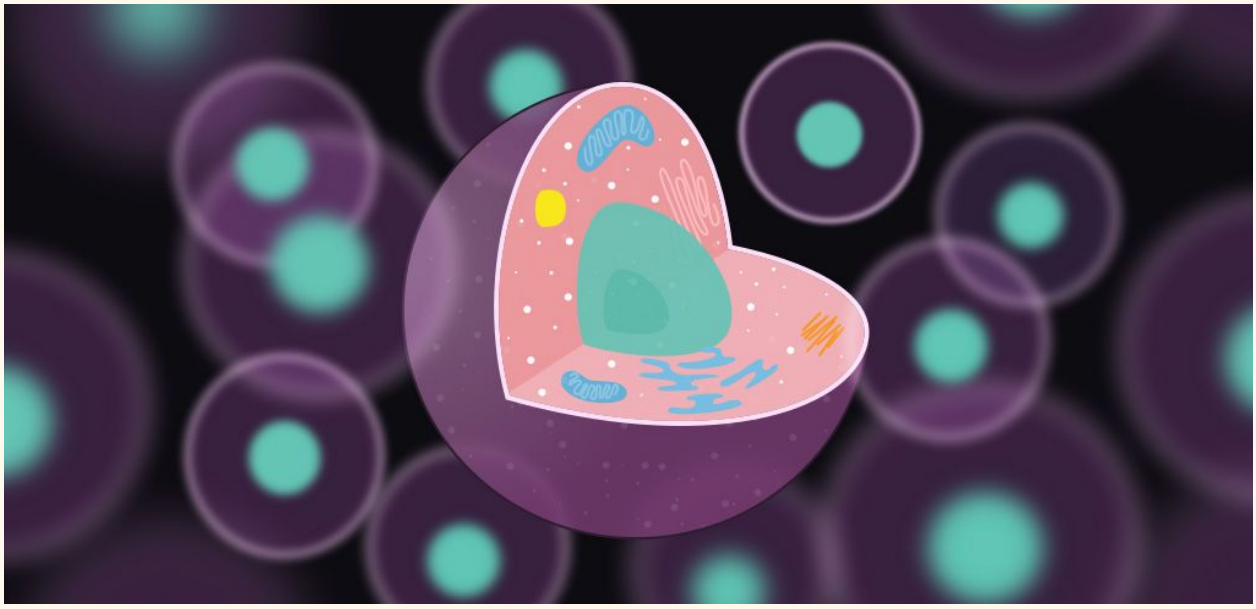
Atul Tyagi, Pallavi Kapoor, Rahul Kumar
Kumardeep Chaudhary, Ankur Gautam and G.P.S. Raghava

IN SILICO MODELS
FOR DESIGNING AND DISCOVERING NOVEL ANTICANCER PEPTIDES

By: Rhythm Nagpal (MT17048), Himanshu Aggarwal (MT17015), Rohit Kumar Verma (MT17145)



ABSTRACT

Biology has entered into a new era of genomics that has far-reaching consequences in human medicine and health. In the past few decades , it has been observed that the use of therapeutic peptides in the field of cancer treatment has received considerable attention . We have tried to implement the paper 'In Silico Models for designing and discovering novel Anticancer Peptides' in order to study the development of computational models which can help in predicting novel anticancer peptides. It has been found that Cys, Gly, Ile, Lys, and Trp are dominated at various

positions in anticancer peptides as similar to the result provided in the paper. In order to build the models for the design and discovery purpose, we have used Support vector machine, Multilayer Perceptron(MLP), Adaboost, Random forest, Quadratic Discriminant Analysis, Extreme Learning machines using amino acid composition and binary profiles and dipeptide composition as input features on main dataset.

INTRODUCTION

In the present world where biology, biological sciences and biological treatments have entered the nook and corner of every household, workplace, restrooms,etc which is all due to our living conditions , cancer has become and is one of the most deadliest diseases. Millions of people around the world are suffering from this chronic disease. It has increased the death toll, in other world the human mortality rate . Chemotherapy treatment of cancer has an adverse effect on the normal cells. Development of resistance to drug targets is a challenge to the cancer treatment. Due to the emerging complexities in diagnosis and treatment, there is an earnest need to come up with sophisticated but reliable solutions.

Anti Cancer Peptides(ACPs) have the property to entangle cancer selective toxicity which has henceforth replaced the conventional way of treatment of cancer i.e., Chemotherapy. It has been found that the peptide drug designing has revolutionized the field of peptide-based drug designing.

ACPs are small (5–30 amino acids) peptides and are basic in nature. ACPs are mainly derived from the antimicrobial peptides (AMPs) and therefore they are in similar in nature to AMPs. The surface of the cancer cells are negatively charged as similar to those of bacterial type of cells and therefore the antimicrobial peptides show a broad spectrum toxicity towards the cancer cells. ACPs interact electrostatically with the negatively charged membrane of cancer cells and therefore it plays an important role in cancer selective toxicity of ACPs. ACPs are further classified into mainly two groups i.e., one those are toxic to cancer cells and bacterial cells but not toxic to the normal cells and second one which are toxic to cancer cells, bacterial cells and as well as normal cells.

Till date, ACPs have not been studied extensively as the mechanism is not fully read so far. It has been found that most of the ACPs like AMPs performs membrane-lytic mode of action.

Furthermore, it has been found that few of ACPs are performing other modes of action such as apoptosis that includes the destruction of mitochondrial membrane. Researchers have found that the selective toxicity of many anticancer peptides towards cancer cells is found to be the difference in the lipid content and other components of normal cells and cancer cells. Peptide-based therapy has numerous advantages over small molecules that involve high specificity, low production cost, high tumor penetration, ease of synthesis and modification etc. In addition, few ACPs induce apoptosis (program cell death) by disrupting mitochondrial membrane when delivered into the cancer cells. Many peptide-based therapies to treat various tumor types are currently being evaluated in various phases of preclinical and clinical trials. The success of these peptides in clinics has open the door for ACPs to reach clinical settings.
We have implemented the paper in an attempt to develop in silico methods for the prediction and designing of ACPs. In order to build the models we have used Support vector machine, multilayer perceptron(MLP), AdaBoost, Random Forest, Quadratic Discriminant analysis, and Extreme learning machine using various features of peptides like amino acid composition, dipeptide composition and binary profile patterns.

BACKGROUND

Some of the papers have discussed a least squares version for support vector machine (SVM) classifiers. Due to equality type constraints in the formulation, the solution follows from solving a set of linear equations, instead of quadratic programming for classical SVM''s. The approach is illustrated on a two-spiral benchmark classification problem.

Others suggest that artificial Neural networks are outstanding tools able to generate generalizable models in many disciplines. They present the multi-layer perceptron (MLP) which is the most common neural network.

Previous work has shown that the difficulties in learning deep generative or discriminative models can be overcome by an initial unsupervised learning step that maps inputs to useful intermediate representations. Some introduce and motivate a new training principle for unsupervised learning of a representation based on the idea of making the learned representations robust to partial corruption of the input pattern. This approach can be used to train autoencoders, and these denoising autoencoders can be stacked to initialize deep architectures. The algorithm can be motivated from a manifold learning and information theoretic perspective or from a generative model perspective. Comparative experiments clearly

show the surprising advantage of corrupting the input of autoencoders on a pattern classification benchmark suite

It has been found that potential peptides are used in cancer treatment and it has been evident from a variety of different strategies that are available to address the progression of tumor growth and propagation of the disease. Use of peptides that can directly target cancer cells without affecting normal cells (targeted therapy) is evolving as an alternate strategy to conventional chemotherapy. Peptides can be utilized directly as a cytotoxic agent through various mechanisms or can act as a carrier of cytotoxic agents and radioisotopes by specifically targeting cancer cells. Peptide-based hormonal therapy has been extensively studied and utilized for the treatment of breast and prostate cancers. Tremendous amount of clinical data is currently available attesting to the efficiency of peptide-based cancer vaccines.
The decreasing number of approved drugs produced by the pharmaceutical industry, which has been accompanied by increasing expenses for R&D, demands alternative approaches to increase pharmaceutical R&D productivity. This situation has contributed to a revival of interest in peptides as potential drug candidates. New synthetic strategies for limiting metabolism and alternative routes of administration have emerged in recent years and resulted in a large number of peptide-based drugs that are now being marketed. The report is about considerable number of peptides that are currently available as drugs and the chemical strategies that were used to bring them into the market.

IMPLEMENTATION

We have calculated the frequency of amino acids in the main dataset in which we have tried to differentiate ACPs from non-ACPs, as well as ACPs from AMPs. Firstly, we have determined the frequency of occurrence of all 20 amino acids in these peptides. For this, percent average composition of amino acids in ACPs, non-ACPs (random peptides) and AMPs were calculated and compared and are results are quite similar with the paper as shown in Fig 1. Certain residues, including Gly, Lys, Cys, Phe, Ile, and Trp were found to be abundant in ACPs compared to non-ACPs while Gly, Ala, Lys and Leu were abundant in AMPs compared to ACPs

and non-ACPs. Since terminal residues play crucial roles in biological functions of peptides, we computed and compared the percent average amino acid composition of N-terminal and C-terminal residues (split amino acid composition) in these peptides. As shown in Fig 2, average amino acid compositions of terminal residues are more or less similar to whole amino acid composition. However, among N-terminal residues, only Cys was found to be in a higher proportion in ACPs compared to both AMPs and non-ACPs. In C-terminal residue analysis, Tyr and Trp were found to abundant in ACPs compared to both AMPs and non-ACPs (Fig 2B).

