FRED - Version 1.0
Tutorial and API Documentation

14th May 2009

Contents

Chapter 1

Introduction

FRED

This document is structured as follows: In the first chapter, we will give a general introduction to FRED and provide some general comments on the installation and usage. The second chapter provides a tutorial on how to build prediction pipelines using FRED. In the third chapter you can find the documentation of the API of FRED, providing information about the objects and functions of FRED and their input and output parameters. The documentation is also available as html version with the FRED package (FRED/doc).

Over the last decade, immunoinformatics has made significant progress. Computational approaches, in particular the prediction of T-cell epitopes using machine learning methods, are at the core of modern vaccine design. FRED is a an extensible open-source software framework for key tasks in immunoinformatics. In its first version, FRED offers easily accessible prediction methods for MHC binding and antigen processing as well as general infrastructure for the handling of antigen sequence data and epitopes. FRED is implemented in Python in a modular way and allows the integration of external methods.

FRED is a valuable tool to accomplish large-scale analyses in immunoinformatics and also a software framework for the development of novel immunoinformatics methods. Ease of use, extensibility, and openness make it an ideal tool for addressing complex immunoinformatics problems in an uncomplicated manner.

FRED is published under the GNU Lesser General Public License (LPGL). The LPGL is available at (`http://www.gnu.org/copyleft/lesser.html`). See 1 for details.

FRED is available at
`http://www-bs.informatik.uni-tuebingen.de/Software/FRED/`. For questions, comments, or suggestions please contact Magdalena Feldhahn (feldhahn@informatik.uni-tuebingen.de).

FRED provides methods for sequence input, sequence preprocessing, filtering, and display of the results. The general organization of FRED is shown in Fig. 1.1. The single prediction methods are accessed internally via a consistent interface. FRED can handle polymorphic sequences (e.g. for the study of SNPs in an epitope context) and offers the possibility to access different methods simultaneously and to combine, compare, or benchmark the methods. FRED is easily extendable by user-defined prediction methods or methods for filtering of the results. FRED is implemented in Python (V 2.6) (`www.python.org`). All additional software required for FRED is freely available and installation packages are included in the FRED package.

Python is a simple-to-use but powerful programming language that can be used even by non programmers

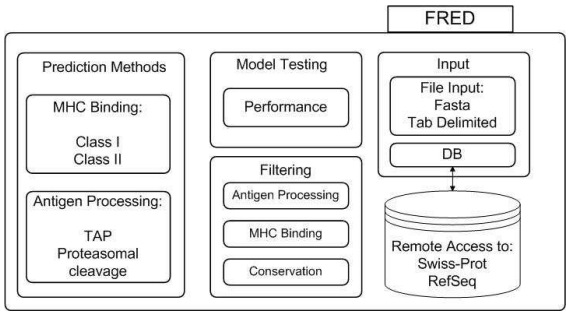


Figure 1.1: FRED is organized in four major parts for sequence input, application of prediction methods, filtering of the results, and model testing. The prediction methods currently included into FRED are listed in Table 1.1.

with a reasonable amount of effort. A general python tutorial and an beginners guide can by found on the Python project website (www.python.org/doc).

MHC Binding:	
SYFPEITHI	[1]
SVMHC	[2]
BIMAS	[3]
NetMHCpan [1]	[4]
NetMHC [1]	[5]
Hammer	[6]
NetMHCIIpan [1]	[7]
Proteasomal Cleavage:	
PCM method from WAPP	[8]
TAP Transport:	
SVMTAP	[8]
Additive Matrix Method	[9]

Table 1.1: Prediction methods currently integrated in FRED. [1]Installation of external software is required. Due to licensing issues, we could not include the stand-alone versions of these methods in the FRED package. See the next section for details.

Installation

The Python code of FRED is platform-independent and does not require a particular operating system, although it has been developed mostly on Linux. We have also successfully installed FRED on other Unix-derived platforms, e.g., MacOS X, and on Windows using cygwin (http://www.cygwin.com/).

First of all, you need to have Python installed (Python 2.6), http://www.python.org/.

FRED requires some additional software:

- *biopython (v1.49)*
- *svmlight (v6.02)* for support vector classification
- *clustalw (v2.0.20)* for sequence alignment

All additional software required for FRED is freely available and installation packages are included in the FRED package (FRED/contrib).

1. Then download FRED (FRED-1.0.tar.gz) from

http://www-bs.informatik.uni-tuebingen.de/Software/FRED/ and unpack it.

2. Install biophython (in FRED/contrib. See README.txt in FRED/contrib for details.)
3. Compile svmlight (in FRED/contrib. See README.txt in FRED/contrib for details.)
4. Install clustalw (in FRED/contrib. See README.txt in FRED/contrib for details.)
5. Optional: download and install the CBS methods NetMHC, NetMHCpan, and NetMHCIIpan (see below for details).
6. Call *'python setup.py install'* to build and install FRED.

Installing FRED under Windows using cygwin:
Install cygwin and make sure to include python, a C++ compiler, and the make tool. The two latter are needed to compile and make the svm tools and clustalw. Then follow the installation instructions for Linux.

We included the code to use some prediction methods from the Center of Biological Sequence Analysis (CBS) at the Technical University of Denmark, since these methods belong to the best ones currently available in this field. We provide code to use NetMHC [5], NetMHCpan [4], and NetMHCIIpan [7] within the FRED framework. We did not include all CBS methods because the architecture of FRED requires predictions for single peptides and not all of the CBS methods provide the "Peptide input" option. Due to licensing issues, we cannot provide the software directly. Academic users can download the stand-alone versions from the respective websites that are linked on (http://www.cbs.dtu.dk/services/). If you intend to use these methods, download the software and install it on you system. Follow the installation instructions included with the software. If you install the software in the correct directory before installing FRED, they will be detected automatically upon the installation of FRED and will then be available for use within FRED. For detailed instructions please refer to the README included with the FRED package.

Copyright

Why use FRED?

In this section we will present some possible applications of FRED. A more detailed description how to use FRED can be found in the tutorial in Chapter 2 More examples along with the required input data are included with the FRED package (FRED/examples).

The selection of peptides for epitope-based vaccines is a typical application for large-scale predictions of MHC binding peptides. Short and simple programs based on FRED can accomplish these prediction tasks.

The following program implements a typical scenario in the selection of conserved peptide candidates for a vaccine against a virus. The scenario is based on the paper by Toussaint et al. [10]: a set of sequences of the hepatitis C virus core protein from four different subtypes are used. All peptides that occur in at least 90% of the input sequences are considered candidates for conserved epitopes. Predictions are made for 29 HLA alleles using the BIMAS method [3].

```
1 models = [('MHC_I_BIMAS','A_0201_9'),('MHC_I_BIMAS','A_1101_9'),...
2 prot_set= Fred.Fred.ProteinSet()
3 prot_set.readFasta("hcv-core-1a1b2a3a.fasta")
4 pep_set = protset.getPeptides(9, 0.9, 0)[0]
5 pep_set.predict(models)
6 cands = Fred.Fred.FindCandidates_HalfmaxThresholds(pep_set,models,0.9,1)
7 cands.writeTabDelimitedFile('results_hcv_core.txt')
```

Alternatively, all nonameric peptides from a consensus sequence of the input sequences can be used as candidate peptides for the prediction by replacing lines 3 and 4 in the above example with:

```
1 seq = prot_set.getConsensus()
2 prot = Fred.Fred.ProteinSet()
3 prot.setFromStrings([seq])
4 pep_set = prot.getPeptides(9,0.0,0)[0]
```

Integration of new methods and performance evaluation:

The expected increase in available data will lead to the development of new prediction methods for MHC binding and immunogenicity. FRED allows the convenient integration of new methods. Methods not implemented in Python can be plugged in using command line calls. Using the example of state of the art prediction methods form CBS group at Copenhagen (NetMHC, NetMHCpan, NEtMHCIIpan; available from http://www.cbs.dtu.dk/services/) we demonstrate how new methods can be integrated into FRED FRED provides a number of standard measures to compare different prediction methods and to evaluate the performance w.r.t. experimental values (Matthews correlation coefficient, accuracy, sensitivity, specificity, area under the ROC curve, correlation, and rank correlation). Different prediction methods can thus be compared with ease.

Web server development:

Using FRED as basis for new applications in computational immunomics leads to a significant reduction of development time and allows the convenient combination of new methods with existing ones. An example for an application based on FRED is EpiToolKit (www.epitoolkit.org, [11], [12]). Only the web-based user interface and the data management in the web server had to be newly implemented. Prediction functionality of EpiToolKit is completely provided by FRED. Through the use of Python FRED can be integrated seamlessly in web servers/content management systems like Plone (http://www.plone.org/).

Remarks on the selection of the appropriate prediction method

The selection of the appropriate tool for epitope prediction is a critical and non-trivial task. The right choice of prediction methods largely depends on the application. We therefore do not want to recommend one prediction method, but provide a uniform interface for different methods. FRED offers simultaneous access to different methods and the possibility to assess their performance with respect to the given application. The user can thus use FRED to determine the best prediction methods for the respective task.

An extensive evaluation of different methods is beyond the scope of this work and has been done elsewhere. Large scale evaluations of different prediction method can e.g. be found in [13] and [14].

Chapter 2

Tutorial

The first part of this tutorial will guide you through the basics steps of implementing a prediction pipeline with FRED. The basic pipeline is structured as follows:

1. Getting peptides
2. Making predictions
3. Postprocessing the predictions

In the second part we shortly describe the analysis and evaluation tools available in FRED.

We included some examples into the FRED package (FRED/examples) that show how typical problems in computational immunomics can be solved using FRED. A short description of what these examples do is given in Section 2.3.

All code in FRED is well documented and commented. The documentation of the API of FRED is available as pdf (Chapter 3 in this document) or as html (FRED/doc). For details on the usage of specific functions in FRED please refer to the documentation.

2.1 Building a basic prediction pipeline

2.1.1 Getting Peptides

Before making predictions, we first need peptides to make predictions for. Peptides are stored in PeptideSets.

There are different possibilities to get peptides into a PeptideSet. A PeptideSet can either be derived from a set of proteins (ProteinSet) or the peptides can directly be obtained from a list of sequences. First, we will show how to get proteins into a ProteinSet and how to derive a PeptideSet from this.

1. Sequences from a list of strings
2. Sequences from a fasta file
3. Sequences from Swiss-Prot

4. Sequences from RefSeq

All four different possibilities are shown in the following code example. This example is also included into the FRED package (FRED/examples/example0_GettingProteins.py).

```
import sys
import Fred.Fred

# 1. Set sequences from a list of strings

seqs = ["MEVKGKKQFTGKSTKTAQEKNRFHKNSDSGSSKTFPTRKVAKEGGPKVTSRNFEKSITKLGKKGVKQFKNKQQGDKSPKNKFQPANKFNKKRKF
QPDGRSDESAAKKPKWDDFKKKKKELKQSRQLSDKTNYDIVVRAKQMWEILRRKDCDKEKRVKLMSDLQKLIQGKIKTIAFAHDSTRVIQCYIQYGNEEQRKQ
AFEELRDDLVELSKAKYSRNIVKKFLMYGSKPQIAEIIRSFKGHVRKMLRHAEASAIVEYAYNDKAILEQRNMLTEELYGNTFQLYKSADHRTLDKVLEVQPE
KLELIMDEMKQILTPMAQKEAVIKHSLVHKVFLDFFTYAPPKLRSEMIEAIREAVVYLAHTHDGARVAMHCLWHGTPKDRKVIVKTMKTYVEKVANGQYSHLV
LLAAFDCIDDTKLVKQIIISEIISSLPSIVNDKYGRKVLLYLLSPRDPAHTVREIIEVLQKGDGNAHSKKDTEVRRRELLESISPALLSYLQEHAQEVVLDKS
ACVLVSDILGSATGDVQPTMNAIASLAATGLHPGGKDGELHIAEHPAGHLVLKWLIEQDKKMKENGREGCFAKTLVEHVGMKNLKSWASVNRGAIILSSLLQS
CDLEVANKVKAALKSLIPTLEKTKSTSKGIEILEKLST","MINSTSTQPPDESCSQNLLITQQIIPVLYCMVFIAGILLNGVSGWIFFYVPSSKSFIIYLKN
IVIADFVMSLTFPFKILGDSGLGPWQLNVFVCRVSAVLFYVNMYVSIVFFGLISFDRYYKIVKPLWTSFIQSVSYSKLLSVIVWMLMLLLAVPNIILTNQSVR
EVTQIKCIELKSELGRKWHKASNYIFVAIFWIVFLLLIVFYTAITKKIFKSHLKSSRNSTSVKKKSSRNIFSIVFVFFVCFVPYHIARIPYTKSQTEAHYSCQ
SKEILRYMKEFTLLLSAANVCLDPIIYFFLCQPFREILCKKLHIPLKAQNDLDISRIKRGNTTLESTDTL"]
proteinSet = Fred.Fred.ProteinSet()
proteinSet.sequencesFromStrings(seqs)
print "\nProteins from strings:"
for protein in proteinSet.getMembers():
        print protein.getSequence()

# 2. Set sequences from a fasta file
proteinSet = Fred.Fred.ProteinSet()
proteinSet.readFasta('input_example0.fasta')
print "\nProteins from fasta:"
for protein in proteinSet.getMembers():
        print protein.getSequence()

# 3. Set sequences from Swiss-Prot
accessions = ['Q15397',"Q15391"]
proteinSet = Fred.Fred.ProteinSet()
proteinSet.accessSwissProt(accessions)
print "\nProteins from Swiss-Prot:"
for protein in proteinSet.getMembers():
        print protein.getSequence()

# 4. Set Sequences from RefSeq
accessions = ['NP_055693', 'NP_055694']
proteinSet = Fred.Fred.ProteinSet()
proteinSet.accessRefSeq(accessions)
print "\nProteins from RefSeq:"
for protein in proteinSet.getMembers():
        print protein.getSequence()
```

