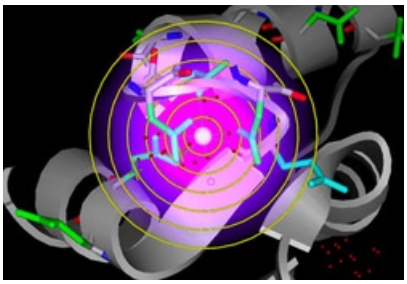


FEATURE User's Manual

Published: Tuesday 8:06 PM, February 18, 2014
Website: simtk.org/home/feature

Copyright and Permission Notice

Acknowledgements

Feature is funded by National Laboratory of Medicine R01 LM05652.

SimTK software and all related activities are funded by Simbios National Centerfor Biomedical Computing through the National Institutes of Health Roadmap for Medical Research, Grant U54 GM072970. Information on the National Centers can be found at http://nihroadmap.nih.gov/bioinformatics.

Table of Contents

Chapter 1. FEATURE 3.1

1.1. Introduction

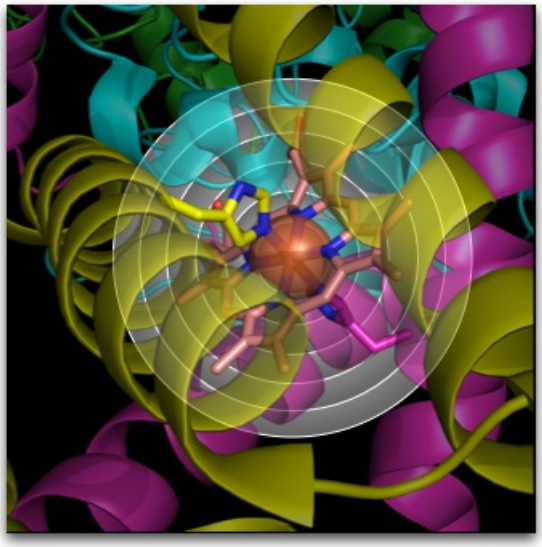
This document discusses the FEATURE Software version 3.1, which was released February, 2014. There is a glossary (Chapter 7.1) to help you understand how we use some terms such as *model*, *classifier* or *force field*.

1.1.1. About FEATURE

FEATURE is software for modeling and finding functional domains in proteins. In other words, FEATURE learns to associate protein function with protein structure by analyzing the structures of functionally similar proteins. Using what it has learned, FEATURE can predict protein functionality in a given protein structure. References related to FEATURE such as research papers are covered in References (Chapter 8).

1.1.2. FEATURE Represents Protein Structure as Computable Vectors

FEATURE analyzes protein structure at the atomic level, sampling tiny spherical volumes around each atom or any given set of points. These volumes are called *microenvironments* (shown in the figure on the right). Microenvironments are represented by a vector of real numbers called a *feature vector*. Given a point in a protein of unknown function, if the feature vector of that point is statistically similar to a feature vector of a known functional domain, then FEATURE predicts that the unknown protein has similar functionality at that point.



Conceptual diagram of a microenvironment in a protein, showing the concentric shells around a metal ion ligand.

Feature vectors are created by tallying the physicochemical properties of the atoms in a microenvironment. These tallies form the numbers in the feature vector. For more information on feature vectors, see feature vector file format. Examples of physicochemical properties include: atom name, partial charge, Van der Waals radii, and solvent accessiblity. For more information on properties, see Physicochemical Properties used by FEATURE (Chapter 6.1).

1.1.3. FEATURE Learns to Associate Protein Function with Structure

FEATURE uses supervised machine learning algorithms to associate protein function with structure. The generalized association is called a *model*; FEATURE uses functional models to make predictions of protein function for any protein structure. Model building involves two training sets: a positive set (based on microenvironments of active sites within a given functional class) and a negative control set (based on microenvironments of proteins that don't belong to the given functional class).

1.1.4. FEATURE Uses What It Has Learned to Predict Protein Function

FEATURE 3.1 uses a Bayesian scoring function to predict protein function for any given microenvironment. The Bayesian scoring function treats each property as an independent event. Over- or underrepresented properties in the positive training data relative to the control data capture the significant features of the modeled active site. FEATURE 4.0 will include Support Vector Machine (SVM) classifiers and Random Forest classifiers, with good results. These additional classifiers will be released pending publication.

1.1.5. About this User Guide

The purpose of this user manual is to help researchers learn to download, install, and use the FEATURE 3.1 software on their own computers. FEATURE is the result of cross-disciplinary research and intended for education. Therefore this user manual emphasizes simple explanations that graduate and undergraduate biologists, engineers, and computer scientists can collectively understand.

WebFEATURE (http://feature.stanford.edu) is online software that offers some of the capabilities of FEATURE 4.0. This guide does not cover WebFEATURE; the usage information for WebFEATURE is available on the WebFEATURE website. For more information on WebFEATURE, see Two Ways to use FEATURE: Online vs. Desktop or visit the website: http://feature.stanford.edu.

FEATURE users can contribute to the FEATURE source code by contacting the FEATURE *developers* at feature-dev@simtk.org. Users who want to contribute should also read the FEATURE Developer Manual.

For more general information on FEATURE, or to download the latest release of FEATURE, see the official FEATURE website at https://simtk.org/home/feature. If you have comments or questions, please e-mail us at feature-users@simtk.org.

1.1.6. Two Ways to use FEATURE: Online vs. Desktop

FEATURE is available in two forms: on the web at http://feature.stanford.edu and as freely downloadable open source software available at http://simtk,org/home/feature. *Online users* will use WebFEATURE, the web version of FEATURE, and *Desktop users* will download and install the open source FEATURE software. Once installed, desktop users can use FEATURE command-line programs on their computer.

Use WebFEATURE if:

- You intend to use one of WebFEATURE's built-in site models; and
- You want to use the protein function search capabilities of FEATURE; and
- You don't want to install FEATURE on your computer

Download, install, and use FEATURE if:

- You want to create your own site model(s); or
- Your data set is large and you want dedicated computational performance; or
- Your data set is sensitive and cannot be shared

1.1.7. Online Users

If you just want to try out FEATURE and don't want to install the software on your computer, you can use WebFEATURE online at http://feature.stanford.edu/webfeature. WebFEATURE's user interface has links to instructions on how to use WebFEATURE.

1.1.8. Desktop Users

If you want to download and install FEATURE binaries on your computer, see Installation (Chapter 2) and Running the Serine Protease Example (Chapter 3.1)

Users can freely download and install the released binaries under the provisions of the FEATURE License (Chapter 9.1). Users can also freely download a copy of the source code. Both binaries and source code are available from the SimTK website (https://simtk.org/home/feature).

1.2. Conventions Used in this Document

Specific names of software programs and files are *italicized*. Software that is to be run in a command-line window will be shown as follows:

```
$
```

For example, if the instructions ask for you to make a listing of the directory, you will see:

```
$ ls
```

Don't forget to hit the *return* key after every command.

Menus choices are shown in bold along with carats to indicate submenus, *e.g.* Menu > Submenu.

1.3. Change Log

1.3.1. FEATURE 3.1 Changes

- Simpler, more reliable build
- Option to optimize speed using modern C++ platforms

1.3.2. FEATURE 3.0 Changes

- Improved accuracy
- User-selectable properties controlled by a human-readable Selectable Properties File

- Default property list is the same as FEATURE 2.0
- Provides example Selectable Properties File to emulate FEATURE for Metals
- Unifies the property lists from FEATURE 2.0 and FEATURE for Metals.
- Uses smart caching techniques to achieve speeds up to 20x faster than FEATURE 1.9
- Improved reading/writing file performance; no arbitrary file name or file size limits!
- FEATURE files (Feature Vector Files, Model Files, and Score Files) provide new header information to avoid mismatching properties
- Easier process for developers to add new properties; change 3 files instead of changing 10 files.
- Simplified architecture reduces likelihood of bugs (60% fewer files, 60% less code, 70% more documentation)
- Improved stability and memory footprint (max memory used: ~300 MB)
- Thorough nightly testing now includes behavior comparisons with FEATURE for Metals
- Improved, user-tested documentation
- Deprecated the LISP model format

1.4. The People Behind FEATURE

FEATURE was created by Prof. Russ Altman and the Helix Group at Stanford University. Prof. Altman continues to lead FEATURE development today. Currently, FEATURE is a collaborative project between Stanford and San Francisco State University's Center for Computing for Life Sciences (CCLS, http://cs.sfsu.edu/ccls) under the direction of CCLS Director Prof. Dragutin Petkovic and Development Lead Mike Wong.

Chapter 2. Installation

This section describes how to download, install, and prepare your computer to run the FEATURE programs as a desktop user. You will need a web browser, a PDF Reader, and an Internet connection. This tutorial assumes you are familiar with how to use e-mail, a web browser, and basic computer tasks. If you have questions, please contact us at feature-users@simtk.org.

2.1. Download the Latest Version from SimTK

FEATURE software is available as source code, and instructions to compile FEATURE for your computer are included below. If you need help, contact us at feature-users@simtk.org.

To download FEATURE, go to the FEATURE Home Page https://simtk.org/home/feature and click on the Downloads link on the left to download one of the following binary distribution files:

For all other platforms you can compile the source code
`feature-3.1.0-src.tar.gz`

2.2. Compiling FEATURE from Source

FEATURE is written in C++ and as such should perform across any platform that provides a C++ compiler. Our primary target platforms are UNIX systems (e.g. Mac OS X, Debian Linux, or Red Hat Linux), and secondarily Windows with Cygwin. We appreciate any feedback to help us make the compilation process as accurate as possible! Contact us at: feature-dev@simtk.org.

2.2.1. FEATURE is Designed to Compile with Open Source Software

FEATURE is designed to be compiled using the GNU Compiler Collection (GCC) and GNU Make. The GCC is software to compile source code into executable binaries. GNU Make manages all the compilation details (such as command line options and paths) for you.

GCC is available online from the GNU website: http://www.gnu.org/software/gcc. GNU Make is also available from the GNU website: http://www.gnu.org/software/make. When possible, we will show you how to install both GCC and the GNU Build Tools using a package management system (e.g. `apt-get` for Debian/Ubuntu Linux, `yum` for Red Hat Linux, and community-supported MacPorts for Mac OS X) because that is vastly easier. However, you can always download the GCC and GNU Make from the GNU website and build them from source.

FEATURE also relies on zlib (website: http://zlib.net), a free, open source compression and decompression library. There are separate instructions on how to get zlib for some systems below. Otherwise, download the zlib source code code from the zlib website and compile the library according to the instructions on the zlib website.

2.2.2. Pre-Compile Instructions for Mac OS X

Most of our developers build FEATURE using Mac OS X, specifically Mac OS X 10.5 and later. You will need to Apple's Developer Toolkit, Xcode, which provides a C++ compiler, make, and zlib, available here:

https://developer.apple.com/technologies/tools

Recent versions of Xcode provides another C++ compiler, Clang, instead of GCC. While Clang is intended to be a drop-in substitute for GCC, some releases of Clang fall short of this goal. When you type `g++` in Terminal, older versions of Xcode will run GCC; newer versions will run Clang.

FEATURE is designed to compile with GCC and FEATURE will usually compile with Clang as well. After installing Xcode, try to compile FEATURE by following the general compilation and installation instructions below; if this doesn't work, then install GCC and try the general compilation and try the custom compiler instructions. You can download binaries for GCC from High Performance Computing on SourceForge (http://hpc.sourceforge.net), MacPorts (http://www.macports.org), or build GCC from source (http://www.gnu.org/software/gcc).

To find out if you have Clang or GCC, type:

```
$ g++ --version
```

The response should be similar to one of the two following lines:

1. `i686-apple-darwin10-g++-4.2.1 (GCC) 4.2.1 (Apple Inc. build 5666)` or
2. `Apple LLVM version 5.0 (clang-500.2.79) (based on LLVM 3.3svn)`

If the response is similar to line 1, then you have Xcode with GCC, and FEATURE will build just fine. Otherwise if the response is similar to the line 2, then you have Xcode with Clang, and FEATURE will probably build. If it doesn't, get GCC from the sources described above and try the custom compiler instructions.

2.2.3. Pre-Compile Instructions for Debian/Ubuntu

Some of our developers have build FEATURE using Debian Linux, specifically Ubuntu 10.10 and later. You will need to run the following commands to fulfill some of the dependencies for FEATURE:

```
$ sudo apt-get install build-essential zlib zlib-devel
```

After you have completed installing the dependencies, follow the general compilation and installation instructions below.

