Neural-network-based ultra-short-term wind forecasting

Mladen Dalto, Mario Vašak, Mato Baotić, 
Jadranko Matuško
University of Zagreb, Faculty of Electrical Engineering and Computing, 
mladen.dalto@fer.hr, mario.vasak@fer.hr, 
mato.baotic@fer.hr, jadranko.matusko@fer.hr

Kristian Horvath
Meteorological and Hydrological Service; 
Zagreb, Croatia 
kristian.horvath@cirus.dhz.hr

Abstract

In recent years rapid growth of wind power generation in many countries around the world has highlighted the importance of wind prediction. In this work neural networks are used for ultra-short-term wind prediction. In many instances reported in the literature neural network exhibits poor performance very often because no complexity reduction methods were considered. To that end, in this paper two input variable selection algorithms based on partial mutual information are compared for further use with nonlinear models such as neural networks. Performance improvements of the proposed prediction system are compared to neural networks without input variable selection, and valid for locations near Split, Croatia. The use of neural network drastically outperforms simple persistence estimator on 3 hour horizon.

1. Introduction

In recent years, environmental considerations have prompted the use of wind power as a renewable energy resource. Rapid growth of wind power generation in many countries around the world has highlighted the importance of wind prediction. The reasons are twofold: facilitating integration of wind energy in the electricity system and advanced control of wind turbines and farms. Numerical weather prediction lacks ultra-short-term prediction capabilities mainly because of the time-consuming simulations of atmospheric models, especially if data assimilation is a component of the prediction system. Besides providing improved forecast up to 3 hours ahead, applications of ultra-short-term wind prediction are in optimal wind turbine and wind farm control [6].

Frequently used time-series models with well-developed theoretical background lack structural capabilities for modelling complex dynamics such as that of the wind [11]. In this work neural networks are trained on historical measurement and then used online for ultra-short-term wind forecasting as in [11], [1], [3], [2] and others.

Large amount of redundant and non-informative data inhibits the training algorithm performance. Often neural networks used do not include appropriate complexity reduction methods which results in lower performance [8]. Input variable selection based on partial mutual information (PMI) is found appropriate for use with nonlinear models such as neural networks. PMI based on k Nearest Neighbours (kNN) [5] and probability kernel density estimation (KDE) [9] are compared. Developed neural network methodology can be used efficiently at a low computational cost for any location with varying number of input variables and historical data samples. Performance improvements of the proposed prediction system relative to simple persistence and to commonly used neural prediction methods are evaluated for locations near Split, Croatia.

2. Neural networks

Multi-Layer Perceptron (MLP) structure of Neural Network (NN) was chosen for this work. This structure is a feedforward map between inputs to outputs. Every node that is not in the input layer is a neuron (processing element) with a nonlinear activation function. In our case only hyperbolic tangent functions are used as activation functions. Connection strength between neurons is represented by synaptic weight coefficients which give certain weights to input signals. There is no theoretical upper limit to the number of layers a MLP network can have, but here MLP NNs with one hidden layer are used. In the following work we make use of notation to denote dimension of some vector \( v \): \( n(v) \), or \( n(l) \) when denoting the dimension of a layer, where \( 0 \leq l \leq L \). Mathematical description of MLP NN with \( L \) layers can be formulated as follows:

\[
y_0 = x. \tag{1}
\]

\[
x_l = [y_{l-1}, 1]^T, \quad 1 \leq l \leq L, \tag{2}
\]

\[
v_l = W_l \cdot x_l, \quad 1 \leq l \leq L, \tag{3}
\]

\[
y_l = \psi_l(v_l), \quad 1 \leq l \leq L, \tag{4}
\]

where:

\( x = [x_1, x_2, \ldots, x_n(z)]^T \in \mathbb{R}^{n(z) \times 1} \) - input vector;

\( y_0 = [y_{0,1}, y_{0,2}, \ldots, y_{0,n(0)}]^T \in \mathbb{R}^{n(0) \times 1} \) - layer 0 output vector;

\( x_l = [x_{l,1}, x_{l,2}, \ldots, x_{l,n(l-1)}, x_{l,n(l-1)+1}]^T \in \mathbb{R}^{(l-1)+1 \times 1} \)
of choosing input variables is simplified with a priori assumptions of the functional form of the model based on the physical process being modelled. In the case of NN, and other similar data based models, there is no assumption of a specific input-output mapping structure, so a general structure is assumed. In that case several problems occur:

- large number of input variables,
- correlation amongst available variables,
- uninformative variables.

