STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LX**, Number 1, 2015
DOI:

PREDICTING INDOOR RADON LEVELS IN ROMANIA USING MACHINE LEARNING ALGORITHMS

O. A. TRIFAN, D. TODAŞCĂ, AND A. SUSCIUC

Abstract. Predicting the radon levels represents a pertinent problem in the Machine Learning universe. Although this problem has been approached sparsely in the past, numerous studies recently surfaced in order to find a satisfactory model. In this paper, we are comparing results from three different ML algorithms (Deep Neural Networks, Support Vector Machines and Random Forests) in order to find a satisfactory model. The data used was provided by researchers from Babes-Bolyai University, Environmental Science and Engineering Faculty, who recorded and measured radon levels periodically in multiple locations. (Here we add a bit more about the data, where it was taken from and how, when we get that data). Using this data, we trained our models and analyzed the results, seeking to enhance the public health and safety in Transylvania. The aim is to outline the regions where there may exist a risk of high indoor radon concentration, without installing physical sensors to detect those threats. Finding a reliable model has benefits because it significantly reduces the cost associated with the sensors and the time required to gather the physical data.

1. Introduction

Radon is a radioactive gas and is considered to be the second major factor in causing lung cancer. In fact, around 3 to 15 percent of lung cancer cases are attributed to faulty Radon exposure. It is part of the scientific consensus that Radon emanates from rocks and soil. Indoor radon levels are highly influenced by groundwater, ventilation systems, weather, architecture and the construction materials used for the building. Addressing the indoor radon problem is of high importance both in the mitigation of radon-provoked harm in already existing buildings and also in the prevention of high-radon levels in future buildings. Measurements of mean radon concentrations have also yielded very different results, depending on the country they have been carried out, due to differences in geographical attributes. The principal interest in

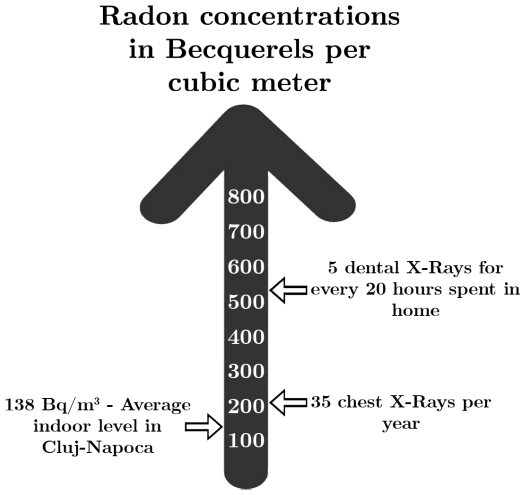
Key words and phrases. radon, deep learning, indoor radon levels, lung cancer, R2-score.

present studies is finding efficient materials in terms of cost to decrease radon exposure (WHO, [12]).

Radon is a radioactive high-energy, alpha particle emitter. Once it enters the human body, a series of short-lived radioactive progeny (isotopes of polonium, lead and bismuth) is generated. Trapped in the human lungs, the progenies themselves bring about health damages rather than the pure inhaled radon (Nilson, [11]).

An epidemiological study that analysed data from 13 European countries found out that the risk of lung cancer is increased by 8.4% per 100 Bq/m^3 increase in measured radon. On average, for cases of lung cancer, the exposure over a considerable part of a person's lifetime is somewhere near 104 Bq/m^3 (Darby, [3]). Therefore a method to predict the level of indoor radon can bring on saving lives, as a Romanian study (Timar, [14]) found out that on average radon exposure is responsible for about 1200-1800 annual cancer deaths inside the country. In Cluj-Napoca, about 17% percent of homes have been found to have radon levels higher than 300 Bq/m3, which is largely considered to be the milestone for pernicious radon levels. The average indoor concentration is 138Bq/m^3. Comparatively, on Romanian territory an average mean value is considered to be between 60 and 80 Bq/m^3, (Florica, [4]).

FIGURE 1



The factors that influence the level of indoor radon concentration are qualitative and also quantitative. Therefore it is not possible to apply any single

metric in order to predict the level of indoor radon. Numerous computer science algorithms take advantage of computers' calculation power and under the model of human ability of learning by gradually improving the accuracy. Consequently, a virtual model to predict the anomalies and the potential danger zones can be established.