Peptide information was encapsulated in a vector of 20 dimensions, using amino acid composition of the peptide. The amino acid composition is the fraction of each amino acid type within a peptide. The fractions of all 20 natural amino acids were calculated by using the following equation:

$$Comp(i)=(Ri/N)\times 100$$

Where Comp (i) is the percent composition of amino acid (i); R i is number of residues of type i, and N is the total number of peptide residues in the data.

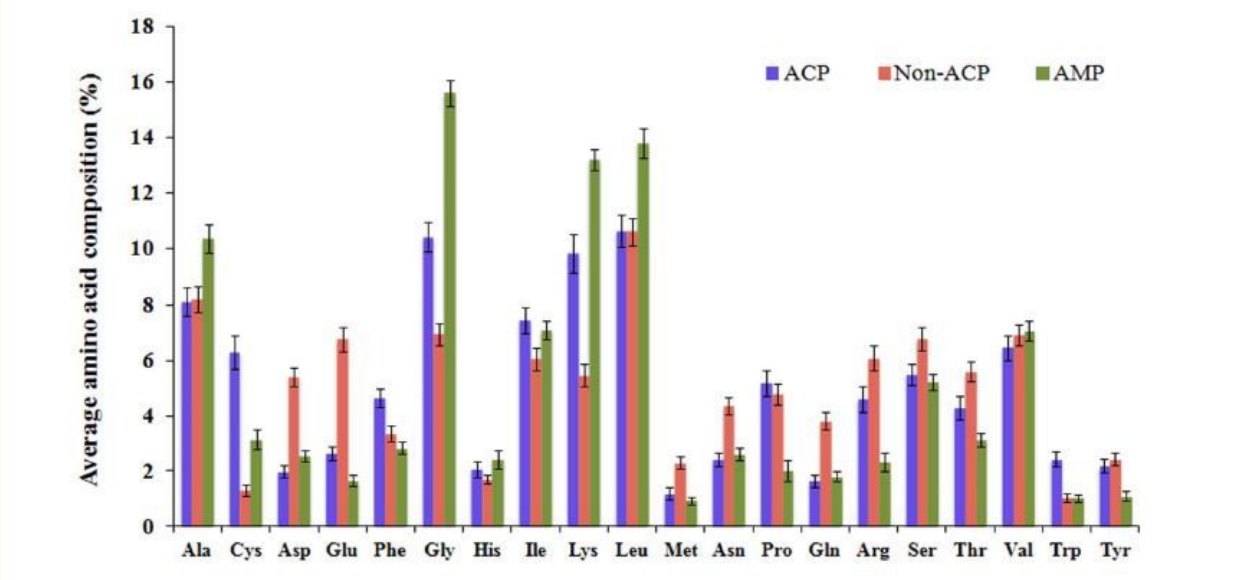
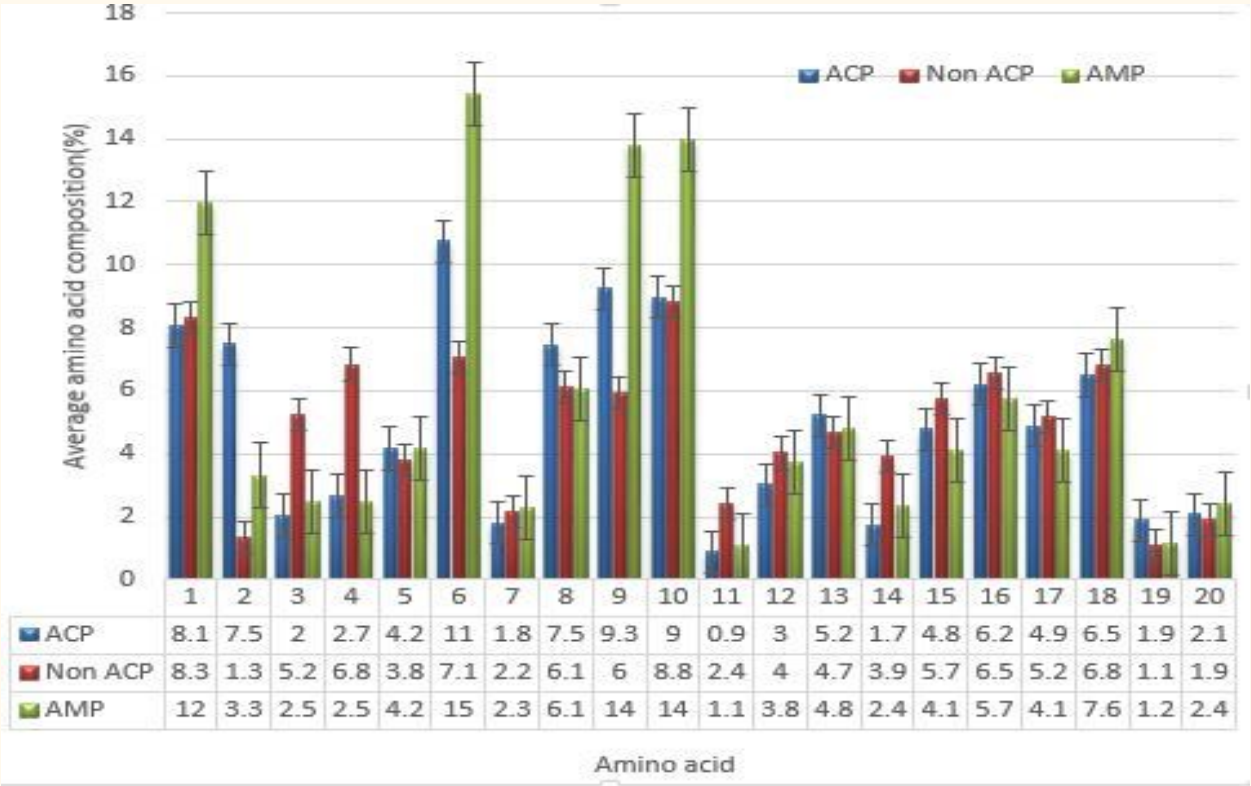


Fig (from paper): Comparison of average whole amino acid composition of anticancer, non-anticancer, and antimicrobial peptides.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
ACP	8.1	7.5	2	2.7	4.2	11	1.8	7.5	9.3	9	0.9	3	5.2	1.7	4.8	6.2	4.9	6.5	1.9	2.1
Non ACP	8.3	1.3	5.2	6.8	3.8	7.1	2.2	6.1	6	8.8	2.4	4	4.7	3.9	5.7	6.5	5.2	6.8	1.1	1.9
AMP	12	3.3	2.5	2.5	4.2	15	2.3	6.1	14	14	1.1	3.8	4.8	2.4	4.1	5.7	4.1	7.6	1.2	2.4

Fig (Our result). Comparison of average whole amino acid composition of anticancer, non-anticancer, and antimicrobial peptides.

