AN INFORMATION RETRIEVAL APPROACH TO PREDICTING METEOROLOGICAL DATA

A. Kidron and S. T. Klein
Department of Computer Science
Bar Ilan University
Ramat Gan 52900, Israel
(972-3) 531 8865

adiramk@netvision.net.il tomi@cs.biu.ac.il

ABSTRACT

An Information Retrieval approach to the prediction of values of time series is suggested. Instead of trying to model the data, a pattern is searched for in a history file serving as "text", and relevant occurrences are used as basis for extrapolation. The application is to meteorological data, providing short-term predictions for certain weather features. The experiments show that the algorithm has a good ability of predicting the requested information, and that the predictions are far more accurate than those provided by an analytical prediction procedure, based on approximations using Fourier transforms.

Keywords
Meteorological data, prediction, time series, pattern matching, multivariate prediction, Information Search and Retrieval.

1. Introduction

Classical Information Retrieval is concerned, on the one hand, with procedures to help a user satisfy her information needs by facilitating her access to large amounts of data, on the other hand, with techniques to evaluate her (dis)satisfaction with whatever data the system provided. The underlying database is usually a large collection of textual documents, written in some natural language. The present study is a spin-off of the classical IR paradigm, trying to apply Information Retrieval techniques to a similar, yet different, environment.

The database we consider consists of meteorological data, collected over some period of time. The "text" thus is a long sequence of measurements, taken at regular intervals, and representing temperature, wind speed, wind direction, etc. Such data is typically used in weather forecasts and has obvious importance to many aspects of our lives. One way to predict the weather in the near future is to try to model the (latest) measured values as some function f, and then assume that future values will follow the same function f, which allows extrapolation. We shall try to predict the future using an Information Retrieval approach: instead of formulating a query consisting of keywords and searching for their occurrences within a given text, our query consists of a given weather pattern, e.g., some meteorological values collected in the past three hours. This pattern is then looked for in the database, with the underlying assumption that if a similar pattern has been observed at some earlier point in time, the continuation of the earlier occurrence is a good estimate to the continuation of the current occurrence.

There are of course some major differences which have to be taken into account. Our data is continuous, so that an exact match of the pattern is not expected to be found. We shall therefore have to define metrics and use approximate matching techniques. Instead of relying on subjective user evaluation as to the relevance of the retrieved items, the degree of relevance can then actually be measured, and the highest-ranking matches can be combined with appropriate weights to form the requested extrapolation.

The organization of the paper is as follows: in the next section, we formalize our definitions and review some of the previous work. Section 3 presents the algorithm used and deals with implementation issues, and Section 4 addresses the comparison with analytical prediction. Finally, Section 5 describes the experiments, comparing all the methods.

2. Background and Definitions

Quantitative methods of future forecasting are based on the assumption that tomorrow's world will be similar to today's, and that patterns observed in the past will continue into the future. If historical patterns can be expected to persist into the future, one should be able to produce a relatively accurate forecast.

A *time series* consists of measurements or observations of a natural, technical or economic process that are made sequentially in time. In general, consecutive samples of time series are dependent on each other to an extent dictated by the underlying process in question. Because of this dependency, it makes sense and it is possible to predict the future of the series [1]. Making a prediction of time series is in fact modeling the dependencies between consecutive samples made by a process. There are many statistical methods that specialize in this, whether the dependencies are linear or nonlinear in nature. However, there are two major problems relating to these measurements. The first problem is noise that entered during the measuring process. The second problem is the possibility that the process involves unknown variables, which have not been measured.

In order to overcome these problems some analytical models and theories based on the available measurements only were developed in the 1980s. Packard et al. [2] and Takens [3] introduced the first formal demonstration of the connection between the observed measurements and the underlying process. Takens' Time-Delay Embedding Theorem implies that there exists, in principle, a non-linear autoregression of the form $x(t) = g[x(t-1), x(t-2), \ldots, x(t-T)]$ which models the series exactly. The problem that remains is how to find this non-linear regression.

In 1987, based on Takens' and Packard et al. works, Farmer and Sidorowich [4] have found that chaotic time series prediction using local approximation techniques is several orders of magnitude better than using global approximations. This gave rise to the use of local approximation methods including nearest neighbor methods or pattern imitation for time series prediction.

