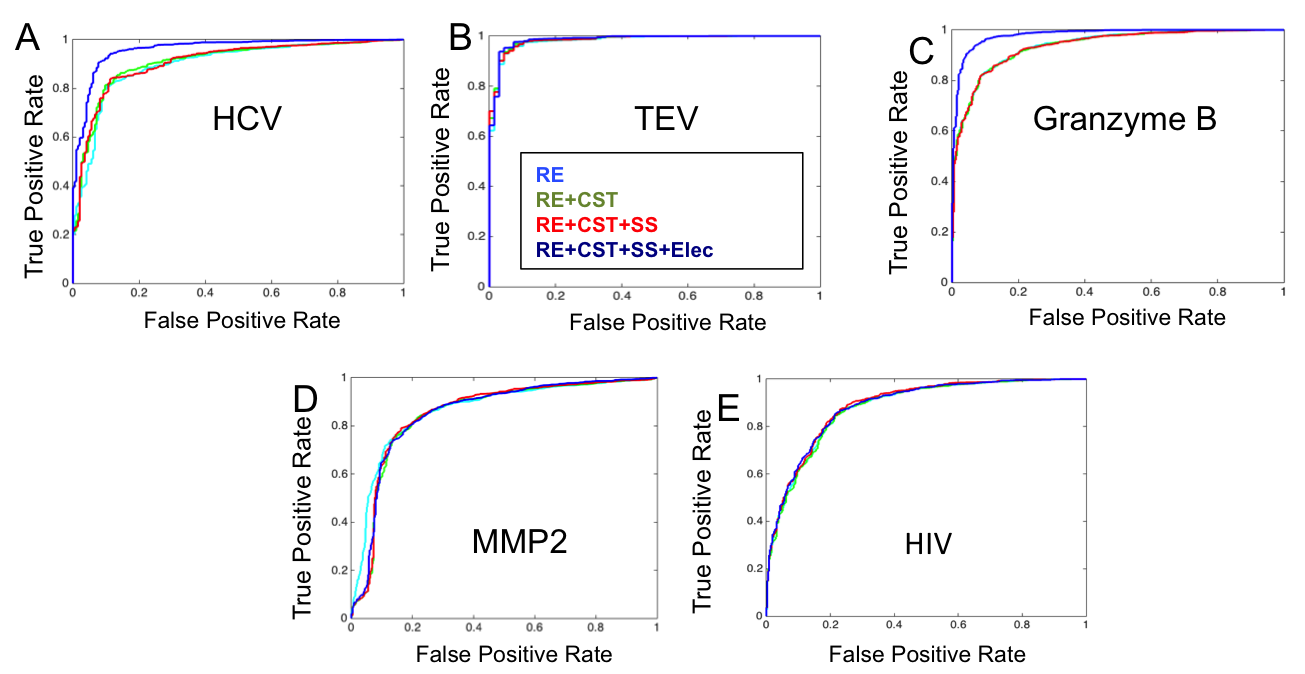
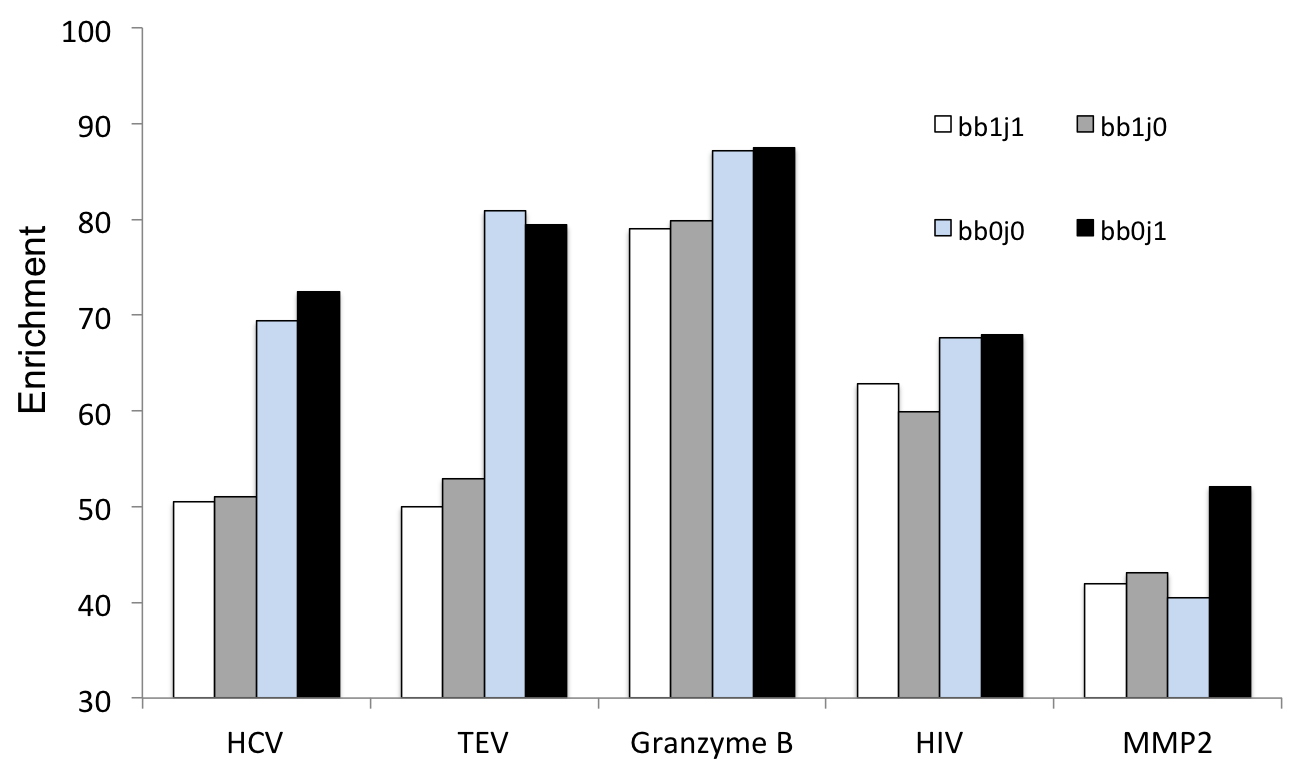