The next step is to derive a PeptideSet form the ProteinSet. Therefor we have to make some decisions:

- What peptide length do we want?
- Do we want to process the proteins separately? (The alternative is to put all peptides from all the proteins into one set.)
- Do we want to use a conservation threshold to filter the peptides? This only makes sense if we do not process peptides separately.

Let's assume we already have a ProteinSet called proteinSet. The function *getPeptides(peptidelength, conservation, protein_by_protein)* extracts peptides from the proteinSet. The parameters indicate the peptide length, the conservation threshold and how we want to process the proteins (set *protein_by_protein = 1* to handle proteins separatly, 0 otherwise.)

Note that *getPeptides()* will always return a list of peptideSets. If we set *protein_by_protein* = 0, this list will only contain one single peptideSet.

The following code will derive a single peptideSet from our proteinSet (actually it produces a list of pep-

tideSets , but we directly take the first element of the list).

```
peptideSets = proteinSet.getPeptides(9, 0.0, 0)
peptideSet = peptideSets[0]
```

As mentioned above we can also directly read in peptide sequences from a set of strings.

This is done with the following code. The second parameter of *getFromStrings* is the peptide length :

```
peptides = ['SYFPEITHI', 'SIINFEKL', 'ALALALALA']
peptideSet = Fred.Fred.PepSet()
peptideSet.setFromStrings(peptides, 9)
```

2.1.2 Making predictions

Now we have to decide which models we want to use to make predictions.

The different prediction methods along with references to the original publications can be found in Table 1.1. The allelic models that are available in in FRED are listed in Table 2.1.

Some remarks on choosing the right prediction method for your application can be found in 1.

A prediction model in FRED is a combination of a prediction method and an allelic model, that specifies the HLA allele and the peptide length, e.g. *('MHC_I_SYFPEITHI','A_0201_9')* for the Syfpetihi prediction for allele HLA-A*0201 and peptides of length 9.

The prediction methods for TAP transport and proteasomal cleavage are allele independent. Here, the allele part of the model is replaced with ", e.g. *('TAP_ADDITIVE',")*

FRED provides functions to read out available methods and models.

```
import Fred.Fred
print Fred.Fred.getAvailableMethods()
print Fred.Fred.getAvailableModels('MHC_I_SVMHC')
```

If no method is specified in *getAvailableModels*, the models for all methods are returned.

First we have to specify the prediction models, than we can finally make a prediction (assuming we constructed a peptideSet as described above):

```
models = [('MHC_I_SVMHC','A_2402_9'),('MHC_I_SYFPEITHI','A_2402_9'),('MHC_I_BIMAS','A_2402_9')]
peptideSet.predict(models)
```

Note that the length of the models has to match the peptide length!

Now, for each peptide the predictions are available and accessible.

```
for peptide in peptideSet.getMembers():
        print peptide.getPredictions()
```

To access the predictions for one method (e.g.)'MHC_I_SYFPEITHI' use:

```
for peptide in peptideSet.getMembers():
        print peptide.getPredictions()['MHC_I_SXFPEITHI']
```

We can also write the prediction results to a tab delimited file *'results.txt'*:

```
peptideSet.wirteTabDelimitedFile('results.txt')
```

Class I					Class II	
SYFPEITHY	BIMAS	Epidemix	SVMHC		HAMMER	Syfpeihti
A_0101_10	A_0101_9	A_01_9	A_01_9		DRB1_0101	DRB1_0101_9
A_0101_11	A_0201_9	A_0201_10	A_0201_9		DRB1_0102	DRB1_0301_9
A_0101_9	A_0205_9	A_0201_11	A_0201_10		DRB1_0301	DRB1_0401_9
A_0201_10	A_0301_9	A_0201_9	A_03_10		DRB1_0305	DRB1_0701_9
A_0201_9	A_1101_9	A_03_10	A_03_9		DRB1_0306	DRB1_1101_9
A_0301_10	A_2402_9	A_03_9	A_1101_9		DRB1_0307	DRB1_1501_9
A_0301_9	A_3101_9	A_1101_10	A_2402_9		DRB1_0308	
A_1101_10	A_3302_9	A_1101_9	A_24_9		DRB1_0309	
A_1101_9	A_6801_9	A_2402_9	A_25_10		DRB1_0311	
A_2402_10	B_0401_9	A_24_9	B_07_9		DRB1_0401	
A_2402_9	B_0702_9	A_25_10	B_08_9		DRB1_0402	
A_2601_10	B_0801_8	A_25_9	B_1501_9		DRB1_0404	
A_2601_9	B_0801_9	B_07_9	B_1501_10		DRB1_0405	
A_6801_10	B_1501_9	B_08_8	B_1801_8		DRB1_0408	
A_6801_9	B_2702_9	B_08_9	B_1801_9		DRB1_0410	
B_0702_10	B_2705_8	B_1801_8	B_2705_9		DRB1_0421	
B_0702_9	B_2705_9	B_1801_9	B_27_9		DRB1_0423	
B_0801_8	B_3501_8	B_2705_9	B_3501_9		DRB1_0426	
B_0801_9	B_3501_9	B_27_10	B_3701_9		DRB1_0701	
B_1402_8	B_3701_9	B_27_9	B_44_9		DRB1_0703	
B_1402_9	B_3801_9	B_3701_9	B_5101_9		DRB1_0801	
B_1501_10	B_3901_8	B_44_9			DRB1_0802	
B_1501_9	B_3901_9	B_5101_8			DRB1_0804	
B_1510_9	B_3902_9	B_5101_9			DRB1_0806	
B_1801_8	B_4001_9				DRB1_0813	
B_1801_9	B_4006_8				DRB1_0817	
B_2705_9	B_4006_9				DRB1_1101	
B_3701_8	B_40_9				DRB1_1102	
B_3701_9	B_4403_9				DRB1_1104	
B_3801_10	B_5101_8				DRB1_1106	
B_3801_9	B_5101_9				DRB1_1107	
B_3901_9	B_5102_8				DRB1_1114	
B_3902_9	B_5102_9				DRB1_1120	
B_4001_9	B_5103_9				DRB1_1121	
B_4101_9	B_5201_8				DRB1_1128	
B_4402_10	B_5201_9				DRB1_1301	
B_4402_9	B_5801_9				DRB1_1302	
B_4501_9	Cw_0301_9				DRB1_1304	
B_4701_9	Cw_0401_9				DRB1_1305	
B_4901_9	Cw_0602_9				DRB1_1307	
B_5001_9	Cw_0702_9				DRB1_1311	
B_5101_8					DRB1_1321	
B_5101_9					DRB1_1322	
					DRB1_1323	
					DRB1_1327	
					DRB1_1328	
					DRB1_1501	
					DRB1_1502	
					DRB1_1506	
					DRB1_5_0101	
					DRB1_5_0105	

Table 2.1: MHC binding prediction models in FRED 1.0.

2.1.3 Filtering for potential binders

In the previous sections we already learned how to get a peptideSet and make predictions and how to access the predictions.

Most of the prediction methods are regressions, i.e. the results are values that in some manner represent the binding affinity of the peptides. A regression can be turned into a classification by applying a threshold. The choice of an appropriate threshold is a critical task since the performance of the resulting classification depends on this threshold. The thresholds should be chosen carefully and depend of the respective task.

Different ways to define a threshold are available in FRED:

- *Halfmax:* For all matrix based prediction methods, the halfmax-threshold is half of the maximum score obtainable by the respective matrix. For machine learning methods, a halfmax value is not available. In that case, the halfmax value is replaced by the 2%-threshold (see below).
- *Percentages:* The thresholds to distinguish binders from non-binders are determined using a percentage value. The thresholds are computed on a large set of randomly chosen proteins (background distribution of scores). The thresholds mean that only the given percentage of the peptides in the background protein set would be classified as binders. The advantage of this threshold method is that thresholds become comparable between models. Based on our experience, a 2%-threshold is a reasonable choice.
- *user defined:* The third possibility to define a threshold is to explicitly define values for each model.

The result of the application of a threshold filtering is a set of candidates. We will give an example for each of the filtering methods.

Each of the filtering methods takes as arguments a peptideSet, a list of models, a conservation threshold and a promiscuity value. The conservation threshold allows filtering for conserved epitopes. The promiscuity value specifies by how many of the applied prediction models a peptide has to be defined as binder to pass the filtering step. The promiscuity threshold allows for consensus prediction (in the case of different models for the same allele) or for the prediction of promiscuous epitopes. promiscuity = 1 means that all peptides that are predicted to bind by at least one model are reported. The default values are conservation = 0.0 and promiscuity = 0.

In the following example we will use a conservation threshold of 0.0 (i.e. no filtering for conservation is applied) and a promiscuity of 3 (we only want peptides that are predicted to bind by all three models specified earlier).

Filtering using percentage values or explicit thresholds requires additional parameters: the percentage value or the thresholds, respectively.

```
# Filtering using halfmax thresholds:
c = Fred.Fred.FindCandidates_HalfmaxThresholds(pepset,models,0.0,3)

# Filtering using percentage values, here we use 2 percent (0.02):
percent = 0.02
c = Fred.Fred.FindCandidates_Percentage(pepset,models,percent,0.0,3)

# Filtering using explicit thresholds. We first have to specify the thresholds:
thresholds = {('MHC_I_SVMHC','A_2402_9'):0.258,('MHC_I_SYFPEITHI','A_2402_9'):15.5,('MHC_I_BIMAS','A_2402_9'):3.68}
c = Fred.Fred.FindCandidates_Percentage(pepset,models,thresholds,0.0,3)
```

We can now write the potential epitopes (peptides that fulfilled all filtering criteria) into a file:

```
c.writeTabDelimitedFile('output_filtering.txt')
```

Please notice that no halfmax or percentage values are available for methods not included in the original

version of FRED, i.e. methods added by the user. FRED provides a method to compute percentage values (see Section 2.2.2)

2.2 Additional tools in FRED

2.2.1 Evaluation tools

FRED provides a number of standard measures to compare different prediction methods and to evaluate the performance w.r.t. experimental values (Matthews correlation coefficient, accuracy, sensitivity, specificity and ares under the ROC curve, correlation, rank correlation). Different prediction methods can thus be compared with ease.

How these tool can be applied is demonstrated in the example *example4_Performance.py* that can be found in the FRED/examples.

2.2.2 Score distributions

FRED provides a class to compute score distributions. One application for this is the computation of percentage thresholds. This class is called *scoreDistribution*. Take care which peptides/proteins you use to compute the score distributions. For details on this class please refer to the documentation.