Many IVS procedures do not give good results because linear relationships are assumed or because of redundancy in the input candidate set. The degree of correlation is frequently measured by Pearson correlation coefficient, which is sensitive only to a linear relationship between two variables (which may exist even if one is a nonlinear function of the other). Correlation analysis is the basis for principal component analysis which is often used in IVS. Two main problems of this and similar analysis are the sensitivity of Pearson correlation coefficient to data transformations and noise [8]. Recently Mutual information (MI) is being used as a more suitable measure for IVS. Mutual information is defined in [4] as:

\[
I(X,Y) = \int_X \int_Y p(x,y) \log \left( \frac{p(x,y)}{p(x)p(y)} \right) dx dy,
\]

where \(p(x,y)\) represents joint density estimation and \(p(x)\) and \(p(y)\) marginal distributions of \(X\) and \(Y\). In practical context probability distributions in 10 are rarely known. Base of the logarithm function is usually 2 or \(e\), here we use \(e\). MI advantage is in the fact that it does not make assumptions about the structure of the analysed variables and is invariant to transformations and noise [8]. Naive application of MI cannot detect redundancy between candidate input variables and lacks a stopping criterion. Partial mutual information (PMI) is a nonlinear IVS algorithm that can completely or partially overcome these problems. PMI between the output \(Y\) and candidate \(C_j\) with preselected predictors \(S\) is defined as:

\[
PMI(Y,C_j|S) = H(Y,S) + H(S,C_j) - H(S) - H(Y,S,C_j),
\]

where \(Y, C_j\) are random scalars, \(S\) a random vector and \(H(\cdot)\) represents Shannon entropy:

\[
H(X) = -\sum_i p(x_i) \log p(x_i).
\]

A detailed formulation of entropy for multiple arguments is provided in [5]. PMI is symmetric under the same condition \(PMI(Y,C_j|S) = PMI(C_j,Y|S)\) which follows from MI symmetry (\(I(X,Y) = I(Y,X)\)). In a special case, when no predictors are preselected, PMI becomes \(PMI(Y,C_j|\emptyset) = I(Y,C_j)\). Two variants of PMI were implemented and tested: one based on KDE from [8] and another based on kNN from [5].

Consider a system with output \(Y\), and with \(S\) and \(C_j\) as predictor candidates. If \(I(Y;S) \geq I(Y;C_j) \forall j\),
Then the following Gaussian noise was added: data was mapped to have zero mean and unit variance. As a preprocessing step to partial mutual information, all predictor $S$ is selected as a first candidate because it introduces more information required to predict $Y$. Now we perform a second step of the algorithm with preselected $S$. The information contribution of any candidate $C_j$ is represented by $I(C_j; Y|S)$ as in Figure 1. Throughout this paper a set of real measurements are used to test the algorithms. To simplify notation we use $D = \{ (x_i, y_i) \}_{i=1}^{N}$. As a preprocessing step to partial mutual information, all data was mapped to have zero mean and unit variance. Then the following Gaussian noise was added: 

$$
\left[ z_{1,i}, z_{2,i}, \ldots, z_{n(x),i} \right]^T = \left[ z_{1,i}, z_{2,i}, \ldots, z_{n(x),i} \right]^T + \mathcal{E},
$$

$$
\mathcal{E} \sim \mathcal{N}(0_{n(x)}, 10^{-4} \cdot I_{n(x)}).
$$

As the final preprocessing step all data was normalized to $[-1, 1]$. Although mutual information is invariant to transformations and noise it has positive effects on potential quantization problems [13] when performing KDE and simplifies implementation because now we can assume all elements of $O$ are unique and hence the cardinal number $|O|$ is equal to the number of samples $N$. If we, for example, take quantized time values and add noise, we can assume unique samples. This allows simpler KDE implementation because the size of the time values set does not change.