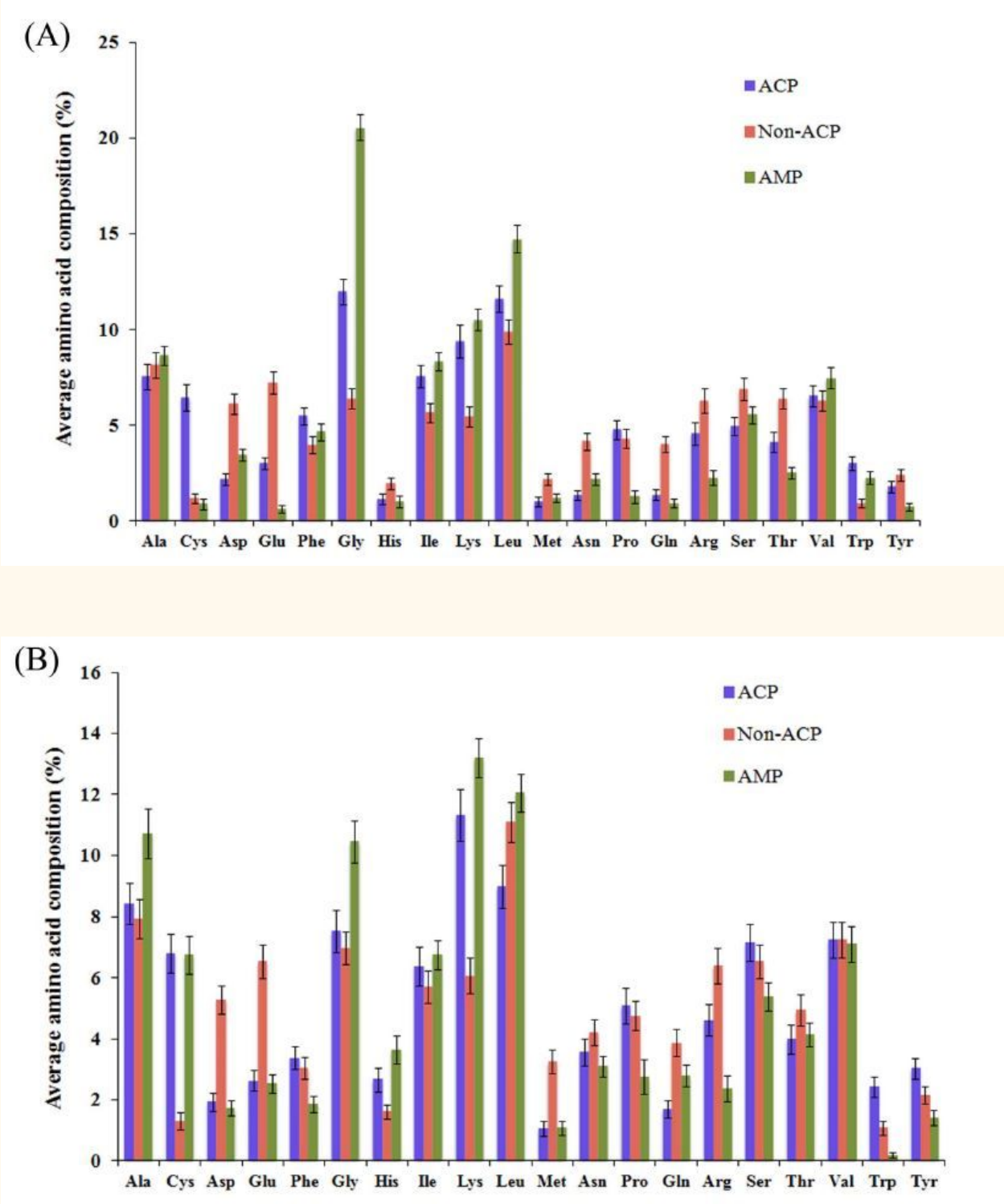
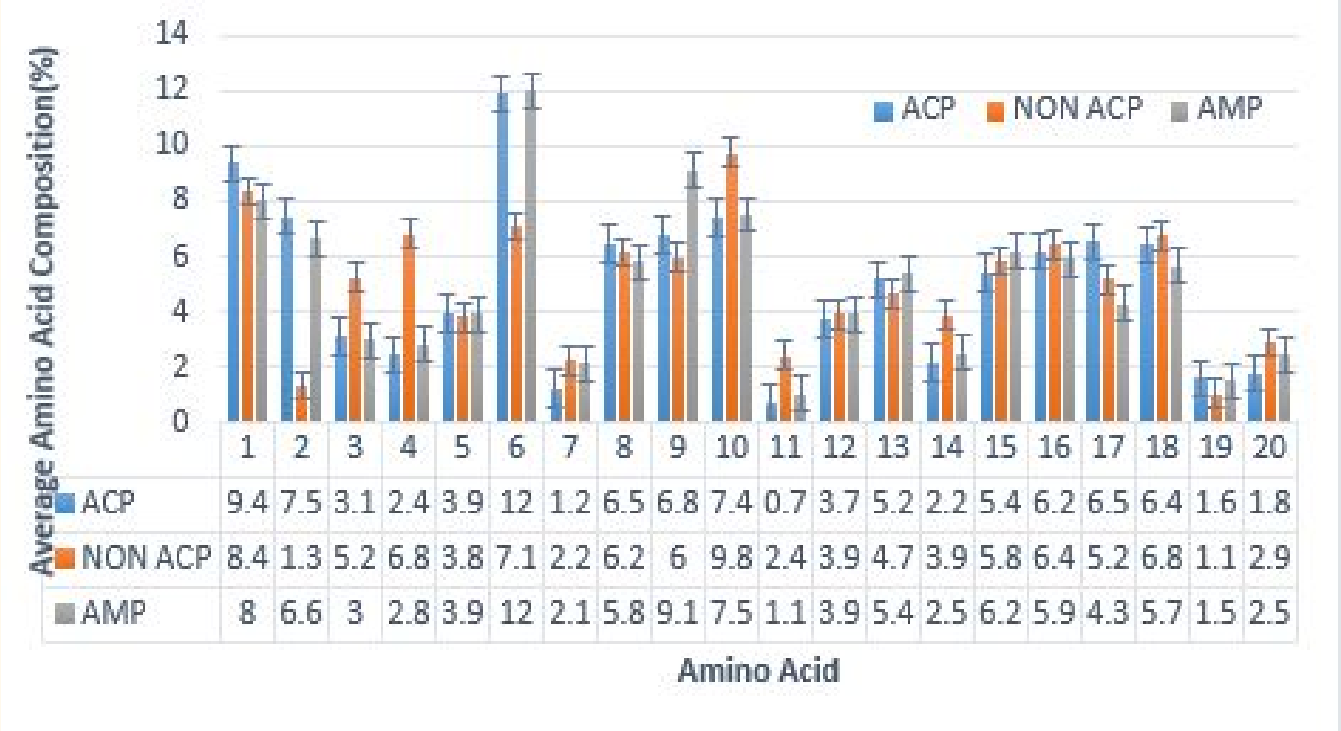


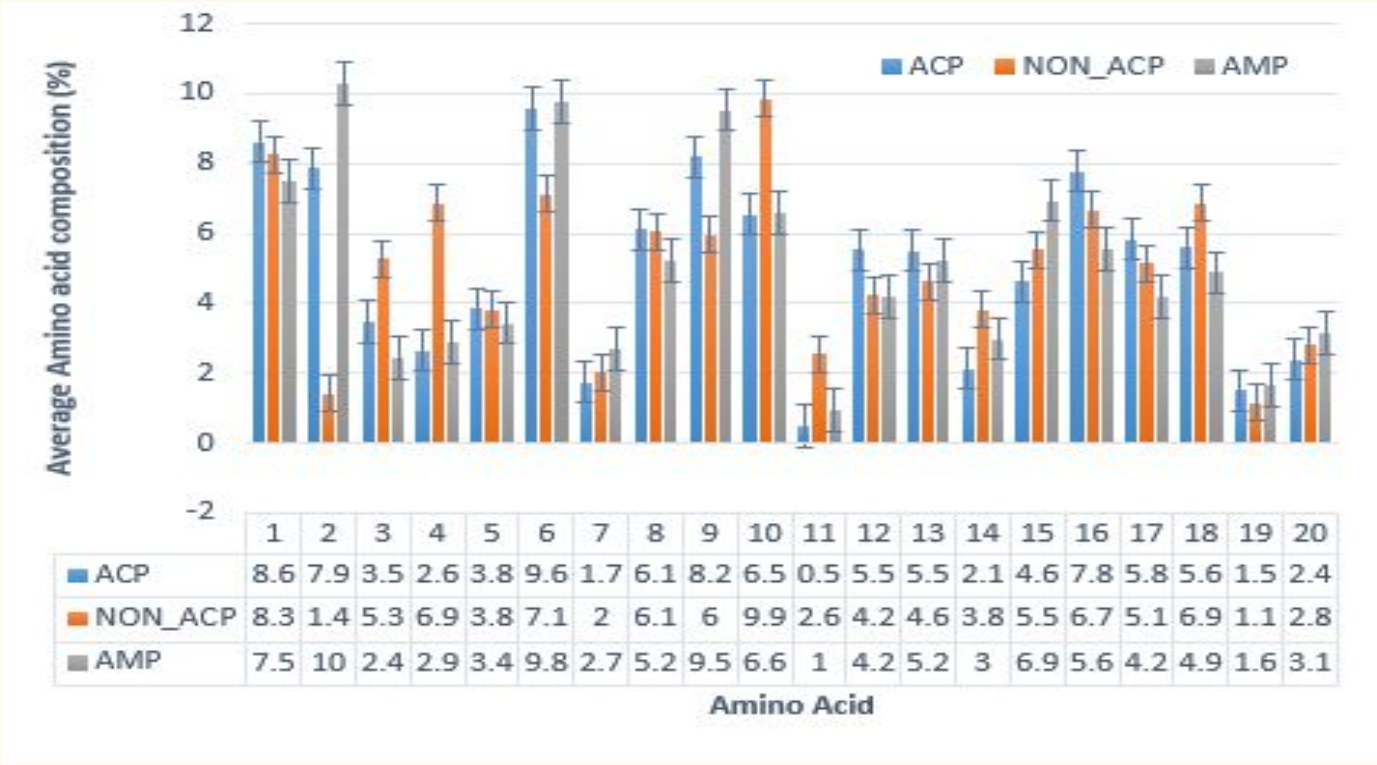
Fig (from paper). Comparison of average amino acid composition of ten (A) N- and (B) C-terminal residues of anticancer, non-anticancer, and antimicrobial peptides.