A recent study aimed to accomplish a universal model to improve the existing metrics used in assessing the Indoor Air Quality by predicting the radon with respect to geospatial and built environmental attributes (Khan, [5]). An artificial neural network with random weights and polynomial statistic models has been implemented and provided an accurate prediction. It has proven to be more powerful than any other traditional Artificial Neural Networks. However, this method has some drawbacks, specifically its inability to correctly establish the build features that cause the radon levels within housing stock to increase based on the build year.

Another study made a comparison between a neural network model and a bagged neural network, trained to predict the indoor radon levels in Czech Republic (Timkova, [15]). Classifying the prediction in three classes with respect to the mean radon concentration, the error was 50%, 54%, 28%. A positive bias has been observed in the predicted radon concentration by the trained model.

Aimed to not only predict indoor concentrations, but also understand what causes high radon levels inside buildings, Kropat ([7]) used data from Switzerland and built maps of concentration for all its regions, using both supervised (kernel regression, ensemble regression trees and random forests) and unsupervised (k-medoids) machine learning techniques. The authors conclude that concrete foundations decrease the indoor radon levels compared to earth buildings, and regarding lithology, that the igneous rocks and carbonate rocks in the Jura Mountains increase the radon levels. Random forest algorithms also provided the best predictability with a coefficient of determination of 33%, compared to Bayesian additive regression trees (29%) and Kernel estimation (28%).

The study of radon in Computer Science has potential for other applications other than predicting indoor radon levels. As a subject of further study, it has been concluded that Radon is an accurate precursor of seismic events (Rafique, [13]). Rafique proposed delegated boosting for regression (DRM) to find anomalies in radon concentration values and accurately predict the occurrence of an earthquake, days before it actually comes about. This method provided promising results, evincing a finer accuracy than the three other algorithms used, XGBoost, Support Vector Machine Linear and Support Vector Machine Radial. While DRM showed a RMSE of 2667.437, the other three

algorithms largely surpassed this result, obtaining scores of 3757.667, 7618.878 and 10881.65 respectively.

2. Methods and experiments

In this paper, we choose three Machine Learning algorithms and compare the results across the fields: Deep Neural Networks (DNN), Support Vector Machines (SVM), and Random Forest (RF). In all of the three methods used we have considered a regression problem.

2.1. **Dataset.** The dataset used in our experiments is publicly available online here. For every square-shaped geographical area, we are given 15 input values, and an estimation of the percentage of homes affected by risky radon levels. We have theorized that this estimation of homes was computed by means of ML algorithms as well, as it will be discussed below. We must also note that such a high dimension of the input space is rarely found in real life datasets.

2.2. **Scoring.**

2.2.1. *R2. R2* (R squared, coefficient of determination) is the first scoring method we use to rank our models. If y_i is our target vector of length n and $\hat{y}_i$ our prediction vector, R2 is computed in the following way:
$R^2 = 1 - \frac{\Sigma_i (y_i - \hat{y}_i)^2}{\Sigma_i (y_i - \mu)^2}$, where $\mu = \Sigma_i \frac{\hat{y}_i}{n}$ (the average of our predictions)
Best possible value for *R2* is 1, when $y_i = \hat{y}_i$. Negative values are possible. The lower the value, the worse the performance.

2.2.2. *MAE.* MAE (Mean Average Error) is the second scoring method we use, as a way to further rank models with a similar coefficient of determination. With the aforementioned notations, MAE is computed in the following way:
$MAE = \frac{1}{n} \Sigma_i |y_i - \hat{y}_i|$
Best possible value for MAE is 0, when $y_i = \hat{y}_i$. Only positive values are possible. The higher the value, the worse the performance.

2.3. **K-fold Cross-Validation.** Cross-validation is a technique used in Machine Learning to prevent overfitting and assist in parameter tuning (Berrar, [2]). Optimizing our models' performance based solely on training data in the fitting phase of the algorithm is what makes a model learn too much from existing noise. Therefore, we introduce some validation data, and use it to better estimate the model's performance on unknown data.

K-fold Cross-Validation is an efficient technique for tuning parameters in the model selection phase [1], although it may increase existing biases (Kohavi, [6]). It works by splitting the dataset into k equal disjoint subsets - any such

subset is called *fold*. The main idea is to use k-1 distinct folds as training data, and score its performancee on the kth left-out fold (validation data). We sequentially execute this procedure k times, so that every fold is eventually used as validation data. The final performance of the model is the average performance of all k steps. Averaging a model's performance on different samples of the existing dataset is meant to guarantee us a more accurate scoring on new data.