Beside the traditional statistical methods, one can also find analytical methods for calculating approximations. While considering the time series as a set of samples of a continuous process acting like a continuous function, one can calculate approximations for the function at a certain point using techniques such as interpolation, Taylor series expansion (local approximation), Fourier analysis (global approximation), etc. More recently, advanced methods on prediction such as neural networks, genetic algorithms, fuzzy methods [5] and general regression neural networks [6] have also been frequently used.

A large number of studies have been done in order to predict financial data such as stock and currency rates, development of production and consumption etc. In this research we concentrate on predicting local meteorological data such as temperature, wind speed and direction. We focus on this problem because of its vast impact on our life: from the influence on our daily activity, through more important issues such as planning the activity of naval fleets or aerial forces, to critical influence on people’s lives such as the ability to analyze the spreading of chemical and biological materials during unconventional warfare.

Given a time-series $\vec{x}(t),\ t = 0,1,2,...,$ the *current* state of the series is defined as the last (current) element of the series. One can expand this definition to the vector that contains the currently last *m* elements of the series. We will index the current state of a series with the index of the last element in the sequence:

$$\vec{x}_t = \{\vec{x}(t), \vec{x}(t-1), ..., \vec{x}(t-m+1)\}$$

The current state is in fact our *query pattern*. Similarly, any sequence of m elements from the series will be defined as a *state*:

$$\vec{x}_j = \{\vec{x}(j), \vec{x}(j-1), \ldots, \vec{x}(j-m+1)\} \qquad \text{where } m \le j \le t.$$

Providing an estimation of only one value into the future of the time series is called *simple prediction*, while the prediction of several consequent values into the future is called *multiple forecasting*. One can carry out multiple forecasting by repeating the simple prediction process p times, where each time the predicted value is added to the end of the series and updates the current state (iterative forecast / prediction), or alternatively, one can perform p-steps ahead prediction (direct forecast / prediction).

In the domain of meteorology, forecasts usually refer to a *lead time* (in most cases 12 or 24 hours, but sometimes it could be a few minutes — depending on the purpose of the forecast). Producing predictions according to a known lead time is actually performing multiple forecasting with a desired length.

3. The Suggested Algorithm

In contrast to some of the above mentioned methods, our suggested algorithm produces a prediction in a simple manner without the need to perform complicated calculations such as calibrating and maintaining neural networks. The algorithm is based on the k-nearest neighbors method. This method is usually used as a general classification technique, and we use it in order to make prediction in the following way: for a given time series with n elements, $\vec{x}(t),\ t = 0,1,2,\ldots,n-1$ (each element is a vector with one or more values), and for the current point in time t, we consider the current state vector of length m $\vec{x}_t$ as a *query* or a *pattern*, and try to find along the time series k past states $\vec{x}_{t_i}, 1 \le i \le k,$ which are the closest or the most similar to the current state. The matching grade will be determined according to a chosen method for measuring distances between patterns (i.e. Euclidian, Voronoi, Hamming, Manhattan etc.)

After finding the k nearest neighbors of the query pattern, we retrieve the continuation of each one of them. Now we have k new sub-sequences, and we use these to calculate the sequence that represents the future of the current state. At the end of this process, we can evaluate the success of prediction by calculating the distance between the prediction produced and the true values measured (assuming they exist).

There are two common methods for implementing pattern matching on multi-attribute series. The first method is serializing the series. Using this method, we break the time series into multi-dimensional pieces (past states). The values of each dimension of a state are written next to the values of a previous dimension, forming a one-dimensional sequence, but l times longer, where l is the number of dimensions. The second method is a repetition of the pattern matching process separately for each dimension of the query pattern.

A wide range of algorithms has been developed for the general pattern-matching problem over a given alphabet. There are particularly elegant linear-time $O(n+m)$ algorithms (by Knuth-Morris-Pratt and Boyer-Moore [7]) and practical searching utilities for more general patterns instead of query strings Q (e.g., regular patterns in the Unix utility grep). However, time series data in continuous domains is inherently inexact, due to the unavoidable imprecision of measuring devices and clocking strategies. This forces us to work with the approximate version of the various matching problems [8]: Given a tolerance $\varepsilon \geq 0$ and a distance metric D between sequences, sequences S_1 and S_2 match approximately within tolerance ε when $D(S_1, S_2) \leq \varepsilon$.

3.1 Defining a distance

There are many methods for measuring the distance between sequences or vectors, some of which are more complex as every element in each vector is a vector itself. We shall deal with this problem by dividing it into two sections: first, defining a distance function between two discrete points of a time series, second, defining a distance measure between two subsequences.