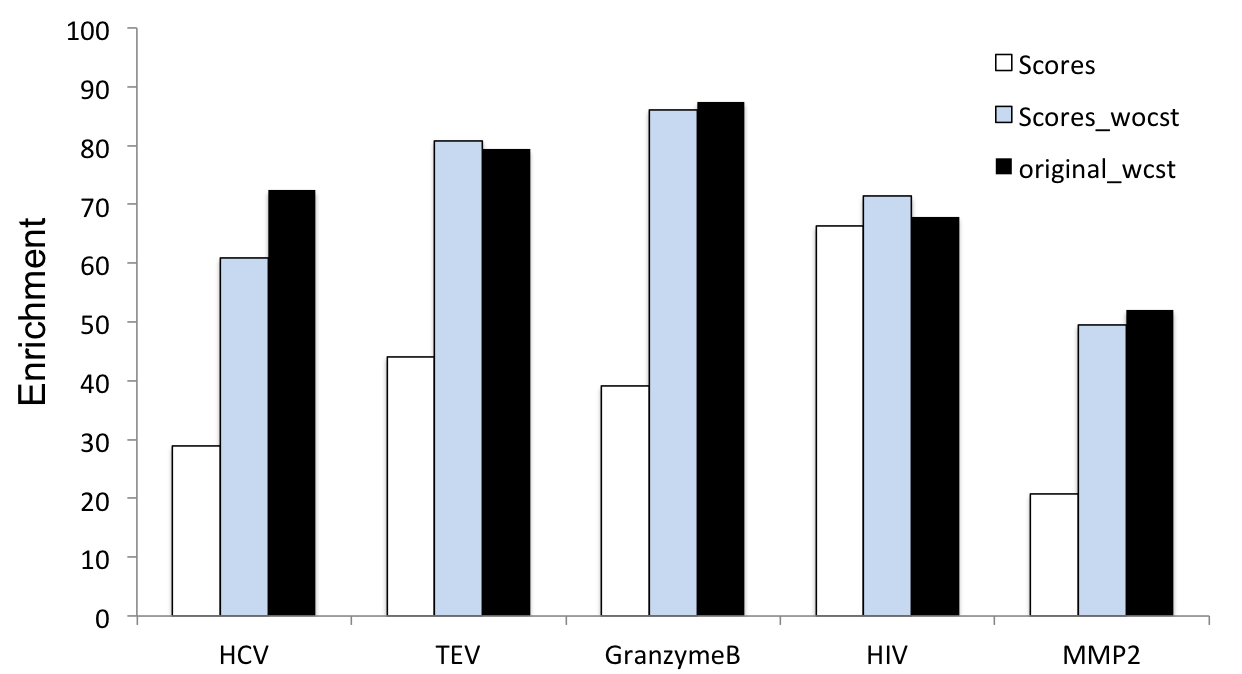
**Supplementary Figures:**