2.2.4. Pre-compile instructions for Red Hat-compatible Linux

Some of our developers have build FEATURE using one of several Red Hat- compatible Linux distributions, specifically CentOS 5.4 and later. You will need to run the following commands as root to fulfill some of the dependencies for FEATURE:

```
$ yum groupinstall 'Development Tools'
$ yum install zlib zlib-devel
```

After you have completed installing the dependencies, follow the general compilation and installation instructions below.

2.2.5. Pre-Compile Instructions for Windows

Some of our developers have had success compiling FEATURE on Windows using Cygwin's GNU libraries and utilities (http://www.cygwin.com). We recommend you start by installing Cygwin, including GCC, GNU make, and the zlib packages.

After you have completed installing the dependencies, open a shell terminal and follow the general compilation and installation instructions below.

2.3. General Compilation and Installation Instructions

You can unpack the source code distribution with the following command:

```
$ tar -xvzf feature-3.1.0-src.tar.gz
```

To build FEATURE 3.1 binary executables, use the following command:

```
$ make
```

FEATURE 3.1 has some speed optimizations that are available only to C++ compilers that comply with the 2011 C++ revision (c++11) or the 2003 C++ revision (c++03). Primarily, FEATURE supports GCC 4.6 and later. The fast version of FEATURE is about 33% faster. The build system for FEATURE will try to compile fast FEATURE for the 2011 C++ revision first; if this fails then it will try to compile fast FEATURE for the 2003 C++ revision next. If this fails, it will try to build standard FEATURE which relies on the 1998 C++ revision.

If the attempt to build the fast version hangs indefinitely (takes more than an hour), or if you only want to build the standard version of FEATURE, instead of typing `make`, type `make build-standard`.

2.3.1. Custom Compiler Instructions

If the general compilation instrutions don't work, you can tell the FEATURE 3.1 build system to use a different C++ compiler. For example, if a GCC C++ compiler is installed at `/usr/local/bin/g++`, use the following command:

```
$ make CXX=/usr/local/bin/g++
```

2.3.2. Installing FEATURE With Root Access

If you have root access (i.e. `sudo`) on your system and you want to install FEATURE in the default location (`/usr/local/feature`), use the following commands:

```
$ sudo make install
```

Add the following lines to `/etc/bashrc` or the appropriate shared shell resource file.

```
export PDB_DIR=/usr/local/feature/data/pdb
export DSSP_DIR=/usr/local/feature/data/dssp
export FEATURE_DIR=/usr/local/feature/data
export PYTHONPATH=$PYTHONPATH:/usr/local/feature/tools/lib
```

2.3.3. Installing FEATURE in a Directory of Your Choice

If you don't have root access on your system or don't want to install FEATURE in the default directory, you can install FEATURE in the directory of your choice (such as `~/feature`) with the following commands:

```
$ mkdir -p <path>
$ cp -Rf bin data tools README <path>
```

Add the following lines to your `.bashrc`, or `.bash_profile`, or the appropriate shell resource file.

```
export PATH=$PATH:<path>/bin:<path>/tools/bin
export PDB_DIR=<path>/data/pdb
export DSSP_DIR=<path>/data/dssp
export FEATURE_DIR=<path>/data
export PYTHONPATH=$PYTHONPATH:<path>/tools/lib
```

2.3.4. Testing FEATURE After Compiling from Source

The tests will automatically be run as part of the compilation process and you will be warned if the tests fail. If you want to re-run the tests, type:

```
$ cd tests
$ make clean
$ make
```

2.4. Troubleshooting

If you are having trouble installing FEATURE, check the FEATURE Users' Group Mailing List Archives at https://simtk.org/pipermail/feature-users/.

If you still need help, please post to the Group mailing list at feature-users@simtk.org; users and developers read this mailing list and will be able to help you. The `/usr/local/feature/tests/installtest.log` file may be useful to troubleshoot the installation problems; if you post the results of this file, you will improve the chances that someone on the mailing list can help you.

3.1. Learning to Use FEATURE

The best way to learn to use the FEATURE software is to try an example. This exercise is intended for Desktop Users to learn the basics of how to run the FEATURE command line tools and learn how to read and interpret the results from FEATURE. This section builds on concepts presented in the Introduction (Chapter 1) and the section on Installation (Chapter 2).

3.1.1. Conceptual Overview

This section describes briefly what FEATURE does and how to use the programs that FEATURE provides to search for functional domains in protein structures and model functional domains. Note that FEATURE is primarily used for proteins, but it can also find RNA domains.

You can use FEATURE to:

1. Characterize microenvironments in biomolecular structures
2. Model biomolecular functional classes
3. Search for functionality within a biomolecular structure

FEATURE describes structures at the atomic level by accumulating physicochemical properties in highly localized spherical volumes called microenvironments. FEATURE provides the `featurize` program to characterize microenvironments in protein structures in PDB files and write out the descriptions as feature vector files.

A functional class can be modeled using structure descriptions of all domains in the given class. These structure descriptions form the positive training data. In contrast, a sample of structure descriptions of protein structures outside the given class form the negative training data. FEATURE provides the `buildmodel` program to train a machine learning algorithm on the positive and negative training data.

If a protein contains a structure similar to one being modeled, then the protein may belong to the same functional class as the model. FEATURE provides the `scoreit` program that compares all points in a protein structure to a model, effectively searching for functional domains in the protein structure that match the model.

Software Overview

FEATURE provides three command line programs:

`featurize`	Characterizes microenvironments within biomolecular structures
`buildmodel`	Train a machine learning classification model positive and negative training data
`scoreit`	Evaluates the likelihood that points within a biomolecule match a modeled functional class

3.1.2. Calling FEATURE Programs from the Command Line

FEATURE command line programs and useful tools are configured to be directly accessible from the command line. For example, you can run the `featurize` program by typing the `featurize` command in a terminal window as follows:

```
$ featurize
```

If you need help with any of the programs, simply run the program from the command line, without any arguments.

```
$ ./featurize
Error: ERROR: No PDB IDs requested

FEATURE 3.1.0 (build 1622) - featurize

Usage: featurize [OPTIONS] [PDBID...]

Must exclude .ent or other filename extension in PDBID

Options:
    -v  Increase verbosity
    -n SHELLS [Default: 6]
        Set number of shells to NUMSHELLS
    -w SHELL_WIDTH [Default: 1.25]
        Set thickness of each shell to SHELLWIDTH Angstroms
    -x EXCLUDED_RESIDUES [Default: HETATM]
        Set residues to exclude to comma separated list of
        EXCLUDERESIDUES
    -f PDBID_FILE
        Read PDBIDs from PDBIDFILE
    -P POINT_FILE
        Read point list from POINTFILE
    -l PROPERTIES_FILE
        Read a property list from PROPERTIESFILE
    -s SEARCH_PATH
        Look for protein files in SEARCHPATH first before system directories
```

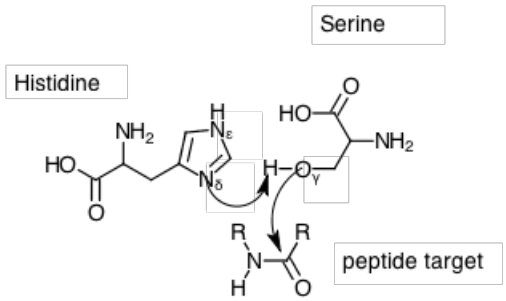
How to Use FEATURE to Build a Model

This section presents:

- An introduction on how training data points are selected
- Instructions to characterize positive training data points
- Instructions to characterize negative training data points
- Instructions to build the model

The most important step for building a protein functional model is the selection of the training data. One source of training data is joining the Prosite database (http://expasy.org/prosite) and UNIPROT database (http://www.uniprot.org) and datamining the resultant join for groups of proteins with a similar function. The details for this method is beyond the scope of this document and the motivated user may review the references (http://feature.stanford.edu/publications.php) for a complete description of this approach.

A functional mechanism is the chemical pathway that shows how a functional class catalyzes reactions. Functional mechanisms typically show electron transfers, step-by-step, with all intermediate molecular states, until the catalytic reaction is complete and the domain is restored.



Serine Protease Mechanism First Step

Note that the histidine imidazole ring can flip over so that the two nitrogens are interchangeable.

This figure shows the electron transfers for the histidine nitrogens (Nδ, Nε) and serine gamma oxygen (Oγ) in the first step to initiate protease activity on the peptide target. FEATURE uses each of these reactive atoms to model serine protease activity.

FEATURE models one reactive atom for a given functional mechanism, such as the serine gamma oxygen (Oγ) or histidine delta or epsilon nitrogens (Nδ or Nε) for serine protease functionality. The first step of the serine protease mechanism is shown in the figure above. Another example of reactive atoms includes any of three or four gamma sulfur (Sγ) in cysteine residues for ferredoxin functionality. Therefore a given functional class may have multiple models. Each match between a given protein and the models of a functional class is evidence that the protein may also belong to the modeled functional class.

The end result of training data selection is a list of positions of the key atom in proteins belonging to a given functional class (positive training data points) and a list of positions of the reactive atom within proteins that are not in the given functional class (negative training data points). These lists are stored in separate files, one for the positive training data, one for the negative training data. They must follow the Point File format.

The first step is to characterize the microenvironments at the points for functionally similar active sites. This becomes the positive training data. The command to do this is as follows:

```
$ featurize -P positive_training_data.ptf > positive_training_data.ff
```

Similarly, the command to characterize the negative training is as follows:

```
$ featurize -P negative_training_data.ptf > negative_training_data.ff
```

The last step is to train the model on the data. The command to build a model based on training data is as follows:

```
$ buildmodel positive_training_data.ff negative_training_data.ff > domain-atom.model
```

You may now use the resulting FEATURE model to search other protein structures for functional activity.

How to Use FEATURE to Search for Functionality

This section explains how to use a given model (`domain-atom.model`) to search for active sites in a given protein structure (`pdb_id`) using the following steps:

- Select relevant atoms in the given protein structure
- Characterize the microenvironments around the relevant atoms
- Use the model to calculate scores for matches

Because each model is based on the microenvironment around one reactive atom, it saves time to look at each microenvironment of all occurrences of the same reactive atom in a given protein. This is a recommended practice, not a requirement; users are free to score every atom in a protein structure.

To select all occurrences of the reactive atom, use the `atomselector.py` utility as follows:

```
$ atomselector.py -r residue -a atom pdb_id > pdb_id.ptf $ featurize -P pdb_id.ptf > pdb_id.ff
```

Optionally, you can look at every atom in the structure.

```
$ featurize -P pdb_id > pdb_id.ff
```

Once you have c

To search for active sites, type the following command:

```
$ scoreit -a domain-atom.model pdb_id.ff > pdb_id.hits
```

The search results are called *hits* and are stored in the `pdb_id.hits` file which is in the Score File format. The Score File format assigns a score to each atom in the given protein structure The scores in the second column `pdb_id.hits` file indicate the relative goodness of the match at the . Higher scores indicate a stronger match. To see the top 5 scores, you can use the following commands:

```
$ sort -k2 -n pdb_id.hits | tail -n 5
```

Each atom that scores high

3.1.3. Running the Serine Protease Example

The Serine Protease example is included in FEATURE binaries for the purpose of education and testing. This example uses FEATURE to:

1. Build a model of the serine protease domain based on the microenvironment surrounding the OG atom of the conserved serine residue.
   The required inputs are:
   - Positive Training Data: PDB structures of serine protease.
   - Negative Training Data: PDB structures of non-serine protease.
2. Identify potential serine protease active sites by scoring Serine residues from other structures. For this example, we will score a known serine protease, TSV-PA (PDB ID 1bqy), which is an active component in the venom of the Chinese green tree viper and a hydrolase (PDB ID 1ufo) from an extremophile, Thermus thermophilus.

Open a Command Prompt/Terminal Window

FEATURE is run from the command prompt within a terminal window. To open a terminal window, select:

Mac OS X
Macintosh HD > Applications > Utilities > Terminal

Red Hat Linux
Applications > System Tools > Terminal

Copy the Example Directory to Your Home Directory

The example is included in the FEATURE distribution which is available from the FEATURE Home Page https://simtk.org/home/feature. Type the following command to navigate to the example directory.

```
$ cp -Rf /usr/local/feature/examples/serprot ~
$ cd serprot
```

Build the model

The example directory contains data files for building a model of trypsin-like serine proteases. These data files contain the positive training set point file (`trypsin_ser_og.pos.ptf`) and the negative training set point file (`trypsin_ser_og.neg.ptf`).

