Introduction to machine learning
Project 3 - Challenge

Adrien Vinders - s194594
Emil Geleleens - s191407

December 16, 2022

Introduction

The asked task is to predict the wind power production of a wind farm in Australia. A dataset is given over 10 zones and the goal is to predict the power production of the wind farm for the next 24 hours in those 10 zones.
The implementation is made in Python using the sci-kit-learn library for supervised learning [1]
Various regression methods were tested to make the best prediction possible, including:

1. k Nearest Neighbors, with different values for k,
2. Random Forest with 100 and 500 trees,
3. Support Vector Regression,
4. Neural Networks.

Some of the above methods were tested with and without the following features management methods:

5. Features Extraction:
   (a) Univariate variance threshold feature extraction,
   (b) Correlation threshold feature extraction.
6. Our implementations.
   Note : our implementations are available on our github repository

1 kNN

The first thing to note is that we made one model per zone. Then, our dataset is split into three subsets:

- The training set which contains the samples that we will use to fit the model,
- The test set which contains the samples that we will use to test our model,
- The gradescope set which contains the inputs for which we have to predict the outputs (dataset on gradescope).

1.1 Feature selection

The first thing that we had to do was to determine what the feature vectors should contain.

1. Firstly, we chose to only take the speed profile of the wind as input $(U_{10}, U_{100}, V_{10}, V_{100})$, because these four features are the most relevant to compute the wind power. We tried to normalize these values between 0 and 1, but our results were less accurate, so we kept the default values.

2. Secondly, we chose to add another value in the feature vector which would represent the time at which the speed profile was measured. The idea behind this is that to predict the wind power at a certain time, training set samples that are close in time are more relevant for the prediction than samples corresponding to measurements that are far away in time. Indeed, we would expect the wind power to be a linear function of time. However, passed a certain point, ... The value representing the time was computed by taking the timestamp corresponding to the tuple (Hour, Day, Month, Year) from which we subtracted the smallest timestamp. We then normalized this value between 0 and 1 by dividing the obtained value by the difference between the biggest timestamp and the smallest. Finally, we scaled this value by a factor β which can be adjusted to obtain the best accuracy. We found that $\beta = 0.1$ improved slightly the model.

1.2 Choice of k

Since we have one model per zone, we don't just have one value for k, but actually 10 different values. Let us define the vector $\mathbf{k} = (k_1, k_2, \ldots, k_{10})$, where k_i is the number of nearest neighbors used for the model describing zone i. We tried two techniques to choose our vector $\mathbf{k}$.

1. Firstly, we decided to use the output from gradescope to adjust the vector $\mathbf{k}$. We made several submissions until we had a vector for which the MAE reached a minimum, this vector is

$$\mathbf{k} = (24, 23, 987, 600, 61, 50, 64, 3676, 380, 589). \tag{1}$$

   This technique is the one we used for the challenge. It allowed us to be at the third position at the end of the challenge, before the MAE with the full dataset was revealed which made us fall to the 21st place. This method allowed us to have a MAE over the public gradescope set of 0.17675193757633528 and over the full set: 0.1827168321537757. We note that for the challenge we used a concatenation of the training set and the test set as training set, because we did not use the test set for anything else. Finally, this approach could be criticized because we used the gradescope set as test set to improve the model to then be able to compute better predictions for the gradescope set. This is like a snake that bites its own tail to feed itself. So we wanted to try another approach.

2. A second approach was to use the test set to evaluate what the MAE could be. We used dichotomic search to find a k that minimizes the MAE of the test set. This method does minimize the MAE over the test set, but the results over the gradescope set are less uplifting. We obtained a MAE of 0.18425858186114824 with the predictions over the public dataset and 0.18704408273365225 over the full dataset. The vector $\mathbf{k}$ obtained is the following:

$$\mathbf{k} = (11, 20, 19, 19, 21, 15, 16, 14, 24, 19). \tag{2}$$

   As we can see this vector is quite different from the one we found with the first method. We decided to also use the dichotomic search with the gradescope public set for which the outputs were revealed and we obtained

$$\mathbf{k} = (24, 141, 1261, 556, 55, 59, 79, 3747, 439, 712). \tag{3}$$

   As we can see this time the vector is far closer to the one we obtained by trying manually all the possibilities. This means that either our test set is incomplete or the k values obtained with the gradescope public set were

only well adjusted for this particular set. This would explain why we went from 3rd place to 21st. Indeed, as soon as we went to the full dataset the vector **k** was not suited anymore.