2.3 Examples

We included some examples into the FRED package where we demonstrate how typical problems in computational immunomics can be solved using FRED. These examples and all required input data can be found in FRED/examples. Here we shortly describe what these examples do.

- *exapmle0_GettingProteins.py*
  In this example we demonstrate the different ways to obtain protein sequences:

  1. Sequences from a list of strings
  2. Sequences from a fasta file
  3. Sequences from Swiss-Prot
  4. Sequences from RefSeq

- *example1_ConservedEpitopes.py*
  In this example we demonstrate how to predict conserved epitopes.

- *example2_ConsensusPrediction.py*
  In this example we demonstrate how to make consensus predictions.

- *example3_mHags.py*
  In this example we demonstrate how FRED can deal with polymorphic protein sequences. The polymorphic sequence is given in a special polymorphic fasta format. The polymorphic positions and the

observed amino acids are specified in the header. The sequence is given as reference sequence.

```
> sequence | [5:A,G;10:R,M,V]
XXXXAXXXXRXXXXXXXXXXXXXXXXXXXXX

codes for
XXXXAXXXXRXXXXXXXXXXXXXXXXXXXXX
XXXXAXXXXMXXXXXXXXXXXXXXXXXXXXX
XXXXAXXXXVXXXXXXXXXXXXXXXXXXXXX
XXXXGXXXXRXXXXXXXXXXXXXXXXXXXXX
XXXXGXXXXMXXXXXXXXXXXXXXXXXXXXX
XXXXGXXXXVXXXXXXXXXXXXXXXXXXXXX
```

FRED can treat the polymorphic sequence as non-polymorphic sequence. Then, the reference sequence (the one specified in the fasta file) is used as normal protein sequence, the polymorphisms specified in the fast header are ignored.
Alternatively FRED can consider all polymorphic peptides resulting from the polymorphic sequence. Both alternatives are shown in the example.

- *example4_Performance.py*
  In this example we show how FRED can be used to assess the performance of prediction methods.

  1. Assess the performance w.r.t. binary binding information (0 for non binding peptide, 1 for binding peptides).
     The binary binding information is read in from a file (tab delimited: peptide sequence -tab- binding [0|1]).
     Predictions can be made for the peptides using different prediction models. These predictions can then be compared to the original values. The following performance measures are available:

     - Matthews Correlation Coefficient (MCC)
     - Accuracy (ACC)
     - Sensitivity (SE)
     - Specificity (SP)
     - Area under the ROC curve (AUC)

     Except for AUC, these performance measures are threshold dependent! The predictions results have to be filtered using one of the threshold methods first. Additionally, the threshold that would yield the highest MCC on this dataset can be obtained, along with the corresponding ACC, SE, and SP values.

  2. Compute the correlation between predictions.
     FRED can be used to compute the correlation between prediction methods or between experimental values and prediction values. The reference prediction values can be read in from a file (see input_example4_scores.txt for details). The pairwise correlation (Pearson's Correlation Coefficient or Rank correlation) between these reference values and predictions can be computed.

- *example5_hcv.py*
  In this example we show how FRED can be used to solve a typical problem in vaccine design against highly variable viruses. We search for epitopes that are highly conserved in a set of HCV core protein sequences.
  This application of FRED is based on the paper by Toussaint et al. [10].

- *example6_AvailableModels.py*
  In this model we show how to read out the prediction models available in FRED.

- *example7_IntegratingNetMHCpan.txt*
  We use the portable version of NetMHCpan 2.0 (http://www.cbs.dtu.dk/services/NetMHCpan/) to demonstrate how an external method can be integrated into FRED.

Bibliography

[1] H. Rammensee, J. Bachmann, N. P. Emmerich, O. A. Bachor, and S. Stevanović. SYFPEITHI: database for MHC ligands and peptide motifs. *Immunogenetics*, 50(3-4):213–219, Nov 1999. 4

[2] Pierre Dönnes and Oliver Kohlbacher. SVMHC: a server for prediction of MHC-binding peptides. *Nucleic Acids Res*, 34(Web Server issue):W194–W197, Jul 2006. 4

[3] K. C. Parker, M. A. Bednarek, and J. E. Coligan. Scheme for ranking potential HLA-A2 binding peptides based on independent binding of individual peptide side-chains. *J Immunol*, 152(1):163–175, Jan 1994. 4, 6

[4] Morten Nielsen, Claus Lundegaard, Thomas Blicher, Kasper Lamberth, Mikkel Harndahl, Sune Justesen, Gustav Roder, Bjoern Peters, Alessandro Sette, Ole Lund, and Soren Buus. NetMHCpan, a method for quantitative predictions of peptide binding to any HLA-A and -B locus protein of known sequence. *PLoS ONE*, 2(8):e796, 2007. 4, 5

[5] S. Buus, S. L. Lauemller, P. Worning, C. Kesmir, T. Frimurer, S. Corbet, A. Fomsgaard, J. Hilden, A. Holm, and S. Brunak. Sensitive quantitative predictions of peptide-MHC binding by a 'Query by Committee' artificial neural network approach. *Tissue Antigens*, 62(5):378–384, Nov 2003. 4, 5

[6] T. Sturniolo, E. Bono, J. Ding, L. Raddrizzani, O. Tuereci, U. Sahin, M. Braxenthaler, F. Gallazzi, M. P. Protti, F. Sinigaglia, and J. Hammer. Generation of tissue-specific and promiscuous HLA ligand databases using DNA microarrays and virtual HLA class II matrices. *Nat Biotechnol*, 17(6):555–561, Jun 1999. 4

[7] Morten Nielsen, Claus Lundegaard, Thomas Blicher, Bjoern Peters, Alessandro Sette, Sune Justesen, S Buus, and Ole Lund. Quantitative predictions of peptide binding to any HLA-DR molecule of known sequence: NetMHCIIpan. *PLoS Comput Biol*, 4(7):e1000107, 2008. 4, 5

[8] Pierre Dönnes and Oliver Kohlbacher. Integrated modeling of the major events in the MHC class I antigen processing pathway. *Protein Sci*, 14(8):2132–2140, Jun 2005. 4

[9] Irini Doytchinova, Shelley Hemsley, and Darren R Flower. Transporter associated with antigen processing preselection of peptides binding to the MHC: a bioinformatic evaluation. *J Immunol*, 173(11):6813–6819, Dec 2004. 4

[10] Nora C Toussaint, Pierre Dönnes, and Oliver Kohlbacher. A mathematical framework for the selection of an optimal set of peptides for epitope-based vaccines. *PLoS Comput Biol*, 4(12):e1000246, Dec 2008. 6, 14

[11] Magdalena Feldhahn, Philipp Thiel, Mathias M Schuler, Nina Hillen, Stefan Stevanović, Hans-Georg Rammensee, and Oliver Kohlbacher. Epitoolkit–a web server for computational immunomics. *Nucleic Acids Res*, 36(Web Server issue):W519–W522, Jul 2008. 6

[12] Nora C Toussaint and Oliver Kohlbacher. OptiTope - A Web Server for the Selection of an Optimal Set of Peptides for Epitope-based Vaccines. *Nucleic Acids Res - accepted*, 2009. 6

[13] Hong Huang Lin, Surajit Ray, Songsak Tongchusak, Ellis L Reinherz, and Vladimir Brusic. Evaluation of MHC class I peptide binding prediction servers: applications for vaccine research. *BMC Immunol*, 9:8, 2008. 7

[14] Hong Huang Lin, Guang Lan Zhang, Songsak Tongchusak, Ellis L Reinherz, and Vladimir Brusic. Evaluation of MHC-II peptide binding prediction servers: applications for vaccine research. . *BMC Bioinformatics*, 9 Suppl 12:S22, 2008. 7

Chapter 3

API Documentation for FRED

3.1 Package Fred

FRED Framework for T cell Epitope Detection.

Version: 1.0

3.1.1 Modules

3.2 Module Fred.Fred

3.2.1 Functions

getAvailableMethods()

Returns all available prediction methods.

getAvailableModels(*method=''*)

Returns all available predictions models in FRED format. IF no method is specified, the models for all methods are returned.

Parameters

`method:` Method for which all available models are returned. If left empty, the models for all methods are returned.

3.2.2 Variables

Name	Description
__docformat__	**Value:** `'epytext en'`
path_prefix	**Value:** `'/nfs/wsi/bs/share/usr/feldhahn/FRED'`
available_models	**Value:** {`'MHC_II_HAMMER': ['DRB1_0101', 'DRB1_0102', 'DRB1_0301',...`

3.2.3 Class Sequence

Known Subclasses: Fred.Fred.Protein, Fred.Fred.Peptide

Class to handle Sequence objects

Methods

__init__(*self*)

Initializes empty sequence object. Default values: _sequence = '', _length = 0 , _name = ''

setSequence(*self*, *seq*)

Set sequence.

Parameters

`seq:` (String) A sequence.

getSequence(*self*)

Return sequence.

Return Value
sequence (String)

getLength(*self*)

Return length of the sequence.

Return Value
length (integer)

3.2.4 Class Protein

Fred.Fred.Sequence
Fred.Fred.Protein

Known Subclasses: Fred.FredPolymorphicExtension.PolymorphicProtein

Class to handle Protein objects

Methods

__init__(*self*)

Call Sequence constructor for empty object.

Overrides: Fred.Fred.Sequence.__init__

setName(*self*, *name*)

Set name.

Parameters
`name:` (String) Name of the Protein.

getName(*self*)

Return name.

Return Value
name (String) Name of the Protein.

cleanSequence(*self*)

Cleans sequence from invalid symbols.

All characters not belonging to the standard amino acid one letter code are changed to 'X'. When generating subsequences (peptides) from the sequence, all subsequences containing an 'X' are omitted!

Inherited from Fred.Fred.Sequence(Section 3.2.3)

getLength(), getSequence(), setSequence()

3.2.5 Class Peptide

Fred.Fred.Sequence
└
Fred.Fred.Peptide

Class to handle Peptide objects.

Methods

__init__(*self*, *pepSet*=`None`)

Initializes empty Peptide object.

Overrides: Fred.Fred.Sequence.__init__

addOrigin(*self*, *protein*, *startpos*)

Adds an origin (source protein and startposition) to the peptide object. Increases number of occurrences of the peptide. A peptide can occur in more than one protein in the protein set. Every peptide in the peptide set has a unique sequence. Origins keeps track of all occurrences of the peptide.

Parameters

`protein:` (Protein) The ID of the source protein.

`startpos:` (int) The starting position of the peptide in the respective protein.

setOrigins(*self*, *origins*)

Sets the origins of the peptide from a list. A peptide can occur in more than one protein in the protein set. Every peptide in the peptide set has a unique sequence. Origins keeps track of all occurrence of the peptide.

Parameters

`origins:` (list) A list of origins [(Protein ID, startpos),...]

getOrigin(*self*)

Return origins. A peptide can occur in more than one protein in the protein set. Every peptide in the peptide set has a unique sequence. Origins keeps track of all occurrence of the peptide.

Return Value
origins (dictionary)

getNumber(*self*)

Return the number of occurrences of the peptide in the ProteinSet. A peptide can occur in more than one protein in the protein set. Every peptide in the peptide set has a unique sequence.

Return Value
(int) The number of occurrences of the peptide in the ProteinSet.

getInSeqs(*self*)

Return the number of sequences the peptides is contained in. A peptide can occur in more than one protein in the protein set. Every peptide in the peptide set has a unique sequence.

Return Value
(int) The number of sequences the peptide is contained in.

getPeptideSet(*self*)

Return the PeptideSet object the peptide belongs to.

Return Value
(PeptideSet) The PeptideSet the peptide belongs to.

setPrediction(*self*, *method*, *allele*, *score*, *rank*)

Set MHC prediction result.

Parameters
`method:` (String) The name of the prediction method.
`allele:` (String) The name of the allele.
`score:` (float) The prediction score.
`rank:` (int) The rank of the peptides w.r.t. the PeptideSet.

getPredictions(*self*)

Return the predictions associated with the peptide.

Return Value

predictions (dictionary) The predictions for the peptide. '{}' if no predictions were made.

setTap(*self*, *method*, *score*, *rank*, *c*)

Sets TAP transport prediction result.

Parameters

`method:` (String) The name of the prediction method.

`score:` (float) The prediction score.

`rank:` (int) The rank of the peptides w.r.t. the PeptideSet.