### 3.1 Kernel based PMI

To estimate $I(X; Y)$ from (10) Monte Carlo integration is used and the following approximation is derived:

$$
I(X; Y) \approx \frac{1}{|D|} \sum_{i=1}^{|D|} \log \left( \frac{f(x_i, y_i)}{f(x_i) \cdot f(y_i)} \right).
$$

Because we do not know the probability distribution functions, their estimates are used:

$$
\hat{I}(X; Y) = \frac{1}{|D|} \sum_{i=1}^{|D|} \log \left( \frac{\hat{f}(x_i, y_i)}{\hat{f}(x_i) \cdot \hat{f}(y_i)} \right),
$$

where individual and pair variables are drawn (recorded) from the dataset $D$. For probability density estimation in (16) Gaussian kernels are used:

$$
\hat{f}(z) = \frac{1}{\det(\Sigma)|O|^{n(z)+1}} \sum_{i=1}^{|O|} \exp \left( -\frac{\|z - z_i\|_2^2}{2h^2} \right),
$$

where $\|z - z_i\|_2$ is Mahalanobis distance defined as:

$$
\|z - z_i\|_2 = (z - z_i)^T \Sigma^{-1} (z - z_i) \in \mathbb{R}
$$

and $\Sigma = \mathbb{E}[(z - \mathbb{E}[z])(z - \mathbb{E}[z])^T] \in \mathbb{R}^{n(z) \times n(z)}$ is the sample covariance matrix. Kernel density estimation is sensitive to the choice of bandwidth $h$. In our work bandwidth is chosen as in [12]:

$$
h = \left( \frac{4}{|O|(|n(z) + 2|)} \right)^{1/(n(z)+4)}.
$$

For $z \in \mathbb{R}$ in (17) we obtain a Gaussian kernel density estimator of a single random variable. Using the analogy to partial correlation coefficient PMI can be estimated from residuals:

$$
u = [Y - \hat{m}_Y(S)] \in \mathbb{R}^{|O| \times 1},
$$

$$
v = [C_j - \hat{m}_{C_j}(S)] \in \mathbb{R}^{|O| \times 1},
$$

where $Y^T = [y_1, y_2, \ldots, y_O]^T$ is the output sample vector and $S \in \mathbb{R}^{n(S) \times m(S)}$ a sample matrix of preselected predictors. Vectors $u$ and $v$ represent realizations of random variables $Y$ and $U$, respectively. The Nadaraya-Watson kernel estimate $\hat{m}_Y: \mathbb{R}^{1 \times m(S)} \rightarrow \mathbb{R}$ is given by:

$$
\hat{m}_Y(s_k) = \frac{\sum_{i=1}^{|O|} \hat{f}(s_k - s_i)^T \cdot y_i}{\sum_{i=1}^{|O|} \hat{f}(s_k - s_i)^T}.
$$

Use of this estimator is derived as an approximation $\mathbb{E}(Y|S)$ using KDE. It is useful to write (22) in matrix form:

$$
\hat{Y} = H \cdot \hat{Y},
$$

where $H$ is the hat matrix (sometimes also called projection matrix):

$$
H = \left[ \begin{array}{c} \hat{f}(s_k - s_i)^T \\ \sum_{i=1}^{|O|} \hat{f}(s_k - s_i)^T \end{array} \right]_{k,i} \in \mathbb{R}^{|O| \times |O|}.
$$

To obtain $PMI(Y, C_j|S)$ finally we calculate $\hat{I}(V; U)$. The Akaike Information Criterion (AIC) is given in Algorithm 1. The Akaike Information Criterion (AIC) is given by:

$$
AIC_k(u) = |O| \cdot \ln \left( \frac{1}{|O|} \sum_{i=1}^{|O|} u_i^2 \right) + 2 \cdot \text{trace}(H).
$$
and it was used as a stopping criteria. With every iteration $k$ in Algorithm (1) a new predictor is joined to $S$, giving residuals $u_i$. AIC is used as a measure of the relative quality of a statistical model, for a given set $S$. AIC deals with the trade-off between the model performance and the complexity of the model measured in effective number of parameters $\text{trace}(\mathbf{H})$. Increase in AIC value indicates that we have selected enough candidates for predicting $Y$.

### 3.2 kNN based PMI

The basis of this approach is laid out in [7]. The authors use Kozachenko-Leonenko estimate for Shannon entropies:

$$
\hat{H}(X) = \psi(N) - \psi(k) + \log(c_n(X)) + \frac{n(X)}{N} \sum_{i=1}^{N} \log(\epsilon(i))
$$

(26)

where $\epsilon(i)$ is twice the distance from sample $x_i$ to its $k$-th neighbour, $c_n(X)$ is the volume of $n(X)$ dimensional unit norm ball and $\psi(\cdot)$ is the digamma function. Entropy estimator (26) was generalized in [5] to include an estimator for PMI:

$$
P\hat{MI}(Y,C_j|S) = H(k - 1)
$$

$$
- \frac{1}{N} \sum_{i=1}^{N} \left( H_{N_Y} + H_{N_{C_j}} - H_{N_Y}(i) + H_{N_{C_j}}(i) \right).
$$

