernels, this technique affords a further generalization that is particularly suitable when defining kernels for domains. We can view the sum of the counts of an n-gram in a domain as its average count after normalization. One could extend this idea to consider higher moments of the counts of an n-gram, as this could capture useful information about how similarity varies across the sequences within a single domain.

Remarkably, it is possible to design efficiently computable domain kernels based on these quantities by

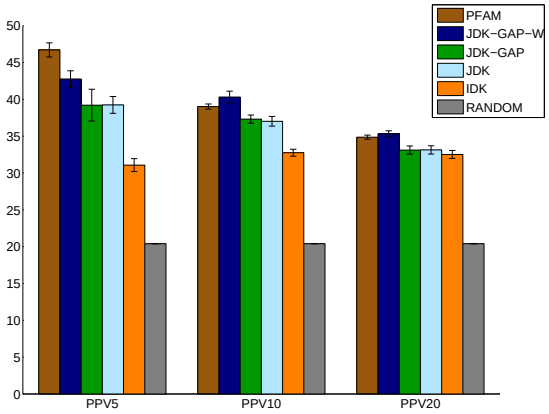

Figure 5. SVM's performance with various kernels averaged over all datasets.

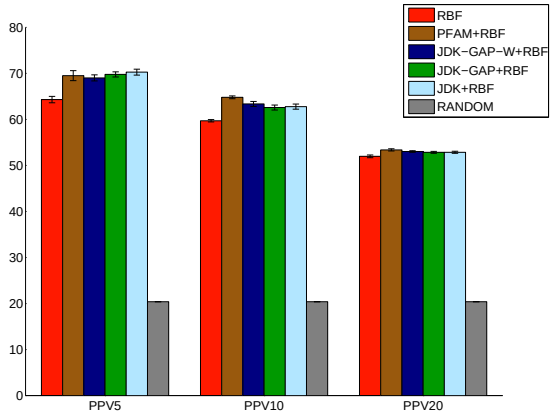

Figure 6. SVM's performance with various kernels combined with RBF kernel averaged over all datasets.

generalizing the domain kernels from Sections 3.3 and 3.4 in a way similar to what is described by Cortes and Mohri (2005). Let m be a positive integer. We can define the *m-th moment of the count* of the sequence z in a weighted automata A, denoted by $c_{A,m}(z)$, as:

$$c_{A,m}(z) = \sum_{u \in \Sigma^*} c_u^m(z) A(u). \tag{11}$$

Both of our kernel families can then be generalized to a similarity measure based on the m-th moment of the n-gram counts as follows:

$$K_m^I(x,y) = \sum_{i=1}^{P} \Big(\sum_{|z|=n} c_{x,m}(z) c_{D_i,m}(z) \Big) \Big(\sum_{|z|=n} c_{y,m}(z) c_{D_i,m}(z) \Big)$$

$$K_m^J(x,y) = \sum_{i=1} \sum_{|z|=n} c_{x,m}(z) c_{D_i,m}^2(z) c_{y,m}(z).$$

These kernels can be efficiently computed by using, in place of T_n, a transducer T_m^n that can be defined such that $T_m^n(x,z) = (c_x(z))^m = c_{x,m}(z)$ for all $x, z \in \Sigma^*$ with $|z| = n$.

4. Experimental Results

We evaluated the domain-based kernels described in Section 3 (with $n = 3$) using an experimental design similar to Section 2.3. In order to test these kernels under various conditions, we chose to work with datasets sampled from the yeast dataset used in Section 2. We constructed 10 datasets, each containing 500 sampled data points randomly chosen from the 4728 initial points, subject to the constraint that we maintained the same ratio of positively and negatively labeled points. We worked on a large cluster machines and used the OpenFst and OpenKernel libraries to construct similarity matrices for each sample dataset for varying kernels (Allauzen et al., 2007; Allauzen & Mohri, 2007). Generating similarity matrices took less than 30 minutes for IDK, 1 hour for JDK, and 2.5 hours for JDK with gaps treated as wildcards. We do not show results for the top 1% since it is an unreliable statistic when working with 500 points. In all reported results we exclude results from one sampled dataset that generated pathological results for all sequence kernels.

Figure 5 shows the average prediction performance over the sampled datasets for various kernels. The figure shows that the average performance of the JDK with gaps treated as wildcards (JDK-GAPS-W) is slightly better than the Pfam kernel, as it outperforms the Pfam kernel for the top 10% and 20% predictions. The figure also presents results for variants of the JDK that either ignore gaps in the seed alignment (JDK) or treat them as a distinct symbol (JDK-GAPS). The results show that, regardless of the treatment of gaps, the JDK drastically outperforms the IDK.