`c:` Tapclass - one of [high, intermediate, low, non] for SVMTAP, or 'no class defined'.

getTap(*self*)

Gets score for TAP transport.

Return Value

Tap predictions (dictionary).

setProteasome(*self*, *method*, *score*, *percent*)

Sets score for generation of the C-terminus of the peptide by proteasomal cleavage. The percentage values (percentage of peptides with score > score of this peptide) is based on a background distribution. This distribution was computed on a large set of randomly chosen natural peptides.

Parameters

`method:` (String) The name of the prediction method.

`score:` (float) The prediction score.

`percent:` (float) The percentage of peptides with score > score of this peptide.

getProteasome(*self*)

Gets score for generation of the C-terminus of the peptide by proteasomal cleavage. The percantge values (percentage of peptides with score > score of this peptide) is based on a background distribution. This distribution was computed on a large set of randomly chosen natural peptides.

Return Value

(dictionary) Proteasomal cleavage prediction. (Key = method name, Value = (score, percentage of peptides with score > score of this peptide))

setPeptideSet(*self*, *peptideSet*)

Associates the Peptide with an PeptideSet.

setCTerms(*self*, *l*)

Computes all the c-terminal extensions of the peptides. More than one c-terminal extension is possible if the peptide occurs in more than one sequence.

Parameters

`l:` (int) Length of c-terminal extension.

getCTerms(*self*)

Gets c-terminal extensions. return format: [(source protein ID, startposition, extension), ...]

Return Value

(list) C-terminal extensions of the peptide.

getConservation(*self*)

Gets conservation of peptide.

If protein_by_protein = 1 for the generation of PeptideSets, the conservation is 1/number of protein sequences.

Return Value

(float) Percentage of input sequences the peptide occurs in.

Inherited from Fred.Fred.Sequence(Section 3.2.3)

getLength(), getSequence(), setSequence()

3.2.6 Class SequenceSet

Known Subclasses: Fred.Fred.PeptideSet, Fred.Fred.ProteinSet

Class SequenceSet. Set of sequences...

Methods

__init__(*self*)

Initializes SequenceSet.

setSequences(*self*, *sequences*)

Sets sequences. Deletes all sequences already contained in the set!

Parameters

`sequences`: (list) List of sequence ojects.

addSequence(*self*, *sequence*)

Adds a sequence object to the set.

Parameters

`sequence`: (Sequence) Sequence to be added.

setFromStrings(*self*, *sequences*)

Sets sequences from strings. Deletes all sequences already contained in the set!

Parameters

`sequences`: (list) List of strings.

getMembers(*self*)

Gets List with sequence objects.

Return Value

(list) List of Sequence objects

getSequences(*self*)

Gets list with sequences from sequence set.

Return Value

(list) List of sequences (strings) in SequenceSet.

getLength(*self*)

Gets number of sequence objects in the SequenceSet.

Return Value
(integer) Number of Sequence objects in SequenceSet.

3.2.7 Class ProteinSet

Fred.Fred.SequenceSet
└ **Fred.Fred.ProteinSet**

Known Subclasses: Fred.FredPolymorphicExtension.PolymorphicProteinSet

Class ProteinSet

Methods

__init__(*self*)

Initialize empty ProteinSet object.

Overrides: Fred.Fred.SequenceSet.__init__

accessRefSeq(*self*, *accessions*)

Retrieves Sequences from RefSeq database. Valid accessions are: accession for RefSeq, e.g. 'NP_055693'

Parameters
`accessions:` a list of valid RefSeq accessions

accessSwissProt(*self*, *accessions*)

Retrieves Sequences from SwissProt database. Valid accessions are: Primary accession for SwissProt, e.g. 'O93182'

Parameters
`accessions:` a list of valid SwissProt accessions

readFasta(*self*, *fastafile*)

Reads sequences from fastafile.

Parameters
`fastafile:` (String) filename and path to Fasta-file

sequencesFromStrings(*self*, *sequences*)

Sets sequences from List of Strings.

Parameters

`sequences:` List of Strings.

getPeptides(*self*, *l*=`9`, *conservation*=`0.0`, *prot_by_prot*=`0`)

Generates a PeptideSet from the proteins in the ProteinSet.

Options : Peptide length: desired length of the peptides. Only one peptide length can be handled at once! prot_by_prot : Generate a PeptideSet for each Protein (prot_by_prot = 1) or one PeptideSet for all Proteins (prot_by_prot =0). conservation threshold: Only peptides that occur in more than the given percentage of the Proteins in the ProteinSet are added.

Parameters

`l:` (integer) Peptide Length. Only one peptide length can be handled at once!

`conservation:` (float) Conservation Threshold. Only peptides that occur in more than the given percentage of the Proteins in the ProteinSet are added.

`prot_by_prot:` Generate a PeptideSet for each Protein (prot_by_prot = 1) or one PeptideSet for all Proteins (prot_by_prot = 0).

writeFasta(*self*, *filename*)

Writes the sequences into a fasta-file.

Required for multiple alignment if the protein sequences were given as strings. Fasta file is needed as input for alignment algorithm.

Parameters

`filename:` (string) Name of the Fasta file, including path.

setupClustalw(*self*)

computeAlignment(*self*)

Computes a clustalW alignment from the sequences using BioPython.

Requires that the sequences are given in a Fasta file. Sets _alignment attribute to alignment_summary object (see BioPython Tutorial for details) Computes variability of positions and a consensus sequence.

clustalw has to be instaled on your system! To check your clustalw installation type 'clustalw' in your command line. If the program does not start, there is something wrong...

getAlignment(*self*)

Returns sequence alignment.

Return Value
(string) Alignment of the Sequences.

getAlignmentObject(*self*)

Returns sequence alignment object.

Format: alignment summary object (see BioPython Tutorial). Returns an empty string if no alignment was computed.

Return Value
BioPython alignment summary object

getConsensus(*self*)

Returns consensus sequence of the alignment

Return Value
(string) The consensus sequence

getVariability(*self*)

Returns variability of single positions.

For each position in the precomputed alignment, the variability (entropy) is returned.

Return Value
(list) A List of the variability of the positions in the alignment.

Inherited from Fred.Fred.SequenceSet(Section 3.2.6)

addSequence(), getLength(), getMembers(), getSequences(), setFromStrings(), setSequences()

3.2.8 Class BenchmarkAssess

Class for comparing models with binary binding data.

This class implements methods to compare predictions to binary binding information.

The following measures are implemented:

Matthews correlation Coefficient (MCC), Sensitivity (SE), Specificity (SP), Accuracy (ACC)

ROC curve (the values to plot ROC curves)

Area unter the ROC curve (AUC)

Methods

AUC(*self*, *dat*)

Calculates the Area under the ROC curve from 1-SP/SE pairs using the trapezoid method.

Parameters

`dat`: (list) A list of (1-SP)/SE pairs.

Return Value

(float) The Area under the ROC curve (AUC).

MCC(*self*, *dat*)

Calculates the number of True Positives (TP), True Negatives (TN), False Positives (FP), False Negatives (FN) and ACC,SE,SP,MCC from a list of real class/predicted class pairs.

Parameters

`dat`: (list) A list containing binary binding information [(predicted class, true class),...].

Return Value

(list) A list containing the following values: [MCC,SE,SP,ACC,TP,FP,TN,FP].

ROC(*self*, *dat*)

Calculates (1-SP)/SE pairs for ROC curves from a list of real class/predicted score pairs.

Parameters

`dat:` (list) A list containing value pairs: dat = [(true class, predicted score),...].

Return Value

(list) A list of (1-SP)/SE pairs.

__init__(*self*)

calcAUC(*self*, *pepSet*)

Computes the Area under the ROC curve (AUC) using experimental binary binding information (_predictions['EXP'])

Computes pairwise AUC with EXP for every prediction model.

Parameters

`pepSet:` (PeptideSet) A PeptideSet.

Return Value

AUC - dictionary containing the AUC for each model

calcPerformance(*self*, *candidates*)

Computes the performance of the prediction models.

For each prediction model the performance (MCC, SE, SP, ACC) is compared w.r.t. the given experimental binding data.

Parameters

`candidates:` (list) A list of candidates for which the performance should be computed.

Return Value

(list) A list containing four dictionaries (for MCC, SE, SP, ACC).
Key: model, value = the respective performance measure.

calcThresholds(*self*, *pepSet*)

Determines the threshold that yields the best MCC on this dataset for each model.

The threshold is varied over the entire range of predicted values (one model, one PeptideSet). Only Values that actually occur in the dataset are candidates for a threshold.

For each of the thresholds the MCC is calculated. The threshold yielding the best MCC is returned.

Parameters

`pepSet:` (PeptideSet) A PeptideSet.

Return Value

(dictionary) A dictionary containing the best-MCC-threshold for each model. Key: model, value: (best mcc cutoff, alist of mcc,se,...) for the best mcc threshold, (best-mcc-threshold ,[MCC,SE,SP,ACC,TP,FP,TN,FP]).

writePerformanceSummary(*self*, *pepSet*, *candidates*, *filename*)

Calculates the performance. Writes Performance Summary into a file.

Parameters

`pepSet:` a PeptideSet

`candidates:` a list of candidates

`filename:` filename of output file

3.2.9 Class BenchmarkCompare

Class for comparing prediction results.

This class contains methods to compare predictions to other predictions or experimental data.

The comparison is made on the prediction scores, not on binary binding information.

This class implements the following measures:

Pearsons Correlation Coefficient

Spearmans Rank Correlation

Methods

__init__(*self*, *peptideSet*)

calcCorrelation(*self*, *dat*)

Computes the correlation between two predictions.

Parameters

`dat`: (list) A list of tuples of corresponding predictions. Only accessed internally, so you do not have to care about this..

computePearsonCorr(*self*)

Computes pairwise Pearsons correlation coefficient.

Return Value

corrTable (dictionary) A dictionary containing the pairwise correlation coefficients. key: (model1, model2), value: correlation.

computeSpearmansRank(*self*)

Computes the pairwise Spearmans rank correlation.

Return Value

rankCorr (dictionary) A dictionary containing the pairwise rank correlations. key: (model1, model2), value: rank correlation.

writeCorrelationTable(*self*, *corr*, *filename*, *dist*=`0`)

Writes pairwise correlations into a file.

Parameters

`corr`:	A dictionary with pairwise distances/correlations)
`filename`:	Name and path to outputfile.
`dist`:	Indicates if the correlation should be converted into distances (1-correlation). Default: dist = 0 (no conversion)

3.2.10 Class Candidate

Class for candidate objects. Candidate object has attributes sequence, alleles with 1/0 binding information, TAP information (low, medium, high), Proteasomal cleavage information (low, medium, high), conservation; is linked with corresponding peptide for all other information.

Methods

__init__(*self*, *peptide*)

Initializes Candidate Object.

getAllModels(*self*)

Return list of all prediction models.

self._all_models contains all prediction models (MHC, TAP, Protesomal cleavage)

Return Value
(list) A list of prediction models.

getBinding(*self*)

Return binding information.

For each MHC model a binary binding information is returned. 1 = binder, 0 = non binder.

Return Value
(dictionary) Binding information for all models. Keys of the dictionary: model, value: 1/0

getMHCModels(*self*)

Return MHC prediction models.

Return Value
(list) A list of MHC prediction models.

getMethod(*self*)

Return method for threshold generation.

Three ways of getting thresholds for the prediction methods are available: 'Explicit threshold': An explicit threshold for each MHC prediction threshold has to be specified by the user. 'percentage': The thresholds are generated for a percentage value. 'halfmax': Halfmax thresholds are used.

Return Value
(string) One of'Explicit threshold', 'percentage', or 'halfmax'.

getMinConservation(*self*)

Get the conservation threshold for candidates. Only candidates that are conserved in more than a given percentage of the input sequences are reported as candidates.

Return Value
(float) The percentage a of input sequences the peptide has to be conserved in.

getMinPromiscuity(*self*)

Return the promiscuitiy threshold for candidates. The minimum number of MHC models for which a peptide has to be classified as binder to be reported as candidate.

Return Value
(integer)

getPeptide(*self*)

Return the peptide object the candidate is associated with.

Each candidate is associated with a Peptide. The Peptide object holds all additional information to the caniddate. (predictions, origin,...)