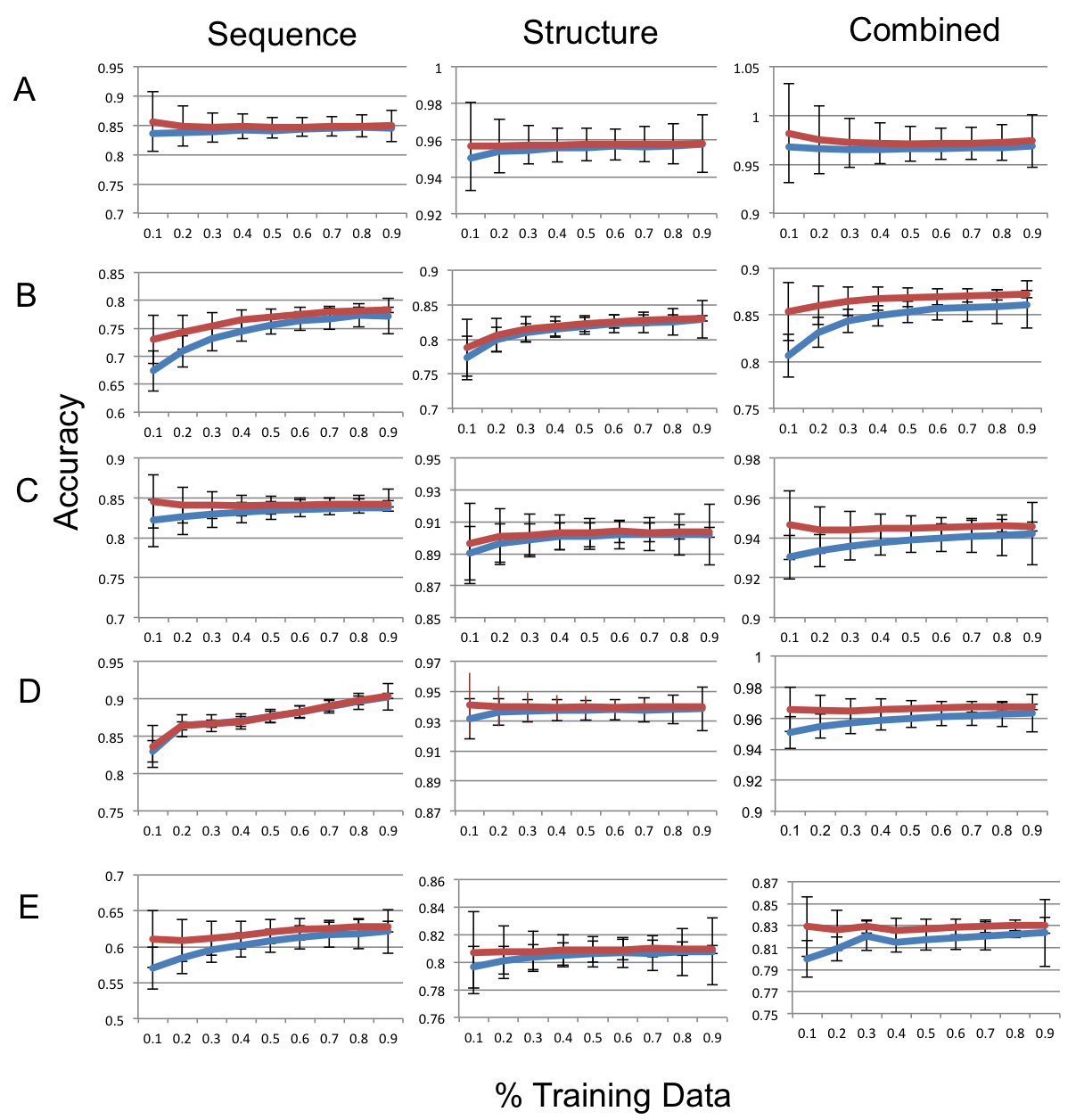
**Figure S1: The additive effect of each energy term to the auROC.** Each plot shows the representative ROC curve for Rosetta Energy (sum total of peptide and protease interface energy; depicted in light blue), Rosetta Energy + constraint score (Green), Rosetta Energy + constraint score + secondary structure propensity (red), Rosetta Energy + constraint score + secondary structure propensity + Electrostatic binding energy (dark blue). All score terms are seen to contribute to the discriminative efficiency of the score function.