3. Another method that we could have used is k-fold cross validation.

1.3 Weights

Sklearn provides a way to weight the different contributions of the k nearest neighbors to the prediction. By default each of the k nearest neighbors bring the same contribution to the prediction. We decided to try using a function w that assigns a weight which decreases when the distance d to the neighbors increases. This means that distant neighbors will have a smaller impact on the prediction than neighbors that are closer. We defined our function $w(d)$ as follows:

$$w(d) = \frac{1}{d + \gamma}, \tag{4}$$

where γ is positive parameter. We found that a value of $\gamma = 10$ did improve our model.

1.4 Single model for all zones

As we have said earlier, we built one model per zone. This allowed us to avoid having to put the ZONEID into the feature vector to differentiate between zones, which would not make any sense. Indeed, as kNN computes the euclidean distances between the feature vectors, if we want to predict the wind power in zone 10, kNN would consider that feature vectors in ZONEID 1 are less relevant than samples in ZONEID 9. This is bad because the ZONEID does not give any information about the relations between zones. Having one model per zone allows us to adjust k for each zone separately. However, as we have seen it was kind of tricky to adjust correctly the values for k.

One possible assumption might be that all the feature vectors for which the ZONEID corresponds to the zone in which we want to make a prediction have a higher relevance than the other ones. And for these other zones, we consider that they all have the same relevance with respect to the zone. To represent this in our model, we could tweak the feature vectors by adding 10 features z_i that are either 0 our α each. We would then build our feature vectors such that if a sample is in zone i, then the $z_i = \alpha$ and the other nine $z_j = 0$. With this technique the parameter α would be a measurement of how much zones are "distant" (i.e. how much the data in one zone is not relevant for the prediction of the wind power in another zone). Let us take an example feature vector to illustrate:

$$(z_1, z_2, z_3, \ldots, z_{10}, U_{10}, U_{100}, V_{10}, V_{100}, \ldots) = (0, 0, \alpha, \ldots, 0, 3.27, -1.2, 2.4, 3.19, \ldots). \tag{5}$$

This vector corresponds to a measurement from zone 3.

There are three problems with this approach. First, the "distance" between zones is certainly not the same between each zone pair. Another problem is that if we use this technique, then we will have one single model for all the zones and we won't be able to adjust k for each zone separately. And finally, adding 10 features to the feature vectors will make the program very slow.

However, we still decided to try. We did the test with $k = 24$. And the results are pretty good. However, the computation time is quite long. For the public set MAE, we have: 0.15068269343228907 and for the full gradescope set we have: 0.1805573633478381.

2 Random forest

A Random Forest (RF) is an ensemble machine learning algorithm making a large number of decision trees trained using a random subset of the data. RF are effective at handling large and complex datasets and are resistant to overfitting.

2.1 Results

RF were implement using default values of sklearn.ensemble.RandomForestRegressor [2]
- i.e. squared error loss function, nodes expanded until all leaves are pure, minimum number of samples required to split an internal node = 2, minimum number of samples required to be at a leaf node = 1, minimum weighted fraction of the sum total of weights required to be at a leaf node = 0.0, number of features to consider when looking for the best split = 1.0, unlimited number of leaf nodes, bootstrap sample for building trees, jobs run sequentially,

fit a whole new forest when a call is made, no pruning, samples to draw from the data to train each base estimator : all the data-

10 implementations have been done - including 7 trained only on the last zone - (see Table 1) from which observations can be made :

- Train the regressor on the whole dataset leads obviously to a smaller MAE,
- RF trained with only speed features - i.e. U10, U100, V10 and V100 -are more accurate than RF trained with all features except U100 and V100. It is explained by the fact that ZoneID does not bring any information for models learned on specific zones and 4 features for the time - i.e. Hour, Day, Month, and Year- lead to similar importance of the time and the speeds. Perform univariate variance threshold feature extraction and transformation of the speed vector into a timestamp would thus lead to a better model,
- 1000 trees instead of 500 slightly decrease MAE - $\sim 0.00002\%$ - and is then not worth it.