Although there have been many distance functions proposed, by far the most commonly used is the Euclidian Distance, defined as:

(1)
$$L_2(\vec{x}, \vec{y}) = d(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{l} (x_i - y_i)^2}$$

where $\vec{x}$ and $\vec{y}$ are vectors of length l and x_i is the i-th component of vector $\vec{x}$. Euclidean distance is a special case of the L_p norms: $L_p(\vec{x},\vec{y}) = \left(\sum_{i=1}^{l} |x_i - y_i|^p \right)^{1/p}$ where p=2. The Euclidian distance function has two major weaknesses:

1. If one of the input attributes has a relatively wide range, then it can dominate the other attributes. For example, if an application has just two attributes, *A* and *B*, and *A* can have values from 1 to 1000, and *B* has values only from 1 to 10, then *B*'s influence on the distance function will usually be dominated by that of *A* [9].
2. If the amplitude of changes in one of the attributes of each vector is intrinsically much larger or smaller than that of the others, then the distance function could reflect a wrong interpretation of changes in each attribute. For instance, consider a meteorological application that contains data based on vectors of two attributes, one attribute being the wind speed and the other wind direction. Direction can be measured in degrees in the range 0 to 360, whereas speed values, measured in meters per second, exceed 40 only in hurricanes. The Euclidian distance function does not reflect this difference.

To overcome these problems while using Euclidian distance, it is necessary to use normalization [8]. We shall normalize each attribute in a vector based on its average and standard deviation. We shall therefore define:

$$\vec{x}_{norm} = (x_{1_{norm}}, x_{2_{norm}}, \ldots, x_{l_{norm}}) \text{ where } x_{i_{norm}} = \frac{x_i - \bar{x}_i}{\sigma_i}$$

with $\bar{x}_i$ being the average value of attribute i, $i = 1, \ldots, l,$ over all elements of a time series, and σ_i its standard deviation. Based on equation (1) we get the following definition for the distance between two points of a time-series:

$$d(\vec{x}_{norm}, \vec{y}_{norm}) = \sqrt{\sum_{i=1}^{l} (x_{i_{norm}} - y_{i_{norm}})^2} = \sqrt{\sum_{i=1}^{l} \left(\frac{x_i - \bar{x}_i}{\sigma_i} - \frac{y_i - \bar{y}_i}{\sigma_i} \right)^2} = \sqrt{\sum_{i=1}^{l} \frac{(x_i - y_i)^2}{\sigma_i^2}}$$

Given two time series S_1 and S_2 of m elements each, denote by $S_i[j]$ the j-th element of the i-th series. The distance between the sequences will be defined using L_p norms. We examine two cases:

1. Using L_p with p=2, i.e., Euclidean distance:

(2) $$L_2(S_1, S_2) = \sqrt{\sum_{j=1}^{m} d(S_1[j], S_2[j])^2}$$

This method suffers from the problem that the distance between two sequences which have a large number of points and in which the sequences are relatively close to each other could be greater than the distance between two short and very dissimilar sequences [10]. This may be overcome by dividing by the number of points:

$$D_2(S_1, S_2) = \frac{1}{m}\sqrt{\sum_{j=1}^{m} d(S_1[j], S_2[j])^2}$$

Because we are dealing with sequences of identical length, we will use this technique only when calculating the error measure at the end of the prediction process.

2. Using L_p with $p = \infty$. This gives:

(3) $$D_\infty(S_1, S_2) = \max_{1 \le j \le m} (d(S_1[j], S_2[j]))$$

When looking for a sequence that matches a query approximately within tolerance ε, the meaning of this distance function is searching for a sequence inside an m-dimensional sphere of radius ε around the query. We have tested both distance metrics in order to find their impact on the prediction effectiveness.

The natural way of searching for the location of a query pattern in a large database is the naïve sequential scan, also known as the sliding window method: a window of length m elements (as the length of the query) is sliding along the time series, and the distance between the query and the subsequence inside the window is measured. The indices of the k nearest neighbors of the query are dynamically kept in a heap. The time complexity of the comparisons is: $O((n-m+1)*m)=O(mn)$, updating the heap takes $O(n \log k)$.

Since the values at hand are not discrete, standard pattern matching is not an option here. Some attempts to deal with time series were based on the promising techniques of multidimensional indexing (R-Tree, R+-Tree, R*-Tree, X-Tree, SR-Tree, etc.) [11][12]. While using these methods, similar objects could be retrieved in sublinear time.