**Fig S2: Impact of sampling flexibility of the protease backbone and sidechain degrees of freedom.** The peptide backbone and sidechains were flexible in all of the simulations depicted in the figure. “bb” refers to the backbone of the protease such that bb=0 indicates that the backbone was not allowed to relax, bb=1 that backbone was allowed to relax. ”j” refers to the rigid body freedom of the peptide with respect to the protease. j=0 means that rigid body freedom was constrained during the simulation; j=1 rigid body flexibility allowed during simulation. The highest efficiency of discrimination was observed when the protease backbone was not allowed to relax, and the protease sidechains were flexible during the simulation.



**Figure S3: Contribution of maintaining near attack conformation with respect to protease catalytic machinery.** Three FastRelax protocols were performed to compare the effect of the presence of catalytic constraints during the Fastrelax and scoring stage. Scores (white bars) depict enrichment values obtained when enzymatic constraints were excluded in the FastRelax step but were included in the scoring step. Scores\_wocst (blue) depict experimental results where constraints were excluded from the FastRelax step as well as from the scoring calculation. Original\_wcst (black) depict experimental results where FastRelax was performed with constraints and the constraint score was included in calculation of Enrichment. Highest enrichment is observed when catalytic constraints are included in both the FastRelax as well as scoring steps.



**Fig S4. To avoid over-fitting we performed a jack-knifing procedure where classification and generalization was performed by randomly splitting the datasets into training and test.** The Figure illustrates Accuracy versus Training Data size plots for Sequence, Structure and Combination SVMs. (A) TEV (B) HIV (C) HCV (D) Granzyme B (E) MMP2

**Supplementary Tables:**

Table S1: True Positive and False Positive Rates observed for critical point of auROC

|  |  |  |
| --- | --- | --- |
| **Protease** | **TPR** | **FPR** |
| HCV | 0.92 | 0.08 |
| TEV | 0.96 | 0.04 |
| HIV | 0.82 | 0.18 |
| Granzyme B | 0.93 | 0.07 |
| MMP2 | 0.76 | 0.24 |

Table S2: Results of a calculation to investigate the additive effect of each score term in the discriminatory score function

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Protease** |  | **RE+CST** | **RE+CST+Elec** | **RE+CST+Elec+SS** |
|  |  |  |  |  |
| Granzyme B | Enrichment | 0.70 | 0.68 | 0.87 |
|  | Fold increase | 4.6 | 4.5 | 5.7 |
|  | AUC | 0.93 | 0.93 | 0.98 |
|  |  |  |  |  |
| HCV | Enrichment | 0.64 | 0.76 | 0.80 |
|  | Fold increase | 6.2 | 7.3 | 7.6 |
|  | AUC | 0.93 | 0.97 | 0.97 |
|  |  |  |  |  |
| TEV | Enrichment | 0.72 | 0.72 | 0.80 |
|  | Fold increase | 16.68 | 16.68 | 18.35 |
|  | AUC | 0.98 | 0.98 | 0.98 |
|  |  |  |  |  |
| HIV | Enrichment | 0.69 | 0.68 | 0.69 |
|  | Fold increase | 3.2 | 3.2 | 3.2 |
|  | AUC | 0.90 | 0.90 | 0.90 |