(27)

where $H_{\epsilon}$ = $\gamma + \psi(n)$ is the harmonic number with previously stated connection to digamma function for integers $n$. If $S = \emptyset$ then (27) reduces to:

$$
\hat{I}(Y,C_j) = H(k - 1) + H(N - 1)
$$

$$
- \frac{1}{N} \sum_{i=1}^{N} \left( H_{N_Y} + H_{N_{C_j}} \right).
$$

(28)

Number of elements $N_{z_1,\ldots,z_{n(z)}}$ in $z \in \mathcal{O}$ with distance less than the distance to the $k$-th neighbour is defined by:

$$
N_{z_1,\ldots,z_{n(z)}}(i) = |\{||z_i - z_k|| < \epsilon_k(i) \; | \; i^* \in [1,|\mathcal{O}|]\}|,(29)

where the norm is defined as:

$$
||z|| = \max(||z_1||,\ldots,||z_{n(z)}||)
$$

(30)

and the nested norms we use are also $||.||_{\infty}$. The entire algorithm is described by Algorithm 2. As it can be noted, kNN based algorithm lacks a stopping criteria and hence goes trough the entire candidate set $C$ until it is sorted in $S$ by partial mutual information values.

### 3.3 PMI algorithms comparison

For PMI algorithm testing, samples of a linear system with noise were generated. Advantage of using synthetically generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforehand. Commonly generated examples for testing is a known targeted dependency beforeh.
concluded that kernel based PMI has a great benefit of a stopping criteria which is especially emphasized when using large candidate set, as the algorithm does not have to check and sort every candidate by its partial mutual information value. Besides having a stopping criteria our implementation of KDE based PMI was faster than kNN based PMI and was used in Section 4.2 for NN IVS. Although the previous analysis was sufficient to motivate the choice of KDE PMI algorithm, most of the interesting dependencies for neural networks are nonlinear and hence PMI algorithms should be tested on such examples. Alongside AR9, nonlinear autoregressive tests were performed in [9].

4. Data and features

Data consists of physical quantities measured with sampling time of $T_s = 10$ min. They represent either 10-min mean, maximum or 10-min interval terminal values of various quantities; times of maximum, minimum within $T_s$ and other derived features. Use of supervised learning in (8) requires a proper series of cause-consequence data pairs. Data pairs need to be formed from historic data which contain considerable amount of missing values and other faulty measurements. These were labeled as such, and removed from the set. Original measurements of outputs $Y$ represent wind speed and direction. Because of seemingly abrupt changes in wind direction measurements $\theta(i)$ when crossing $360^\circ$ or $0^\circ$ we use orthogonal wind components given by:

\[
\begin{align*}
    v_x(i) &= ||v(i)|| \cdot \cos \left( \frac{\pi \theta(i)}{180} \right), \\
    v_y(i) &= ||v(i)|| \cdot \sin \left( \frac{\pi \theta(i)}{180} \right),
\end{align*}
\]

as illustrated in Figure 3. By doing so smooth functions are used for mapping direction and wind speed change. Among all measured data some is unnecessary (redundant, not useful, unknown,...) and some needs to be artificially derived (e.g. turbulence, gustiness) to increase the chance of successful neural network training. Three years (2010-2012) of measurement data were obtained for wind on 10 m relative hight from the ground in main meteorological stations of Split, Sibenik and Knin locations in Croatia and used for neural network training, validation and testing. Data was divided according to Table 1. All of the dataset elements were randomly sampled from the initial data.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Training</th>
<th>Validation</th>
<th>Testing</th>
<th>Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Split</td>
<td>70%</td>
<td>15%</td>
<td>15%</td>
<td>95756</td>
</tr>
<tr>
<td>Sibenik</td>
<td>70%</td>
<td>15%</td>
<td>15%</td>
<td>9498</td>
</tr>
<tr>
<td>Knin</td>
<td>70%</td>
<td>15%</td>
<td>15%</td>
<td>4959</td>
</tr>
</tbody>
</table>

Table 1: Dataset division

4.1 Dataset for NN without IVS

A simple way to choose NN inputs is to use as much data as available and, in addition, generate some artificial features to present new predictors with high relevance. The predictors candidate set $\mathcal{C}$ consists of variables in Table 2 i.e. $d + 1$ of their historic values $\mathcal{C}_d = [z_0, z_{-1}, ..., z_{-d}]$:

\[
\mathcal{C} = \{ \mathcal{C}_{d,i} \mid i \in [d+1, N] \},
\]

In this way we provide the NN predictors $X = \mathcal{C}_d$ with samples from $\mathcal{C}$.