Return Value
(Peptide) The Peptide associated with the candidate.

getPromiscuity(*self*)

Gets promiscuitiy.

Return Value
(integer) Number of MHC models for which the peptide is classified as binder.

getSequence(*self*)

Return the sequence of the candidate.

Return Value
(string) The sequence of the candidate.

getThresholds(*self*)

Return the MHC prediction thresholds.

Return Value

(dictionary) The thresholds that were used to separate binder form non binders. key: model, value: float.

setAllModels(*self*, *models*)

Set list of all prediction models.

self._all_models contains all prediction models (MHC, TAP, Protesomal cleavage)

Parameters

`models:` (list) A list of prediction models.

setBinding(*self*, *binding*)

Set binding inforamtion.

For each used MHC model binary binding information is stored. 1 = binder, 0 = non binder.

Parameters

`binding:` (dictionary) Binding information for all models. Keys of the dictionary: model, value: 1|0

setMHCModels(*self*, *models*)

Sets MHC prediction models.

Parameters

`models:` (list) A list of MHC prediction models.

setMethod(*self*, *method*)

Set Method for threshold generation.

Different ways of getting thresholds for the prediction methods are available: 'Explicit threshold': An explicit threshold for each MHC prediction threshold has to be specified by the user. 'percentage': The thresholds are generated from a percentage value. 'halfmax': Halfmax thresholds are used.

Parameters

`method:` (string) Must be one of 'Explicit threshold', 'percentage', or 'halfmax'.

setPromiscuity(*self*, *promiscuity*)

Set promiscuity.

Parameters

`promiscuity`: (integer) Number of MHC models for which the peptide is classified as binder.

3.2.11 Class FindCandidates

Known Subclasses: Fred.Fred.FindCandidates_ExplicitThresholds, Fred.Fred.FindCandidates_Halfm
Fred.Fred.FindCandidates_Percentage

A base class for selecting Peptides from a PeptideSet.

Methods

__init__(*self*)

findCandidates(*self*, *peptideSet*)

For all peptides in the PeptideSet: check if the peptide satisfies all criteria.

Parameters

`peptideSet`: (PeptideSet)

getAllModels(*self*)

Return the models that were considered for finding candidates

Return Value

(list) A list of models.

getCandidates(*self*)

Return a list of Candidate objects.

Return Value

(list) A list of Candidate objects (one per peptide that satisfies the criteria).

getMHCModels(*self*)

Return all MHC models.

Return Value

(list) A list of MHC models.

getThresholds(*self*)

Return the thresholds used for distinction between binders and nonbinders.

Return Value
(dictionary) A dictionary containing the threshols.

setMHCModels(*self*)

Selects MHC prediction models from all models.

Only MHC models are considered for promiscuity filtering.

writeTabDelimitedFile(*self*, *filename*)

Writes prediction values into tab delimited file.

Parameters
`filename:` filename (and path) of the output file.

3.2.12 Class FindCandidates_ExplicitThresholds

Fred.Fred.FindCandidates
└ **Fred.Fred.FindCandidates_ExplicitThresholds**

Selects Peptides form PeptideSet using the specified criteria.

Methods

__init__(*self*, *peptideSet*, *models*, *thresholds*, *conservation*=0.0, *promiscuity*=0)

Peptides are filtered using explicitly given thresholds.

The user has to specify a threshold for each MHC prediction model.

Parameters

`peptideSet:` (PeptideSet) A PeptideSet.

`models:` (list) A list of prediction models.

`conservation:` (float) The conservation thresholds. Default is 0.0 (no conservation).

`thresholds:` (dictionary) A dictionary that contains an explicit threshold for each MHC prediction model. Key: model,value: threshold (float).

`promiscuity:` (integer) A peptide has to be classified as binder for the given number of MHC prediction models. Default is 0 (all peptides are reported).

Overrides: Fred.Fred.FindCandidates.__init__

isCandidate(*self*, *peptide*)

Checks if a peptide satisfies all specified criteria.

Inherited from Fred.Fred.FindCandidates(Section 3.2.11)

findCandidates(), getAllModels(), getCandidates(), getMHCModels(), getThresholds(), setMHCModels(), writeTabDelimitedFile()

3.2.13 Class FindCandidates_HalfmaxThresholds

Fred.Fred.FindCandidates
└ **Fred.Fred.FindCandidates_HalfmaxThresholds**

Selects Peptides form PeptideSet using the specified criteria.

Methods

__init__(*self*, *peptideSet*, *models*, *conservation*=`0.0`, *promiscuity*=`0`)

Peptides are filtered using a halfmax threshold.

For matrix based methods, the halfmax values is half of the sum over the maximal entries of all columns (half of the maximal score obtainable by that matrix). For SVM based methods, a halfmax threshold is not defined. The 2 percent threshold is used instead.

Parameters

`peptideSet:`	(PeptideSet) A PeptideSet.
`models:`	(list) A list of prediction models.
`conservation:`	(float) The conservation thresholds. Default is 0.0 (no conservation).
`promiscuity:`	(integer) A peptide has to be classified as binder for the given number of MHC prediction models. Default is 0 (all peptides are reported).

Overrides: Fred.Fred.FindCandidates.__init__

isCandidate(*self*, *peptide*)

Checks if a peptide satisfies all specified criteria.

setThresholds(*self*)

Sets classification thresholds form percentage value.

Inherited from Fred.Fred.FindCandidates(Section 3.2.11)

findCandidates(), getAllModels(), getCandidates(), getMHCModels(), getThresholds(), setMHCModels(), writeTabDelimitedFile()

3.2.14 Class FindCandidates_Percentage

Fred.Fred.FindCandidates
└ **Fred.Fred.FindCandidates_Percentage**

Selects Peptides form PeptideSet using the specified criteria.

Methods

__init__(*self*, *peptideSet*, *models*, *percentage*=`0.02`, *conservation*=`0.0`, *promiscuity*=`0`)

Peptides are filtered using a percentage threshold.

The thresholds to distinguish binders from non-binders are determined using a percentage values. The thresholds are computed on a large set of randomly chosen proteins.

The thresholds mean that only the given percentage of the peptides in the background protein set would be classified as binders.

Parameters

`peptideSet:`	(PeptideSet) A PeptideSet.
`models:`	(list) A list of prediction models.
`conservation:`	(float) The conservation thresholds. Default is 0.0 (no conservation).
`percentage:`	(float) The percentage value to determine the thresholds. Default: 0.02 (2 percent)
`promiscuity:`	(integer) A peptide has to be classified as binder for the given number of MHC prediction models. Default is 0 (all peptides are reported).

Overrides: Fred.Fred.FindCandidates.__init__

isCandidate(*self*, *peptide*)

Checks if a peptide satisfies all specified criteria.

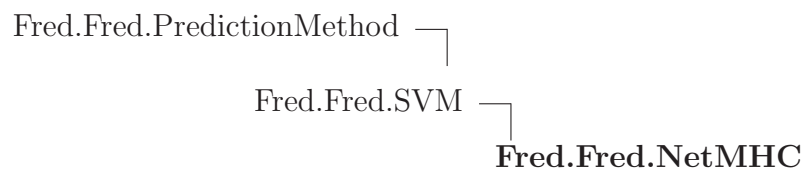
setThresholds(*self*)

Sets classification thresholds form percentage value.

Inherited from Fred.Fred.FindCandidates(Section 3.2.11)

findCandidates(), getAllModels(), getCandidates(), getMHCModels(), getThresholds(), setMHCModels(), writeTabDelimitedFile()

3.2.15 Class NetMHC



Class for NetMHC prediction.

Buus S, Lauemoller SL, Worning P, Kesmir C, Frimurer T, Corbet S, Fomsgaard A, Hilden J,Holm A, Brunak S. Sensitive quantitative predictions of peptide-MHC binding by a 'Query by Committee' artificial neural network approach. Tissue Antigens., 62:378-84, 2003.

webserver: http://www.cbs.dtu.dk/services/NetMHC/

FRED uses the log score values (1-log50k(aff)) returned by the netMHC prediction.

Method name = 'MHC_I_NETMHC'

Methods

__init__(*self*, *peptideSet*)

Initializes SVM Prediction object.

Parameters

`peptideSet:` A PeptideSet

Overrides: Fred.Fred.PredictionMethod.__init__ extit(inherited documentation)

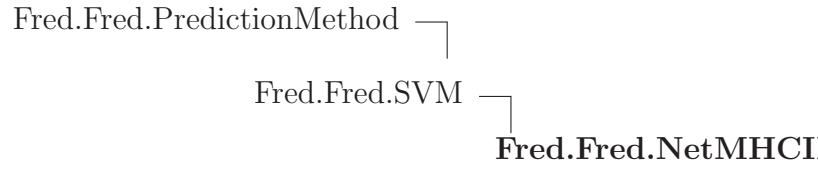
predict(*self*)

Makes netMHC predictions for all peptides and all alleles.

Inherited from Fred.Fred.SVM(Section 3.2.28)

encoding_sparse(), execute(), setAlleles()

3.2.16 Class NetMHCIIPan



Class for NetMHCIIpan prediction

Nielsen M, Lundegaard C, Blicher T, Peters B, Sette A, Justesen S, Buus S, and Lund O. PLoS Comput Biol. 2008 Jul 4;4(7). Quantitative predictions of peptide binding to any HLA-DR molecule of known sequence: NetMHCIIpan.

webserver: http://www.cbs.dtu.dk/services/NetMHCIIpan/

FRED uses the 1-log50k(aff) values returned from the netMHCIIpan prediction.

Method name = 'MHC_II_NETMHCIIPAN'

Methods

__init__(*self*, *peptideSet*)

Initializes SVM Prediction object.

Parameters

`peptideSet:` A PeptideSet

Overrides: Fred.Fred.PredictionMethod.__init__ extit(inherited documentation)

predict(*self*)

Makes netMHCpan predictions for all peptides and all alleles.

setAlleles(*self*)

Overwrites the general setAlleles method.

Overrides: Fred.Fred.SVM.setAlleles

Inherited from Fred.Fred.SVM(Section 3.2.28)

encoding_sparse(), execute()

3.2.17 Class NetMHCPan

Fred.Fred.PredictionMethod
└ Fred.Fred.SVM
└ **Fred.Fred.NetMHCPan**

Class for NetMHCpan prediction method.

NetMHCpan - MHC class I binding prediction beyond humans Ilka Hoof, Bjoern Peters,

John Sidney, Lasse Eggers Pedersen, Ole Lund, Soren Buus, and Morten Nielsen

webserver: http://www.cbs.dtu.dk/services/NetMHCpan/

FRED uses the 1-log50k(aff) values returned from the netMHCpan prediction.

Method name = 'MHC_I_NETMHCPAN'

Methods

__init__(*self*, *peptideSet*)

Initializes SVM Prediction object.

Parameters

`peptideSet:` A PeptideSet

Overrides: Fred.Fred.PredictionMethod.__init__ extit(inherited documentation)

predict(*self*)

Makes netMHCpan predictions for all peptides and all alleles.

Inherited from Fred.Fred.SVM(Section 3.2.28)

encoding_sparse(), execute(), setAlleles()

3.2.18 Class PSSM

Fred.Fred.PredictionMethod
└ **Fred.Fred.PSSM**

Known Subclasses: Fred.Fred.PSSM_Bimas, Fred.Fred.PSSM_Epidemix, Fred.Fred.PSSM_Hammer, Fred.Fred.PSSM_Proteasome, Fred.Fred.PSSM_Syfpeithi, Fred.Fred.PSSM_Syfpeithi_II, Fred.Fred.PSS

Baseclass for PSSM predictions.

Methods

__init__(*self*, *peptideSet*)

Initializes a PSSM prediction method object.

Parameters

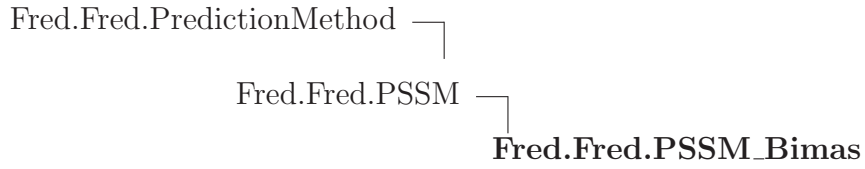
`peptideSet:` A PeptideSet

Overrides: Fred.Fred.PredictionMethod.__init__

predict(*self*)

Make predictions for the specified models.