(A)



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
ACP	9.4	7.5	3.1	2.4	3.9	12	1.2	6.5	6.8	7.4	0.7	3.7	5.2	2.2	5.4	6.2	6.5	6.4	1.6	1.8
NON ACP	8.4	1.3	5.2	6.8	3.8	7.1	2.2	6.2	6	9.8	2.4	3.9	4.7	3.9	5.8	6.4	5.2	6.8	1.1	2.9
AMP	8	6.6	3	2.8	3.9	12	2.1	5.8	9.1	7.5	1.1	3.9	5.4	2.5	6.2	5.9	4.3	5.7	1.5	2.5

(B)

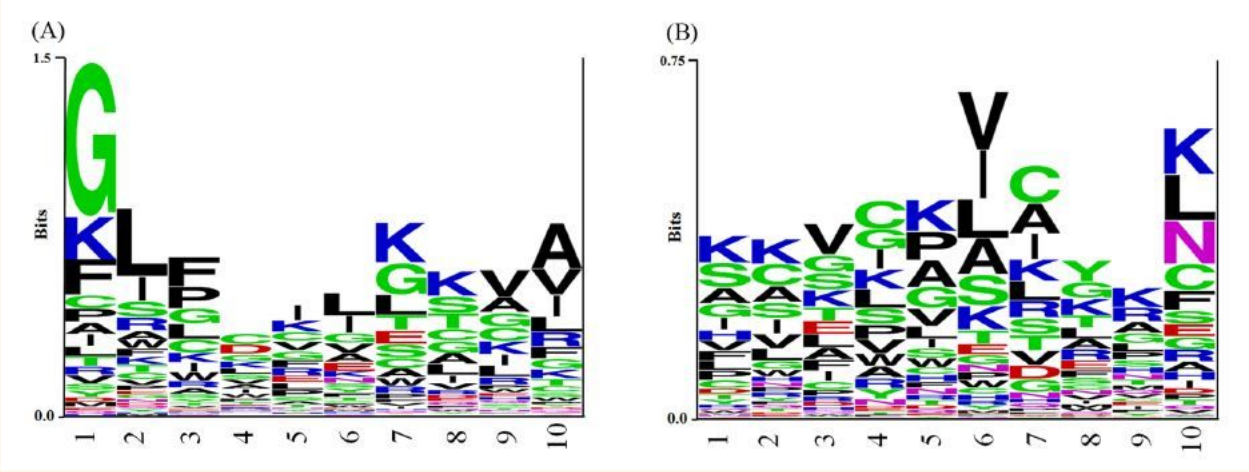


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
ACP	8.6	7.9	3.5	2.6	3.8	9.6	1.7	6.1	8.2	6.5	0.5	5.5	5.5	2.1	4.6	7.8	5.8	5.6	1.5	2.4
NON_ACP	8.3	1.4	5.3	6.9	3.8	7.1	2	6.1	6	9.9	2.6	4.2	4.6	3.8	5.5	6.7	5.1	6.9	1.1	2.8
AMP	7.5	10	2.4	2.9	3.4	9.8	2.7	5.2	9.5	6.6	1	4.2	5.2	3	6.9	5.6	4.2	4.9	1.6	3.1

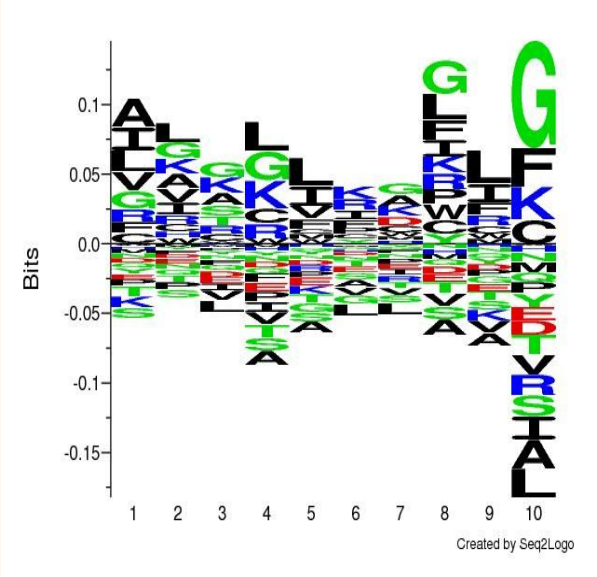
Fig (Our results) Comparison of average amino acid composition of ten (A) N- and (B) C-terminal residue of anticancer, non-anticancer, and antimicrobial peptides.

Residue Preference

We have programmatically illustrated the computation of residue preference at both termini of peptides. Sequence logos are a well formatted and labelled graphical representation depicting residue preference. The sequence logos of 10 N-terminal and 10 C-terminal residues are shown in the figure below. As shown, no exclusive preference of residues was observed except Gly at the first position at N-terminus. However, there are few residues like Leu, Lys, Ala and Phe at N-terminus and Val, Cys, Leu and Lys at C-terminus which are also preferred but relatively less preferred than Gly at various positions. We have found almost similar results with the paper. The sequence logos, which provide information about the position specific frequency of amino acids in peptide, were generated using the WebLogo software 3. The sequence logos give the position specific frequency of amino acids in peptides. Each logo consists of stacks of symbols, one stack for each position in the sequence. The overall height of the stack indicates the sequence conservation at that position, while the height of symbols within the stack indicates the relative frequency of each amino acid at that position.



(Results from the paper)



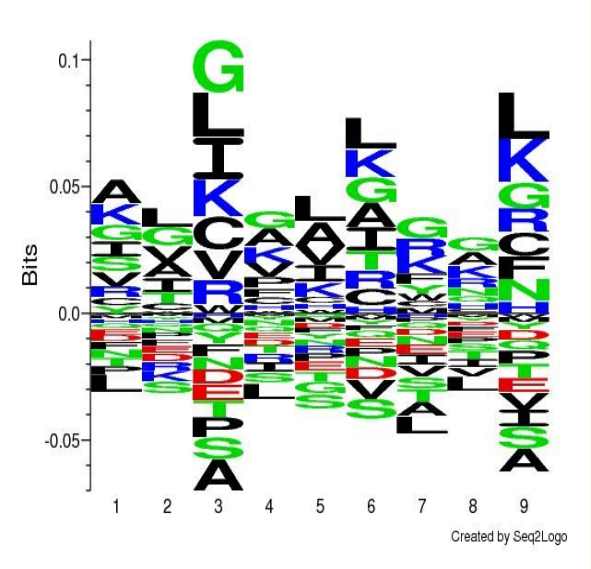


Fig. Sequence logo (A) N-Terminal (B) C-Terminal

(Our Results)

Residue Preference

The dipeptide composition provides composition of pair of residues (e.g. Ala-Ala, Ala-Leu, etc.) present in peptide, and are used to transform the variable length of peptides to fixed length vectors. It gives a fixed pattern length of 400 (20 × 20), and encapsulates information about the fraction of amino acids as well as their local order. It is calculated using following equation: Fraction of Dipeptide(i) =Total number of Dipeptide(i)/Total number of all possible dipeptides Where dipeptide (i) is one out of 400 dipeptides.

Binary Profile of Patterns

Binary profiles were generated for each peptide, where each amino acid is represented by a

vector of dimensions of 20 (e.g. Ala by 1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0). A pattern of window length W was represented by a vector of dimensions 20 × W. We have created binary profile for first 5 and 10 residues from N-terminus, similarly for last 5 and 10 residues from C-terminus of peptides in all datasets. The binary profile has been used in a number of existing methods.

Support Vector Machine Models

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

Multilayer Perceptron

A multilayer perceptron (MLP) is a class of feedforward artificial neural network. An MLP consists of at least three layers of nodes. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable. Multilayer perceptrons are sometimes colloquially referred to as “vanilla” neural networks, especially when they have a single hidden layer.

Adaboost

AdaBoost, short for Adaptive Boosting, is a machine learning meta- algorithm formulated by Yoav Freund and Robert Schapire, who won the 2003 Godel Prize for their work.It can be used in conjunction with many other types of learning algorithms to improve performance. The output

of the other learning algorithms(‘weak learners’) is combined into a weighted sum that represents the final output of the boosted classifier. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. AdaBoost is sensitive to noisy data and outliers. In some problems it can be less susceptible to the overfitting problem than other learning algorithms. The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing, the final model can be proven to a strong learner.

Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes(classification) or mean prediction(regression) of the individual trees. Random decision forests correct for decision trees’ habit of overfitting to their training set.

Quadratic Discriminant Analysis

A quadratic classifier is used in machine learning and statistical classification to separate measurements of two or more classes of objects or events by a quadric surface. It is a more general version of the linear classifier.

Extreme learning machine

Extreme learning machines are feedforward neural networks for classification, regression, clustering, sparse approximation, compression and feature learning with a single layer or multiple layers of hidden nodes, where the parameters of hidden nodes (not just the weights connecting inputs to hidden nodes) need not be tuned. These hidden nodes can be randomly assigned and never updated (i.e. they are random projection but with nonlinear transforms), or can be inherited from their ancestors without being changed. In most cases, the output weights of hidden nodes are usually learned in a single step, which essentially amounts to learning a linear model.