Table S3: Results of a grid-based optimization scheme to maximize enrichment

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | *Enrichment* | *protease* | *peptide* | *cst* | *ss* | *Elec* |
| TEV | 0.8088 | 1 | 0.3 | 2.5 | 0.005 | 0.25 |
| HCV | 0.7806 | 1 | 1 | 3.5 | 0.001 | 0.5 |
| HIV | 0.7112 | 1 | 0.8 | 3.4 | 0.013 | 0.1 |
| GrB | 0.8867 | 1 | 0.5 | 2.5 | 0.005 | 0.3 |
| MMP2 | 0.6747 | 1 | 0.7 | 2.6 | 0.007 | 0.1 |

Table S4: Primers used for molecular cloning the sequences to be tested in the YESS assay into the assay (LY104) vector using RF cloning

|  |  |
| --- | --- |
| Sequence | Primers |
| LEEFFCSG | FOR: CGGTAGCGGAGGCGGAGGGTCGTTGGAAGAATTCTTCTGTTCAGGC |
|  | REV: CTGCCTTTATCATCATCATCTTTATAATCACTGCCGCCTGAACAGAAGAATTCTTCC |
| LEEYQCSG | FOR: CGGTAGCGGAGGCGGAGGGTCGTTGGAAGAATATCAATGTTCAGGCG |
|  | REV: CTGCCTTTATCATCATCATCTTTATAATCACTGCCGCCTGAACATTGATATTCTTCCAA |
| CEDYFCSG | FOR: CGGTAGCGGAGGCGGAGGGTCGTGTGAAGATYMTTTCTGTTCAGGCG |
|  | REV: CTGCCTTTATCATCATCATCTTTATAATCACTGCCGCCTGAACAGAAAKRATCTTCACA |
| FEDFQCSG | FOR: CGGTAGCGGAGGCGGAGGGTCGTTCGAAGATTTCCAATGTTCAGGC |
|  | REV: CTGCCTTTATCATCATCATCTTTATAATCACTGCCGCCTGAACATTGGAAATCTTCG |

**Supplementary Methods:**

The MMPBSA calculation includes the following steps:

1. Preparation of AMBER input .pdb files
2. Preparation of input parameter and topology files
3. MMPBSA Calculation

Description of each of the steps below:

In order to transform a pdb file into an AMBER readable format the hydrogens and virtual atoms are stripped. The subsequent file is loaded into AMBER using the following script using a tleap interface.

source leaprc.gaff

source leaprc.ff12SB

loadamberparams frcmod.ionsjc\_tip3p

d$i = loadpdb "toload\_$i.pdb"

addions d$i Cl- 0

charge d$i

saveamberparm d$i d$i.prmtop d$i.inpcrd

quit

The files saved as d$i.prmtop and d$i.inpcrd are inputs to the ante-MMPBSA.py program which generates the receptor-ligand, receptor only and ligand only topology files. An AMBER topology file is used to specify atom types, charges, etc. The inpcrd / input coordinate file is used to build the connections which forms the overall structure of the pdb.

ante-MMPBSA.py -p d$i.prmtop -c d\_c$i.prmtop -s @Cl-

ante-MMPBSA.py -p d\_c$i.prmtop -r d\_r$i.prmtop -l d\_l$i.prmtop -n : “residue range”

Residue range: specify the pose numbering of the peptide

The final step involves using the inpcrd and prmtop files to calculate the MMPBSA contribution of the complex. This is done by calculating the electrostatic energy of the peptide and protease separately as well as in a bound state

The following commandline is used for MMPBSA calculation

MMPBSA.py -O -i mmpbsa.in -o FINAL\_RESULTS\_MMPBSA.dat -sp d$i.prmtop -cp d\_c$i.prmtop -rp d\_r$i.prmtop -lp d\_l$i.prmtop -y \*.inpcrd

For MMP2: The pdbs in these cases needed to be analyzed differently because of the presence of heteroatoms such as Zinc and Water that are involved in the active sites respectively.