One problem which arises when trying to index sequences with spatial access methods is the so-called ***dimensionality curse*** [12][13]: spatial access methods typically work only when the number of dimensions is low, and as the number of dimensions grows, the effectiveness of these methods is not better than the sequential scan. This leads to the idea of reducing the number of dimensions of the indexed vectors. Dimensionality reduction is generally done by extracting a *signature* of low dimensionality from the original sequences, in a manner which to some extent preserves the distances between them, and then perform the indexing and the searching in the signature space [13]. Faloutsos et al. [11][14] describe the general solution of extracting and indexing signatures. They show that in order to guarantee completeness (no false alarms), the distance function used in the signature space must underestimate the true distance measure.

Some methods for extracting signatures have been developed including spectral signatures, minimal bounding rectangles and piecewise constant approximation [13]. There are some applications in which the dimensionality reduction may in itself speed up the sequential scan [10][13]. In our algorithm we chose to try and improve performance by pruning irrelevant solutions during scanning using an *aggregate distance measure*. The principle of this method is avoiding unnecessary calculations, when the partial evaluation shows that the current comparison will not be useful for the global solution, and should therefore be stopped. Implementing this in our algorithm is done as follows: using the distance definitions (equations 2,3), it is possible to use iterative calculations of the distance between the query pattern Q and any subsequence S. In each iteration, the distance between consecutive points of each sequence is added to the partial distance that was accumulated so far. In each step we have to compare the aggregated distance to a known distance measure (in our case – the distance R of the k^{th} closest neighbor to the query pattern Q). If the aggregated distance is greater than R, we should stop calculating the distance between the query and the current subsequence, and shift the window to the next subsequence.

3.2 Calculating prediction errors

We chose to use an error measure, which is quite common in prediction researches [1][15], and shall refer to it as the *error ratio*. To define it, we need two other measures: an absolute prediction error and its "standard deviation". Let *t* be the current point in time and *p* the size of the requested prediction. The true values of the time series in the immediate future are $S = \vec{x}(t+1), \vec{x}(t+2), ..., \vec{x}(t+p)$ and the corresponding series of predictions is $\hat{S} = \hat{\vec{x}}(t+1), ..., \hat{\vec{x}}(t+p)$; the absolute prediction error is then defined as the distance between these sequences: $E = D(S, \hat{S})$. The "standard deviation" is defined as $SD = D(S, \overline{S})$, the distance between the test-set series S and the series of averages $\overline{S} = \overline{\vec{x}}, \overline{\vec{x}}, ..., \overline{\vec{x}}$, consisting of *p* identical elements, each being a vector of *l* components $\overline{\vec{x}} = (\overline{x}_1, \overline{x}_2, ..., \overline{x}_l)$ where the *i*-th component is the average of the *i*-th components of all the elements in the series up to point *t*, $\overline{x}_i = \frac{1}{t}\sum_{j=1}^{t} \overline{x}_i(j)$ (recall that each element $\vec{x}(j)$ of the time series is itself a vector, denoted by $(x_1(j), \ldots, x_l(j))$). This "standard deviation" is in fact the absolute error in the case where the prediction was the average of the time-series itself. The *error ratio* is then defined as:

(4) $$Error = \frac{E}{SD} = \frac{D(S, \hat{S})}{D(S, \overline{S})}.$$

This measure shows how much using the prediction results is more effective than using the plain average of the series. If this value is zero, the prediction is perfect. However, if it is larger than 1, then the predictions are not better than the average of the time series.

3.3 Forming the prediction pattern

One of the most common methods of calculating the prediction is a projection of the continuation of the nearest sequences to the continuation of the current state through averaging [15][16]:

(5) $$\hat{\vec{x}}(t+d) = \frac{1}{k}\sum_{i=1}^{k} \vec{x}(t_i + d) \text{ for } d=1,\ldots, p,$$

where t_i are the ending indices of the *k* nearest matches to the current state. This, however, does not take the various distances of the *k* neighbors into account. In fact the average should be weighted, with weights that are inversely proportional to the distance. Actually we took them inversely proportional to the squares of the distances to put a larger penalty on more distant patterns. For example, suppose the k=3 closest patterns are at distances a, b and c, the corresponding weights should then be $\frac{1/a^2}{1/a^2+1/b^2+1/c^2}$, $\frac{1/b^2}{1/a^2+1/b^2+1/c^2}$ and $\frac{1/c^2}{1/a^2+1/b^2+1/c^2}$. Formally:

$$(7) \qquad \hat{\vec{x}}(t+d) = \frac{\sum_{i=1}^{k} w_i \vec{x}(t_i + d)}{\sum_{i=1}^{k} w_i} \qquad \text{where} \qquad w_i = \left(\frac{1}{D(\vec{x}_t, \vec{x}_{t_i})} \right)^2$$

Care is needed in case one or more of the neighbors match exactly (distance zero), in which case the corresponding w_i should be set to 1 and the w_i of the other neighbors should be zero.