DATASETS

We have used the datasets which is 225 experimentally validated anticancer peptides from the

(http://crdd.osdd.net/raghava/anticp/). AMPs with a broad spectrum anticancer activities. All these peptides were unique and considered as positive examples. For non-anticancer peptides, we have used 2250 random peptides from the datasets. We have used AMPs datasets which were around 1375 from the dataset which were extracted from above databases like APD, CAMP.

1. Main Dataset

This dataset contains 225 experimentally validated anticancer (positive examples) and 2250 random or potential non-anticancer peptides (negative examples).

2. Alternate Dataset

This dataset contains 225 experimentally validated anticancer peptides and 1372 non-anticancer (AMPs without anticancer activities, negative examples).

3. Balanced Dataset

It is a well known fact that classification techniques, particularly machine learning techniques performed best on balanced datasets. The main balanced dataset contains 225 anticancer and 225 non-anticancer or random peptides (randomly obtained 2250 SwissProt peptides). Similarly, alternate balanced dataset contains 225 anticancer and 225 non-anticancer or AMPs (randomly obtained from 1372 AMPs).

4. Independent Dataset

This dataset contains 50 experimentally validated ACPs and an equal number of random peptides considered as negative examples. None of the peptides in independent dataset is identical to peptides in training or testing dataset.

Residue composition as input features

In order to develop the various models based on machine learning techniques, one needs fixed length input features. So we have fixed the datasets similar to the information provided in the paper. The dataset contains peptides of variable length; thus we have computed composition profile of peptides. In this study, we computed amino acid and dipeptide composition where information is encapsulated in a vector of 20 and 400 dimensions respectively. The calculation of amino acid and dipeptide composition was described previously.

Binary profile of patterns

Binary profiles is a key feature and has been used in a number of existing methods. It encapsulates information of both composition and order of amino acid in peptides. Therefore, binary profiles for first 5 and 10 residues from N- and C-terminus were generated for each peptide, where each amino acid is represented by a vector of dimensions of 20 (e.g. Ala by 1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) as described previously.

PERFORMANCE MEASURE

Cross-validation technique

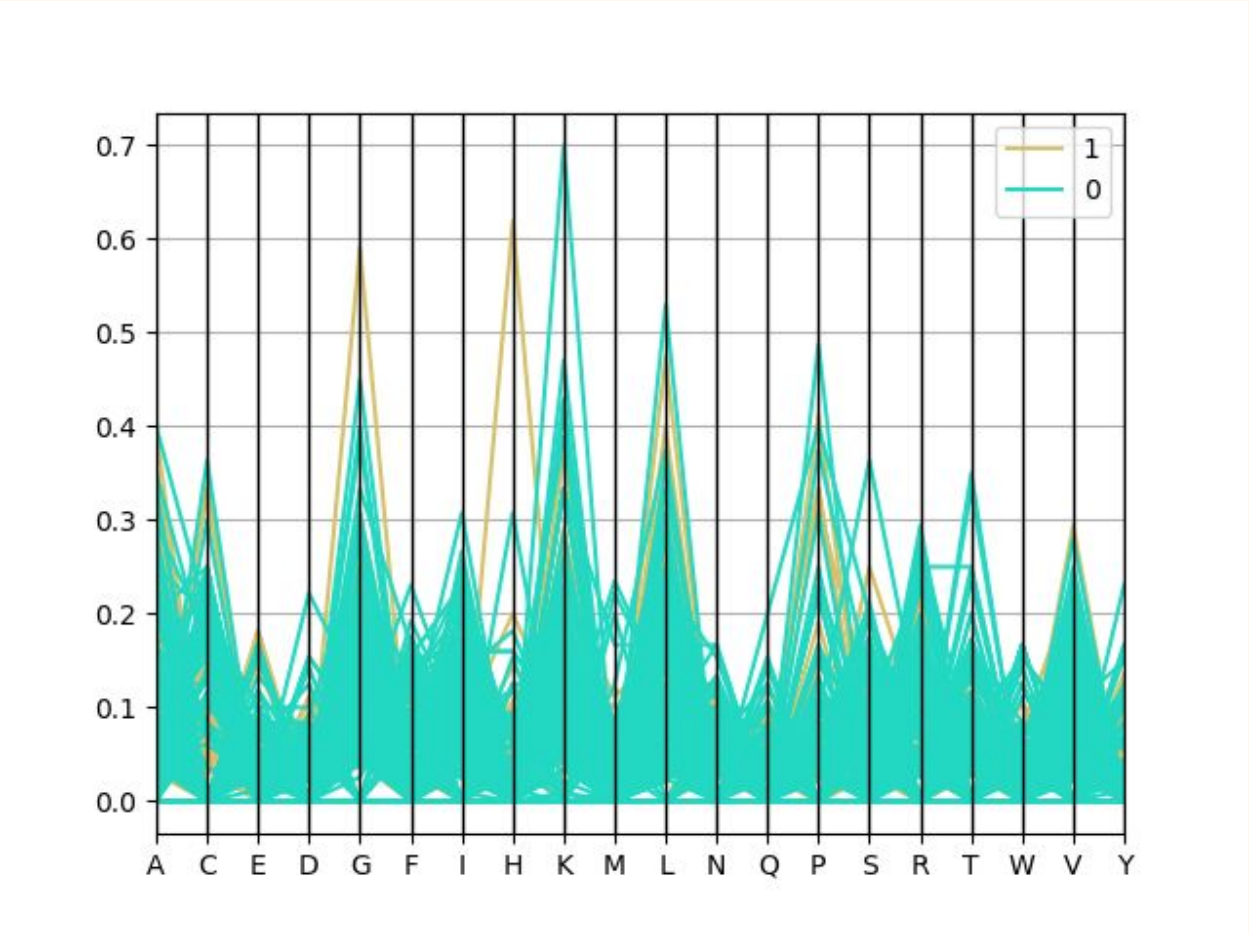
The validation of any prediction method is very essential part. In the present study, five-fold cross-validation technique was used to evaluate the performance of all the models. Here, sequences are randomly divided into five sets, of which four sets are used for training and the remaining fifth set for testing. The process is repeated five times in such a way that each set is used once for testing. In this technique, one sample is used for testing and remaining samples for training, this process is repeated in such a manner that each sample is used only once for testing.

RESULTS

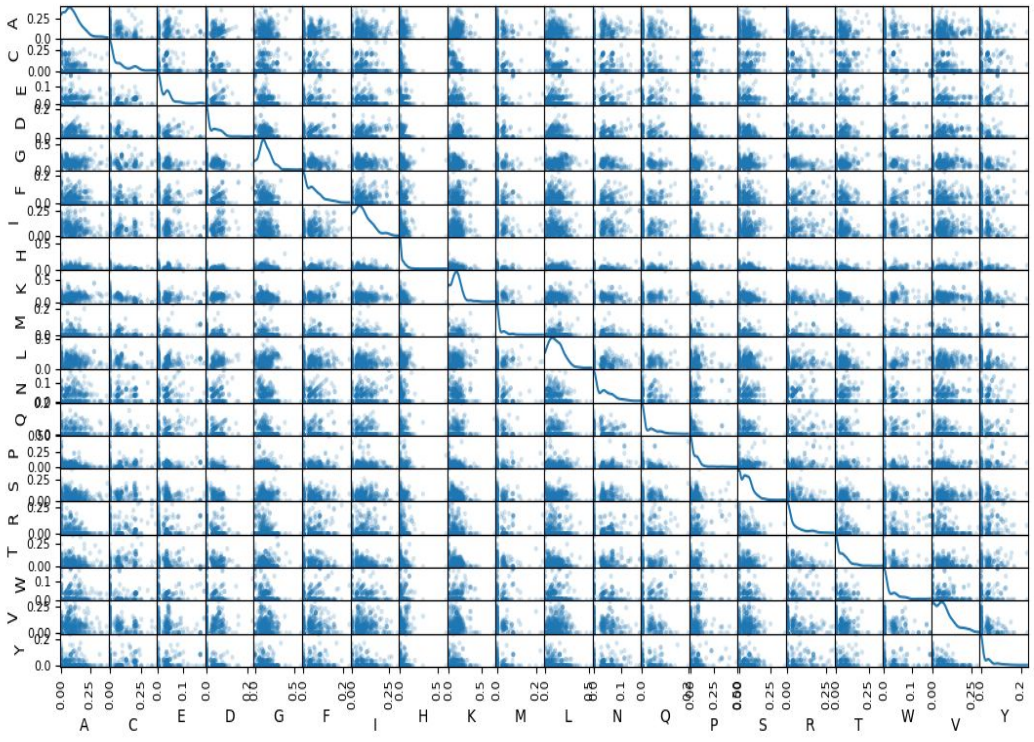
The above explained models were trained separately on all the features extracted from the data, i.e. amino acid composition, dipeptide composition, and binary profiles of the peptides. Models are trained on each feature separately and results from them have been observed, which are as follows.

Firstly, the data at hand was observed thoroughly. The features formed from the data is also observed for analysis of important features within and to understand if the data is even distinguishable.

Below plot shows the importance among the features in the vector representing the peptides by amino acid composition for the ‘main’ dataset. Here, the legend 0 represent ACP and 1 represent AMP.



