

# Peptide programs: applying fragment programs to protein classification

André Falcão ([afalcao@di.fc.ul.pt](mailto:afalcao@di.fc.ul.pt))  
*Department of Informatics  
Faculty of Sciences  
University of Lisbon, Portugal*

**Daniel Faria**  
**António Ferreira**

ACM CIKM2008 - DTMBIO'08 Workshop  
October 30 2008

## Summary

- Protein function determination with machine learning
  - Instance based
  - Model based
- Peptide programs
  - the basic idea
  - Implementation details
- Test data
- Results
- Conclusions

## Machine learning for protein function determination

- The most common approach to determining protein function without *in vivo* or *in vitro* experimenting is through machine learning
- Proteins are unstructured data
  - Arbitrary sizes
  - Difficult to extract attributes and work on a vector space
  - Working in metric spaces is easier:
    - Instance based learning
    - Not dependent on the extraction of attributes
    - There are several way to compare proteins

## Instance based methods

- Proteins get their properties from their sequence
  - Sequences fold into secondary structures
  - And then into tertiary structures
  - Idea is Proteins with similar primary, secondary or tertiary structures CAN have similar functions
- All methods that purport to compare proteins are instance based
  - For primary structure data: BLAST, FASTA or Smith and Waterman
  - For proteins with determined tertiary structure the problem is more complex as it is still difficult and expensive to produce this data

## Model based methods

- Model based methods in machine learning use a set of features from known instances in a dataset to learn a model
- The generated model uses those same features in unknown data to estimate the unknown attributes
- Proteins are unstructured data with arbitrary sizes. Using the amino acids in the sequence will simply do not work except for the simplest problems
  - It is naturally difficult to identify and to extract features and work in a vector space

## Model based methods

- Several approaches have been tried using almost all the known Methods
  - Neural Networks
  - Support Vector Machines
  - Decision Trees
  - ...
- Most methods arbitrarily set attributes that have a difficult biological justification
- Some authors use secondary or tertiary information which is not directly available for a large majority of proteins

## Which is best?

- Instance based methods:
  - Slow - based on the number of known instances
  - Incapable of predicting really different samples
    - When the variable space is large, a K-nearest neighbors approach becomes very prone to errors (curse of dimensionality)
  - Yet these methods have so far constructed the ground work for protein function classification (BLAST!)
- Model based methods
  - Can be faster - models do not usually rely on the existing sample data
  - If the model really learns the process it will be capable of predicting no matter how different samples
  - So far have not yet proven to be efficient
    - Difficult to select and use attributes
    - Low classification accuracies

## The idea of Peptide Programs

- Proteins are formed sequentially by the ribosome. RNA is translated and the protein is folded sequentially
- The idea of protein programs is to mimic this initial folding process as if the protein was a computer program and the output of its execution is the protein function
  - The same instructions are applied to each residue
- The problem is then just to determine the instructions attributed to the residues
- A model can then be built:
  - Does not have explicit attributes derived from the proteins
  - Is independent of the protein size

# Peptide programs

- The concept is analogous to fragment programs in computer graphics
  - For a given material type and lightning conditions a specific small program is ran that produces a consistent look and feel
- Amino acid residues are interpreted as instructions. The same residue type will execute the same functions
- Problems:
  - What is the instruction set?
  - How do I use such model for classification?
  - How it is possible to choose the appropriate instructions for each residue?

## The Instruction set (I)

R0	<
R1	=
R2	≥
R3	
R4	
R5	<b>NOP</b>
R6	+
R7	-

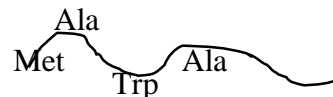
- The first efforts used a very simple virtual register based machine
  - It was fundamental to maintain simplicity for the searching process
  - Each instruction composed of a **condition** and an **operation** (1 byte each)
  - 8 integer registers with 8 bits (signed values) [-128, +127] - clamping occurs above and below
  - add and subtraction operations with values or between registers
- At the end of the program (protein) a number of registers is evaluated, if all are > 0, then the program outputs a positive.

## The Instruction Set (II)

Mnemonic	OpCode	Description
NOCOND	0	No condition
IFS <i>rega, regb</i>	1	IF $regb - B < rega < regb + B$ , do
IFL <i>rega, regb</i>	2	IF $rega < regb$ , do
IFGE <i>rega, regb</i>	3	IF $rega \geq regb$ , do

Mnemonic	OpCode	Description
NOP	0	No operation
ADDR <i>rega, regb</i>	1	$rega = rega + regb$
ADDV <i>reg, val</i>	2	$reg = reg + val$ , where $val \in [-4, 3]$
SUBR <i>rega, regb</i>	3	$rega = rega - regb$

## An example



### A peptide program:

```

Ala
  ifge r1, r3: adr r5, r6
  ifl r0, r8 : adv r1, -3
Met
  nocond      : nop
  ifs r2, r5 : adv r6, 2
Trp
  nocond      : adr r5, r5
  ifge r4, r5: adv r0, 2
...
  