The water is modeled using the TP5.lib and the following command is added to the prep script

*Sample Scripts:*

*Sample xml for initial Relax:*

<dock\_design>

<SCOREFXNS>

<myscore weights=enzdes.wts/>

</SCOREFXNS>

<TASKOPERATIONS>

<ProteinInterfaceDesign name=pido design\_chain2=0 modify\_after\_jump=1/>

<InitializeFromCommandline name=init/>

<ReadResfile name=rrf filename="PATH TO RESFILE"/>

</TASKOPERATIONS>

<FILTERS>

</FILTERS>

<MOVERS>

<AddOrRemoveMatchCsts name=cstadd cst\_instruction=add\_new/>

<FastRelax name=fastrelax scorefxn=myscore repeats=8 task\_operations=pido,init>

<MoveMap name=mm>

<Chain number=2 chi=1 bb=1/>

<Chain number=1 chi=1 bb=1/>

<Jump number =1 setting=1/>

</MoveMap>

</FastRelax>

<TaskAwareMinMover name =min\_pro task\_operations=rrf scorefxn=myscore chi=1 bb=0 jump=0/>

<PackRotamersMover name=repack task\_operations=rrf/>

<ConstraintSetMover name=protease\_cst cst\_file="PATH\_TO\_PROTEASE\_BACKBONE\_HEAVY\_ATOM\_CONSTRAINT\_FILE"/>

</MOVERS>

<APPLY\_TO\_POSE>

</APPLY\_TO\_POSE>

<PROTOCOLS>

<Add mover\_name=protease\_cst/>

<Add mover\_name=repack/>

<Add mover\_name=min\_pro/>

<Add mover\_name=cstadd/>

<Add mover\_name=fastrelax/>

</PROTOCOLS>

</dock\_design>

*Command line:*

~<PATH\_TO\_ROSETTA\_BIN> rosetta\_scripts.static.linuxgccrelease -jd2:ntrials 1 -nstruct 20 -parser:protocol <PATH\_TO\_RELAX\_XML> -database <PATH\_TO\_DATABASE> -out::prefix Job\_${i}\_ -s <PATH\_TO\_STARTING\_PDB> -run:preserve\_header -enzdes::cstfile <PATH\_TO\_CONSTRAINT\_FILE> -out:file:output\_virtual @<PATH\_TO\_FLAGS\_FILE>

*Sample Script For Mutate, FastRelax, Scoring*

#MUTATERUN

<PATH\_TO\_EXECUTABLE>/rosetta\_scripts.static.linuxgccrelease -nstruct 10 -jd2:ntrials 1 -parser:protocol <PATH\_TO\_XML> -database <PATH\_TO\_DATABASE> -out::prefix $1\_mut\_ -s <PATH\_TO\_STARTING\_PDB> -enzdes:cstfile <PATH\_TO\_CSTFILE> -run:preserve\_header @<PATH\_TO\_FLAGSFILE> > design.log

find `pwd` -name "$1\_mut\_\*00\*pdb" > tlist

cp ~/Rosetta/main/database/scoring/weights/talaris2013 ./

#SCORINGRUN

~/Rosetta/main/source/bin/rosetta\_scripts.static.linuxgccrelease -jd2:ntrials 1 -parser:protocol <PATH\_TO\_SCORING\_XML> -database <PATH\_TO\_DATABASE> -out::prefix Scores\_ -l tlist -in:file:native <PATH\_TO\_STARTINGPDB> -run:preserve\_header @<PATH\_TO\_FLAGSFILE> -score:weights talaris2013 > scoring.log

ls Scores\_\*.pdb > slist

#CSTRUN

~/Rosetta/main/source/bin/rosetta\_scripts.static.linuxgccrelease -jd2:ntrials 1 -parser:protocol <PATH\_TO\_XML> -database ~/Rosetta/main/database/ -out::prefix $1\_cst\_ -l tlist -enzdes:cstfile <PATH\_TO\_CSTFILE> -run:preserve\_header @<PATH\_TO\_FLAGSFILE> -jd2:enzdes\_out > cst.log

Protease Mutate:

<dock\_design>

<SCOREFXNS>

<myscore weights=enzdes.wts/>

</SCOREFXNS>

<TASKOPERATIONS>

<ProteinInterfaceDesign name=pido design\_chain2=0 modify\_after\_jump=0/>

<InitializeFromCommandline name=init/>

<ReadResfile name=rrf filename="PATH\_TO\_RESFILE"/>

</TASKOPERATIONS>

<FILTERS>

</FILTERS>

<MOVERS>

<MutateResidue name=mut1 target=Res#1 new\_res=DM1/>

<MutateResidue name=mut2 target= Res#2 new\_res=DM2/>

<MutateResidue name=mut3 target= Res#3new\_res=DM3/>

<MutateResidue name=mut4 target= Res#4 new\_res=DM4/>

<MutateResidue name=mut5 target= Res#5 new\_res=DM5/>

<MutateResidue name=mut6 target= Res#6 new\_res=DM6/>

<AddOrRemoveMatchCsts name=cstadd cst\_instruction=add\_new/>

<FastRelax name=fastrelax scorefxn=myscore repeats=8 task\_operations=pido,init>

<MoveMap name=mm>

<Chain number=2 chi=1 bb=1/>

<Chain number=1 chi=1 bb=0/>

<Jump number =1 setting=1/>

</MoveMap>

</FastRelax>

<TaskAwareMinMover name =min\_pro task\_operations=rrf chi=1 bb=0 jump=0/>

<PackRotamersMover name=repack task\_operations=rrf/>

</MOVERS>

<APPLY\_TO\_POSE>

</APPLY\_TO\_POSE>

<PROTOCOLS>

<Add mover\_name=mut1/>

<Add mover\_name=mut2/>

<Add mover\_name=mut3/>

<Add mover\_name=mut4/>

<Add mover\_name=mut5/>

<Add mover\_name=mut6/>

<Add mover\_name=repack/>

<Add mover\_name=cstadd/>

<Add mover\_name=fastrelax/>

</PROTOCOLS>

</dock\_design>

SCORING XML

CST XML

<dock\_design>

<SCOREFXNS>

<myscore weights=enzdes.wts/>

</SCOREFXNS>

<TASKOPERATIONS>

<InitializeFromCommandline name=init/>

</TASKOPERATIONS>

<FILTERS>

<EnzScore name="cstenergy" scorefxn=myscore whole\_pose=1 score\_type=cstE energy\_cutoff=99999.0/>

</FILTERS>

<MOVERS>

<AddOrRemoveMatchCsts name=cstadd cst\_instruction=add\_new/>

</MOVERS>

<APPLY\_TO\_POSE>

</APPLY\_TO\_POSE>

<PROTOCOLS>

<Add mover\_name=cstadd/>

<Add filter\_name=cstenergy/>

</PROTOCOLS>

</dock\_design>

AMBER MMPBSA

cat >tleap.in <<EOF

source leaprc.gaff

source leaprc.ff12SB\_manasi

loadamberparams frcmod.ionsjc\_tip3p

loadamberparams frcmod.ionslrcm\_hfe\_tip3p

d$i = loadpdb "toload\_$i.pdb"

charge d$i

saveamberparm d$i d$i.prmtop d$i.inpcrd

quit

EOF

tleap -f tleap.in

ante-MMPBSA.py -p d$i.prmtop -c d\_c$i.prmtop -s @Cl-

ante-MMPBSA.py -p d\_c$i.prmtop -r d\_r$i.prmtop -l d\_l$i.prmtop -n :199-208

MMPBSA.py -O -i mmpbsa.in -o FINAL\_RESULTS\_MMPBSA.dat -cp d\_c$i.prmtop -rp d\_r$i.prmtop -lp d\_l$i.prmtop -y d$i.inpcrd

MATLAB

function [test, testlab, ttcleaved, to, ts, train, trainlab, a, f, X, Y, T, AUC, AUCav, Std, Performanceav,Stdp] = coduh(A, LABELS, cleaved, uncleaved, boxconstraint, rbfsigma)

clearvars -except A LABELS cleaved uncleaved boxconstraint rbfsigma TABLE

X = [];

Y = [];

T = [];

AUC = [];