Also, interfeature relation is also observed by the below graph. However, this information is not used fully for training the models. But this can be very useful for the future development.



This is again for the ‘main’ dataset. This shows the relationship between the features and the dependencies between them can be studied from this.

Now, once we understood the data, we developed few commonly used and few advanced machine learning models for the classification purpose. We applied MLP classifier, Support Vector Machine, Adaboost classifier, Random Forest Model and Quadratic discriminant analysis. Some of these models outperformed the existing models mentioned in the paper. Below are few results:

These are the results for amino acid composition feature on several datasets.

Alternate Dataset					
	Accuracy	MCC	AUC	Sensitivity	Specificity
MLP	0.8690702087	0.291959646	0.5742516296	0.9890829694	0.1594202899
SVM	0.834323344	0.305541	0.5234499	0.98434355	0.234553
ABC	0.8523423232	0.31345334	0.6104422	0.9752343434	0.184543322
RFC	0.8956356736	0.4387562143	0.6506866654	0.9825327511	0.3188405797
QDA	0.8690702087	0.4387562143	0.5	0.9825327511	0.3188405797

Main Dataset					
	Accuracy	MCC	AUC	Sensitivity	Specificity
MLP	0.9424724602	0.5835527461	0.7515112764	0.9812834225	0.5217391304
SVM	0.9424724602	0.5567600696	0.7054657832	0.9906417112	0.4202898551
ABC	0.9326805386	0.5273269791	0.7395857553	0.9719251337	0.5072463768
RFC	0.9571603427	0.6853885457	0.7595326668	0.9973262032	0.5217391304
QDA	0.9155446756	0	0.5	1	0

Balanced 1					
	Accuracy	MCC	AUC	Sensitivity	Specificity
MLP	0.8187919463	0.6382480173	0.8191240087	0.8026315789	0.8356164384
SVM	0.8590604027	0.7183616632	0.8585976929	0.8815789474	0.8356164384
ABC	0.8657718121	0.7315536081	0.8654470079	0.8815789474	0.8493150685
RFC	0.9127516779	0.8255541673	0.912851478	0.9078947368	0.9178082192
QDA	0.8791946309	0.7582912761	0.8791456381	0.8815789474	0.8767123288

Balanced 2					
	Accuracy	MCC	AUC	Sensitivity	Specificity
ELM	0.9461444308	0.5955183983	0.7337828412	0.9893048128	0.4782608696
MLP	0.6979865772	0.4161040171	0.6944844989	0.8684210526	0.5205479452
SVM	0.8120805369	0.6310861702	0.8103821197	0.8947368421	0.7260273973
ABC	0.8389261745	0.6794140293	0.8380497477	0.8815789474	0.7945205479
RFC	0.8523489933	0.7052787296	0.8517483778	0.8815789474	0.8219178082
QDA	0.7919463087	0.5943101137	0.7898341745	0.8947368421	0.6849315068

Main					
	Accuracy	MCC	AUC	Sensitivity	Specificity
ELM	0.9461444308	0.5955183983	0.7337828412	0.9893048128	0.4782608696
MLP	0.9424724602	0.5835527461	0.7515112764	0.9812834225	0.5217391304
SVM	0.9424724602	0.5567600696	0.7054657832	0.9906417112	0.4202898551
ABC	0.9326805386	0.5273269791	0.7395857553	0.9719251337	0.5072463768
RFC	0.9571603427	0.6853885457	0.7595326668	0.9973262032	0.5217391304
QDA	0.9155446756	0	0.5	1	0

Similar was done with the dipeptide composition feature.

ALternate Dataset					
	Accuracy	MCC	AUC	Sensitivity	Specificity
ELM	0.8690702087	0.3772644483	0.6723308651	0.9388646288	0.4057971014
MLP	0.8823529412	0.3912910484	0.6553540915	0.9628820961	0.347826087
SVM	0.8690702087	0.132254	0.5542	0.997342223	0.321023444
ABC	0.8747628083	0.2006477485	0.5340484779	0.9956331878	0.07246376812
RFC	0.893738140	0.433704448	0.655749636	0.978165938	0.333333333

	4		1	9	3
QDA	0.8633776091	0.02003982155	0.5028795646	0.9912663755	0.01449275362

Balanced 1					
	Accuracy	**MCC**	**AUC**	**Sensitivity**	**Specificity**
ELM	0.7718120805	0.5434390771	0.7717195386	0.7763157895	0.7671232877
MLP	0.8791946309	0.7596466415	0.8796863735	0.8552631579	0.904109589
SVM	0.8657718121	0.7322032827	0.8651766402	0.8947368421	0.8356164384
ABC	0.8187919463	0.6375923693	0.818853641	0.8157894737	0.8219178082
RFC	0.8791946309	0.7596466415	0.8796863735	0.8552631579	0.904109589
QDA	0.4832214765	-0.02898537633	0.4861211247	0.3421052632	0.6301369863

Balanced 2					
	Accuracy	**MCC**	**AUC**	**Sensitivity**	**Specificity**
ELM	0.7919463087	0.5855369226	0.7909156453	0.8421052632	0.7397260274
MLP	0.8590604027	0.717994998	0.8588680606	0.8684210526	0.8493150685
SVM	0.8657718121	0.7333904038	0.8649062725	0.9078947368	0.8219178082
ABC	0.8523489933	0.7064022165	0.8514780101	0.8947368421	0.8082191781
RFC	0.8657718121	0.7333904038	0.8649062725	0.9078947368	0.8219178082
QDA	0.5771812081	0.1530957753	0.5760634463	0.6315789474	0.5205479452

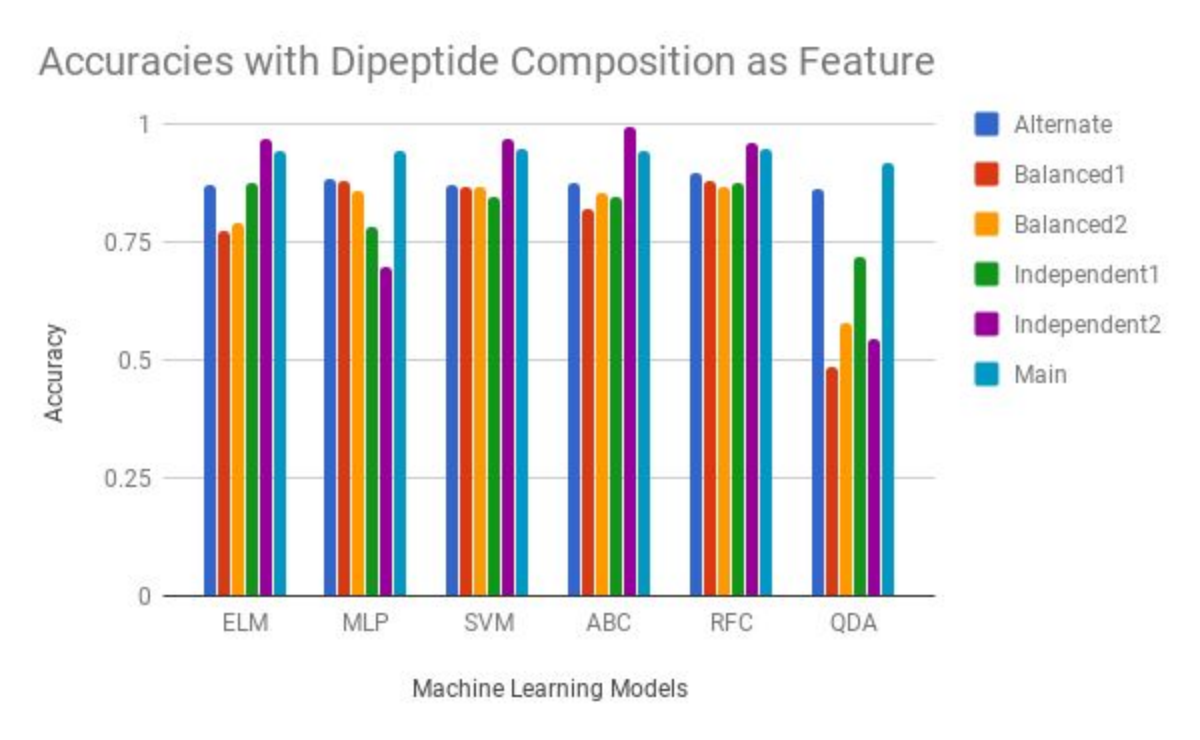
Main					

	Accuracy	MCC	AUC	Sensitivity	Specificity
ELM	0.9436964504	0.5789253969	0.7324459428	0.986631016	0.4782608696
MLP	0.94124847	0.6151027977	0.8034662482	0.9692513369	0.6376811594
SVM	0.9485924113	0.618350424	0.7482755948	0.9893048128	0.5072463768
ABC	0.94124847	0.5592128834	0.7245311168	0.9852941176	0.4637681159
RFC	0.9461444308	0.5849700985	0.6811594203	1	0.3623188406
QDA	0.9155446756	0	0.5	1	0