2.4. **Deep Neural Network.** Neural Networks mimic the mechanics of the human brain, using "neuron-like" structures. The models are trained using several layers, each parameter activating each layer in a mathematically decided way. The dendrites in the brain receive sensory inputs from the exterior sending the signal to the axons, which are responsible for feeding the output. This "information" propagates from one neuron to another, a structure that is emulated in the form of neural network algorithms.

Using the Keras library in Python, we have constructed a Deep Neural Network and adjusted its properties, protecting ourselves from underfitting and overfitting and decreasing the overall MAE. In our tests, we have taken the 15 input parameters and performed a batch normalization. A common problem with datasets is the high range and diversity of the input data, making it hard for the training to decide a certain weight for each parameter. Using Batch Normalization, we can regularize the input data, conveying in a lower MAE and lower training steps. We use this method when entering every layer of our network.

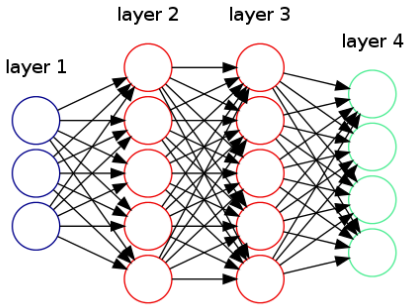
The Keras model is Sequential, with 1 input layer and 3 hidden layers: the input with 30 weights and sigmoid as an activation function, and the hidden layers with 20, 15, and respectively 10 weights, all using *relu* as the activation function. We conclude with the output layer, also using *relu*. The model was compiled using *adam* as the optimizer and *MAE* as the loss function. We have also run tests using *MSE*, but *MAE* showed more promising results.

We trained the model with a batch size of 16 for 100 epochs, as the model showed no signs of overfitting. The loss converges for both the validation and the training data after approximately 70 epochs, making our choice of 100 epochs commonsensical.

By trial and error and educated guessing, we have concluded that:

- Batch normalization is essential before each layer, improving both training name and overall accuracy
- A number of 3-5 layers is yielding satisfactory results
- A training time of 100 epochs gives similar results to the model trained for 2000 epochs, for both the training data and the validation data

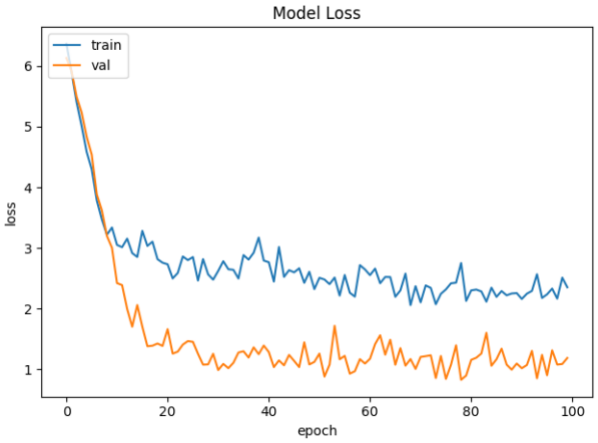
FIGURE 2. Neural Network Layer Scheme



Layer 1 is the input layer, Layer 2 and 3 are Hidden while Layer 4 is the output

- MAE and MSE both show promising results, although MAE has a slight advantage in terms of the average loss

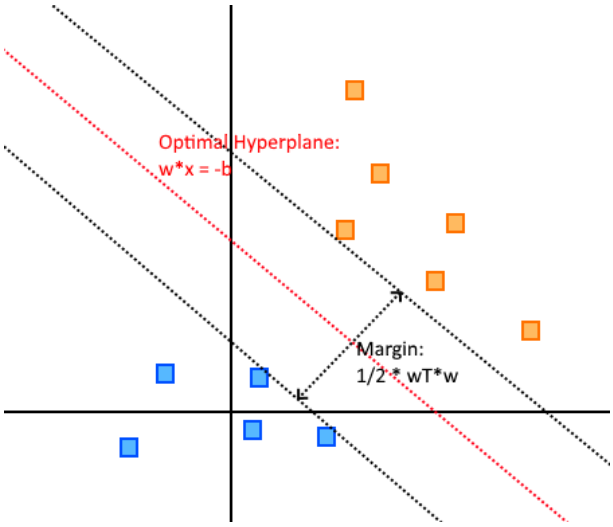
FIGURE 3. Model MAE



2.5. **SVR (Support Vector Regression).** Support Vector Machine (SVM) is a well-researched supervised machine learning algorithm. Due to computational costs, it does not scale well with large datasets. It is argely resistant to overfitting - we do not deal with model overfitting, but rather model selection and parameter tuning (Lameski, [8]).
SVM is classically used in dual classification problems, and it is worth seeing