Based on these results, we next tested the effectiveness of the JDK combined with the RBF kernel. Similar to the results in Figure 1, average prediction performance over the sampled datasets was better using combination kernels in contrast to any kernel alone.[3] Figure 6 shows that the combined JDK is comparable to the combined Pfam kernel. Further, in contrast to the results in Figure 5, the treatment of gaps by the JDK does not significantly alter prediction efficiency. In other words, the JDK is able to match the best results of the Pfam kernel using only raw Pfam sequence data (JDK), while completely ignoring the hand-curated multiple sequence alignments that are vital to parameterizing the HMMs of the Pfam Library. We did not perform experiments using higher moments of the count, as described in Section 3.5, though we suspect

[3] As in Figure 1, RBF alone outperforms all sequence kernels alone, possibly due to the phyletic retention feature.

that these more refined kernels would lead to further improvements over the Pfam kernel.

5. Conclusion

We presented novel domain-based sequence kernels that require no hand-crafted information, in contrast to the Pfam kernel. The joint domain kernels we defined were shown to match or outperform the best previous results for predicting protein essentiality. These kernels and their generalization based on moments of counts can be used for any application requiring similarity between sequences that may be extracted from proximity to several large sequence domains. In bioinformatics, such applications may include remote homology prediction, subcellular localization, and prediction of protein-protein interaction.

6. Acknowledgments

This work was partially supported by the National Science Foundation award number MCB-0209754 and the New York State Office of Science Technology and Academic Research (NYSTAR), and was also sponsored in part by the Department of the Army Award Number W81XWH-04-1-0307. The U.S. Army Medical Research Acquisition Activity, 820 Chandler Street, Fort Detrick MD 21702-5014 is the awarding and administering acquisition office. The content of this material does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

References

Allauzen, C., & Mohri, M. (2007). OpenKernel library. `http://www.openkernel.org`.

Allauzen, C., Riley, M., Schalkwyk, J., Skut, W., & Mohri, M. (2007). OpenFst: a general and efficient weighted finite-state transducer library. *CIAA 2007* (pp. 11–23). Springer. `http://www.openfst.org`.

Ben-Hur, A., & Brutlag, D. L. (2003). Remote homology detection: a motif based approach. *ISMB (Supplement of Bioinformatics)* (pp. 26–33).

Ben-Hur, A., & Noble, W. (2005). Kernel methods for predicting protein-protein interactions. *Bioinformatics*, *21*, 38–46.

Chang, C.-C., & Lin, C.-J. (2001). *LIBSVM: a library for support vector machines*. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Chen, Y., & Xu, D. (2005). Understanding protein dispensability through machine learning analysis of high-throughput data. *Bioinformatics*, *21*, 575–581.

Collins, M., & Duffy, N. (2001). Convolution kernels for natural language. *NIPS 2001* (pp. 625–632).

Cortes, C., Haffner, P., & Mohri, M. (2004). Rational Kernels: Theory and Algorithms. *Journal of Machine Learning Research*, *5*, 1035–1062.

Cortes, C., & Mohri, M. (2005). Moment kernels for regular distributions. *Machine Learning*, *60*, 117–134.

Cortes, C., & Vapnik, V. N. (1995). Support-Vector Networks. *Machine Learning*, *20*, 273–297.

Eskin, E., & Snir, S. (2005). The Homology Kernel: A Biologically Motivated Sequence Embedding into Euclidean Space. *CIBCB* (pp. 179–186).

Gustafson, A., Snitkin, E., Parker, S., DeLisi, C., & Kasif, S. (2006). Towards the identification of essential genes using targeted genome sequencing and comparative analysis. *BMC:Genomics*, *7*, 265.

Haussler, D. (1999). *Convolution Kernels on Discrete Structures* (Technical Report UCSC-CRL-99-10). University of California at Santa Cruz.

Lanckriet, G., Deng, M., Cristianini, N., Jordan, M., & Noble, W. (2004). Kernel-based data fusion and its application to protein function prediction in yeast. *Pacific Symposium on Biocomputing* (pp. 300–311).

Leslie, C. S., & Kuang, R. (2004). Fast String Kernels using Inexact Matching for Protein Sequences. *Journal of Machine Learning Research*, *5*, 1435–1455.

Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., & Watskins, C. (2002). Text classification using string kernels. *Journal of Machine Learning Research*, *2*, 419–44.

Platt, J. (2000). Probabilities for support vector machines. In *Advances in large margin classifiers*. Cambridge, MA: MIT Press.

Salomaa, A., & Soittola, M. (1978). *Automata-Theoretic Aspects of Formal Power Series*. Springer.

Sonnhammer, E., Eddy, S., & Durbin, R. (1997). Pfam: A comprehensive database of protein domain families based on seed alignments. *Proteins: Structure, Function and Genetics*, *28*, 405–420.

Zien, A., Rätsch, G., Mika, S., Schölkopf, B., Lengauer, T., & Müller, K.-R. (2000). Engineering Support Vector Machine Kernels That Recognize Translation Initiation Sites. *Bioinformatics*, *16*, 799–807.