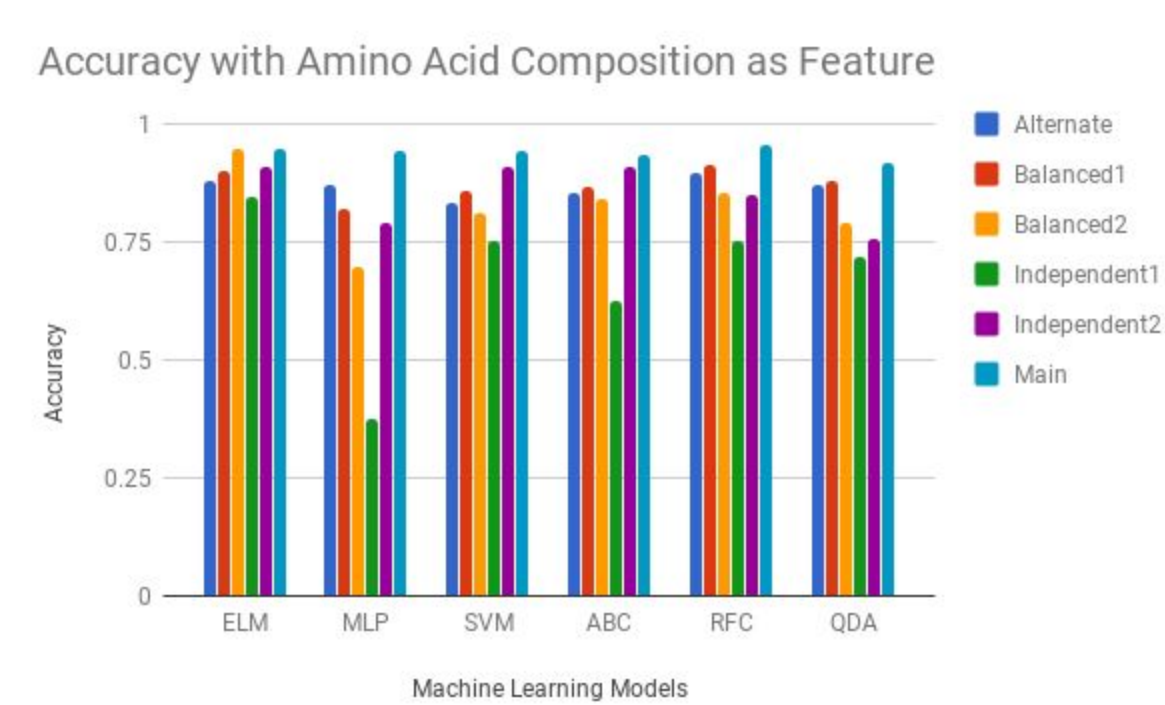
Results obtained were good and also showing improvement over the existing results, but in order to develop an even better model, we headed to deep learning models.

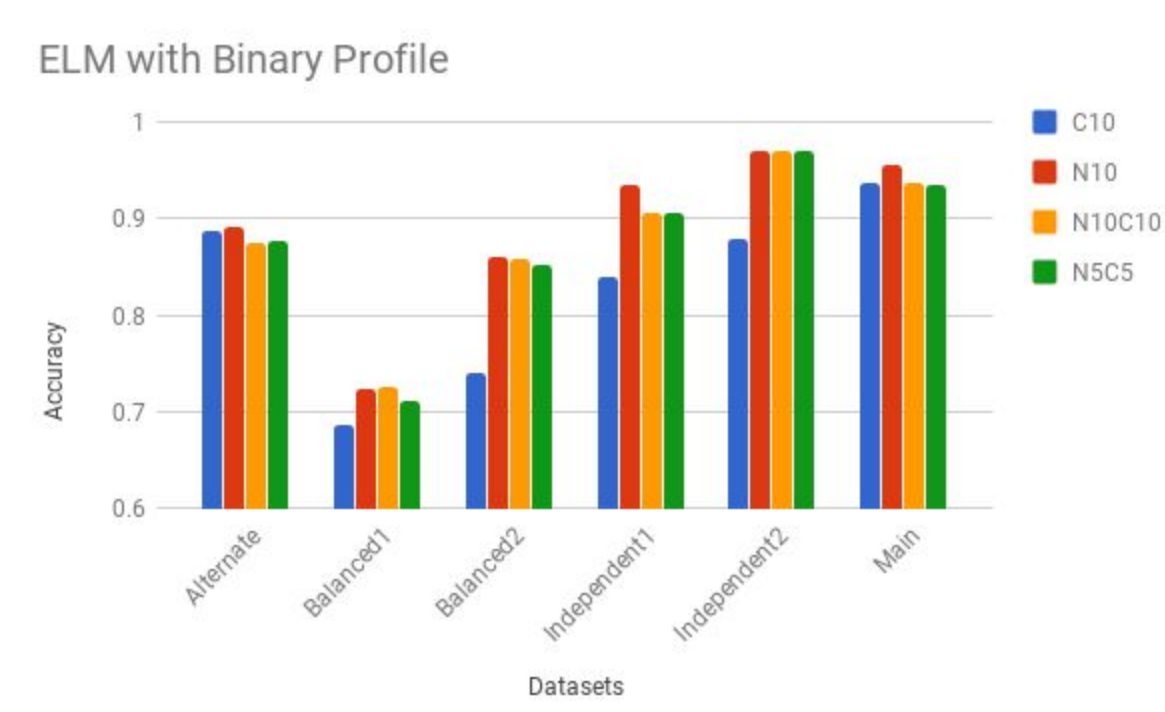
Extreme Learning Machine is a very useful deep learning model. It is a feedforward neural network used for the purpose of classification and regression. As expected, the results obtained from this model outperformed our own results from previous models.

Below are the plots showing the comparison of the accuracies obtained from the models used.



As can be seen from the plot above, all the models performed similar, but Random forest, SVM and ELM performed really well on the data. However, with the binary profile feature, the ELM model showed increase in accuracies of around 5%-7% from the results mentioned in the paper.


Accuracy with Amino Acid Composition as Feature
1
0.75
0.5
0.25
0
Accuracy
ELM
MLP
SVM
ABC
RFC
QDA
Machine Learning Models
Alternate
Balanced1
Balanced2
Independent1
Independent2
Main


ELM with Binary Profile
1
0.9
0.8
0.7
0.6
Accuracy
Alternate
Balanced1
Balanced2
Independent1
Independent2
Main
Datasets
C10
N10
N10C10
N5C5

PROBLEMS FACED AND HOW WERE THEY RESOLVED

The paper turned out to be much more difficult than anticipated, so we had to spend a considerable amount of time in understanding it. We had multiple weekly meetings with our group members to get a deeper insight into the paper. We have attempted to get the results of the datasets mentioned in the paper using the data set stated in the paper.
Now, the challenge that we're facing is

FUTURE WORK

There are a couple of things we would like to accomplish in future work aspect. Firstly, we would like to find the rest of the ACPs which has been found after the webserver created so that the research community will get updated. Regular update should be maintained order to train the ACPs and the Non ACPs. Currently we have 225 ACPs and 2250 non ACPs and we have used the classifier to the provided data sets. Secondly, after

CONCLUSION

There is a rapid growth in the field of peptide based drug designing and research in response to the demand for novel drug delivery systems. AntiCP is one such efficient method.
Highly efficient ACPs and help to find newer ACP analogues more speedily and conveniently. We hope that establishment of such method will speed up the pace of identifying improved and efficacious ACPs in future. The present study demonstrates that features like amino acid composition, binary profile, dipeptide can be used to train an SVM classifier that can predict cancer peptides with higher accuracy. The hybrid model described in this study achieved more accuracy than the previous methods and thus may complement the existing methods. Based on the above study, a user- friendly web server AntiCP has been developed to help the biologists,

where a user can predict and design ACPs with much ease. AntiCP web server is freely accessible at (http://crdd.osdd.net/raghava/anticp/).