how it works in that case first. For a multi-dimensional input vector x with target values t, a SVM tries to find the best separating hyperplane for the two classes that is computed by multiplication with a weight vector w (see figure below). In this context, 'best' means to correctly separate the two classes and minimize the margin, which is the distance between the hyperplane and the nearest points of both classes. The nearest points are called Support Vectors. Such separating hyperplanes can be efficiently found by mathematically solving a set of constraints (see figure 4).

FIGURE 4. (x, t) - input features and target values, w - weight vector



minimise $\frac{1}{2}w^T w$ (the margin) with the condition $t_i(w^T x_i + b) \geq 1$ (points are properly separated)

However, finding a separating hyperplane means finding a linear solution in a multi-dimensional space to separate our input, whereas our dataset might not be linearly separable! We solve this issue in practice by employing two measures. First, we introduce a new integer parameter C and minimize $\frac{1}{2}w^T w + C \cdot$(distance to the misclassified points). The distance to the misclassified points is tracked by what we call slack variables. We may intuitively understand the value of C how much we care about each misclassification, with higher values placing more emphasis on every misclassified points and possible outliers.

Second, we transform our data into a higher dimension and make use of kernels for efficient computation. The intuition of building extra dimensions is that it might allow for the data to become linearly separable, so we choose a meaningful function ϕ and work further on $\phi(x)$. Increasing the number of dimensions might seem to also increase the computation time, but we can overcome this issue with the so-called Kernel trick: it is possible to simplify the mathematical equation by writing down the kernel function $K(x, y) = \phi^T(x) \cdot \phi(y)$, where x, y - input elements.

SVMs can be used for regression, in which case we talk about Support Vector Regression (SVR). The main difference from classification is that our algorithm will minimize an error function, instead of the distance to the support vectors. Common practice is to take the least-squares error function and turn it into an epsilon-insensitive error function, with the intuition that we still imagine a margin around our prediction of width epsilon (see figure 5).

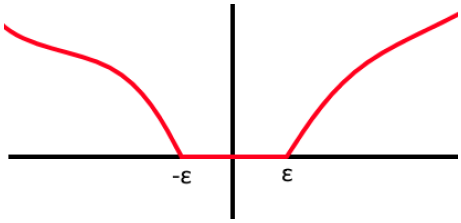
Kernels used in this paper:
Polynomial kernel: $K(x, y) = (\gamma \cdot < x, y > + coef_0)^{degree}$
$\phi(x) = \{1, x_1, ..., x_n, x_1^2, ..., x_n^2, ..., x_1^{degree}, ..., x_n^{degree}\}$
Radial Basis Function (rbf) kernel: $K(x, y) = exp(-\gamma ||x - y||^2)$
Sigmoid kernel: $tanh(\gamma \cdot < x, y > + coef_0)$

Now we can understand the challenge of building a smart SVM: selecting good values for C, the kernel, and the kernel's parameters.

In case of regression, we talk about Support Vector Regression (SVR). The main difference from classification is that our algorithm will minimize an error function, instead of the distance to the support vectors. Common practice is to take the least-squares error function and turn it into an ϵ-insensitive error function, with the intuition that we still imagine a margin around our prediction of width ϵ ([10]).

2.5.1. *Software.* We have implemented and tuned our own SVR model using the Sklearn library, freely available in Python. The implementation found in the Sklearn library is based on LIBSVM, an open source machine learning library, and makes use of all the highlights presented in the Methods section above: slack variables, epsilon-sensitive error functions, kernel trick.
Sklearn provides special default values for γ inside the rbf kernel, *auto* and *scale*: *auto* means $\gamma = \frac{1}{n}$, while *scale* means $\gamma = \frac{1}{n} \cdot v$, where v - the variance

FIGURE 5. ϵ-insensitive function: for input in range $[-\epsilon, \epsilon]$, value is 0.



of the dataset.
For plotting all the graphs that appear in this paper, we have used the matplotlib Python library.