`trypsin_ser_og.pos.ff`

XYZ coordinates of the conserved serine gamma oxygen atom from the PDB files of serine proteases. This is the positive training set point file. Note, 22 points are used here, but as few as 5 points can be used if enough examples are not available.

`trypsin_ser_og.neg.ff`

XYZ coordinates of the conserved serine OG atom from the PDB files of non-proteases. This is the negative training set point file. Note, 49 points are used here, but experiments show that more points give better results.

To learn more about point files (.ptf), see Point File.

To create a model, the microenvironments (local environment) of these points need to be characterized using `featurize`:

```
$ featurize -P trypsin_ser_og.pos.ptf > trypsin_ser_og.pos.ff
$ featurize -P trypsin_ser_og.neg.ptf > trypsin_ser_og.neg.ff
```

To learn more about the `featurize` program, see featurize and FEATURE Vector File. Note that `featurize` needs a `.dssp` file for each `.pdb` file to function properly. The necessary files have already been created for you. To learn more about creating `.dssp` files, see DSSP File

After characterizing the environment, create a model using `buildmodel`

```
$ buildmodel trypsin_ser_og.pos.ff trypsin_ser_og.neg.ff > trypsin_ser_og.model
```

To learn more about the `buildmodel` program, see buildmodel.

Use the Model to Find Serine Protease Activity

Once a model is built, you can use it to identify locations with similar environments on another protein structure. For instance 1bqy is a serine protease that was not in the original training set. You can also try 1ufo.

1. You'll need to make sure that there's a DSSP file for the PDB file of your protein-of-interest. For `1bqy` and `1ufo`, this step has already been done for you and you can skip this step (we are not the developers of `dsspcmbi`, so installing and testing `dsspcmbi` is beyond the scope of this tutorial).
   For other PDB files (such as `4hhb`), you can create the DSSP file with this command:
   ```
   $ dsspcmbi pdb4hhb.ent pdb4hhb.dssp
   ```

2. Select the residues at which to scan. Generally these are the residues and atoms that were used in the model. In this case, it is the OG atom of the conserved serine residue:
   ```
   $ atomselector.py -r ser -a og 1bqy > 1bqy_ser_og.ptf
   ```

3. That will create a point file which can be fed into `featurize` to characterize the microenvironments:
   ```
   $ featurize -P 1bqy_ser_og.ptf > 1bqy_ser_og.ff
   ```

4. You can then use the model you built earlier to score all the micro-environments and then sort the results in ascending order.
   ```
   $ scoreit -a trypsin_ser_og.model 1bqy_ser_og.ff > 1bqy_ser_og.hits
   $ sort -k2 -n 1bqy_ser_og.hits > 1bqy_ser_og.hits.sorted
   ```
   To learn more about the `scoreit` program and its output, see scoreit and Score File respectively.

5. You can see the top 5 scoring hits in 1bqy by using:
   ```
   $ tail -n 5 1bqy_ser_og.hits.sorted
   ```
   You should see the highest scoring hits. The first field is just a unique label for the site. The second field is the score for the site. The next three fields are the x,y,z coordinates for the hit. The last field shows the residue. SER195 is the catalytic serine for `1bqy`.

   ```
   Env_1bqy_0      -110.586184     11.959  25.312  62.893  #       SER29:A@OG
   Env_1bqy_32     -36.637977      37.557  8.635   139.522 #       SER214:B@OG
   Env_1bqy_15     -28.324812      22.436  37.11   53.493  #       SER214:A@OG
   Env_1bqy_31     149.060025      35.096  9.444   145.234 #       SER195:B@OG
   Env_1bqy_14     176.033218      19.642  39.631  59.232  #       SER195:A@OG
   ```

Summary of Steps

Here's the entire sequence of commands to issue, repeated for your convenience.

```
$ cd /usr/local/feature/examples/serprot
$ featurize -P trypsin_ser_og.pos.ptf > trypsin_ser_og.pos.ff
$ featurize -P trypsin_ser_og.neg.ptf > trypsin_ser_og.neg.ff
$ buildmodel trypsin_ser_og.pos.ff trypsin_ser_og.neg.ff > trypsin_ser_og.model
$ dsspcmbi ../../data/pdb/pdb1bqy.ent ../../data/dssp/pdb1bqy.dssp
$ atomselector.py -r ser -a og 1bqy > 1bqy_ser_og.ptf
$ featurize -P 1bqy_ser_og.ptf > 1bqy_ser_og.ff
$ scoreit -a trypsin_ser_og.model 1bqy_ser_og.ff > 1bqy_ser_og.hits
$ sort -k2 -n 1bqy_ser_og.hits > 1bqy_ser_og.hits.sorted
$ tail -5 1bqy_ser_og.hits.sorted
```

We recommend that after downloading and installing the FEATURE desktop binaries you run this test and check that the example output matches the above values.

3.1.4. Using Selectable Properties

One of the biggest improvements to FEATURE 3.1 is the ability to selectable properties. This means that you can customize how FEATURE characterizes microenvironments. To select properties, either copy `proteins.properties` or `metals.properties` and edit the file.

For example:

```
$ cp proteins.properties my.properties
$ vim my.properties
# ================================================================================
# SELECTABLE PROPERTIES FILE
# ================================================================================
# This is a system-level file. To customize your own selectable properties,
# copy this file to your directory.
#
# To indicate that you don't want to use a property, add a hash mark (#) in
# front of the property name
#
# Example:
# # ATOM_TYPE_IS_XYZ
# ================================================================================

  ATOM_TYPE_IS_C
  ATOM_TYPE_IS_CT
  ATOM_TYPE_IS_CA
  ATOM_TYPE_IS_N
```

You can then comment out properties you don't want by adding a hash mark (#) in front of the property name.

```
# ================================================================================
# SELECTABLE PROPERTIES FILE
# ================================================================================
# This is a system-level file. To customize your own selectable properties,
# copy this file to your directory.
#
# To indicate that you don't want to use a property, add a hash mark (#) in
# front of the property name
#
# Example:
# # ATOM_TYPE_IS_XYZ
# ================================================================================

# ATOM_TYPE_IS_C
# ATOM_TYPE_IS_CT
# ATOM_TYPE_IS_CA
  ATOM_TYPE_IS_N
```

To add new properties, simply type the property name. In the example below, we add the property `AMIDE_CARBON`.

```
# ================================================================================
# SELECTABLE PROPERTIES FILE
# ================================================================================
# This is a system-level file. To customize your own selectable properties,
# copy this file to your directory.
#
# To indicate that you don't want to use a property, add a hash mark (#) in
# front of the property name
#
# Example:
# # ATOM_TYPE_IS_XYZ
# ================================================================================

  AMIDE_CARBON
# ATOM_TYPE_IS_C
# ATOM_TYPE_IS_CT
# ATOM_TYPE_IS_CA
  ATOM_TYPE_IS_N
```

Once you have the properties customized as you like, call `featurize` with the `-l` option as shown below.

```
$ featurize -l my.properties -P my_pointfile.ptf
```

For more information on how to use `featurize`, see featurize. For more information on selectable properties, see selectable properties file. For more information on properties in general, see physicochemical properties (Chapter 6.1). As always if you have questions, feel free to e-mail feature-users@simtk.org.

Chapter 4. Programs

FEATURE is made of three programs, each with a specific function.

- `featurize` Characterizes the microenvironments of a protein for a given set of Cartesian points near or within the protein
- `buildmodel` Builds a model which can be used to identify similar active sites in unknown proteins
- `scoreit` Shows log probability scores that a point of the protein is similar to the model

Each of the above sections describes a program in detail. It includes information about what files you will need for input, what is generated as output, an example on how to run the program on the command-line, and a diagram on the underlying algorithm for the program.

These programs use a number of different files as input and output. These files are explained in Files and Extensions (Chapter 5).

4.1. `featurize`

Generates feature vectors at given Cartesian points in or near one or more proteins

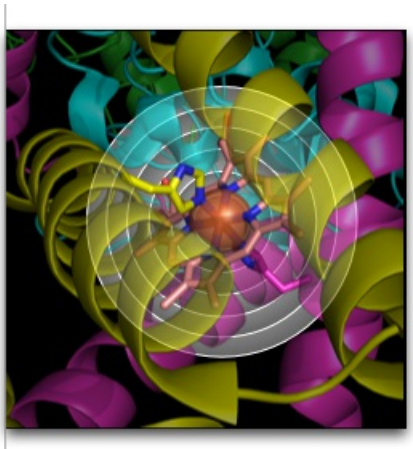


Figure A.
Conceptual diagram of a microenvironment, showing the concentric shells around a metal ion ligand.

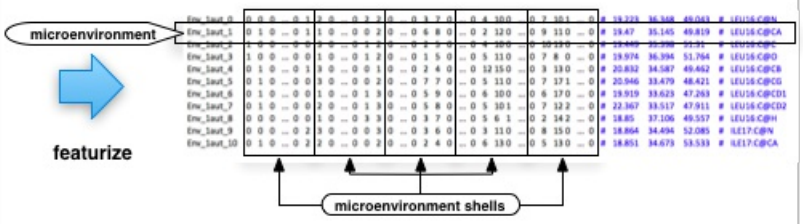


Figure B.
Computational representation of a microenvironment. The number of columns per shell have been abbreviated.
Feature Vector files look exactly like this.

The *featurize* program takes points in a biomolecule to build microenvironments (Figure A) and produce computational representations of said microenvironments (Figure B).

The *featurize* program analyzes mutually exclusive concentric spherical volumes (called shells) around a given point in a biomolecule. These shells collectively describe a microenvironment. The *featurize* program tallies physicochemical properties for each atom contained in each shell. These tallies form the computational representation describing the microenvironment. Groups of microenvironment computational representations are called Feature Vectors and are stored in Feature Vector Files.

Machine Learning can be trained on Feature Vectors to produce biomolecule functional class models. Biomolecules of unknown function can be characterized as Feature Vectors and scored against functional class models to predict functionality.

4.2. `featurize` Input

- An optional Point File and
- One or more PDB files and their related DSSP files. A PDB file contains protein structure and function data. A DSSP file contains protein secondary structure data; it is derived from a PDB file.

You have three options for specifying the PDB and related DSSP files:

1. Provide one or more PDB IDs as a command-line argument.
   ```
   $ featurize 1bqy [...]
   ```

2. Specify one or more PDB IDs in a text file, using the `-f` option.
   ```
   $ featurize -f pdb_files.txt
   ```

3. Specify a Point File wth the `-P` option. A point file specifies a set of points for one or more PDB IDs.
   ```
   $ featurize -P sites.ptf
   ```

`featurize` will look for the PDB and DSSP files in three locations: the current working directory, and the paths specified by the environment variables `PDB_DIR` and `DSSP_DIR`. These environment variables are set when FEATURE is installed; to change them, see Environment Variables

4.3. `featurize` Output

- Prints a feature vector

4.4. `featurize` Usage

```
Usage: featurize [OPTIONS] [PDBID...]
Must exclude .ent or other filename extension in PDBID
Options:
-v Increase verbosity
-n SHELLS [Default: 6]
Set number of shells to NUMSHELLS
-w SHELL_WIDTH [Default: 1.25]
Set thickness of each shell to SHELL_WIDTH Angstroms
```

-x EXCLUDED_RESIDUES [Default: HETATM]
Set residues to exclude to comma separated list of
EXCLUDED_RESIDUES
-f PDBID_FILE
Read PDBIDs from PDBID_FILE
-P POINT_FILE
Read point list from POINT_FILE
-l SELECTABLE_PROPERTIES_FILE
Read a property list from SELECTABLE_PROPERTIES_FILE

4.4.1. `featurize` Options Detail

The verbosity option (`-v`) increases the number of output messages, some of which are useful for understanding how `featurize` works and for debugging.

The `NUMSHELLS` variable controls the number of radial shells `featurize` will evaluate. Larger values will create more data and take longer to compute; however the data may be noisy. The default value is strongly recommended.

The `SHELLWIDTH` variable controls the size of the shells that `featurize` will evaluate. Again, larger values will create more data, but may contribute noise. The default value is strongly recommended.

The `EXCLUDERESIDUES` option excludes one type of PDB-specified atoms. For a list of PDB-specified atom types, see PDB files. The default value is strongly recommended.