3.2.19 Class PSSM_Bimas



MHC class I Bimas prediction method.

Matrix based prediction for MHC class I binding peptides. http://www-bimas.cit.nih.gov/molbio/hla_bi
Parker, K. C.; Bednarek, M. A. & Coligan, J. E. Scheme for ranking potential HLA-A2 binding peptides based on independent binding of individual peptide side-chains. J Immunol, 1994, 152, 163-175.

The original matrices are log-transformed to obtain an additive scoring scheme. Therefore the resulting scores are the logs of the scores obtained by the original matrices. Method name = 'MHC_I_BIMAS'

Methods

__init__(*self*, *peptideSet*)

Initializes a PSSM prediction method object.

Parameters

`peptideSet`: A PeptideSet

Overrides: Fred.Fred.PredictionMethod.__init__ extit(inherited documentation)

predict(*self*)

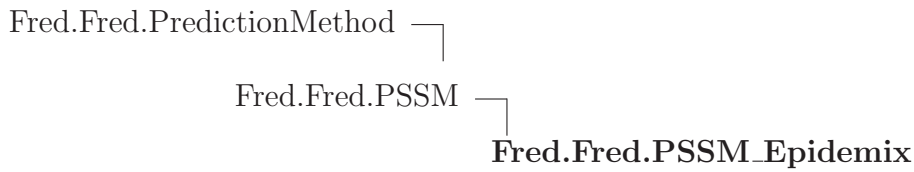
Overwrites the predict method from class PSSM to handle the constant factor in log-transformed Bimas matrices.

Overrides: Fred.Fred.PSSM.predict

setAlleles(*self*)

Extracts Bimas alleles from PeptideSet models.

3.2.20 Class PSSM_Epidemix



MHC class I Epidemix prediction.

MHC class I prediction matrices trainied on SYFPEITHI binders (positive training sets from SVMHC training). Henikoff&Henikoff pseudocount correction with swissprot background distribtions. Trained by M. Feldhahn.

FRED - A Framewoek for T cell Epitope Detection Diploma Thesis, M.Feldhahn, 2006, University of Tuebingen

Method name = 'MHC_I_EPIDEMIX'

Methods

__init__(*self*, *peptideSet*)

Initializes a PSSM prediction method object.

Parameters

`peptideSet:` A PeptideSet

Overrides: Fred.Fred.PredictionMethod.__init__ extit(inherited documentation)

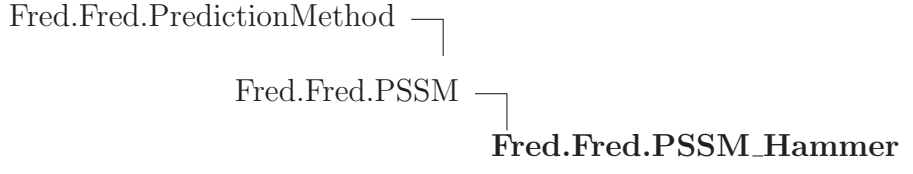
setAlleles(*self*)

Extracts Epidemix alleles from PeptideSet models.

Inherited from Fred.Fred.PSSM(Section 3.2.18)

predict()

3.2.21 Class PSSM_Hammer



Prediction of MHC class II binding peptides based on virtual matrices.

Sturniolo, T., Bono, E., Ding, J., Raddrizzani, L., Tuereci, O., Sahin, U., Braxenthaler, M., Gallazzi, F., Protti, M. P., Sinigaglia, F. and Hammer, J. Generation of tissue-specific and promiscuous HLA ligand databases using DNA microarrays and virtual HLA class II matrices. Nat Biotechnol, 1999, 17, 555-561

Method name = 'MHC_II_HAMMER'

Methods

__init__(*self*, *peptideSet*)

Initializes a PSSM prediction method object.

Parameters

`peptideSet:` A PeptideSet

Overrides: Fred.Fred.PredictionMethod.__init__ extit(inherited documentation)

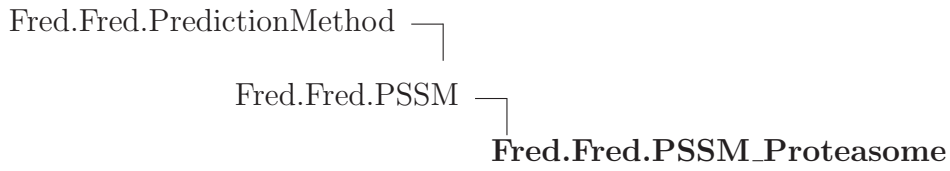
setAlleles(*self*)

Extracts Hammer alleles form PeptideSet models.

Inherited from Fred.Fred.PSSM(Section 3.2.18)

predict()

3.2.22 Class PSSM_Proteasome



Proteasomal cleavage prediction using PCM method.

Doennes, P. and Kohlbacher, O. Integrated modeling of the major events in the MHC class I antigen processing pathway. Protein Sci, 2005.

Takes into account the c-terminal extensions of the peptides. C-terminal extension must be available for all peptides!

Method name = 'PROTEASOME_PCM'

Methods

__init__(*self*, *peptideSet*)

Initializes a PSSM prediction method object.

Parameters

`peptideSet:` A PeptideSet

Overrides: Fred.Fred.PredictionMethod.__init__ extit(inherited documentation)

predict(*self*)

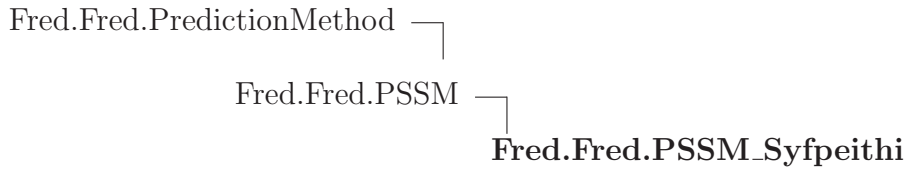
Overwrites the predict method from PSSM to handle the pcm matrix.

Predicts score for cleaving after position 4 in a peptide of length 6: input at last 4 position of a peptide + 2 positions of c-terminal extension.

Computes cleavage score for each occurrence of the peptide.

Overrides: Fred.Fred.PSSM.predict

3.2.23 Class PSSM_Syfpeithi



MHC class I Syfpeithi prediction method.

Matrix based prediction for MHC class I binding peptides. http://www.syfpeithi.de. Rammensee, H., Bachmann, J., Emmerich, N. P., Bachor, O. A. and Stevanovic, S. SYFPEITHI: database for MHC ligands and peptide motifs. Immunogenetics, 1999, 50, 213-219.

Method name = 'MHC_I_SYFPEITHI'

Methods

__init__(*self*, *peptideSet*)

Initializes a PSSM prediction method object.

Parameters

`peptideSet:` A PeptideSet

Overrides: Fred.Fred.PredictionMethod.__init__ extit(inherited documentation)

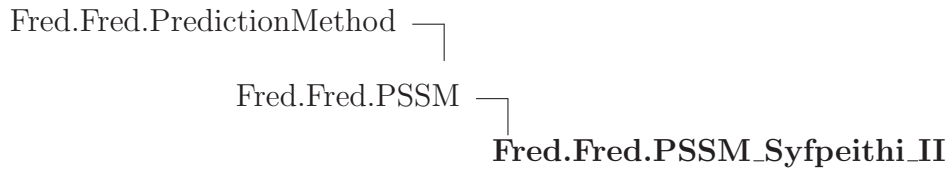
setAlleles(*self*)

Extracts Syfpeithi class I alleles from PeptideSet models.

Inherited from Fred.Fred.PSSM(Section 3.2.18)

predict()

3.2.24 Class PSSM_Syfpeithi_II



MHC class II Syfpeithi prediction method.

Matrix based prediction for MHC class I binding peptides. http://www.syfpeithi.de. Rammensee, H., Bachmann, J., Emmerich, N. P., Bachor, O. A. and Stevanovic, S. SYFPEITHI: database for MHC ligands and peptidemotifs. Immunogenetics, 1999, 50, 213-219.

Method name = 'MHC_II_SYFPEITHI'

Methods

__init__(*self*, *peptideSet*)

Initializes a PSSM prediction method object.

Parameters

`peptideSet:` A PeptideSet

Overrides: Fred.Fred.PredictionMethod.__init__ extit(inherited documentation)

setAlleles(*self*)

Extracts Syfpeithi class II alleles form PeptideSet models.

Inherited from Fred.Fred.PSSM(Section 3.2.18)

predict()

3.2.25 Class PSSM_TAP_Additive

Fred.Fred.PredictionMethod
└ Fred.Fred.PSSM
└ **Fred.Fred.PSSM_TAP_Additive**

Class for TAP prediction, additive method from Doytchinova et al. (Jenner institute).

Doytchinova, I., Hemsley, S. and Flower, D. R. Transporter associated with antigen processing preselection of peptides binding to the MHC: a bioinformatic evaluation. J Immunol, 2004, 173, 6813-6819.

Method name = 'TAP_Additive'

Methods

__init__(*self*, *peptideSet*)

Initializes a PSSM prediction method object.

Parameters

`peptideSet:` A PeptideSet

Overrides: Fred.Fred.PredictionMethod.__init__ extit(inherited documentation)

predict(*self*)

Makes TAP prediction using additive method.

Overrides: Fred.Fred.PSSM.predict

3.2.26 Class PeptideSet

Fred.Fred.SequenceSet
└ **Fred.Fred.PeptideSet**

Known Subclasses: Fred.FredPolymorphicExtension.PolymorphicPeptideSet

PeptideSet object. Set of Peptide objects.

Methods

__init__(*self*, *parent*=`None`)

Initialize PeptideSet.

Each PeptideSet is associated with a ProteinSet that contains the source Proteins of the Peptides. If no parent ProteinSet is given, _parent is set to None

_tmpdir: Directory for temorary data. default: ./tmp/ Please do not put important data in here!

_svmdir: Directory for svm executables

_svmModeldir: Directory for SVM model files.

Use the methods setTmpDir(dir), setSvmDir(dir), or setSvmModelDir(dir) to change the direcotires.

Parameters

`parent`: (ProteinSet) ProteinSet object the peptides are extracted from.

Overrides: Fred.Fred.SequenceSet.__init__

addModels(*self*, *models*)

Adds prediction models.

A model has to be ('Method_Name','allele') for MHC models or ('Method_Name','') Method_Name has to match one of the prediction methods. Unknown models are ignored during prediction.

Models already contianed in self._models are not overwritten.

Parameters

`models`: (list) A list of prediction models.

cleanTmpDir(*self*)

Method deletes temporary files in self._tmpdir when predictions are finished.

This function is more or less obsolete since the temporary files are associated with the porces id and removed directly after usage.

getModels(*self*)

Return prediction models.

For all these models predictions are made. Unknown models are ignored during prediction. Prediction models ('Method_Name','allele') for MHC models or ('Method_Name','') for TAP and proteasomal cleavage prediction.

Return Value
(list) A list of prediction models.

getParent(*self*)

Return the ProteinSet object the PeptideSet is derived from. If no parent ProteinSet is specified, the return value is None.

Return Value
(ProteinSet) ProteinSet object the PeptideSet is derived from.

getPeplen(*self*)

Return the peptide length.

Return Value
(integer) the length of the peptides.

predict(*self*, *models*)

Makes predictions for the specified prediction models.

Calls prediction methods and makes predictions for the specified models. Unknown models are ignored without warning! Check the spelling of the models!! The results are set to the respective attributes in each Peptide.

readBinaryBinding(*self*, *file*)

Reads in binary binding data for a set of peptides from a file.

File format: tab delimited. peptide sequence 'tab' [1|0]. See examplefile_readBinary.txt

Experimental Values are stored in _predictions['EXP'] ('exp',score,1) Everything except the score will be ignored in further processing.

Parameters
`file:` (string) (Path to) and name of input file

readValues(*self*, *file*, *ascending*=1)

Reads in a set of peptides with values (predicted or experimental) in tab-separated file.

File format: tab delimited. the first three lines contain additional information.

Lines beginning with # are ignored

PEPLEN - specifies the peptide length.

METHOD - Name of the method.

ALLELE - defines the allelic models. tab delimited! Make sure that the order of the alleles matches the values given in the following lines!