2.5.2. *Data Normalization.* SVM is not scale invariant, as it is with NNs. At the beginning of our experiemnts, our models' performance was underwhelming, because we did not normalize the values from our input. We have used sklearn's Pipeline tool, which encapsulates the model together with a normalization scaler from Standard Scaler, Maximum Absolute Scaler, Minimum Maximum Scaler.
Consider our dataset with all input features as a matrix A, with each feature on a column. We denote by $A_{,j}$ the jth column inside this matrix.
Standard Absolute Scaler: $A_{i,j} = \dfrac{A_{i,j} - \mu_j}{\sigma_j}$, where μ_j is the jth column and σ_j is the variance on the jth column, scales the data so the mean of each column is 0 and the variance is 1;
Maximum Absolute Scaler: $A_{i,j} = \dfrac{A_{i,j}}{max(A_{,j})}$, making the maximum of each column to be equal to 1;
Minimum Maximum Scaler: $A_{i,j} = \dfrac{A_{i,j} - minA_{,j}}{(max(A_{,j}) - min(A_{,j}))}$ making the range of values from all columns to be inside [0, 1].

2.5.3. *Trial and Error.* At first, we tried to obtain our best model by randomly splitting the dataset into training data (70%) and testing data (30%), then manually trying out values for our model's parameters and seeing which scores best on the testing data.
By means of trial and error, we have made the following observations:

- greater value of C increase runtime for all kernels, but also improve the model's performance, up until a point (aprox. 1000) where performance starts decreasing again as a consequence of overfitting; this suggests almost total lack of noise in the dataset
- in increasing order of computational time, the kernels are: sigmoid, rbf, poly
- the addition of sklearn Scalers significantly increase the score, and choosing between the three of them also
- regarding polynomial kernel, a higher degree always brought longer training times, but didn't always improve the model performance
- high values for the *coef0* parameters used by sigmoid and poly greatly increase runtime and decrease model performance; values less than 1 are preferred
- our best models train very fast on the dataset, under 300.000 iterations, suggesting there is little complexity in the solution

2.5.4. *Grid Search.* There is a possible problem we may have with our previous technique: manually overfitting our model for the given dataset. For reliable results, we have always maintained the same random seeds for our random procedures. At the end of our selection, we realized the model performs much worse on other random seeds. Therefore, to verify our observations and select a model that will be reliable on new data, we need cross-validation (Mantovani, [9]).

GridSearchCV from Sklearn formalizes the process of trying out all possible combinations of parameters with values taken from lists given by the user, using a K-fold cross-validation technique, and ranking them according to their performance. We decided to use a 5-fold cross-validation technique, with an R2 scoring, and we compiled part of the results of our GridSearchCV in a descriptive table.

We must note that here we did not allow for a polynomial kernel to use values for C higher than 1, because fitting them took too long. Even though our table supports our former statements from section 2.5.1, the reader must realize the importance of making sure we have a scientific basis under our results, as well as making sure we build accurate models that perform as best as they can on possible new data:

- no sigmoid kernel made it to the top 40, and the best sigmoid GridSearchCV score was 0.722

rank	kernel	scaler	gamma	C	coef0	degree	R2 score
1	rbf	Std Scaler	auto	100	-	-	0.939736
2	rbf	Std Scaler	scale	100	-	-	0.939736
3	rbf	Max Abs Scaler	scale	100	-	-	0.937483
4	poly	Min Abs Scaler	-	1	0.5	4	0.936993
5	poly	Max Abs Scaler	-	1	0.5	4	0.936713
6	rbf	Max Abs Scaler	scale	300	-	-	0.936368
7	rbf	Min Abs Scaler	scale	100	-	-	0.935566
8	poly	Max Abs Scaler	-	1	0.1	4	0.935563
24	rbf	Std Scaler	scale	1000	-	-	0.927811

- StandardScaler works well in combination with rbf, but poorly with poly
- a polynomial degree of 5 downgrades the results
- increasing the value of C too much eventually decreases performance
- generally, a scaling gamma provides better results

2.6. **Random Forest.** The Random Forest algorithm has a slightly different approach than the other supervised learning algorithms. It is based on one of the most basic data structures in the whole computer science world: binary trees. Their main advantage is that the cost of building one is reasonably low $\mathcal{O}(n_{samples} n_{features} \log(n_{samples})))$ and the cost of using it is even lower $\mathcal{O}(\log(n_{samples}))$, where $n_{samples}$ refers to the number of records in the data set and $n_{features}$ to the dimension of the input space.