2.2 Possible improvements

Perform univariate variance threshold feature extraction and transformation of the speed vector into a timestamp would thus lead to a better model. The learning dataset for a zone would be composed of the following features :

- U10,
- U100,
- V10,
- V100,
- timestamp.

Add correlation feature extraction to this improvement might not increase the accuracy of the model - not tested - but the learning dataset of a zone would then be composed of the following features :

- U10,
- V10,
- timestamp.

3 Support Vectors Regressor

Support Vector Regressor is a variant of the Support Vector Machines (SVMs) algorithm, which is a classification algorithm - SVM work by finding the hyperplane in a high-dimensional space that maximally separates the different classes -. In the case of SVR, the goal is to find the hyperplane that best fits the data, rather than separating the classes. SVR Find the hyperplane that has the smallest error with respect to the training data (the support vectors) - this is the hyperplane that best fits the data - and compute the error between the hyperplane and the training data using a loss function SVR has the ability of find non-linear hyperplanes, is robust to overfitting and handles high-dimensional data **Epsilon-SVR** had been implemented. It allows for a small amount of deviation from the predicted values by introducing a margin of error, or "epsilon". The epsilon determines the size of the margin of error and controls the trade-off between the simplicity of the predicted function and the accuracy of the predictions. **Radial Basis Function (RBF) kernel** - i.e. Gaussian function of the Euclidean distance between two input points - had been used in the implementation [1]. Radial kernels are based on the distance between the input data points. RBF kernels are well-suited to SVR because it can learn complex and non-linear functions.

[1] kernel function: operation applied to the input data to transform it into a higher-dimensional space. The data is then more easily separable allowing the SVM algorithm to learn more complex and flexible functions that better model the relationship between input and output data

3.1 Results

Our implementation of epsSVR train the model on the whole learning dataset - i.e. on the 10 zones -. and fit it on each prediction dataset. As can be seen in Table 1 it did not has a good MAE (0.243774153590 and 0.2325289309807 respectively for global and zone 1). This might be explained by the fact that a small value of epsilon - 0.1 in our implementation - leads to overfitting as it defines the width of the error volume around the regression line/hyperplane. Better tuning of this parameter could be done through cross-validation.

4 Neural network

4.1 Architecture

An Artificial Neural Network (ANN) is a network of neurons organized in layers. We choose to use a dense neural network. It means that each neuron in a layers is connected to each neuron in the previous and next layer. The idea is to adapt the weights of links between each neuron to better approximate the model. A general illustration can be found in figure 1

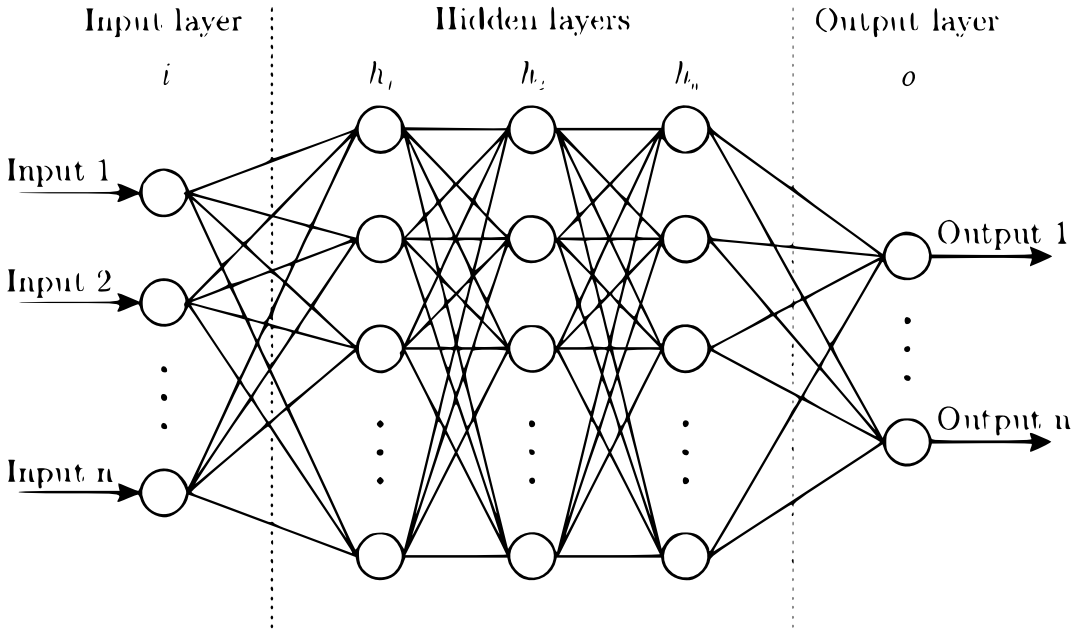

Figure 1: Dense Neural Network [3]