This block is followed by the actual data: Peptide sequence and the respective values, tab delimited.

Example for inputfile format: see file 'examplefile_readValues.txt'

Parameters

`file:` (string) (Path to and) name of input file

`ascending:` (integer) default = 1. This variable indicates if higher scores stand for higher binders. This is needed for assigning ranks to the peptides. Default: high scores = high binders. For experimental values (IC50) set ascending = 0

removeModel(*self*, *model*)

Removes a model for the set of models.

This function is needed for unknown models.

Parameters

`model:` A Model in format ('method', 'allele')

setFromStrings(*self*, *sequences*, *peplen*)

Set peptide sequences directly without doing the detour over a protein Set. This is useful if preprocessing of the sequences by Fred is not needed.

Parameters

`sequences:` (list of strings) a list of peptide sequences

`peplen:` (integer) peptide length

Overrides: Fred.Fred.SequenceSet.setFromStrings

setModels(*self*, *models*)

Sets the prediction models.

For all models specified here, predictions will be made. A model has to be ('Method_Name','allele') for MHC models or ('Method_Name','') for TAP/Proteasome, where so allelic models are available.

Method_Name has to match one of the prediction methods. Unknown models are ignored during prediction. Overwrites self._models!

Parameters

`models:` (list) A list of models.

setParent(*self*, *parent*)

Associates the PeptideSet with the ProteinSet the peptides are derived form.

Parameters

`parent:` (ProteinSet) ProteinSet object the peptides are extracted from.

setPeplen(*self*, *peplen*)

Sets the peptide length.

Parameters

`peplen:` (integer) The length of the peptides contained in the PeptideSet.

setSvmDir(*self*, *svmdir*)

Sets the path to the SVM executables.

Sets the directory where the svm executables are stored. Default value 'path_prefix/svms/'

Parameters

`svmdir:` (string) Path to the SVM executables.

setSvmModelDir(*self*, *svmModeldir*)

Sets the path to the SVM models.

Sets the directory where the svm classify modelfiles are stored. Default value 'path_prefix/svms/'

Parameters

`svmModeldir:` (string) Path to the SVM model files.

setTmpDir(*self*, *tmpdir*)

Sets the temporary directory. Mostly needed for the Support Vector Machine results and encoding files.

The default directory is 'path_prefix/tmp/'. Prediction files are removed directly after usage.

Parameters

`tmpdir`: (string) location of tmp directory.

writeTabDelimitedFile(*self*, *filename*)

Writes prediction values into tab delimited file.

Parameters

`filename`: name of the output file.

Inherited from Fred.Fred.SequenceSet(Section 3.2.6)

addSequence(), getLength(), getMembers(), getSequences(), setSequences()

3.2.27 Class PredictionMethod

Known Subclasses: Fred.Fred.SVM, Fred.Fred.PSSM

Baseclass for Prediction methods.

Methods

__init__(*self*, *peptideSet*)

Initializes PredictionMethod object with PeptideSet.

Parameters

`peptideSet`: A PeptideSet

3.2.28 Class SVM

Fred.Fred.PredictionMethod
└ **Fred.Fred.SVM**

Known Subclasses: Fred.Fred.NetMHC, Fred.Fred.NetMHCIIPan, Fred.Fred.NetMHCPan, Fred.Fred.SVMHC, Fred.Fred.SVMTAP

Baseclass for SVM prediction method.

All subclasses use the svmlight package, version 6.0.

Methods

__init__(*self*, *peptideSet*)

Initializes SVM Prediction object.

Parameters

`peptideSet:` A PeptideSet

Overrides: Fred.Fred.PredictionMethod.__init__

encoding_sparse(*self*, *peptides*, *mer*, *filename*)

Computes sparse encoding of the peptides.

Sparse encoding: leading 1 in each line, one line per peptide. 20 'positions' per amino acid. Only the positions containing a one are listed.

Example for nonameric peptide 'SYFPEITHI': 1 16:1 40:1 45:1 73:1 84:1 108:1 137:1 147:1 168:1

The encoding is written in a file that is used as input for the SVM classifier.

Parameters

`peptides:` (list) A list of sequences.

`mer:` (int) The peptide length.

`filename:` (string) The path and filename of the output file.

execute(*self*, *command*, *outfile*)

Executes 'command' from the command line. Results are read in from the outfile.

Parameters

`command:` (string) The command to execute svm_classify (including path, modelfile, datafile and outputfile).

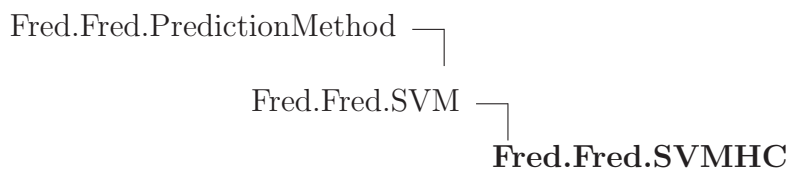
`outfile:` (string) the file from which the results are read.

Return Value

(list) The results of the svm prediction in a list.

setAlleles(*self*)

3.2.29 Class SVMHC



Class for SVMHC prediction. An SVM method for prediction of MHC class I peptide binding prediction.

Doennes, P. and Kohlbacher, O. SVMHC: a server for prediction of MHC-binding peptides. Nucleic Acids Res, 2006, 34, W194-W197.

Method name = 'MHC_I_SVMHC'

Methods

__init__(*self*, *peptideSet*)

Initializes SVM Prediction object.

Parameters

`peptideSet:` A PeptideSet

Overrides: Fred.Fred.PredictionMethod.__init__ extit(inherited documentation)

predict(*self*)

Makes SVMHC predictions for all peptides and all alleles.

setAlleles(*self*)

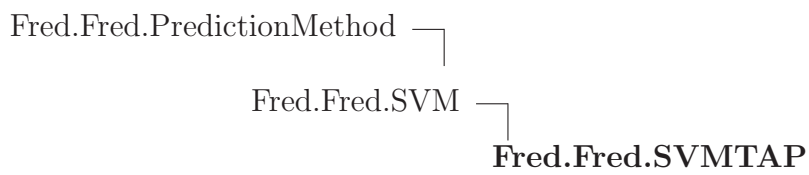
Extracts SVMHC alleles form PeptideSet models.

Overrides: Fred.Fred.SVM.setAlleles

Inherited from Fred.Fred.SVM(Section 3.2.28)

encoding_sparse(), execute()

3.2.30 Class SVMTAP



Class for SVMTAP prediction. An SVM method for prediction of TAP affinities.

Doennes, P. and Kohlbacher, O. Integrated modeling of the major events in the MHC class I antigen processing pathway. Protein Sci, 2005

Method name = 'SVMTAP'

Methods

__init__(*self*, *peptideSet*)

Initializes SVM Prediction object.

Parameters

`peptideSet:` A PeptideSet

Overrides: Fred.Fred.PredictionMethod.__init__ extit(inherited documentation)

predict(*self*)

Inherited from Fred.Fred.SVM(Section 3.2.28)

encoding_sparse(), execute(), setAlleles()

3.2.31 Class scoreDistribution

Class to compute score distributions and percentage values. Make sure to use a suitable set of peptides/proteins as basis for the score distribution. E.g. all peptides in a set of 100 000 randomly chosen natural proteins. Since this results in a huge number of scores, the computation of the score distributions might take a while. There are different possibilities to get the scores:

1. Read in a file with precomputed scores (one score per line)
2. Pass a list of scores
3. Scores are extracted for a model from PeptideSet with precomputed scores

To Do: descending scores?? Test scores from peptide set.

Methods

__init__(*self*)

computeCoverage(*self*)

Computes the coverage of the equdistant intervals.

computeIntervals(*self*)

Computes the interval boundaries for score distribution. The score range is devided into 100 equdistant intervals.

computePercentages(*self*)

Computes the percentage thresholds.

getCoverage(*self*)

Returns the cumulative coverage of the intervals. 'normal coverage' would be: the percentage of scores in that interval. cumulative coverage: sum of this interval and all intervals that cover lower scores.

Return Value

cumulative coverage in Dictionary. Keys: 1..100, Values: 'cumulative coverage' of the intervals.

getCoverageNoncumulative(*self*)

Returns the coverage of the intervals, i.e. the percentage of scores that lies in that interval.

Return Value

coverage in Dictionary. Keys: 1..100, Values: 'coverage' of the intervals.

getIntervals(*self*)

Returns the intervals boundaries for 100 equidistant intervals of the range the scores are in.

Return Value

interval boundaries. List: [(lower1,upper1),(lower2,upper1),...]

getPercentages(*self*)

Returns the percentage values. This method can be used to compute 'percentage thresholds' from background peptides for new prediction methods. Be careful with the set of peptides you use as

Return Value

coverage values in Dictionary. Keys: percentage (1..100), Values: corresponding threshold

getScores(*self*)

Returns the scores.

Return Value

Scores

scoresFromFile(*self*, *file*)

Reads scores from file. File format: one score per line.

Parameters

`file:` Name on input file containing precomputed scores.

scoresFromList(*self*, *scores*)

Takes scores from a list.

Parameters

`scores:` a list of scores (floats).

scoresFromPeptideSet(*self*, *peptideSet*, *model*)

Takes a peptideSet with precomputed scores. Extracts the scores for a given model. If no predictions are avialable for the given model, the function will try to make the predictions.

Parameters

`peptideSet:` A peptideSet with precomputed scores.

`model:` the model for which the scores should be considered

3.3 Module Fred.FredException

3.3.1 Class InputError

exceptions.Exception
└ **Fred.FredException.InputError**

Known Subclasses: Fred.FredException.NoFile, Fred.FredException.ParserError

Error druing file input.

Methods

__init__(*self*, *filename*)

Initialize InputError.

Parameters

`filename`: name of the onput file.

Overrides: exceptions.Exception.__init__

__str__(*self*)

Overrides: exceptions.Exception.__str__

Inherited from exceptions.Exception

__getitem__()

3.3.2 Class NoFile

exceptions.Exception
└ Fred.FredException.InputError
└ **Fred.FredException.NoFile**

Error druing file input. File not found.

Methods

__init__(*self*, *filename*)

Initialize NoFile Error

Parameters

`filename:` name of the onput file.

Overrides: exceptions.Exception.__init__

__str__(*self*)

Overrides: exceptions.Exception.__str__

Inherited from exceptions.Exception

__getitem__()

3.3.3 Class ParserError

exceptions.Exception

Fred.FredException.InputError

Fred.FredException.ParserError

Error druing file input. Parsing Error.

Methods

__init__(*self*, *filename*)

Initialize NoFile Error

Parameters

`filename:` name of the onput file.

Overrides: exceptions.Exception.__init__

__str__(*self*)

Overrides: exceptions.Exception.__str__

Inherited from exceptions.Exception

__getitem__()

3.3.4 Class ShortInput

exceptions.Exception
└
Fred.FredException.ShortInput

Class handles exceptions for wrong length of input sequences. Exception is thrown if the input sequence is shorter than the specified peptide length.

Methods

__init__(*self*, *inputLen*, *pepLen*)

Initialize ShortRawInputException.

Parameters

`inputLen:` (Integer) The length of sequence raising exception.

`pepLen:` (Integer) Peptide length specified by user.

Overrides: exceptions.Exception.__init__

__str__(*self*)

Overrides: exceptions.Exception.__str__

Inherited from exceptions.Exception

__getitem__()

3.3.5 Class svmError

exceptions.Exception
└
Fred.FredException.svmError

Class that handles execptions for failures is svm predictions. If svm prediction failes, no output is generated and the specified output file does not exist -> raises IOError. IOError is catched and a more specific exception is raised.

Methods

__init__(*self*, *command*)

Overrides: exceptions.Exception.__init__

__**str**__(*self*)
Overrides: exceptions.Exception.__str__

Inherited from exceptions.Exception

__getitem__()

3.3.6 Class pssmAlleleError

exceptions.Exception
└ **Fred.FredException.pssmAlleleError**

Class that handles problems with PSSM predictions: Unknown allele for the respective method.

Methods

__**init**__(*self*, *method*, *allele*)
Overrides: exceptions.Exception.__init__

__**str**__(*self*)
Overrides: exceptions.Exception.__str__

Inherited from exceptions.Exception

__getitem__()

3.3.7 Class BenchmarkError

exceptions.Exception
└ **Fred.FredException.BenchmarkError**

Exception is raised if no exprimental binary binding data is available but an evaluation method is called.