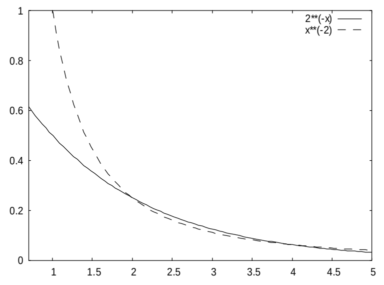

Figure 1: Comparison of decay functions

Another option could be to model weights decaying exponentially with increasing distances. A constant $K > 1$ is chosen, and w_i is set to $w_i = K^{-D(\vec{x}_t, \vec{x}_{t_i})}$. The size of K controls the exponential bias: K = 1 corresponds to uniform weights, and with growing K the elements at closer distances are increasingly dominant. We chose K = 2. Figure 1 shows a plot of w_i as function of the distance d for both $w_i = d^{-2}$ and $w_i = 2^{-d}$. The main difference is close to 0, meaning that if there is a subsequence matching almost perfectly, $w_i = d^{-2}$ will give it a weight tending to infinity which has the effect of practically ignoring the other neighbors, while $w_i = 2^{-d}$ is more moderate, tending to 1. For higher d values, the two functions are very similar.

Each point of the current state is a data vector of different meteorological variables. Since there may be some degree of dependency among the variables, the searches and the prediction process are performed on multivariate series. As comparison, we also split the multivariate pattern into several univariate ones, on which independent searches are conducted in parallel; the resulting predictions are then merged into a unified one.

4. Comparison with Analytical Prediction

We compare the results obtained by the algorithm based on pattern matching to the results produced by analytical approximation of a time series based on the Fourier transform, implemented by the well known Fast Fourier Transform (FFT) algorithm [7]. The FFT approximation algorithm uses the last 2^s sampling points of the time series (until the point from which the prediction starts) in order to calculate a function, which is an approximation to the function that underlies the observed time series. The values of this function for the points beyond the known 2^s points are the predicted values of the time series. We used s=10, that is, 1024 points.

One problem with FFT estimations is that it may succeed in revealing a global trend, but locally some delays are possible. Since we focus on local prediction, one may improve the results by performing a local shift of the values that are generated by the approximation function. This shift will be based on three variables: b, a parameter that denotes the number of the last points of the given series S; p, indicating the prediction length; and r, for possible shift values. Denote the estimated series obtained by the approximated function by EST. A window of size b is placed at the end of the estimated time series (inside the window one can find the last b observed points), and is shifted to the left r times. For each shift, the distance between the sequence inside the window and the last b points of the given series is evaluated, and the shift giving the minimum distance is selected:

$$Dist = \min_{0 \le i \le r} D(EST[1024-b+1-i]\cdots EST[1024-i],\ \ S[1024-b+1]\cdots S[1024])$$

The value of r corresponds to the number of points within one period, if some periodicity, e.g., a full 24 hour day, is known. Denote by i_0 the shift size corresponding to the minimal distance. We will use this size for shifting the estimated series in order to get a better prediction in the following way:

$$pred = EST[1024-i_0+1]\cdots EST[1024-i_0+p]\,.$$

Creating an approximation to a function using FFT is a technique suitable for univariate time-series. Therefore, we will make some comparisons:

1. We will calculate a prediction for each meteorological variable using the suggested algorithm and using FFT and we will compare between the results.
2. We will build a merged prediction using FFT and we will compare this result to the best result that would be achieved by the suggested algorithm using a multivariate prediction or a unified prediction.

5. Experiments and Results

The experiments were performed using a meteorological database, giving at each entry point the following data: wind direction (degrees), wind speed (m/sec), temperature (Celsius) and σ_θ, the standard deviation of wind direction measurements (the wind direction field is the average of all the measures that were done in the last 10 minutes, and this field contains the standard deviation of these measurements). Using the above notation we thus have $l = 4$. The four parameters were chosen because they are quite standard in many meteorological applications, in particular, σ_θ is a known parameter for classifying atmospheric stability [17]. The data points were recorded in regular intervals every 10 minutes.