In our case an ANN of **5 layers** is constructed

- 1 input layer of size N_FEATURES,
- 3 hidden layers of size N_FEATURES,
- 1 output layer of size 1, as only one value is asked to be predicted.

For the input and hidden layers, **activation function ReLU** is used. For the output neuron a linear activation function as a real value is asked in output. As the activation function is ReLU, the weight initializer used is **He initialization**[3]. Its role in a neural network is to set the initial values for the weights of the network's connections. The **optimizer Adam** chosen is in favor of gradient descent because as time-to-convergence was a concern. It also has the advantage of smoothing out the loss of performance due to other misschosen parameters[3]. **Mean square error loss funnction** is the most common or regression and is thus chosen. This function is a measure of the difference between the network's predicted output and the actual output for a given input. and thus allow the ANN to evaluate its prediction. The Neural Network has a relative high amount of tunable parameters including $4 * \text{N\_FEATURES}^2 + \text{N\_FEATURES}$ inter neurons weigths - for input and hidden layers + output layer -. Neurons migth also have inner parameter tunable. It thus lead to $4 * (\mathbf{n\_features}^2 + \mathbf{n\_features}) + \mathbf{n\_features} + 1$ **tunable parameters** - actually observed in practice -. Note that the ANN has $4 * \text{N\_FEATURES} + 1$ neurons.

On figure 2 a representation of the ANN with 9 features - without having to perform features management. It has 37 neurons and 370 tunable parameters.