Methods

__**init**__(*self*)
Overrides: exceptions.Exception.__init__

__**str**__(*self*)
Overrides: exceptions.Exception.__str__

Inherited from exceptions.Exception

__getitem__()

3.3.8 Class BenchmarkErrorMCC

exceptions.Exception
└ **Fred.FredException.BenchmarkErrorMCC**

Exception is raised if MCC can not be calculated because binding is set so -1. This happens if no threshold values for binding prediction are available.

Methods

__**init**__(*self*)
Overrides: exceptions.Exception.__init__

__**str**__(*self*)
Overrides: exceptions.Exception.__str__

Inherited from exceptions.Exception

__getitem__()

3.4 Module Fred.FredPolymorphicExtension

3.4.1 Class PolymorphicPredictions

Class offers funcionality to make epitope predictions for polymorphic proteins (SNP).

Methods

__init__(*self*)

makePredictions(*self*, *pepsets*, *models*)

Method to run predictions for polymorphic protein.

Parameters

`pepsets:` a dict of polymorphic peptidesets. Keys: PolymorphicProteins. For each portein a dictionary with polymporphic position as key and the respective pepset as value.

`models:` a list of prediction models. Format: FRED prediction model format.

Return Value

polymorphic predictions in dictionary. Same format as parameter 'pepsets'

3.4.2 Class PolymorphicProtein

Fred.Fred.Sequence
Fred.Fred.Protein
Fred.FredPolymorphicExtension.PolymorphicProtein

Class representing a polymorphic protein. Extends class Fred.Protein. Additionally the polymorphisms of the protein are stored (position,[residues]).

Methods

__init__(*self*)

Initialize PolymorphicProtein.

Overrides: Fred.Fred.Sequence.__init__

setPolymorphisms(*self*, *polymorphisms*)

Method sets polymorphisms.

Parameters

`polymorphisms:` a list of polymorphisms. List of tuples: [(position,[residues]),...].

getPolymorphisms(*self*)

Yields list of all polymorphisms.

Return Value

a list of polymorphisms. List of tuples: [(position,[residues]),...].

Inherited from Fred.Fred.Protein(Section 3.2.4)

cleanSequence(), getName(), setName()

Inherited from Fred.Fred.Sequence(Section 3.2.3)

getLength(), getSequence(), setSequence()

3.4.3 Class PolymorphicPeptideSet

Fred.Fred.SequenceSet
Fred.Fred.PeptideSet
Fred.FredPolymorphicExtension.PolymorphicPeptideSet

Extends Fred.PeptideSet with some functions to keep track of the unerlying polymorphism.

Methods

__init__(*self*, *ProtSet*)

Initiaize PolymorphicPeptideSet.

Each PolymorphicPeptideSet is associated with one polymorphic position.

Parameters

`ProtSet:` Polymorphic protein set the peptides are derived from.

Overrides: Fred.Fred.SequenceSet.__init__

setPosition(*self*, *pos*)

Sets the polymorphic position.

Parameters

`pos:` The polymorphic position relative to the original protein sequence.

getPosition(*self*)

Return the polymorphic position.

Return Value

The polymorphic position relative to the original protein sequence.

setVariations(*self*, *variations*)

Set the variations of the polymorphic position

Parameters

`variations:` a list with the of amino acids that are observed at the polymorphic position.

getVariations(*self*)

Return the variations of the polymorphic position

Return Value

a list with the of amino acids that are observed at the polymorphic position.

setPolymorphism(*self*, *polymorphism*)

Set polymorphism.

Parameters

`polymorphism:` A FredPolymorphicExtension.Polymorphism object.

writeTabDelimitedFile(*self*, *filename*)

Extends the function od PeptideSet to add information about the polymorphism.

Parameters

`filename:` output file.

Overrides: Fred.Fred.PeptideSet.writeTabDelimitedFile

Inherited from Fred.Fred.PeptideSet(Section 3.2.26)

addModels(), cleanTmpDir(), getModels(), getParent(), getPeplen(), predict(), readBinaryBinding(), readValues(), removeModel(), setFromStrings(), setModels(), setParent(), setPeplen(), setSvmDir(), setSvmModelDir(), setTmpDir()

Inherited from Fred.Fred.SequenceSet(Section 3.2.6)

addSequence(), getLength(), getMembers(), getSequences(), setSequences()

3.4.4 Class PolymorphicProteinSet

Fred.Fred.SequenceSet
Fred.Fred.ProteinSet
Fred.FredPolymorphicExtension.PolymorphicProteinSet

Class represents a polymorphic protein set. Extends Fred.ProteinSet.

Methods

__init__(*self*)

Method initializes PolymorphicProteinSet.

Overrides: Fred.Fred.SequenceSet.__init__

parseFastaHeader(*self*, *header*)

Method parses polymorphic fasta header.

Parameters

`header:` (String) the polymorphic fasta header

Return Value

[protein name, polymorphisms]

readFasta(*self*, *fastafile*)

Reads polymorphic sequences from fastafile.

Format description: If polymorphic sequences should be specified put the polymorphisms in the FASTA header of the according sequence. Please use the following header format:

> ID | [Polymorphismlist] Reference Sequence

Polymorphismlist: Semicolon separated list of Polymorphisms Polymorphism: Position : Amino acid list Position: Polymorphic sequence position (Integer) Amino acid list: Comma separated list of observed amino acids (One letter code)

By default the first specified amino acid is treated as reference residue and should be the amino acid contained in the entered sequence. Any number of amino acids can be specified for one polymorphism.

But notice that for protein sequences containing non-standard amino acids the concerning residues are replaced by a dummy ('X'). Peptides containing an 'X' are not considered in epitope prediction.

Example:

> sequence | [5:A,G;10:R,M,V]

XXXXAXXXXRXXXXXXXXXXXXXXXXXXXXX

codes for

XXXXAXXXXRXXXXXXXXXXXXXXXXXXXXX

XXXXAXXXXMXXXXXXXXXXXXXXXXXXXXX

XXXXAXXXXVXXXXXXXXXXXXXXXXXXXXX

XXXXGXXXXRXXXXXXXXXXXXXXXXXXXXX

XXXXGXXXXMXXXXXXXXXXXXXXXXXXXXX

XXXXGXXXXVXXXXXXXXXXXXXXXXXXXXX

Parameters

`fastafile`: (String) filename and path to Fasta-file

Overrides: Fred.Fred.ProteinSet.readFasta

addUserSequence(*self*, *sequence*, *polymorphisms*, *seq_name=''*)

Method reads a single sequence in FASTA format and the respective snps: This method is used by EpiToolKit

Parameters

`sequence:` a protein sequence (string)

`polymorphisms:` FredPolymorphicExtension.Polymorphism object.

`seq_name:` Name of the protein

getPolymorphicPeptides(*self*, *length*, *tmpdir='./tmp/'*, *svmdir='./svms/'*)

Method creates polymorphic peptide sets. The peotide sets are not returned directly. Each PolymporphicPeptideSet represents one polymorphic position in one protein. To associate the PolymorphicPeptideSets with the a protein and a polymorphic position, the peptide sets are packed into nested dictionaries.

!!The reutn type differs from the return type of proteinSet.getPeptides() in Fred!!

Parameters

`length:` Peptide length (integer)

`tmpdir:` temporary directory (default: './tmp/')

`svmdir:` path to the svm directory (default: './svm/')

Return Value

A dictionary. Keys: the polymporphic proteins in the PolymporphicProteinSet; Values: Dictionary. Keys: Polymorphic positions of the respective protein. Values: a PolymorphicPeptideSet that contains all peptide variants defined by that polymorphic position.

Inherited from Fred.Fred.ProteinSet(Section 3.2.7)

accessRefSeq(), accessSwissProt(), computeAlignment(), getAlignment(), getAlignmentObject(), getConsensus(), getPeptides(), getVariability(), sequencesFromStrings(), setupClustalw(), writeFasta()

Inherited from Fred.Fred.SequenceSet(Section 3.2.6)

addSequence(), getLength(), getMembers(), getSequences(), setFromStrings(), setSequences()

3.4.5 Class Polymorphism

Class representing protein polymorphism.

Methods

__init__(*self*)

setSource(*self*, *source*)

Method sets source of polymorphism data. E.g. dbSNP, swissprot... Just for information

Parameters

`source`: Source of polymorphism

getSource(*self*)

Method returns source of poylmorphism data.

Return Value

Source of polymorphism

setDbID(*self*, *dbid*)

Method sets database id of entry.

Parameters

`dbid`: ID of SNP entry

getDbID(*self*)

Method returns database id polymorphism data.

setPosition(*self*, *pos*)

Method sets position of polymorphims.

Parameters

`pos`: position of the polymorphism

getPosition(*self*)

Method returns position of polymorphism.

Return Value

position of the polymorphism

setReference(*self*, *ref*)

Method sets reference amino acid of polymorphism. If no reference is speciefied self._reference=.

Parameters

`ref:` reference amino acid (default =)

getReference(*self*)

Method returns reference amino acid of polymorphism.

Return Value

reference amino acid

setVariations(*self*, *vars*)

Method sets variations of polymorphism.

Parameters

`vars:` observed variations at this polymorphic position. A list of amino acids.

getVariations(*self*)

Method returns variations of polymorphism.

Return Value

observed variations at this polymorphic position. A list of amino acids.

setHeterozygosities(*self*, *hzygosities*)

Method sets heterozygosity and stderr.

getHeterozygosities(*self*)

Method return heterozygosity values.

3.5 Module Fred.halfmax_thresholds

3.5.1 Variables

Name	Description
halfmax	**Value:** {'MHC_II_HAMMER': {'DRB1_0101_9': 3.0, 'DRB1_0102_9': 3.0...

3.6 Module Fred.matrices_bimas_log

3.6.1 Variables

Name	Description
bimas_matrices	**Value:** {'A_0101_9': [{'con': -4.60517018599}, {'A': 0.0, 'C': 0....

3.7 Module Fred.matrices_epidemix

3.7.1 Variables

Name	Description
epidemix_matrices	**Value:** {'A_01_9': [{'A': -0.4, 'C': 0.3, 'D': -0.6, 'E': -0.3, '...

3.8 Module Fred.matrices_hammer

3.8.1 Variables

Name	Description
hammer_matrices	**Value:** {'DRB1_0101': [{'A': -999.0, 'C': -999.9, 'D': -999.0, 'E...

3.9 Module Fred.matrices_pcm

3.9.1 Variables

Name	Description
pcm_matrices	**Value:** {'all': [[0.473123756582, -0.502526820951, -0.21691300156...

3.10 Module Fred.matrices_syfpeithi

3.10.1 Variables

Name	Description
syfpeithi_matrices	**Value:** {'A_0101_10': [{'A': 1, 'C': 0, 'D': 0, 'E': 0, 'F': 0, '...

3.11 Module Fred.matrices_syfpeithi_II

3.11.1 Variables

Name	Description
syfpeithi_matrices	**Value:** {'DRB1_0101_9': [{'A': 0, 'C': 0, 'D': 0, 'E': 0, 'F': 10...

3.12 Module Fred.matrix_additive_method

3.12.1 Variables

Name	Description
additive_matrices	**Value:** {'all': [{'A': 0.4, 'C': 0.0, 'D': -1.24, 'E': -1.349, 'F...

3.13 Module Fred.percentages_proteasome

3.13.1 Variables

Name	Description
percentages	**Value:** {'PROTEASOME': {0: 20.3836067249, 1: 3.12130410506, 2: 2....

3.14 Module Fred.percentages_svmtap

3.14.1 Variables

Name	Description
percentages	**Value:** {'SVMTAP': {0: -54.57134791, 1: -52.85706405, 2: -51.1427...

3.15 Module Fred.percentages_to_threshold

3.15.1 Variables

Name	Description
percentages	**Value:** {'MHC_II_HAMMER': {'DRB1_0101': {0: 5.0, 1: 1.25, 2: 0.75...

3.16 Module Fred.svmhc_alleles

3.16.1 Variables

Name	Description
svmhc_alleles	**Value:** `['A_01_9', 'A_0201_9', 'A_0201_10', 'A_03_10', 'A_03_9', ...`

Index