PYTHON CODE SNIPPETS

Feature Selection code

```
# ============================================================================
# Single Composition Festures
# ============================================================================

def getComposition(sequences):
    # single peptide count
    composition = []
    for s in sequences:
        if (any(i in s for i in na)):
            continue;
        dtemp = {}
        dtemp.update(d)
        dtemp.update(Counter(s))
        l = len(s)
        dtemp = {k: v/float(l) for k,v in dtemp.iteritems()}
        composition.append(dtemp.values())

    return composition

def getCompositionFeatures(sequencesPos, sequenceNeg):
    compositionPos = getComposition(sequencesPos)
    compositionNeg = getComposition(sequencesNeg)

    X = compositionNeg + compositionPos
    Y = [1]*len(compositionNeg) + [0]*len(compositionPos)

    xx = pd.DataFrame(X)
    xx.columns = d.keys()
    y = pd.DataFrame(Y)
    y.columns = ['Classes']

    return xx, y
```

```
# ============================================================================
# Dipeptide Features
# ============================================================================

def getDipeptideComposition(sequences):
    dicomposition = []
    for s in sequences:
        if (any(i in s for i in na)):
            continue;
        dtemp = {}
        dtemp.update(dp)
        ds = []
        for i in xrange(0,len(s)-1):
            ds.append(s[i:i+2])
        dtemp.update(Counter(ds))
        l = len(s)-1
        dtemp = {k: v/float(l) for k,v in dtemp.iteritems()}
        dicomposition.append(dtemp.values())

    return dicomposition

def getDipeptideFeatures(sequencesPos, sequenceNeg):
    diCompositionPos = getDipeptideComposition(sequencesPos)
    diCompositionNeg = getDipeptideComposition(sequencesNeg)

    X = diCompositionNeg + diCompositionPos
    Y = [1]*len(diCompositionNeg) + [0]*len(diCompositionPos)

    xx = pd.DataFrame(X)
    xx.columns = dp.keys()
    y = pd.DataFrame(Y)
    y.columns = ['Classes']

    return xx, y

# ____________________________________________________________________________
```

```
# ==========================================================================
# Binary Profile (left as N and right as C terminus)
# ==========================================================================
fet = df(d.keys())
fetEnc = pd.get_dummies(fet, prefix="amino")

def getBinary(sequences, n):
    bicomposition = []
    for s in sequences:
        if (any(i in s for i in na)):
            continue;
        sl = list(s)
        if(len(sl)<n): continue;
        sfet = fetEnc["amino_"+sl[0]]
        for i in range(1, n):
            sfet = sfet.append(fetEnc["amino_"+sl[i]], ignore_index=True)
        bicomposition.append(sfet)
    return bicomposition

def getBinaryProfileFeatures(sequencesPos, sequencesNeg, n, terminal):

    if terminal == 'N':
        biCompositionPos = getBinary(sequencesPos, n)
        biCompositionNeg = getBinary(sequencesNeg, n)
    elif terminal == 'C':
        sequencesPos = np.asarray([s[::-1] for s in sequencesPos])
        sequencesNeg = np.asarray([s[::-1] for s in sequencesNeg])
        biCompositionPos = getBinary(sequencesPos, n)
        biCompositionNeg = getBinary(sequencesNeg, n)
    elif terminal == 'NC':
        tempPos=[]
        for s in sequencesPos:
            temp = s[:n]+s[::-1][:n]
            tempPos.append(temp)


        tempNeg=[]
        for s in sequencesNeg:
            temp = s[:n]+s[::-1][:n]
            tempNeg.append(temp)

        sequencesPos = np.asarray(tempPos)
        sequencesNeg = np.asarray(tempNeg)
        biCompositionPos = getBinary(sequencesPos, n)
        biCompositionNeg = getBinary(sequencesNeg, n)
    else:
        biCompositionPos = getBinary(sequencesPos, n)
        biCompositionNeg = getBinary(sequencesNeg, n)
```

```
    X = biCompositionNeg + biCompositionPos
    Y = [1]*len(biCompositionNeg) + [0]*len(biCompositionPos)

    xx = pd.DataFrame(X)
    y = pd.DataFrame(Y)
    y.columns = ['Classes']

    return xx, y
```

```
# ==============================================================================
# Visualizing data using TSNE
# ==============================================================================

def visualizeTSNE(X,Y):
    X_tsne = TSNE(n_components=2, perplexity=50, learning_rate=100, n_iter=2000).fit_transform(X)
    colorset = ['orange', 'blue']

    color=[]
    for row in Y:
        color.append(colorset[row])

    #plot_name = args.plots_save_dir + dataset_name[-1] + '.png'
    fig = plt.figure(figsize=(10,10))
    plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c = color)
    #plt.savefig(plot_name)
    plt.show()

# ==============================================================================
# Visualizing data using pandas scatter_matrix
# ==============================================================================

def visualizeScatter(xx):
    scatter_matrix(xx, alpha=0.2, figsize=(20,10), diagonal='kde')

# ==============================================================================
# Visualizing data using PCA
# ==============================================================================

def visualizeScatterPCA(xx, n = 2):
    pca = PCA(n_components = n)
    trans = pd.DataFrame(pca.fit_transform(xx))
    if n == 2:
        plt.scatter(trans[:][0], trans[:][1])
    else:
        fig = plt.figure()
        ax = plt.axes(projection='3d')
        ax.scatter( trans[:][0], trans[:][1], trans[:][2])

# ==============================================================================
# Visualizing data using Parallel coordinates
# ==============================================================================
# NORMALISE DATA AND TRY THIS
def visualizeParallelCoordinates(xx,y):
#    Classes = ['G','K','C','F','I','W'] #in abundance for ACP
    Classes = d.keys()
    data_norm = pd.concat([xx[Classes], y], axis=1)
    parallel_coordinates(data_norm, 'Classes')   #not working properly
```

Train-Test Code

```
# ============================================================================
# Classifiers (SKLEARN)
# ============================================================================

def evaluate(truth, pred):
    accuracy = accuracy_score(truth, pred)
    conf = confusion_matrix(truth, pred)
    TN = conf[0, 0]
    FP = conf[0, 1]
    spec = TN / float(TN + FP)
    mcc = matthews_corrcoef(truth, pred)
    auc = roc_auc_score(truth, pred)
    sensitivity = recall_score(truth, pred)

    print accuracy, mcc, auc, sensitivity, spec

def run(X, Y):
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, random_state=42)

    X_train=X_train.values
    X_test=X_test.values
    y_train=y_train.values
    y_test=y_test.values

    #Grid Search
    MLPParams={
            'learning_rate': ["constant", "invscaling", "adaptive"],
            'hidden_layer_sizes': [(100,1), (50,2), (10,3)],
            'activation': ["relu", "tanh"]
            }
    SVMParams={'C': [0.001, 0.01, 0.1, 1, 10], 'gamma' : [0.001, 0.01, 0.1, 1]}
    ABCParams={
            "n_estimators": [50, 75],
            "learning_rate": [0.01, 0.1, 0.2, 0.5]
            }
    RFCParams={
            'n_estimators': [200, 700],
            'max_features': ['auto', 'sqrt', 'log2']
            }
    QDAParams={
            'reg_param': [0.0, 0.1, 0.5]
            }
    clf1 = GridSearchCV(MLPClassifier(), MLPParams, cv=5)
    clf2 = GridSearchCV(SVC(), SVMParams, cv=5)
    clf3 = GridSearchCV(AdaBoostClassifier(), ABCParams, cv=5)
    clf4 = GridSearchCV(RandomForestClassifier(), RFCParams, cv=5)
    clf5 = GridSearchCV(QuadraticDiscriminantAnalysis(), QDAParams, cv=5)

    clf1.fit(X_train,y_train)
```

ELM Code

```
# ==============================================================
# ELM Classifier
# ==============================================================

def evaluate(truth, pred):
    accuracy = accuracy_score(truth, pred)
    conf = confusion_matrix(truth, pred)
    TN = conf[0, 0]
    FP = conf[0, 1]
    spec = TN / float(TN + FP)
    mcc = matthews_corrcoef(truth, pred)
    auc = roc_auc_score(truth, pred)
    sensitivity = recall_score(truth, pred)

    print accuracy, mcc, auc, sensitivity, spec
```

```
def run(X, Y):
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, random_state=42)
    #ELM model
    X_train=X_train.values
    X_test=X_test.values
    y_train=y_train.values
    y_test=y_test.values
    print 'ELM tanh'
    for x in range(50, 500, 50):
        elm = ELM(X_train.shape[1], 1, classification='c')
        elm.add_neurons(x, 'tanh')
        elm.train(X_train, y_train)
        pred = elm.predict(X_test)
        temp = []
#        print 'Error(TANH, ', x, '): ', elm.error(y_test, pred)
        for p in pred:
            if p >=0.5:
                temp.append(1)
            else:
                temp.append(0)
        pred = np.asarray(temp)
#        print 'Error(TANH, ', x, '): ', elm.error(y_test, pred)
        evaluate(y_test, pred)

    print 'ELM rbf_linf tanh'
    for x in range(10, 100, 10):
        elm = ELM(X_train.shape[1], 1)
        elm.add_neurons(x, 'rbf_linf')
        elm.add_neurons(x*2, 'tanh')
        elm.train(X_train, y_train)
        pred = elm.predict(X_test)
        temp = []
#        print 'Error(TANH, ', x, '): ', elm.error(y_test, pred)
        for p in pred:
            if p >=0.5:
                temp.append(1)
            else:
                temp.append(0)
        pred = np.asarray(temp)
#        print 'Error(RBF+TANH, ', x, ',', 2*x, '): ', elm.error(y_test, pred)
        evaluate(y_test, pred)
```

REFERENCES

1. Suykens, Johan AK, and Joos Vandewalle. "*Least squares support vector machine classifiers.*" Neural processing letters 9.3 (1999): 293-300.
2. Taud, H., and J. F. Mas. "Multilayer Perceptron (MLP)." *Geomatic Approaches for Modeling Land Change Scenarios*. Springer, Cham, 2018. 451-455.
3. Vincent, Pascal, et al. "*Extracting and composing robust features with denoising autoencoders*." Proceedings of the 25th international conference on Machine learning. ACM, 2008.
4. https://www.wikipedia.org/