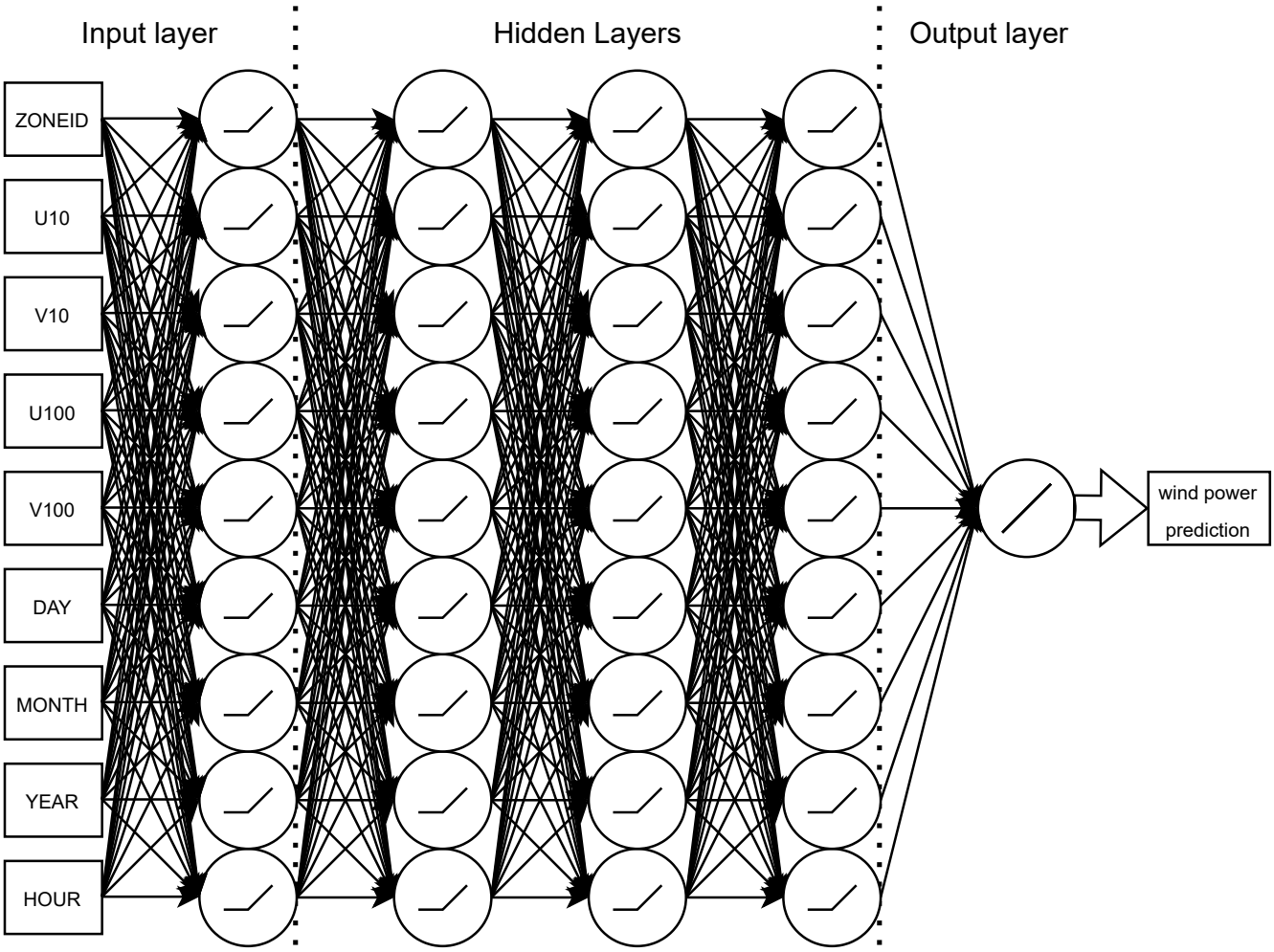

Figure 2: ANN with all the features

The inputs values need to be scaled for the network to converges faster. As the activation function is ReLU a common method is **min-max scaling** - subtracting the minimum and dividing by the range -.

To perform the fit of all these tunable parameters it may be usefull to train several times the networks. The idea is to divide the dataset in batch and train sequentially the network several numbers - epochs -[3]. Our learning dataset is splitted in **10 batches** and sequentially trained in **50 epochs**. A large batch size can make training faster and more stable, but it can also consume more memory and make the network less responsive to changes in the data. A small batch size can make training slower and noisier, but it can also allow the network to be more flexible and avoid overfitting. The number of batches was picked considering this tradeoff. The number of epochs was also picked in a tradeoff. A large number of epochs increases the accuracy on the training set but it highly increases overfitting while a small number of epochs decreases the accuracy of the model.

4.2 Results

Those results had been gathered : From the table above we can see that the neural network has a good accuracy neither on our testing sample nor on the prediction sample. Training it with too many epochs (150) decreases the MAE because it over-fitted the data. The minibatch size was too small. The model was not able to accurately represent the underlying distribution of the data with such a small sample. Then 10 batches (batch size = $\frac{n_{samples}}{10}$). When 1 ANN is constructed per zone (while keeping the same other parameters) the MAE and error increase highly. This is because the model not having sufficient data to learn on. When considering the size of our learning dataset

Method	Expctd error(%)	time(s)	MAE_GLOB	MAE_Z1
Correlation FE, test 10%	7.29	12.077	0.2556024822676563	0.2663099177565241
Correlation FE, tesst 10%, minibatch, 150 epochs	4.59	2892.28	0.202639853415171	0.17902481850330204
test 10%, 10 batches, 50 epochs	5.1577	8.73	0.1978985906637341	0.17943384394762932
1 ANN per zone, test 10%, 10 batches, 50 epochs	19.22	18.673	0.3966516599325793	0.2942257636466016

and the type of data an Artificial Neural Network is not the model to choose to better predict our output.

4.3 Possible improvements

We saw that 1 ANN per zone decreases the accuracy of our ANN model. This may be less the case while performing a transformation of the features Hour, Day, Month, Year into a timestamp in addition to a univariate variance threshold feature extraction - which would remove the ZoneID feature as well-. The network would then have layers of 5 neurons - i.e. 126 tunable parameters instead of 370 -. Fewer parameters mean a lower need for a large learning dataset. In addition to this with correlation feature extractions, features U10 and V10 (with high correlation with respectively U100 and V100) would also be deleted. This would have led to a 3 feature - U100, V100, timestamp - ANN with 52 tunable parameters. Unfortunately, not enough time was left for us to test these 2 models.