```

With B=0

	t0	Met	Ala	Trp	Ala
R0	0	0	0	2	0
R1	0	0	0	0	-3
R2	0	0	0	0	0
R3	0	0	0	0	0
R4	0	0	0	0	0
R5	0	0	2	4	6
R6	0	2	2	2	2
R7	0	0	0	0	0

## Fitting the instruction set

- Required a combinatorial optimisation method
- Simulated annealing is a very robust local search meta-heuristic:
  1. Start with a random attribution of instructions to each amino acid
  2. Verify how well the model fits into the data
  3. Randomly change the solution
  4. If the error ratio is smaller than the original, accept the solution, or else there is a decreasing probability that a worse solution is accepted
  5. Repeat steps 2-4 for a determined number of iterations

## Implementation details

- Developed in C and Python
- Open Sourced - <http://xldb.di.fc.ul.pt/wiki/Peptides>
- Parameters:
  - Number of SA iterations
  - Number of instructions per residue
  - Number of ending registers
  - Number of bits for each register
  - Buffer value for similarity condition
- Training time: ~ 1min/1000 iterations
- Testing time: ~ 31  $\mu$ secs (average)

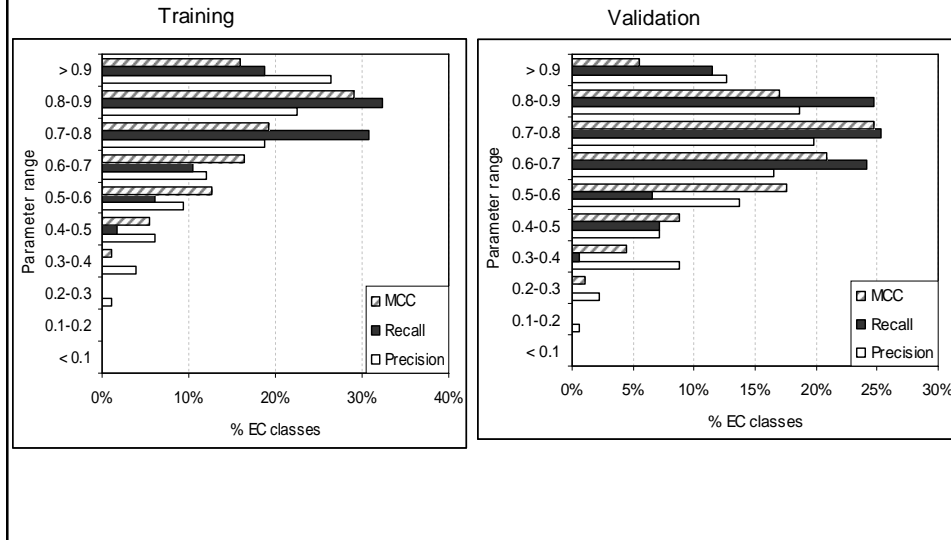
## Data sets

- Data from Uniprot/SwissProt
- 182 different Enzyme Commission (EC) classes with enough number of elements)
- ~33k enzymes on positive datasets for all the classes
  - 25k used for training
  - 8k used for validation
- Negatives datasets included potentially all the definite non-positive elements in SwissProt

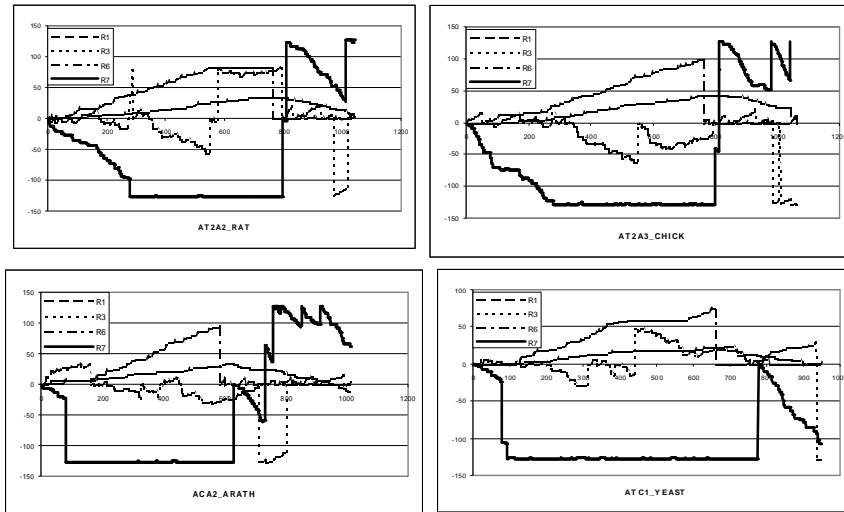
## Global Results

- Average precision:
  - 76% in the training step and
  - 69% in validation
- Average recall
  - 79% in training
  - 74% validation;
- Average Matthews correlation coefficient (MCC)
  - 0.75 in training
  - 0.68 in validation
- Full detailed results:
  - <http://xldb.di.fc.ul.pt/wiki/Peptides>

# Results I



# Register Dynamics



EC:3.6.3.8 (Calcium transporting ATPases).

## Ongoing Work

- Parallelize the algorithm in a cluster of CELL based machines
  - 3 levels of parallelization
- Comparison of PP's results with BLAST in more specific contexts
  - VERY promising results!
- Tightening and experimenting with the instruction set:
  - Reducing the number of possibilities is key

## Future Work

- Fragment programs present a wide range of possibilities for improvement:
  - Testing of different instruction sets, distinct for separate problems
  - Use fragment programs as weak classifiers in a bagged approach
  - Cascading classifiers in order to reduce the number of false positives

## Conclusions

- Fragment programs appear as fast and robust tools for protein classification, providing results comparable to some literature efforts
- Faster than sequence alignment methods! (31  $\mu$ secs average!)
- Very small (average 120 bytes) allowing for complex classification procedures with reduced computational footprints
  - 25,000 PPs could take about 3.0 Mbytes
  - With a single CPU machine each protein could be tested in 0.77 secs

## Acknowledgements

- Research for this project was funded by FCT PhD Scholarship SFRH/BD/29797/2006
- Physical presentation of this research at the CIKM2008/DTMBio Workshop was made possible by FCT Project PTDC/AGR-CFL/64146/2006