4.2 Datasets for NN with IVS

The choice between predictor candidates in $\mathcal{C}$ is optionally made with some of the IVS algorithms implemented and described in Section 3. If we define the NN to have $n(y_{2}) = 2T$ outputs, it is necessary to use PMI Algorithm 1 for every output $y_{2,j}$ and candidates in

![Figure 2: Relative importance of delayed components](image1)

![Figure 3: Wind vector decomposition to orthogonal North (0°) and East (90°) components](image2)
C to obtain selected candidates $S_j$ for $j = 1, \ldots, n(y_2)$. After obtaining $S_j$ we use their union:

$$S = \bigcup_{j=1}^{n(y_2)} S_j$$

(35)

to form the final vector of predictors $X = S$ for the use with the NN. For our concrete application number of variables was reduced according to Table 3.

<table>
<thead>
<tr>
<th>Row</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10 min wind speed mean in North direction</td>
<td>m/s</td>
</tr>
<tr>
<td>2</td>
<td>10 min wind speed mean in East direction</td>
<td>m/s</td>
</tr>
<tr>
<td>3</td>
<td>10 min wind speed maximum in North direction</td>
<td>m/s</td>
</tr>
<tr>
<td>4</td>
<td>10 min wind speed maximum in East direction</td>
<td>m/s</td>
</tr>
<tr>
<td>5</td>
<td>time of maximum wind speed</td>
<td>s</td>
</tr>
<tr>
<td>6</td>
<td>terminal wind speed in North direction on the 10 min interval</td>
<td>m/s</td>
</tr>
<tr>
<td>7</td>
<td>terminal wind speed in East direction on the 10 min interval</td>
<td>m/s</td>
</tr>
<tr>
<td>8</td>
<td>time of day</td>
<td>s</td>
</tr>
<tr>
<td>9</td>
<td>$\sin(2\pi((t_{sec} - 7200)/86400))$</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>$\cos(2\pi((t_{sec} - 7200)/86400))$</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>sun zenith</td>
<td>rad</td>
</tr>
<tr>
<td>12</td>
<td>sun azimuth</td>
<td>rad</td>
</tr>
<tr>
<td>13</td>
<td>moon right ascension</td>
<td>rad</td>
</tr>
<tr>
<td>14</td>
<td>moon declination</td>
<td>rad</td>
</tr>
<tr>
<td>15</td>
<td>moon distance</td>
<td>km</td>
</tr>
<tr>
<td>16</td>
<td>gustiness</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>(terminal-maximal) 10 min wind speed</td>
<td>m/s</td>
</tr>
<tr>
<td>18</td>
<td>(terminal-mean) 10 min wind speed</td>
<td>m/s</td>
</tr>
<tr>
<td>19</td>
<td>(maximal-mean) 10 min wind speed</td>
<td>m/s</td>
</tr>
<tr>
<td>20</td>
<td>$(18$th row)$^2$ +$(19$th row)$^2$</td>
<td>$(\frac{m}{s})^2$</td>
</tr>
<tr>
<td>21</td>
<td>artificial wind speed sum</td>
<td>m</td>
</tr>
<tr>
<td>22</td>
<td>10 min pressure mean</td>
<td>Pa</td>
</tr>
<tr>
<td>23</td>
<td>10 min pressure maximum</td>
<td>Pa</td>
</tr>
<tr>
<td>24</td>
<td>time of maximum pressure</td>
<td>min</td>
</tr>
<tr>
<td>25</td>
<td>terminal pressure</td>
<td>Pa</td>
</tr>
<tr>
<td>26</td>
<td>10 min maximum pressure time</td>
<td>min</td>
</tr>
<tr>
<td>27</td>
<td>(terminal-maximal) pressure</td>
<td>Pa</td>
</tr>
<tr>
<td>28</td>
<td>(terminal-mean) pressure</td>
<td>Pa</td>
</tr>
<tr>
<td>29</td>
<td>(maximal-mean) pressure</td>
<td>Pa</td>
</tr>
<tr>
<td>30</td>
<td>10 min temperature mean</td>
<td>K</td>
</tr>
<tr>
<td>31</td>
<td>10 min temperature maximum</td>
<td>K</td>
</tr>
<tr>
<td>32</td>
<td>maximum temperature time</td>
<td>min</td>
</tr>
<tr>
<td>33</td>
<td>10 min temperature minimum</td>
<td>K</td>
</tr>
<tr>
<td>34</td>
<td>minimum temperature time</td>
<td>min</td>
</tr>
<tr>
<td>35</td>
<td>terminal temperature</td>
<td>K</td>
</tr>
<tr>
<td>36</td>
<td>(terminal-mean) temperature</td>
<td>K</td>
</tr>
<tr>
<td>37</td>
<td>artificial squared temperature</td>
<td>$K^2$</td>
</tr>
<tr>
<td>38</td>
<td>10 min mean relative humidity</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>10 min maximum relative humidity</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>maximum relative humidity time</td>
<td>min</td>
</tr>
<tr>
<td>41</td>
<td>10 min minimum relative humidity</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>minimum relative humidity time</td>
<td>min</td>
</tr>
<tr>
<td>43</td>
<td>terminal relative humidity</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>(terminal-mean) relative humidity</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>artificial squared relative humidity</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Variables derived from data