`PDBFILE` is either a filename or `-` (standard input). If the user provides a filename, `featurize` will read each PDB ID listed in the file and produce a feature vector for each corresponding protein at each atom location.

If the user specifies standard input then the user may type in PDB IDs and type `Control-D` to begin analyzing the specified proteins. Both methods are useful for producing feature vectors for unknown proteins to be classified.

Optionally the user can use `featurize` with one or more PDBID parameters. This causes `featurize` to produce a feature vector for each corresponding protein at each atom location.

`POINTFILE` is a filename of a Point file. This option is useful for creating a feature vector for sites and non-sites. The point file for similar sites can be manually selected or programatically extracted from databases such as ProSite. The datamining approach is left to the user; FEATURE does not provide this service.

`SELECTABLE_PROPERTIES_FILE` is a filename of a Selectable Properties file. As of FEATURE 3.0, a user can specify which properties they want to use for their analysis. Two property sets are provided with FEATURE 3.0: `proteins.properties` and `metals.properties`. The former file has the standard 80 properties for FEATURE. The latter file has 44 properties used in another version of FEATURE, called FEATURE for metal ion Ligands. Users are encouraged to copy these files or even concatenate and edit them to build a customized list of properties. See Physicochemical Properties used by FEATURE (Chapter 6.1) properties for more information about the properties themselves.

4.5. `featurize` Environment Variables

The `PDB_DIR` and `DSSP_DIR` environment variables control where `featurize` will look for pdb and DSSP files. You can change them as follows:

- Mac OS X:
  ```
  $ sudo open /etc/bashrc
  $ . /etc/bashrc
  ```
- Linux:
  ```
  $ sudo vim /etc/bashrc
  $ . /etc/bashrc
  ```
- Windows:
  Start -> Settings -> Control Panel -> System -> Advanced -> Environment Variables
  Find `PDB_DIR` and `DSSP_DIR` under `System variables`; click on them and then click on `Edit`; change the path and click `OK`.

4.5.1. `featurize` Examples

`featurize` can be used in several ways: To characterize the microenvironments of similar (or non-similar) active sites across several proteins

featurize -P trypsin_ser_og.pos.ptf > trypsin_ser_og.pos.ff

To characterize the microenvironments at all points within a protein

featurize 1bqy > 1bqy.ff

To characterize the microenvironments for selected points within a protein using the atomselector.py utility. In this case, the gamma oxygens for all serine residues in the structure 1bqy.

```
atomselector.py -r SER -a OG 1bqy > 1bqy_ser_og.ptf
featurize -P 1bqy_ser_og.ptf > 1bqy_ser_og.ff
```

4.6. `featurize` Selectable Properties

Users can select which physicochemical properties to use for microenvironment characterization. This is done by providing `featurize` with a Selectable Properties File

4.7. `buildmodel`

Creates a Naive Bayesian classifier model from a set of positive and negative training data.

4.8. `buildmodel` Input

- A feature vector file of known similar sites
- A feature vector file of known non-similar sites

A feature vector file is produced by featurize and summarizes the physicochemical properties around one or more points in or near a biomolecule.

4.9. `buildmodel` Output

- A model file that contains information that can be used for protein function classification.

4.10. `buildmodel` Usage

```
Usage: buildmodel [OPTIONS] SITE_FEATURE_VECTOR_FILE NONSITE_FEATURE_VECTOR_FILE

Options:
    -v  Increase verbosity
    -n NUMSHELLS [Default: 6]
        Set number of shells to NUMSHELLS for environment mode
    -b NUMBINS [Default: 5]
        Set number of bins
    -s SITEPRIOR [Default: 0.01]
        Set the site prior
    -B BOUNDSFILE
        Specify the min/max in shell major order
    -x
        Increase extended output
    -l PROPERTIESFILE
  Read a property list from PROPERTIESFILE
```

4.10.1. `buildmodel` Options Detail

The `SITEFEATURES` variable is the filename for a Feature Vector File

The `NONSITEFEATURES` variable is the filename for a Feature Vector File

The verbosity option (`-v`) increases the number of output messages, some of which are useful for understanding how `buildmodel` works and for debugging.

The `NUMSHELLS` variable controls number of shells `featurize` will evaluate. Larger values will create more data and take longer to compute; however the data may be noisy. The default value is strongly recommended.

The `NUMBINS` variable controls the number of discrete probability states to use for the purposes of calculating the Bayesian probability.

The `SITEPRIOR` variable controls the prior probability for the Bayesian classification model for positive site classification.

The `BOUNDSFILE` is a text file that specifies the min/max for each bin for each shell to calculate the Bayesian probability. The default behavior is to linearly discretize the property values into `NUMBINS` bins.

For more details about the theory behind the Bayesian classification model We recommend *Artificial Intelligence: A Modern Approach*, by Russell and Norvig, Prentice Hall Press ©2003. Chapters 13 and 14 as a suitable introductory text for advanced undergraduates and graduate students.

4.11. `buildmodel` Examples

buildmodel model.pos.ff model.neg.ff > SNAKE_TOXIN.2.CYS.SG.model
buildmodel trypsin_ser_og.pos.ff trypsin_ser_og.neg.ff > trypsin_ser_og.model

4.12. `scoreit`

Scores a set of given points in an unknown protein for similarity to a given active site model

4.13. `scoreit` Input

- A model file that contains a protein function model, as produced by buildmodel
- A feature vector file that describes one or more points in or near an unknown protein, as produced by featurize

4.14. `scoreit` Output

- A score file that contains a probability score for each point described by the feature vector file that the given point may have the same function as described by the model

4.15. `scoreit` Usage

```
Usage: scoreit [OPTIONS] MODEL_FILE FEATURE_VECTOR_FILE

Options:
    -v  Increase verbosity
    -n NUMSHELLS [Default: 6]
        Set number of shells
    -p PLEVEL [Default: 0.01]
        Set minimum acceptable p-Value; ignore contributions from scores with lower p-Values
    -c CUTOFF [Default: 0]
        Set minimum score to print out
    -l PROPERTIESFILE
        Read a property list from PROPERTIESFILE
    -a  Shows all scores
    -e  Toggle count empty shells [Default: 1]
```

4.15.1. `scoreit` Options Detail

The verbosity option (`-v`) increases the number of output messages, some of which are useful for understanding how `scoreit` works and for debugging.

The `NUMSHELLS` variable controls number of shells `scoreit` will evaluate. This variable should be consistent with the number of shells used to build the model and generate the feature vectors. The default value is strongly recommended.

The `PLEVEL` variable controls the maximum value for the Mann-Whitney-Wilcoxon (MWW) Rank Sum rho score that will be considered. A MWW Rank Sum rho score of 0.5 indicates that there is no difference between the distributions between the positive and negative training data for a given property. Therefore the property is likely to be unimportant. A MWW Rank Sum rho score that approaches zero indicates that the property distribution has strong correlation to a classification. The default value is strongly recommended.

The `CUTOFF` variable controls the minimum score to print out; because scores are relative, the `-a` option is recommended.

The `-e` option directs `scoreit` to ignore empty shells by default. The default value is recommended.

4.16. `scoreit` Examples

The following example will show only significant hits (positive scores) for the protein structure 2zpb against the KAZAL.3.CYS.SG model.

featurize 2zpb > 2zpb.ff
scoreit KAZAL.3.CYS.SG.model 2zpb.ff > 2zpb.hits

The following example will show all significant hits (all scores) for a subset of the protein structure 1bqy (namely the gamma oxygens for all serine residues) against the trypsin_ser_og model.

atomselector.py -r SER -a OG 1bqy > 1bqy_ser_og.ptf
featurize -P 1bqy_ser_og.ptf > 1bqy_ser_og.ff
scoreit -a trypsin_ser_og.model 1bqy_ser_og.ff > 1bqy_ser_og.hits

Chapter 5. Files and Extensions

FEATURE uses a variety of files for input and output; here are a few commonly used extensions and their corresponding file types.

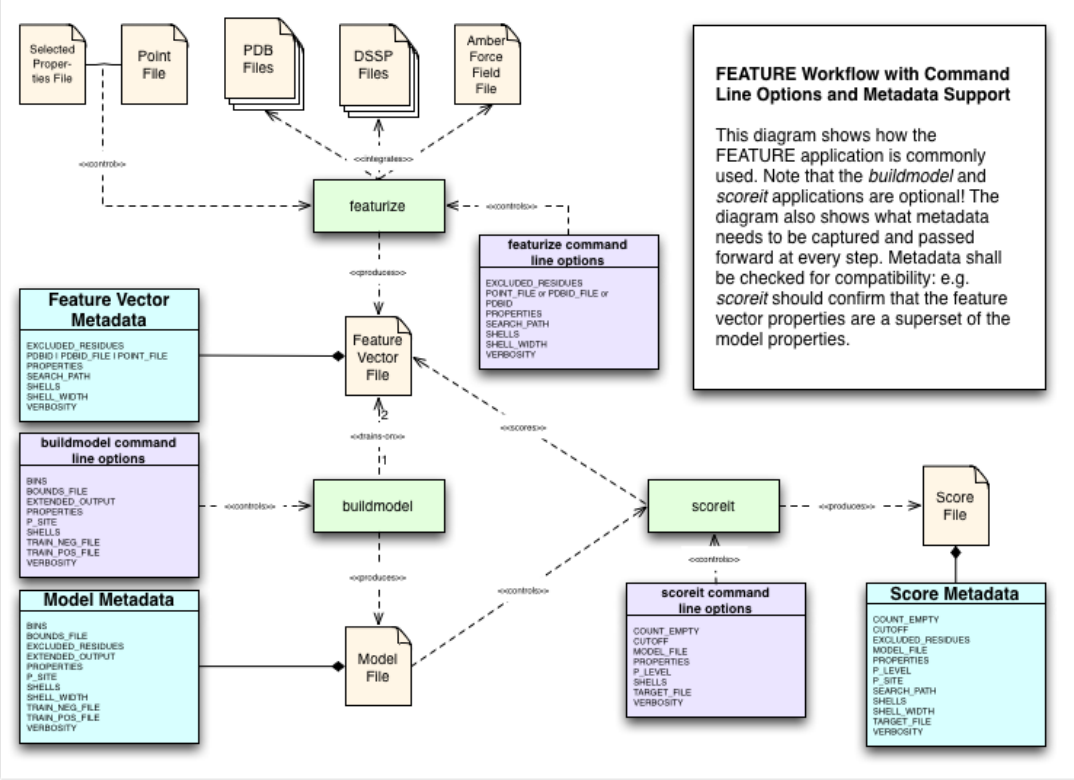
File Extension	File Type
`.brk`	Alias for PDB File
`.dat`	AMBER Files
`.dssp`	DSSP File
`.ent`	Alias for PDB File
`.ff`	Feature Vector File
`.full`	Alias for PDB File
`.hits`	Score File
`.model`	Model File
`.pdb`	PDB File
`.properties`	Selectable Properties File
`.ptf`	Point File