A decision tree is a binary tree where each node represents a test on a certain attribute and each branch represents the outcome of the test. When it comes to building a decision tree, there are a few different algorithms, but they all follow ideas from the same principle: using a greedy approach, at each step, the node with the most informative feature is chosen, starting from the root. Each construction algorithm has its advantages and downsides, therefore comes the idea of building multiple trees, using different approaches, and then ensembling the results.

Bootstrap aggregated (bagged) decision trees use the technique perturb-and-combine which alters each tree so that, when averaged, the resulting forest of trees performs better than each tree individually. When building each tree, we take bootstrap samples from the dataset (a bootstrap sample is a sample taken from the original dataset with replacement, so that we may get some data several times and others not at all) and we apply distinct construction algorithms to obtain trees that are accurate, but also slightly different.

Therefore, a simplified algorithm of building a Random Forest is represented in 'Algorithm 1'.

Algorithm 1 Building a Random Forest

```
function BUILD_RF(𝒩 , m)
    for each tree ∈ 𝒩 do
        choose a new bootstrap sample of the training set
        train a decision tree on the chosen set
        at each node of the decision tree, randomly select m features, compute
    the Gini impurity only on that set of features, selecting the optimal one
        repeat until the tree is complete
```

The Gini Impurity is a metric that aims to compute the probability of incorrectly classifying a randomly chosen element from the dataset, taking into consideration a uniform distribution. It is calculated as

$$G = \sum_{i=1}^{C} p(i) * (1 - p(i))$$

where C is the number of classes and $p(i)$ is the probability of randomly picking an element of class i.

2.6.1. *Experiments.* When it comes to avoiding under- and over-fitting, finding the optimal values of tunning parameters is essential. We have used two types of decision tree forests, particularly RandomForestRegressor (abbreviated as RF) and ExtraTreesRegressor (abbreviated as ER), from SciKit Learn Library. Experimentally testing different parameter values and observing the fluctuation of the MAE, we reached the following results:

- *n_estimators* (the number of trees in the forest) has an optimal value around $log_2(n\_features)$ (for our dataset around 30-40);
- *max_features* (the size of random subsets of features to consider when splitting a node) provides the best results when set to *Auto*. Insignificant improvements in computing time are observed when *max_features* is set to sqrt(*n_estimators*) in spite of an increment of the MAE (averaging all obtained results, from 2.22 to 2.81). As the number of records in our data sample is reasonably low, better result are achieved when max_features is equal to *n_features* (thus is set to *Auto*). On vaster input data, taking *max_features* equal to *n_features* conveys the impression that will improve both the accuracy and the computational time;

Model	*n_estimators*	MAE when $max\_features = n\_features$	MAE when $max\_features = log_2(n\_features)$
rf	800	2.638	2.596
rf	500	2.122	2.619
rf	300	2.181	2.627
rf	100	2.221	2.669
rf	70	1.997	2.2.79
rf	50	2.046	2.045
rf	40	1.613	2.635
rf	30	1.756	3.176
ef	40	3.184	2.959

- max_depth (the maximal depth of a decision tree) should be set to None in combination with *min_samples_split*=2 for an allowment of completely developing the trees;
- *bootstrap*=True (when set to True, bootstrap samples are used) prevents from over-fitting. When set to false (as used in ExtraTreesRegressor), the MAE of the training data was 4.15e-15, while the score obtained for testing data was considerably higher, namely 4.018.

As the process of building a decision tree is an independent process, parallelization can be used. Setting the parameter *n_jobs* to -1 will allow using all processors while building the models, this translating into the possibility of checking a large set of parameters in reasonably low computing time. Using Scikit-Learn's RandomizedSearchCV method, we can define a grid of hyperparameter ranges, and randomly sample from the grid, performing K-Fold cross-validation with each combination of values. This method will not try out all the possibilities from the parameters grid, but rather a fixed number of parameter settings is sampled. From the set represented in fig 8, we tried out a number of 150 combinations and obtained a MAE score of 1.069 using the following parameters $\{'n\_estimators' : 235,' min\_samples\_split' : 5,' min\_samples\_leaf' : 4, 'max\_features' :' log2',' max\_depth' : 10,' bootstrap' : False\}$

The Random Search narrows down the range of each hyperparameter, allowing us to focus the search. This process yielded to the possibility of using another Scikit-Learn method, GridSearchCV, that searches through all possible combinations from the parameter grid. We obtained that the optimal values should be a combination of the parameters from figure 9.