A = Together\_Furin; % change to name of matrix

[numberofelements len] = size(A);

tic

for s = 1:1000

zcleaved = ceil(0.2%\*cleaved);

zuncleaved = ceil(0.2%\*uncleaved);

ttcleaved = randperm(cleaved,zcleaved); %generatingRandomFromNumLength

ttuncleaved = randperm((numberofelements - cleaved), zuncleaved) + cleaved;

t = vertcat(ttcleaved',ttuncleaved');

to(:,s) = vertcat(ttcleaved',ttuncleaved');

ts(s) = length(t);

z = zcleaved + zuncleaved;

test(:,:,s) = A(t,:);

testlab(:,:,s) = LABELS(t,:);

x = numberofelements - z;

train(:,:,s) = zeros(x, len);

trainlab(:,:,s)= cell(x,1);

clear n1;

n1 = 1;

for i = 1:numberofelements

if i ~= t(:)

train(n1,:,s) = A(i,:);

trainlab(n1,s) = LABELS(i);

n1 = n1 + 1;

end

end

svmrbf =[];

svmrbf=svmtrain(train(:,:,s), trainlab(:,s), 'kernel\_function', 'rbf', 'boxconstraint', boxconstraint, 'rbf\_sigma', rbfsigma);

%%TEST%%

V = svmclassify(svmrbf,test(:,:,s));

result = transpose(V);

a(:,s)=transpose(result);

shift = svmrbf.ScaleData.shift;

scale = svmrbf.ScaleData.scaleFactor;

Xnew = bsxfun(@plus,test(:,:,s),shift);

Xnew = bsxfun(@times,Xnew,scale);

sv = svmrbf.SupportVectors;

alphaHat = svmrbf.Alpha;

bias = svmrbf.Bias;

kfun = svmrbf.KernelFunction;

kfunargs = svmrbf.KernelFunctionArgs;

f(:,s) = kfun(sv,Xnew,kfunargs{:})'\*alphaHat(:) + bias;

[X(:,s),Y(:,s),T(:,s),AUC(s)] = perfcurve(testlab(:,:,s), f(:,s) ,'CLEAVED', 'Xcrit','reca', 'YCrit', 'prec' );

AUCav = mean(AUC);

Std = std(AUC);

%ACCURACY

tf(:,s) = strcmp (a(:,s), testlab(:,s));

Performance(s) = sum(tf(:,s)) / numel(a(:,s));

Performanceav = mean(Performance);

Stdp = std(Performance);

% %TRAIN

Vtrain = svmclassify(svmrbf,train(:,:,s));

resulttrain = transpose(Vtrain);

%clear train end

atrain(:,s)=transpose(resulttrain);

shift = svmrbf.ScaleData.shift;

scale = svmrbf.ScaleData.scaleFactor;

Xnew1 = bsxfun(@plus,train(:,:,s),shift);

Xnew1 = bsxfun(@times,Xnew1,scale);

sv = svmrbf.SupportVectors;

alphaHat = svmrbf.Alpha;

bias = svmrbf.Bias;

kfun = svmrbf.KernelFunction;

kfunargs = svmrbf.KernelFunctionArgs;

ftrain(:,s) = kfun(sv,Xnew1,kfunargs{:})'\*alphaHat(:) + bias;

display(f(:,s));

[Xtraintemp,Ytraintemp,Ttraintemp,AUCtrain(s)]= perfcurve(trainlab(:,:,s),ftrain(:,s),'CLEAVED');

[r] = length(Xtraintemp);

Xtrain(1:r, s) = Xtrain(1:r, s) + Xtraintemp;

Ytrain(1:r, s) = Ytrain(1:r, s) + Ytraintemp;

Ttrain(1:r, s) = Ttrain(1:r, s) + Ttraintemp;

clear Xtraintemp Ytraintemp Ttraintemp

[Xtrain(:,s),Ytrain(:,s),Ttrain(:,s),AUCtrain(s)]=

perfcurve(trainlab(:,:,s),ftrain(:,s),'CLEAVED');

AUCtrainav = mean(AUCtrain);

Stdtrain = std(AUCtrain);

tftrain(:,s) = strcmp (atrain(:,s), trainlab(:,s));

Performancetrain (s)= sum(tftrain(:,s)) / numel(atrain(:,s));

Performancetrainav = mean(Performancetrain);

Stdptrain = std(Performancetrain);

s

end

toc