5.1. File Metadata

FEATURE 3.1 annotates each file with the command line options, as provided by the user. For example, the `featurize` program allows users to specify the number of shells to use for the microenvironments, the width of each shell, and other options. FEATURE 3.1 tracks this information and stores it in the resulting feature vector file as *metadata* which is shown below as the lines starting with a hash mark (#). The main file contents follow the metadata.

```
#       EXCLUDED_RESIDUES       HETATM
#       NUM_PROPERTIES  80
#       POINT_FILE      tests/data/metal-models/2FE2S_FERREDOXIN.1.CYS.SG/sites.ptf
#       PROPERTIES      ALIPHATIC_CARBON, AROMATIC_CARBON, PARTIAL_POSITIVE_CARBON,
#       PROPERTIES      ALIPHATIC_CARBON_NEXT_TO_POLAR, AMIDE_CARBON, CARBOXYL_CARBON,
#       PROPERTIES      AMIDE_NITROGEN, POSITIVE_NITROGEN, AROMATIC_NITROGEN,
#       PROPERTIES      AMIDE_OXYGEN, CARBOXYL_OXYGEN, HYDROXYL_OXYGEN, SULFUR,
#       PROPERTIES      PARTIAL_CHARGE, VDW_VOLUME, CHARGE, NEG_CHARGE, POS_CHARGE,
#       PROPERTIES      CHARGE_WITH_HIS, HYDROPHOBICITY, SOLVENT_ACCESSIBILITY,
#       PROPERTIES      ELEMENT_IS_ANY, RESIDUE_NAME_IS_ALA, RESIDUE_NAME_IS_ARG,
#       PROPERTIES      RESIDUE_NAME_IS_ASN, RESIDUE_NAME_IS_ASP, RESIDUE_NAME_IS_CYS,
#       PROPERTIES      RESIDUE_NAME_IS_GLN, RESIDUE_NAME_IS_GLU, RESIDUE_NAME_IS_GLY,
#       PROPERTIES      RESIDUE_NAME_IS_HIS, RESIDUE_NAME_IS_ILE, RESIDUE_NAME_IS_LEU,
#       PROPERTIES      RESIDUE_NAME_IS_LYS, RESIDUE_NAME_IS_MET, RESIDUE_NAME_IS_PHE,
#       PROPERTIES      RESIDUE_NAME_IS_PRO, RESIDUE_NAME_IS_SER, RESIDUE_NAME_IS_THR,
#       PROPERTIES      RESIDUE_NAME_IS_TRP, RESIDUE_NAME_IS_TYR, RESIDUE_NAME_IS_VAL,
#       PROPERTIES      RESIDUE_NAME_IS_HOH, RESIDUE_NAME_IS_OTHER
#       PROPERTIES_FILE metals.properties
#       SHELLS  6
#       SHELL_WIDTH     1.25
#       VERBOSITY       0
Env_1czp_0      0       ...     0       0       0       ...     #       -9.363  18.396  3.46    #       CYS41:A@SG
Env_1doi_0      0       ...     0       0       4       ...     #       3.838   7.404   19.296  #       CYS63:@SG
Env_1fo4_0      0       ...     4       0       0       ...     #       42.551  45.535  25.788  #       CYS43:A@SG
Env_1jq4_0      0       ...     3       0       0       ...     #       -2.825  10.276  11.889  #       CYS42:A@SG
Env_1kf6_0      0       ...     2       0       2       ...     #       9.057   28.643  -27.437 #       CYS57:B@SG
Env_1krh_0      0       ...     2       0       0       ...     #       92.647  74.071  131.301 #       CYS41:A@SG
Env_1o8r_0      0       ...     3       0       3       ...     #       4.551   -13.628 -0.188  #       CYS86:A@SG
Env_1qla_0      0       ...     2       0       1       ...     #       42.164  -14.427 4.869   #       CYS57:B@SG
Env_2pia_0      0       ...     3       0       0       ...     #       6.622   26.803  8.897   #       CYS272:@SG
```

In the above example, you can see the following metadata keywords: `EXCLUDED_RESIDUES`, `NUM_PROPERTIES`, `POINT_FILE`, `PROPERTIES`, `PROPERTIES_FILE`, `SHELLS`, `SHELL_WIDTH`, and `VERBOSITY`.



FEATURE 3.1 supports both command line options and metadata to modify the behavior at each step. For example, you can specify a selectable properties file to specify the properties to be used for `featurize`, `buildmodel`, and/or `scoreit`. `buildmodel` and `scoreit` also reads the metadata from their respective input files and uses that to specify the properties to use.

It's possible for a user to specify different option values than what's in the metadata. This may cause `buildmodel` or `scoreit` to exit with an error and likely give bad or no output. For example, let's say we wish to train a model with feature vectors made with 80 properties and then tell `buildmodel` to only use 40 properties. `buildmodel` will exit with an error. If `buildmodel` or `scoreit` give no errors, then the output will be correct. The bottom line is that you should be consistent with command line options or let the metadata do its job and specify the options for you.

5.2. PDB File

A Protein Data Bank (PDB) file contains a standardized representation for macromolecular structure data, such as sequence and 3-D locations for each atom, typically gathered through techniques such as: NMR, electron microscopy, and X-ray crystallography.

Commonly has one of the following extensions : `.pdb`, `.ent`, `.full`, or `.brk`.

- The Protein Database Website: http://www.wwpdb.org
- PDB Format Definition: http://www.wwpdb.org/docs.html

5.3. DSSP File

A Dictionary of Secondary Structure for Proteins (DSSP) file contains secondary structure information extracted from PDB files. This information is not available directly from the PDB files and are required for some properties (Chapter 6.1).

DSSP is not our work. The DSSP program was designed by Wolfgang Kabsch and Chris Sander to standardize secondary structure assignment. For more information, please visit the official DSSP website http://swift.cmbi.ru.nl/gv/dssp/

The `featurize` program uses DSSP files and PDB files as inputs to create a FEATURE vector file. One can create DSSP files from PDB files using the `dsspcmbi` program as follows:

```
$ dsspcmbi <PDB Filename> <DSSP Filename>
```

For example:

```
$ dsspcmbi pdb1bqy.ent pdb1bqy.dssp
```

5.4. Downloading and Installing **`dsspcmbi`**

Mac OS X:

1. Download the source code here: ftp://ftp.cmbi.ru.nl/pub/molbio/software/dsspcmbi.zip
2. Type the following commands from a command-prompt:

   ```
   $ cd ~/Downloads
   $ unzip dsspcmbi.zip
   $ cd dssp
   $ open .
   ```

3. Edit the second-to-last line of the `DsspCompileGCC` script. You will remove the `-static` flag so that the second-to-last line looks like this:

   ```
   gcc -o dsspcmbi ./DsspCMBI.o ./p2clib.o ./Date.o ./Vector.o ./AccSurf.o ./CalcAccSurf.o ./Contacts.o -lm
   ```

4. Run the `DsspCompileGCC` script:

   ```
   $ ./DsspCompileGCC
   $ sudo mv dsspcmbi /usr/local/bin
   ```

Linux: Ask your system administrator to download and install the binary executable, available here:
http://swift.cmbi.ru.nl/gv/dssp/HTML/dsspcmbi

5.5. AMBER

AMBER refers to a molecular dynamics software package (http://ambermd.org) or a set of force field parameters (http://ambermd.org/#ff) to approximate ab initio electronic structure simulation. FEATURE uses the force field parameters to provide *atom type* information (related to valence hybridization state) and partial charge. FEATURE uses the Cornell et al. (1994) force fields; these files are distributed with FEATURE. If you want to use different force field files, then you'll need to modify the AMBER files to match the format of the force field files distributed with FEATURE.

5.6. AMBER Force Field Files

The AMBER Force Field file provided with FEATURE 3.1 is named `amberM2_params.dat`.

The AMBER Force Field files have a simple format: the *set* keyword describes the relative position of the residue in the peptide; all other entries are simply *residue name*, *atom alias*, the *type* keyword, the *type value*, the *charge* keyword, followed by the *charge value*. The atom alias follows a different nomenclature than PDB atom names. The Residue Template file (described further below) provides the mapping between PDB atom names and AMBER atom aliases.

```
set: normal
     ACE C type: C charge: 0.5972
     ACE CH3 type: CT charge: -0.3662
     ACE O type: O charge: -0.5679
     ...
set: n_terminal
     ALA C type: C charge: 0.6163
     ALA CA type: CT charge: 0.0962
     ALA CB type: CT charge: -0.0597
     ...
set: c_terminal
     ALA C type: C charge: 0.7731
     ALA CA type: CT charge: -0.1747
     ALA CB type: CT charge: -0.2093
     ...
```

5.7. AMBER Residue Template Files

The AMBER Residue Template file provided with FEATURE 3.1 is named `residue_templates.dat`.

AMBER Force Field files use different atom nomenclature than PDB files. The Residue Template provides a mapping to match the AMBER Force Field atom aliases to the PDB atom names.

```
set: normal
     residue: ACE
         aliases:
             1HA 2HH3
             2HA 3HH3
             3HA 1HH3
             CA CH3
             ...
     residue: ALA
         aliases:
             HA2 HA
             HB1 1HB
             HB2 2HB
             ...
     residue: ARG
         aliases:
             1HB 3HB
             1HD 3HD
             ...
```

5.8. Selectable Properties File

A FEATURE file that contains a list of properties to use when characterizing microenvironments. As of FEATURE 3.0, users can now select the physicochemical properties they want to use for their analyses. This file is given to the `featurize` program with the command line option `-l` as follows:

```
$ featurize -l myprops.properties -P mypointfile.ptf
```

5.9. Selectable Properties File Properties Templates

The FEATURE 3.1 release comes with three Selectable Properties Files: `proteins.properties`, `metals.properties`, and `default.properties`. The `default.properties` file contains the default properties, which are the same as the `proteins.properties` file. This file contains the standard 80 properties that were in FEATURE 2.0 and FEATURE 1.9. The `metals.properties` file contains the properties used in FEATURE for Metal Ion Ligands, a separate release of FEATURE in 2008 (see References (Chapter 8), *Robust recognition of zinc binding sites in proteins.* Ebert JC, Altman RB.) You are encouraged to copy these files and modify the copies. Modifying the original `proteins.properties`, `metals.properties` or `default.properties` is strongly discouraged.

5.10. Selectable Properties File Example of a Selectable Properties File

A user can disable properties by placing a hash mark (#) in front of the line. This "comments-out" the property, disabling it for FEATURE. Alternatively, if the property name is deleted from the file, the property will not be enabled. In this example, several properties (e.g. `ATOM_TYPE_IS_N`, `ATOM_TYPE_IS_N2`, `ATOM_TYPE_IS_N3`, etc.) are enabled and `ATOM_TYPE_IS_C` is disabled. Some text editors (e.g. vim and gvim) will highlight the text in blue when the hash mark (#) is placed in the beginning of a line.

```
# ============================================================================
# SELECTABLE PROPERTIES FILE
# ============================================================================
# This is a system-level file. To customize your own selectable properties,
# copy this file to your directory.
#
# To indicate that you don't want to use a property, add a hash mark (#) in
# front of the property name
#
# Example:
# # ATOM_TYPE_IS_XYZ
# ============================================================================

# ATOM_TYPE_IS_C
# ATOM_TYPE_IS_CT
# ATOM_TYPE_IS_CA
  ATOM_TYPE_IS_N
  ATOM_TYPE_IS_N2
  ATOM_TYPE_IS_N3
  ATOM_TYPE_IS_NA
```

FEATURE will warn you if you enable a property that is not currently defined. For a complete list of currently defined properties, see Physicochemical Properties used by FEATURE . Also be aware that it is possible to accidentally mix-and-match properties between analyses. The user is strongly cautioned to create a Selectable Properties File and keep the file and all experiments that use the file in the same directory.

5.11. Point File

A FEATURE file format which describes the location of one or more Cartesian points in or near a protein. The coordinates use the same scale as PDB files, which is typically in Angstroms. Point files are conventionally named with the `.ptf` file extension (Chapter 5).

5.12. Point File Creating A Point File

In the case of known similar active sites, simply record the PDB ID and the 3D location of the active site of interest for each similar active site to manually create a point file. A point file can also be created from parsed information gathered from PDB files of functionally similar proteins.

Point files for known dissimilar active sites can be randomly sampled from the output of atomselector.py on PDB files of known functionally dissimilar proteins.

5.13. Point File Point File Format

```
<PDB ID> <x> <y> <z> [site|nonsite|#] <Atom information comment>
```

Coordinates are measured in Angstroms. The fifth column always has the text: `site`, `nonsite`, or `#` (which indicates a comment).