4.3 Persistent forecast and data

Persistence or persistent estimator is the simplest forecast model and in this work serves for NN forecast evaluation exclusively. It assumes that wind speed and direction will remain the same for the prediction horizon $T$. It is defined as follows:

$$\hat{y}(i + k|\nu) = y(i)$$

(36)

where $k = 1, \ldots, T$ and $i = 1, \ldots, |C|$ samples. For the purpose of estimators comparison, a dataset with constant estimates during the prediction horizon $T$, were generated.

5. Neural network results

Short functional description of the used neural network is given by:

$$\begin{bmatrix} \hat{V}_x(t + T|t) \\ \hat{V}_y(t + T|t) \end{bmatrix} = f_N(X(t), \Theta) \in \mathbb{R}^{2T \times 1},$$

(37)

with:

- $V(t) = [v(t), v(t - 1), \ldots, v(t - (T - 1))]^T$-vector of a wind component $v_x$ or $v_y$;
- $X(t)$-vector of predictors;
- $\Theta$ - model parameters;
- $T$ - prediction horizon.

Use of a single neural network with multiple outputs was motivated by simplicity of having a single output vector for simultaneous prediction of wind speed mean in North and East direction, as well as by simplicity of data preparation, training, testing and use of such a network. Samples of a datasets for NN training, validation and testing were the same for NN with and without IVS, only the number of inputs varies. This ensures unbiased RMSE performance criteria. Results obtained with NN (37) are evaluated with RMSE criterion on a test dataset, which is not used during training, and shown in Figure 4.
When using a neural network with partial mutual information input variable selection we obtained higher performance with lower computational complexity due to reduction in number of inputs. For locations with less data samples neural networks with partial mutual information input variable selection have lower RMSE value then neural networks trained with all predictors as inputs. As the number of samples of a training dataset grows, RMSE decrees and neural networks with PMI IVS have similar performance as neural networks trained with all predictors. However, the key difference is that it uses only a smaller fraction of all available predictors and hence drastically reduces the computational complexity and training time of a neural network with partial mutual information input variable selection. An example of wind speed mean in North direction and forecasted values are presented in Figure 5. As expected, 10 min ahead forecast horizon captures ramp and other events with more accuracy then 3 h ahead prediction.

Figure 5: Measured and predicted wind speed in North direction for 10 min and 3 h prediction horizon on Šibenik location.

6. Conclusions

Classic MLP neural network approach, with all available predictors as inputs, outperforms the persistent estimator except for 10-30 min horizon. The use of neural network with partial mutual information input
variable selection gave higher performance with lower computational complexity than the MLP neural network. Computational complexity and consequently the training time of a neural network is reduced due to use of only a smaller fraction of all available predictors. Developed neural network methodology can be efficiently used on any location with associated sufficiently informative measurement dataset. Wide application to wind prediction on various sites is possible due to neural network structural ability to handle varying number of input variables and model highly nonlinear input-output relationships. Partial mutual information algorithm is successfully used for input variable selection which consequently reduces complexity and neural network training time. There is a great potential of the presented method in improvement of wind farm power delivery prediction and optimal wind farm/wind turbines control – here foremost much can be done for smart yaw control.

Acknowledgement

The investigations described in this paper are part of the project "Weather Intelligence for Wind Energy" (WILL4WIND) supported by the European Union under contract IPA2007/HR/16IPO/001-040507.

References