With 5-fold cross-validation, the total number of fits that were tried out was 5*2880 = 14400, leading us to a final MAE score of 1.0087.

FIGURE 6. Parameters tried out by the RandomizedSearchCV

```
{'bootstrap': [True, False],
 'criterion': ['mse', 'mae'],
 'max_depth': [10, 21, 32, 43, 54, 65, 76, 87, 98, 110, None],
 'max_features': ['auto', 'sqrt', 'log2'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [10, 42, 74, 106, 138, 171, 203, 235, 267, 300]}
```

FIGURE 7. Parameters tried out by the GridSearchCV

```
{'bootstrap': [True, False],
 'max_depth': [None, 10, 30],
 'max_features': ['auto', 'log2', None],
 'min_samples_leaf': [1, 2, 4, 6],
 'min_samples_split': [2, 3, 4, 5],
 'n_estimators': [30, 40, 50, 70, 90, 100, 200, 250, 300, 400]}
```

3. RESULTS

3.1. **DNN Results.** On average, the MAE on both training data and testing data was around 1.100, the model usually being slightly better suited on the training batch (around 0.3% difference between the testing and training data). We considered an accurate prediction and value that was within 5 percent of the target. By these metrics, DNN has an accuracy of 93-96% on both training and testing data, an abnormally good result considering today's ML standards. This result leads us to believe that in fact, the values we predicted were already predicted by an ML algorithm. The versatility of Neural Networks seems to adapt well to different kinds of data, making it a trustworthy way of making predictions.

3.2. **SVM Results.** Our final SVM model uses a rbf kernel with a C value of 100, an 'auto' gamma, and a StandardScaler. It predicts the target values surprisingly well, with very little training time, showcasing an R2 score of 0.956 and a MAE of 0.914. Combined with the fact that high values of C suggest lack of noise in the dataset, this leads us to believe that, indeed, the dataset we used does not present real life data, but rather scientific findings of another ML model.

FIGURE 8. Deep Neural Network Results

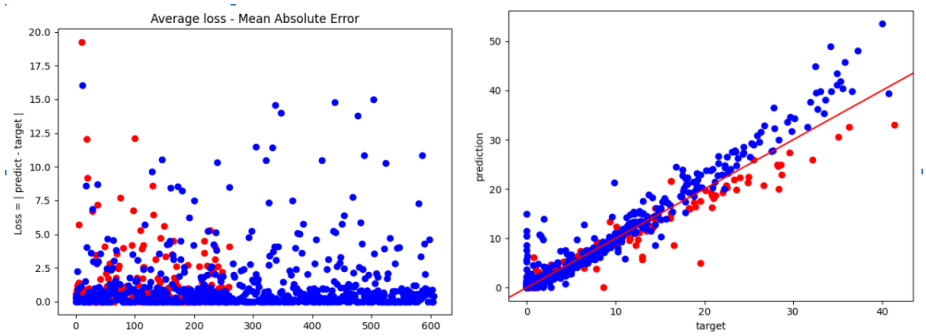
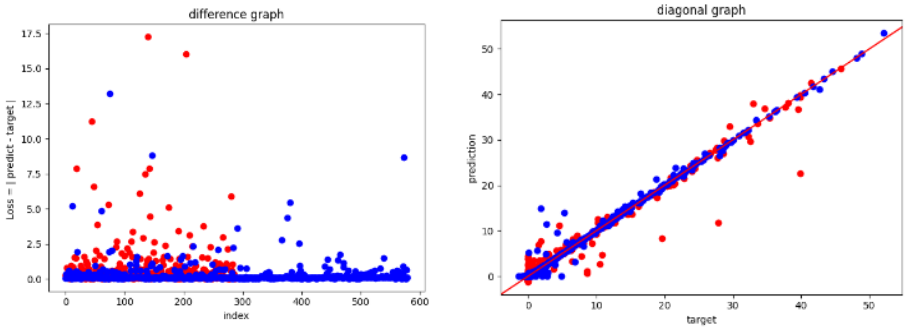