5.14. Point File Example Point File

```
1bqy    11.959  25.312  62.893  #   SER29:A@OG
1bqy    27.066  39.783  78.315  #   SER36:A@OG
1bqy    28.712  39.196  69.556  #   SER60:A@OG
1bqy    5.519   37.209  75.787  #   SER72:A@OG
1bqy    23.814  19.283  76.686  #   SER110:A@OG
1bqy    23.131  16.134  72.115  #   SER111:A@OG
1bqy    16.871  15.681  74.194  #   SER113:A@OG
1bqy    10.131  21.339  74.293  #   SER115:A@OG
1bqy    12.539  18.561  56.672  #   SER122:A@OG
1bqy    14.362  15.956  50.415  #   SER125:A@OG
```

5.15. Feature Vector File

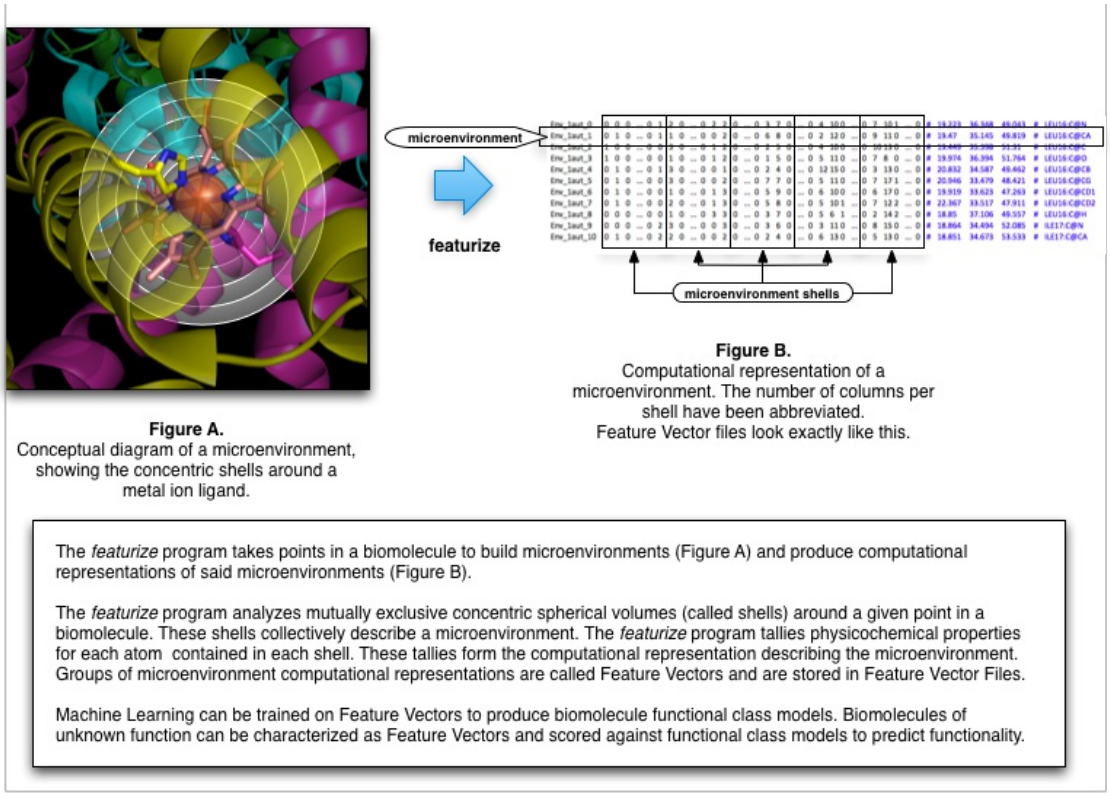


Figure A.
Conceptual diagram of a microenvironment, showing the concentric shells around a metal ion ligand.

Figure B.
Computational representation of a microenvironment. The number of columns per shell have been abbreviated.
Feature Vector files look exactly like this.

A feature vector file contains a summary of the physicochemical properties around one or more points in or near a biomoelcule.

Feature vector files are the output of the featurize program that has been saved to a file. Feature vector files are conventionally named with the `.ff` file extension. Each line of a feature vector file contains a description of the microenvironment around one point; the descriptor is called a *feature vector*.

They are named for either:

- The PDB ID and function of the featurized protein, e.g. `1bqy` becomes `1bqy_ser_og.ff`
- A substrate that the featurized active site acts on or binds to, followed by .pos or .neg for the classification model e.g. `trypsin_ser_og.pos.ff`

5.16. Example Feature Vector

```
Env_1aut_0  0  0  1  0  -0.227  ...  #   11.882  35.097   41.982  site  None
Env_1azz_0  0  0  1  0  -0.227  ...  #  -19.266  66.982  100.441  site  None
Env_1bio_0  0  0  1  0  -0.227  ...  #    7.97    5.368    1.508  site  None
Env_1cgh_0  0  0  1  0  -0.227  ...  #   13.988  35.933    5.966  site  None
Env_1dle_0  0  0  1  0  -0.227  ...  #   -3.778   9.595    6.561  site  None
Env_1eax_0  0  0  1  0  -0.227  ...  #   16.734  61.385   20.44   site  None
```

5.17. Feature Vector File Format

```
<environment> <prop1> <prop2> ... # <x> <y> <z> <site|nonsite|#> <Atom information comment>
```

`environment` is composed of three parts, the text `Env`, the PDB ID, and a unique site index, each separated by underscores.

`prop#` is the value for the given point, protein, and property. Properties can be derived from any information in the PDB File or DSSP File files (e.g. `PARTIAL_CHARGE` sums the partial charges within the given shell volume). The current version of FEATURE analyzes 80 properties.

Coordinates are measured in Angstroms. The column after the hash mark and coordinates always has the text: `site`, `nonsite`, or `#` (which indicates a comment).

5.18. Model File

A model file contains information that can be used for protein function classification.

It is a FEATURE file format that describes the Bayesian posterior probability of a positive site classification for each property. A model file is created by redirecting the output of buildmodel to a file. Models are stored as a tab-delimited format. Model Files use the .model extension.

5.19. Example Model File

Using tab-delimited text format

```
#       NUM_BINS        5
#       P_SITE  0.01
#       NUM_PROPERTIES  80
#       NUM_SHELLS      6
ATOM_TYPE_IS_C-0        0.5     -1e-05  2e-06   0       0       0       0       0
ATOM_TYPE_IS_CT-0       0.5     -1e-05  2e-06   0       0       0       0       0
ATOM_TYPE_IS_Ca-0       0.5     -1e-05  2e-06   0       0       0       0       0
ATOM_TYPE_IS_N-0        0.5     -1e-05  2e-06   0       0       0       0       0
ATOM_TYPE_IS_N2-0       0.5     -1e-05  2e-06   0       0       0       0       0
ATOM_TYPE_IS_N3-0       0.5     -1e-05  2e-06   0       0       0       0       0
ATOM_TYPE_IS_Na-0       0.5     -1e-05  2e-06   0       0       0       0       0
...
```

5.20. Model File Format

Reading a model file requires knowledge of Naive Bayesian Networks, which is beyond the scope of this guide. There are plenty of good texts and papers which describe Bayesian Networks; a quick Internet search should reveal a number of good reads.

```
<property> <p value> <minimum> <range> <bin1 score> <bin2 score> <bin3 score> ...
```

`property` is the property name.

`ρ value` is the ρ value from the Mann-Whitney Rank Sum test, evaluating the correlation between property values and positive or negative classification. Essentially, this measures how important the property is to the model.

`minimum` is the minimum value for the given property for all training data.

`range` is the range (maximum value - minimum value) for the given property for all training data.

`bin1 score` is the log probability for the values that fall within the minimum and maximum ranges for the first bin.

`bin2 score` is the log probability for the values that fall within the minimum and maximum ranges for the second bin.

5.21. Score File

A FEATURE file format that contains a probability score for a given point to match a given modeled functionality.

Score files are created by redirecting the output of scoreit to a file. The `scoreit` program takes a Point File and a Model File and assigns each point a probability score for matching the model. The name of score files often end with `.hits`.

The columns are very similar to the Point File format, with the addition of the 2nd column, which is the log probability score. Higher scores mean that the point in question is more likely to be relevant to the model, and therefore more likely that the given point for the given protein is an active site with the same functionality as that of the model. In the example below, `Env_1bqy_0` is more relevant than `Env_1bqy_1` to the modeled functionality (serine OG protease).

5.22. Example Score File

```
Env_1bqy_0      -117.320526     11.959  25.312  62.893  #       SER29:A@OG
Env_1bqy_1      -191.381713     27.066  39.783  78.315  #       SER36:A@OG
Env_1bqy_2      -164.580861     28.712  39.196  69.556  #       SER60:A@OG
Env_1bqy_3      -250.891138     5.519   37.209  75.787  #       SER72:A@OG
Env_1bqy_4      -214.470383     23.814  19.283  76.686  #       SER110:A@OG
Env_1bqy_5      -201.523516     23.131  16.134  72.115  #       SER111:A@OG
Env_1bqy_6      -210.394876     16.871  15.681  74.194  #       SER113:A@OG
```

5.23. Score File Score File Format

```
<environment> <score> <x> <y> <z> <site|nonsite|#> <Atom information comment>
```

`environment` is composed of three parts, the text `Env`, the PDB ID, and a unique site index, each separated by underscores.

`score` is proportional to the likelihood of a positive classification at the given coordinates. Scores can be positive or negative. A larger value indicates a more likely probability that the given coordinates is a positive match with the model.

Coordinates are measured in Angstroms. The column after the coordinates always has the text: `site`, `nonsite`, or `#` (which indicates a comment).

6.1. Physicochemical Properties used by FEATURE

ALIPHATIC_CARBON	ALIPHATIC_CARBON_NEXT_TO_POLAR	AMIDE
AMIDE_CARBON	AMIDE_NITROGEN	AMIDE_OXYGEN
AMINE	AROMATIC_CARBON	AROMATIC_NITROGEN
ATOM_TYPE_IS_C	ATOM_TYPE_IS_CA	ATOM_TYPE_IS_CT
ATOM_TYPE_IS_N	ATOM_TYPE_IS_N2	ATOM_TYPE_IS_N3
ATOM_TYPE_IS_NA	ATOM_TYPE_IS_O	ATOM_TYPE_IS_O2
ATOM_TYPE_IS_OH	ATOM_TYPE_IS_OTHER	ATOM_TYPE_IS_S
ATOM_TYPE_IS_SH	CARBONYL	CARBOXYL_CARBON
CARBOXYL_OXYGEN	CHARGE	CHARGE_WITH_HIS
ELEMENT_IS_ANY	ELEMENT_IS_C	ELEMENT_IS_N
ELEMENT_IS_O	ELEMENT_IS_OTHER	ELEMENT_IS_S
HYDROPHOBICITY	HYDROXYL	HYDROXYL_OXYGEN
MOBILITY	NEG_CHARGE	PARTIAL_CHARGE
PARTIAL_POSITIVE_CARBON	PEPTIDE	POSITIVE_NITROGEN
POS_CHARGE	RESIDUE_CLASS1_IS_CHARGED	RESIDUE_CLASS1_IS_HYDROPHOBIC
RESIDUE_CLASS1_IS_POLAR	RESIDUE_CLASS1_IS_UNKNOWN	RESIDUE_CLASS2_IS_ACIDIC
RESIDUE_CLASS2_IS_BASIC	RESIDUE_CLASS2_IS_NONPOLAR	RESIDUE_CLASS2_IS_POLAR
RESIDUE_CLASS2_IS_UNKNOWN	RESIDUE_NAME_IS_ALA	RESIDUE_NAME_IS_ARG
RESIDUE_NAME_IS_ASN	RESIDUE_NAME_IS_ASP	RESIDUE_NAME_IS_CYS
RESIDUE_NAME_IS_GLN	RESIDUE_NAME_IS_GLU	RESIDUE_NAME_IS_GLY
RESIDUE_NAME_IS_HIS	RESIDUE_NAME_IS_HOH	RESIDUE_NAME_IS_ILE
RESIDUE_NAME_IS_LEU	RESIDUE_NAME_IS_LYS	RESIDUE_NAME_IS_MET
RESIDUE_NAME_IS_OTHER	RESIDUE_NAME_IS_PHE	RESIDUE_NAME_IS_PRO
RESIDUE_NAME_IS_SER	RESIDUE_NAME_IS_THR	RESIDUE_NAME_IS_TRP
RESIDUE_NAME_IS_TYR	RESIDUE_NAME_IS_VAL	RING_SYSTEM
SECONDARY_STRUCTURE1_IS_3HELIX	SECONDARY_STRUCTURE1_IS_4HELIX	SECONDARY_STRUCTURE1_IS_5HELIX
SECONDARY_STRUCTURE1_IS_BEND	SECONDARY_STRUCTURE1_IS_BRIDGE	SECONDARY_STRUCTURE1_IS_COIL
SECONDARY_STRUCTURE1_IS_HET	SECONDARY_STRUCTURE1_IS_STRAND	SECONDARY_STRUCTURE1_IS_TURN
SECONDARY_STRUCTURE1_IS_UNKNOWN	SECONDARY_STRUCTURE2_IS_BETA	SECONDARY_STRUCTURE2_IS_COIL
SECONDARY_STRUCTURE2_IS_HELIX	SECONDARY_STRUCTURE2_IS_HET	SECONDARY_STRUCTURE2_IS_UNKNOWN
SOLVENT_ACCESSIBILITY	SULFUR	VDW_VOLUME

6.1.1. Properties Description

All properties are initialized to 0 by default.

ALIPHATIC_CARBON

Source: PDB File
Divider: 1.0

Set to 1 if the atom is an aliphatic carbon.