5 Features Extraction

5.1 Univariate variance threshold feature extraction

Univariate Variance threshold feature extraction consists of computing the variance of each feature and dropping each feature for which the variance is lower than a threshold. The threshold chosen was by default $1e-6$ and was tunable if needed.

This method mainly removed the ZoneID feature. For one zone the ZoneID feature is always the same and so its variance is null.

5.2 Correlation threshold feature extraction

Correlation threshold feature extraction consists of computing the correlation between every feature. It results in a correlation matrix containing for every feature its correlation with the other features. A high correlation means that the data are very similar and vary the same way. Then a high correlation between two features leads to the fact that similar information is contained in both of the features. So only one is needed to be kept and the second one is dropped. It is dropped if the correlation is above a threshold, chosen at 0.9 (tunable if needed).

This method shows a high correlation - above the threshold - between features U10 and U100 and features V10 and V100. For each of those two pairs, one feature is dropped.

6 Our implementations

A summary of all our submitted implementation can be found in Table 1. Before the submission "RF, 500 trees regressor on all zones" models were not trained on all the data, but only on the last zone due to an error of implementation. The expected error is the Mean Absolute error (MAE), except when MSE is written and for Neural Networks - where it is the models' evaluation method output. The position is the leader board position of our submission at the time we submitted it - before the end of the challenge then -.

References

[1] Scikit-learn : Supervised learning.

[2] sklearn.ensemble.RandomForestRegressor documentation.

[3] Lavanya Shukla. Designing Your Neural Networks A Step by Step Walkthrough. 2019.

Position	Method	Time(s)	Error(%)	MAE_GLOB	MAE_Z1
27	mean			0.27802968049	0.2707220813
23	kNN with k=10			0.2019747711332	0.163556951130
22	kNN with k = 100]			0.195587279901	0.174476513506
23	10 bagging kNN with k = 100			0.195898057464	0.176044436227
13	Random forest 100 trees	169.53		0.185372021412	0.142365850178
29	RF, 100 trees, with univariate Feature extraction	85.498		0.204967868703	0.16656940560
38	RF, 100 trees, test 10%, correlation FE	46.8149	5.247 (MSE)	0.215089223774	0.18130665921
35	RF, 100 trees, correlation FE	49.909		0.209476560946	0.1747251723557
38	RF, 500 trees, test 10%, correlation FE	244.576	5.1641 (MSE)	0.21380904958	0.178957490960
36	RF, 500 trees, test 10%	346.45	5.15 (MSE)	0.209552155549	0.172885499857228
35	RF, 500 trees	404.994		0.204887672910	0.166712822412
27	RF, 500 trees regressor on all zones	704.69		0.18551711104	0.1441871007726
41	epsSVR, rbf kernel, test 10%	1015.973	7.753 (MSE)	0.243774153590	0.2325289309807
40	RF, 100 trees, test 10%, correlation FE, MAE	109.398	13.82	0.1956323910	0.16641628587
40	ANN, 3 Hidden Layers, test 10%, correlation FE	12.077	7.29	0.2556024822	0.266309917756
38	ANN, 3 HL, test 10%, correlation FE, 150 epochs, minibatch	2892.28	4.59	0.2026398534	0.179024818503
38	ANN, 3 HL, test 10%, 50 epochs, batch 1/10	8.73	5.1577	0.1978985906	0.179433843947
33	random forest with 1000 trees			0.185342051289	0.14334001545
37	random forest with feature_vec = (speeds) and 100 trees			0.186403	0.144280831252
	knn 20 neighbors feature_vec = (speeds)			0.184010	0.140117
	knn 123 neighbors feature_vec = (speeds)			0.182138	0.144743
1	kNN adjusted n_neighbors feature_vec = (speeds)			0.176793	0.1395687
2	kNN adjusted n_neighbors feature_vec = (speeds)			0.176786	0.1395687
2	kNN adjusted neighbors feature_vec = (speeds + timestamp)			0.176752	0.139393
41	ANN, 1 per zone, 3HL, test 10%, 50 epochs, batch 10%	18.673	19.22	0.3966516599	0.2942257636

Table 1: Methods implemented