FIGURE 9. SVM Results



Algorithm	R2	Acc. training	Acc. val	MAE testing	MAE training
DNN	0.891213	96.1685%	96.204%	0.95152	0.97703
SVM	0.95670	99.1379%	96.1672%	0.91407	0.3103
RF	0.90162	98.0030%	96.7741%	1.0822	1.0087

4. CONCLUSIONS

We have found intelligent models with attractive results when it comes to predicting indoor radon levels from ordinary data, such as temperature, humidity, and geographical area. When dealing with datasets with very little noise, it also seems that no matter the method we choose, we may fine-tune it to great accuracy of prediction. The decision of choosing model in such case can come down to the size of the dataset and execution time. Our models may be used to estimate a general threat level for buildings where radon

measurements are virtually impossible. Further experimentation with unsupervised learning methods may yield us information regarding what is exactly the source of high radon levels, and whether prevention is possible.

References

[1] Anguita, D., Ghio, A., Ridella, S., and Sterpi, D. K-fold cross validation for error rate estimate in support vector machines. In *DMIN* (2009), pp. 291–297.

[2] Berrar, D. Cross-validation., 2019.

[3] Darby, S., Hill, D., Auvinen, A., Barros-Dios, J., Baysson, H., Bochicchio, F., Deo, H., Falk, R., Forastiere, F., Hakama, M., Heid, I., Kreienbrock, L., Kreuzer, M., Lagarde, F., Mäkeläinen, I., Muirhead, C., Oberaigner, W., Pershagen, G., Ruano-Ravina, A., and Doll, R. Radon in homes and risk of lung cancer: Collaborative analysis of individual data from 13 european case-control studies. *BMJ: British medical journal 330* (02 2005), 223.

[4] Florica, S., Dicu, T., Burghele, B.-D., Moldovan, M., Szacsvai, K., Tenter, A., Papp, B., Beldean, S., Istrate, A., Catalina, T., et al. Indoor radon related with the geology in romanian urban agglomerations (cluj-napoca).

[5] Khan, S., Taron, J., and Goodarzi, A. Machine learning as a next-generation tool for indoor air radon exposure prediction. *SAGE Open* (01 2020).

[6] Kohavi, R., et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai* (1995), vol. 14, Montreal, Canada, pp. 1137–1145.

[7] Kropat, G. *Predictive analysis and mapping of indoor radon concentrations in Switzerland*. PhD thesis, Université de Lausanne, Faculté de biologie et médecine, 2015.

[8] Lameski, P., Zdravevski, E., Mingov, R., and Kulakov, A. Svm parameter tuning with grid search and its impact on reduction of model over-fitting. In *Rough sets, fuzzy sets, data mining, and granular computing*. Springer, 2015, pp. 464–474.

[9] Mantovani, R. G., Rossi, A. L., Vanschoren, J., Bischl, B., and De Carvalho, A. C. Effectiveness of random search in svm hyper-parameter tuning. In *2015 International Joint Conference on Neural Networks (IJCNN)* (2015), Ieee, pp. 1–8.

[10] Marsland, S. *MACHINE LEARNING, An Algorithmic Perspective*. CRC Press, 2015.

[11] Nilsson, R., and Tong, J. Opinion on reconsideration of lung cancer risk from domestic radon exposure. *Radiation Medicine and Protection 1*, 1 (2020), 48–54.

[12] Organization, W. H. Who handbook on indoor radon: a public health perspective.

[13] Rafique, M., Tareen, A. D. K., Mir, A. A., Nadeem, M. S. A., Asim, K. M., and Kearfott, K. J. Delegated regressor, a robust approach for automated anomaly detection in the soil radon time series data. *Scientific reports 10*, 1 (2020), 1–11.

[14] Timar-Gabor, Alida, K. S., and Dinu, A. "radon exposure and lung cancer risk in romania.". *Journal of Environmental Protection and Ecology 10.1 (2009): 94-103* (2009).

[15] Timkova, J., Fojtikova, I., and Pacherova, P. Bagged neural network model for prediction of the mean indoor radon concentration in the municipalities in czech republic. *Journal of environmental radioactivity 166* (2017), 398–402.

Babeş-Bolyai University, Mihail Kogălniceanu 400000, Cluj-Napoca, Romania
Email address: {toie3033, tdie3030, saie3018}@scs.ubbcluj.ro