ALIPHATIC_CARBON_NEXT_TO_POLAR

Source: PDB File
Divider: 1.0

Set to 1 if atom is an aliphatic carbon next to a polar atom (O or N).

AMIDE

Source: PDB File
Divider: 100.0

Set to 1 if the atom is an N in PRO, ASN, or GLN. Set to 0.5 if the atom is ambiguously N in ASN, GLN or N in HIS or ARG. Set to 0.25 if the atom is ambiguously N in HIS.

AMIDE_CARBON

Source: PDB File
Divider: 1.0

Set to 1 if the atom is a carbon atom in an amide group.

AMIDE_NITROGEN

Source: PDB File
Divider: 1.0

Set to 1 if the atom is a nitrogen atom in an amide group.

AMIDE_OXYGEN

Source: PDB File
Divider: 1.0

Set to 1 if the atom is a oxygen atom in an amide group.

AMINE

Source: PDB File
Divider: 100.0

Set to 1 if the atom is ARG Nε, LYS Nζ, or TRP Nε. Set to 0.5 if the atom is an ARG Nη, HIS Nδ, or HIS Nε. Set to 0.25 if the atom is HIS Aδ or HIS Aε.

AROMATIC_CARBON

Source: PDB File
Divider: 1.0

Set to 1 if the atom is an aromatic carbon.

AROMATIC_NITROGEN

Source: PDB File
Divider: 1.0

Set to 1 if the atom is an aromatic nitrogen.

ATOM_TYPE_IS_C

Source: AMBER Force Field File
Divider: 1.0

Set to 1 if the atom force field type is C.

ATOM_TYPE_IS_CA

Source: AMBER Force Field File
Divider: 1.0

Set to 1 if the atom force field type is Ca.

ATOM_TYPE_IS_CT

Source: AMBER Force Field File
Divider: 1.0

Set to 1 if the atom force field type is Ct.

ATOM_TYPE_IS_N

Source: AMBER Force Field File
Divider: 1.0

Set to 1 if the atom force field type is N.

ATOM_TYPE_IS_N2

Source: AMBER Force Field File
Divider: 1.0

Set to 1 if the atom force field type is N2.

ATOM_TYPE_IS_N3

Source: AMBER Force Field File
Divider: 1.0

Set to 1 if the atom force field type is N3.

ATOM_TYPE_IS_NA

Source: AMBER Force Field File
Divider: 1.0

Set to 1 if the atom force field type is Na.

ATOM_TYPE_IS_O

Source: AMBER Force Field File
Divider: 1.0

Set to 1 if the atom force field type is O.

ATOM_TYPE_IS_O2

Source: AMBER Force Field File
Divider: 1.0

Set to 1 if the atom force field type is O2.

ATOM_TYPE_IS_OH

Source: AMBER Force Field File
Divider: 1.0

Set to 1 if the atom force field type is OH.

ATOM_TYPE_IS_OTHER

Source: AMBER Force Field File
Divider: 1.0

Set to 1 if the atom force field type is none of the other categories.

ATOM_TYPE_IS_S

Source: AMBER Force Field File
Divider: 1.0

Set to 1 if the atom force field type is S.

ATOM_TYPE_IS_SH

Source: AMBER Force Field File
Divider: 1.0

Set to 1 if the atom force field type is SH.

CARBONYL

Source: PDB File
Divider: 10.0

Set to 1 if the atom is an ASN Oδ or GLN Oε. Set to 0.5 if the atom is an ASP Oδ, GLU Oε, or ASN Aδ or GLN Aε

CARBOXYL_CARBON

Source: PDB File
Divider: 1.0

Set to 1 if the atom is a carbon atom in a carboxyl group

CARBOXYL_OXYGEN

Source: PDB File
Divider: 1.0

Set to 1 if the atom is a oxygen atom in a carboxyl group

CHARGE

Source: PDB File
Divider: 1000.0

The absolute value for the following lookup table: -1/3 for ASP Cγ, ASP Oδ, GLU Cδ, GLU Oε; 1.0 for LYS Nζ; and +1/3 for ARG Cζ, ARG Nη

CHARGE_WITH_HIS

Source: PDB File
Divider: 1000.0

Same as CHARGE, but with the additional conditions: +0.5 for HIS Nδ or HIS Nε; 0.25 for HIS Aδ or HIS Aε.

ELEMENT_IS_ANY

Source: PDB File
Divider: 1

Set to 1 for any element. Acts as a counter for atoms.

ELEMENT_IS_C

Source: PDB File
Divider: 1

Set to 1 if the atom is a carbon.

ELEMENT_IS_N

Source: PDB File
Divider: 1

Set to 1 if the atom is nitrogen.

ELEMENT_IS_O

Source: PDB File
Divider: 1

Set to 1 if the atom is oxygen.

ELEMENT_IS_OTHER

Source: PDB File
Divider: 1

Set to 1 if the atom is not C, N, O, or S.

ELEMENT_IS_S

Source: PDB File
Divider: 1

Set to 1 if the atom is sulfur.

HYDROPHOBICITY

Source: PDB File
Divider: 1000.0

Residue	Atom	Value	Atom	Value	Atom	Value
ARG	Cβ, Cγ	1	Cδ, Cζ	0	Nε, $Nη_1$, $Nη_2$	-1
ASN	Cβ	1	Cγ	0	$Oδ_1$, $Nδ_2$, $Aδ_1$, $Aδ_2$	-1
ASP	Cβ	1	Cγ	0	$Oδ_1$, $Oδ_2$	-1
CYS			Cβ	0	Sγ	-1
GLN	Cβ, Cγ	1	Cδ	0	$Oε_1$, $Nε_2$, $Aε_1$, $Aε_2$	-1
GLU	Cβ, Cγ	1	Cδ	0	$Oε_1$, $Oε_2$	-1
HIS	Cβ	1	Cγ, $Cε_1$, $Cδ_2$	0	$Nδ_1$, $Nε_2$	-1
LYS	Cβ, Cγ, Cδ, Nζ	1			Cη	-1
PRO	not Cδ	1	Cδ	0		
SER			Cβ	0	not Cβ	-1
THR	$Cγ_2$	1	Cβ	0	$Oγ_1$	-1
TRP	not $Nε_1$, $Cδ_1$, or $Cε_2$	1	$Cδ_1$, $Cε_2$	0	$Nε_1$	-1
TYR	not OH or Cζ	1	Cζ	0	OH	-1

HYDROXYL

Source: PDB File
Divider: 10.0

Set to 1.0 for SER Oγ, THR Oγ, or TYR Oη. Set to 0.5 for CYS Sγ.

HYDROXYL_OXYGEN

Source: PDB File
Divider: 1.0

Set to 1 if the atom is an oxygen in an hydroxyl group.

MOBILITY

Source: PDB File
Divider: 1

The number of degrees of freedom of the atom. Roughly corresponds to the number of bonds between the atom and the α carbon.

NEG_CHARGE

Source: PDB File
Divider: 1000.0

Total negative charge. See CHARGE

PARTIAL_CHARGE

Source: AMBER Force Field File
Divider: 10000.0

Partial charge of the atom.

PARTIAL_POSITIVE_CARBON

Source: PDB File
Divider: 1.0

Set to 1 if the atom is a carbon next to a carboxyl group or LYS Cε or ARG Cζ.

PEPTIDE

Source: PDB File
Divider: 1

Set to 1 if the atom is part of the peptide backbone (part of a residue but not part of the R group)

POSITIVE_NITROGEN

Source: PDB File
Divider: 1.0

Set to 1 if the atom is a LYS N or ARG N.

POS_CHARGE

Source: PDB File
Divider: 1000.0

Total positive charge. See CHARGE.

RESIDUE_CLASS1_IS_CHARGED

Source: PDB File
Divider: 1

Set to 1 if the residue is: ASP, GLU, LYS, or ARG.

RESIDUE_CLASS1_IS_HYDROPHOBIC

Source: PDB File
Divider: 1

Set to 1 if the residue is: ALA, VAL, LEU, ILE, PRO, PHE, TRP, or MET.

RESIDUE_CLASS1_IS_POLAR

Source: PDB File
Divider: 1

Set to 1 if the residue is: SER, THR, TYR, HIS, CYS, CYX, ASN, GLN, or TRP.

RESIDUE_CLASS1_IS_UNKNOWN

Source: PDB File
Divider: 1

Set to 1 if the residue class 1 isn't charged, hydrophobic, or polar.

RESIDUE_CLASS2_IS_ACIDIC

Source: PDB File
Divider: 1

Set to 1 if the residue is: ASP or GLU.

RESIDUE_CLASS2_IS_BASIC

Source: PDB File
Divider: 1

Set to 1 if the residue is: LYS, ARG, or HIS.

RESIDUE_CLASS2_IS_NONPOLAR

Source: PDB File
Divider: 1

Set to 1 if the residue is: ALA, VAL, LEU, ILE, PRO, PHE, TRP, or MET.

RESIDUE_CLASS2_IS_POLAR

Source: PDB File
Divider: 1

Set to 1 if the residue is: GLY, SER, THR, CYS, CYX, TYR, ASN, or GLN.

RESIDUE_CLASS2_IS_UNKNOWN

Source: PDB File
Divider: 1

Set to 1 if the residue class 2 isn't acidic, basic, polor, or non-polar

RESIDUE_NAME_IS_ALA

Source: PDB File
Divider: 1

Set to 1 if the residues is ALA.

RESIDUE_NAME_IS_ARG

Source: PDB File
Divider: 1

Set to 1 if the residue is ARG.

RESIDUE_NAME_IS_ASN

Source: PDB File
Divider: 1

Set to 1 if the residue is ASN.

RESIDUE_NAME_IS_ASP

Source: PDB File
Divider: 1

Set to 1 if the residue is ASP.

RESIDUE_NAME_IS_CYS

Source: PDB File
Divider: 1

Set to 1 if the residue is CYS.

RESIDUE_NAME_IS_GLN

Source: PDB File
Divider: 1

Set to 1 if the residue is GLN.

RESIDUE_NAME_IS_GLU

Source: PDB File
Divider: 1

Set to 1 if the residue is GLU.

RESIDUE_NAME_IS_GLY

Source: PDB File
Divider: 1

Set to 1 if the residue is GLY.

RESIDUE_NAME_IS_HIS

Source: PDB File
Divider: 1

Set to 1 if the residue is HIS.

RESIDUE_NAME_IS_HOH

Source: PDB File
Divider: 1

Set to 1 if the residue is HOH (water heteroatom).

RESIDUE_NAME_IS_ILE

Source: PDB File
Divider: 1

Set to 1 if the residue is ILE.

RESIDUE_NAME_IS_LEU

Source: PDB File
Divider: 1

Set to 1 if the residue is LEU.

RESIDUE_NAME_IS_LYS

Source: PDB File
Divider: 1

Set to 1 if the residue is IYS.

RESIDUE_NAME_IS_MET

Source: PDB File
Divider: 1

Set to 1 if the residue is MET.

RESIDUE_NAME_IS_OTHER

Source: PDB File
Divider: 1

Set to 1 if the residue isn't one of the other residue names.

RESIDUE_NAME_IS_PHE

Source: PDB File
Divider: 1

Set to 1 if the residue is PHE.

RESIDUE_NAME_IS_PRO

Source: PDB File
Divider: 1

Set to 1 if the residue is PRO.

RESIDUE_NAME_IS_SER

Source: PDB File
Divider: 1

Set to 1 if the residue is SER.

RESIDUE_NAME_IS_THR

Source: PDB File
Divider: 1

Set to 1 if the residue is THR.

RESIDUE_NAME_IS_TRP

Source: PDB File
Divider: 1

Set to 1 if the residue is TRP.

RESIDUE_NAME_IS_TYR

Source: PDB File
Divider: 1

Set to 1 if the residue is TYR.

RESIDUE_NAME_IS_VAL

Source: PDB File
Divider: 1

Set to 1 if the residue is VAL.

RING_SYSTEM

Source: PDB File
Divider: 1

Set to 1 if the atom is part of a ring system (aromatic)

SECONDARY_STRUCTURE1_IS_3HELIX

Source: DSSP File
Divider: 1

Set to 1 if the secondary structure is a 3-turn helix (minimum of 3 residues per turn).

SECONDARY_STRUCTURE1_IS_4HELIX

Source: DSSP File
Divider: 1

Set to 1 if the secondary structure is a 4-turn helix (minimum of 4 residues per turn).

SECONDARY_STRUCTURE1_IS_5HELIX

Source: DSSP File
Divider: 1

Set to 1 if the secondary structure is a 5-turn helix (minimum of 5 residues per turn).

SECONDARY_STRUCTURE1_IS_BEND

Source: DSSP File
Divider: 1

Set to 1 if the secondary structure is a bend.

SECONDARY_STRUCTURE1_IS_BRIDGE

Source: DSSP File
Divider: 1

Set to 1 if the secondary structure is a β-bridge.

SECONDARY_STRUCTURE1_IS_COIL

Source: DSSP File
Divider: 1

Set to 1 if the secondary structure isn't a strand, helix, bend, bridge, turn, or heteroatom but recognizable by DSSP.

SECONDARY_STRUCTURE1_IS_HET

Source: DSSP File
Divider: 1

Set to 1 if the atom is a heteroatom

SECONDARY_STRUCTURE1_IS_STRAND

Source: DSSP File
Divider: 1

Set to 1 if the secondary structure is a strand and/or anti-parallel β-sheet.

SECONDARY_STRUCTURE1_IS_TURN

Source: DSSP File
Divider: 1

Set to 1 if the secondary structure is a turn.

SECONDARY_STRUCTURE1_IS_UNKNOWN

Source: DSSP File
Divider: 1

Set to 1 if the secondary structure is unrecognizable to DSSP.

SECONDARY_STRUCTURE2_IS_BETA

Source: DSSP File
Divider: 1

Set to 1 if the secondary structure is a bend, turn, or bridge.

SECONDARY_STRUCTURE2_IS_COIL

Source: DSSP File
Divider: 1

Set to 1 if the secondary structure is a coil

SECONDARY_STRUCTURE2_IS_HELIX

Source: DSSP File
Divider: 1

Set to 1 if the secondary structure is any kind of helix.

SECONDARY_STRUCTURE2_IS_HET

Source: DSSP File
Divider: 1

Set to 1 if the secondary structure is a heteroatom

SECONDARY_STRUCTURE2_IS_UNKNOWN

Source: DSSP File
Divider: 1

Set to 1 if the secondary structure isn't beta, coil, helix, or het.

SOLVENT_ACCESSIBILITY

Source: DSSP File
Divider: 1

Set to the solvent accessibility value as provided by the DSSP file.

SULFUR

Source: PDB File
Divider: 1.0

Set to 1 if the atom is a sulfur

VDW_VOLUME

Source: PDB File
Divider: 1000.0

The Van der Waals volume of the atom.

7.1. Glossary

AMBER Force Field

Family of force fields used in molecular dynamics. There is also a software package by the same name used to simulate these force fields.

Binary Classification

The classification of a set of items into two distinct groups based on their characteristics, qualities, and measurements.

Classifier

A *classifier* is an algorithm that predicts whether or not a new data point belongs to a given class by statistically extrapolating from training data. For example, if we were to build a classifier for image recognition of apples from other fruit, we could train the classifier on pictures of different apples and pictures of other fruit. We could then use the classifier to tell us if a picture of an orange is an apple (and hope that the classifier says, "no" with strong confidence).

Classifier Model

An instance of a given classification algorithm which has been trained on testing data. Can be used to predict the classification of unknown samples.

Dictionary of Secondary Structure of Proteins (DSSP)

Dictionary used to describe protein secondary structure with single letter codes. See DSSP file

Domain

A highly conserved region of a biomolecule. Domains may demonstrate enzymatic activity or serve an important structural function.

False Negative (FN)

During classification, a protein that the classifier does not deem statistically similar to the model in question but in does indeed share said functionality.

False Positive (FP)

During classification, a false positive is a protein that statistically matches the model in question but is not a candidate for similar functionality.

FEATURE

FEATURE is software for modeling and finding functional domains in proteins. For more information, see the introduction (Chapter 1).

FEATURE 3.1 employs only the Naive Bayes classifier. Support Vector Machine classifier results show remarkable improvement and will be released after publication. Development is underway to incorporate Random Forests.

Feature Vector

N-dimensional vector containing numeric entries calculated by examining the physiochemical properties at varying radial distances from the site center.

For more information, see feature vector file format

Functional Class

A group of biomolecules with identical catalytic function.

Functional Site

Local 3-D environment surrounding a site of interest for a given protein.

Microenvironment

Immediate spherical volume surrounding a point in a protein structure. Microenvironments are divided up into six concentric *shells*, with increasing radius in 1.25 Å increments (by default). Used as the basis for classification within FEATURE.

Model

The term *model* can refer to two primary things within the context of FEATURE:

- See *Prosite Model* for an explanation of how the term *models* is used within the Prosite database.
- See *Classifier Model* for an explanation of how the term *models* is used within the context of Classifiers.

Naive Bayes

A type of classifier which utilizes bayes theorem to create a probability distribution which can be used to predict the classification of a particular item. The algorithm takes in a *training set* containing numeric fields (attributes) and group classes. It then computes the probability distribution for a given class using statistical information about the numeric fields of items contained within it.

Properties

Often referred to as *physicochemical properties*, these are data associated with each atom in a protein structure. Simple examples include:

atom name, partial charge, Van der Waals radius, and solvent accessibility. More complex examples include: if the atom is part of a aromatic ring, if the atom is from a residue that is part of an alpha helix, and the atom mobility (distance from the α carbon).

Prosite Model

Prosite database entry containing protein families, domains, amino acid patterns, and signatures.

For more information, visit the Prosite website: http://prosite.expasy.org

An example of a Prosite model for a ferredoxin-type iron-sulfur binding domain can be found here: http://prosite.expasy.org/cgi-bin/prosite/prosite-search-ac?PDOC00175

Prosite Consensus Pattern

The Prosite database utilizes regular expression patterns to declare the sequence for a specific model. This regular expression can in turn be matched with an appropriate sequence within a given PDB entry. These sequences are also used for similarity filtering.

Pattern	Brief	Description
x	Wildcard	Matches one of any kind of residue
A, C, D, E, F, G, H, I, K, L, M, P, Q, R, S, T, V, W, or Y	Residue	Matches the specified standard IUPAC single letter code for an amino acid residue.
-	Separator	No match effect, simply improves the readability of the pattern.
(*k*)	Count	Matches exactly *k* residues of any kind.
(*i*,*j*)	Range	Matches at least *i* or at most *j* of any kind of residue
[ ]	Choice	Matches one of the given residues in the brackets
{ }	Except	Matches any one residue except the given residues in the brackets

See http://prosite.expasy.org/prosuser.html#convent35 for full details on Prosite patterns.

An example of the Prosite sequence consensus pattern for ferredoxin-type iron-sulfur binding domain is shown below.

```
C-{C}-{C}-[GA]-{C}-C-[GAST]-{CPDEKRHFYW}-C
```

Protein Data Bank

Database containing 3-D structure data for large biological molecules. Curated and maintained by many biologists and biochemists around the world. Hosted by member organizations PDBe, PDBj, and RCSB. Data typically taken using X-ray crystallography or NMR spectroscopy. See \ref pdb "PDB File" for more information.

Point File (.ptf)

Point File, contains positive functional sites for a given model. Contains pdb ids, coordinate information, comments, and chains.

For more information, see Point Files.

Receiver Operating Characteristic (ROC)

Graphical plot of the sensitivity for a binary classifier as it's discrimination 3 threshold is varied. Sensitivity is defined as |TP|, a high sensitivity |TP|+|FP| indicates a low number of false negatives and is thus more desirable.

Secondary Structure

Formally defined by the hydrogen bonds of a biopolymer, for proteins, it's defined by patterns of hydrogen bonds between backbone amide and carboxyl groups.

Shell

A *microenvironment* is composed of concentric shells. A shell is a spherical volume, minus a smaller concentric spherical volume. The radii of the spherical volumes are consecutive integer multiples of some real number (1.25 Å by default).

Similarity Cluster

A similarity cluster groups pdb entries by percent sequence similarity. The clusters used for FEATURE come from the RCSB's weekly BlastClust results.

Support Vector Machine

SVMs were first introduced in 1995 by Vladimir Vapnik and Corinna Cortes. They are a type of classification algorithm which, given a set of training data, constructs a hyperplane or set of hyperplanes in n dimensions. These hyperplanes have a maximized distance from distinct groups of data. During the classification phase, the features for a sample data are compared to the partitions created by the hyperplanes created during training. The classifier can then choose the hyperplane that best separates the sample's features and predict it's classification.

True Negative (TN)

When training a classifier, a negative indicates a residue which does not contribute to a given model's functionality. During the classification stage, a negative means protein in question is not a good candidate to be a structure with similar functionality.

True Positive (TP)

In the context of training a classifier, a positive is a protein model with known functionality and residue information. During the classification phase, a positive indicates a protein with a statistically similar structure and is therefor a possible candidate for the same functionality.

X-ray Crystallography

A technique for finding the 3-D structure of a crystal by analysing diffraction patterns created as x-ray light passes through the sample. In the case of proteins, macromolecules, a crystallized sample is typically grown in solution.

7.1.1. Acronyms

AMBER	Assisted Model Building with Energy Refinement
DSSP	Dictionary of Secondary Structure of Proteins
FN	False Negative
FP	False Positive
NB	Naive Bayes
PDB	Protein Data Bank
PTF	Point Text File
ROC	Receiver Operating Characteristic
SVM	Support Vector Machine
TN	True Negative
TP	True Positive
wwPDB	Worldwide Protein Data Bank

Chapter 8. References

Robust recognition of zinc binding sites in proteins. Ebert JC, Altman RB. Protein Sci. 2008; 17 (1): 54-65

The SeqFEATURE library of 3D functional site models: comparison to existing methods and applications to protein function annotation. Wu S, Liang MP, Altman RB. Genome Biol. 2008; 9 (1): R8

The FEATURE framework for protein function annotation: modeling new functions, improving performance, and extending to novel applications. Halperin I, Glazer DS, Wu S, Altman RB. BMC Genomics. 2008: 9 Suppl 2 S2

Clustering protein environments for function prediction: finding PROSITE motifs in 3D. Yoon S, Ebert JC, Chung EY, De Micheli G, Altman RB. BMC Bioinformatics. 2007: 8 Suppl 4 S10

Structural characterization of proteins using residue environments. Mooney SD, Liang MH, DeConde R, Altman RB. Proteins. 2005; 61 (4): 741-7

WebFEATURE: An interactive web tool for identifying and visualizing functional sites on macromolecular structures. Liang MP, Banatao DR, Klein TE, Brutlag DL, Altman RB. Nucleic Acids Res. 2003; 31 (13): 3324-7

Microenvironment analysis and identification of magnesium binding sites in RNA. Banatao DR, Altman RB, Klein TE. Nucleic Acids Res. 2003; 31 (15): 4450-60

Automated construction of structural motifs for predicting functional sites on protein structures. Liang MP, Brutlag DL, Altman RB. Pac Symp Biocomput. 2003: 204-15

Recognizing protein binding sites using statistical descriptions of their 3D environments. Wei L, Altman RB. Pac Symp Biocomput. 1998: 497-508

Conserved features in the active site of nonhomologous serine proteases. Bagley SC, Altman RB. Fold Des. 1996; 1 (5): 371-9

Characterizing the microenvironment surrounding protein sites. Bagley SC, Altman RB. Protein Sci. 1995; 4 (4): 622-35

Characterizing oriented protein structural sites using biochemical properties. Bagley SC, Wei L, Cheng C, Altman RB. Proc Int Conf Intell Syst Mol Biol. 1995: 3 12-20

The SeqFEATURE library of 3D functional site models: comparison to existing methods and applications to protein function annotation. Wu S, Liang MP, Altman RB. Genome Biol. 2008; 9 (1): R8

The FEATURE framework for protein function annotation: modeling new functions, improving performance, and extending to novel applications. Halperin I, Glazer DS, Wu S, Altman RB. BMC Genomics. 2008: 9 Suppl 2 S2

Improving structure-based function prediction using molecular dynamics. Glazer DS, Radmer RJ, Altman RB. Structure. 2009; 17 (7): 919-29

The FEATURE framework for protein function annotation: modeling new functions, improving performance, and extending to novel applications. Halperin I, Glazer DS, Wu S, Altman RB. BMC Genomics. 2008: 9 Suppl 2 S2

Combining molecular dynamics and machine learning to improve protein function recognition. Glazer DS, Radmer RJ, Altman RB. Pac Symp Biocomput. 2008: 332-43

9.1. FEATURE License