First, the multivariate prediction for the 4 variables was evaluated, then the univariate predictions for each variable independently, which were combined into a unified prediction. We experimented with both weighted and regular averages. Finally, FFT was used for an analytical prediction, and all the results were compared based on their error ratios.

Another set of experiments examined the effectiveness of the techniques to improve searches and accelerate running time. We tried queries of different lengths, from 1 to 144, and prediction length from 1 to 60. We compared the number of performed comparisons between using the pruning by the aggregated distance method and relative to when no pruning is used. In all the experiments we used $k = 10$ nearest neighbors. The reason for choosing $k = 10$ was a practical one, to simplify the programming. It might have been appropriate to try to optimize k by seeking to minimize the error between the predicted and observed values on sample runs, but this could not be exploited for our true predictions, where no actual values are available. In fact, the number k of elements chosen should reflect the variability in the numbers of "closest" patterns: if only 2 or 3 are very close and the others are significantly farther away (call this case 1), prediction should be based just on the very close ones. But if many patterns, say 15, are almost at the same distance from the one searched for (call this case 2), probably all of these should participate in the prediction. Choosing a constant, rather arbitrary, value of *k*, has the advantage of simplifying the program and yet taking this possible difference in variability into account. Indeed, the decaying weighting functions (see Figure 1) imply that in case 1, the more distant values, while participating in the calculations, will get negligible weight and thus have almost no impact. Case 2 will generally occur when no good match for our query window has been found and all the candidates are equally bad. In that case we anyway do not expect a good prediction. Choosing then constant *k* with the given weighting means basing the prediction on a sub-sample and giving to all the elements similar weights.

Analyzing the results of the different experiments performed, it is evident that even though our database was quite small (only 5 months of data), the suggested algorithm is capable of producing very high quality predictions (in the majority of experiments the error ratio was less than 0.5). For larger data collections the results are expected to be improved.

Table 1. Comparison of average error ratios

	D_2+Weighted Avg.	D_2+Regular Avg.	D_∞+Weighted Avg.	D_∞+Regular Avg.
Error Ratio	0.487	0.514	0.745	0.753

Table 1 shows the average of the minimum error ratio (equation 4), averaged over all the experiments, and one can see that the Euclidian distance D_2 provides better results than the use of D_∞, and using weighted averages (equation 6) is better than using regular averages (equation 5).

The average error ratio turned out to be a decreasing function of the prediction length. The following features were observed:

1. Using short query patterns (based on 1 to 22 sample points) usually produces poor quality predictions.
2. Enlarging the query pattern length does not necessarily give a better error ratio.
3. No obvious rule relates the pattern length to obtain the minimal error ratio for a prediction of a certain length. Usually, the query pattern that gave the global minimum could be enlarged without damaging the error ratio.

We therefore suggest the heuristic of using a pattern which is 3 times larger than the requested prediction length, this yielded predictions which are on the average only 12% worse than the optimal ones.

All the experiments were run on a Pentium III 800Mhz processor. On this platform, calculating a multivariate prediction, based on 40 to 80 sample points, and a database of 6000 points (about 2 months of data), takes about 2 seconds. This rose to 4 seconds for 16,000 sample points (5 months). Because the running time of this algorithm is linear in the size of the database, expanding it to contain dozens or even hundreds of months will enable us to receive prediction results in a few minutes at most.

We saw that unified univariate prediction is a plausible alternative, as the predictions yielded error ratios smaller than 1. However, the average error ratio of predictions based on this method is higher by 18% than that for multivariate predictions. This shows that there is a certain dependency among the specific meteorological variables. When separating the variables, we damaged the added value that has come from this dependency, and therefore the results were poorer.

In order to produce the unified prediction we calculated a prediction for each meteorological variable using our suggested algorithm. What emerges from the results is that the temperature was easy to predict (mean error ratio of 0.15) while the σ_θ parameter was the most difficult one (mean error ratio of 0.81). The algorithm produced good predictions for wind speed and wind direction variables (mean error ratio of 0.41, 0.49 respectively).

Comparing the results of the analytical prediction to the multivariate prediction revealed the following facts:

1. The longer the length of prediction, the better its quality. The reason for this is that approximation based on FFT captures the global trends of a time series, but locally there are some delays. Therefore, a prediction based on FFT is better for long-term prediction since global trends are exposed only through long periods of time.
2. The quality of predictions based on the suggested algorithm is 80%-180% higher than predictions made by FFT approximation.

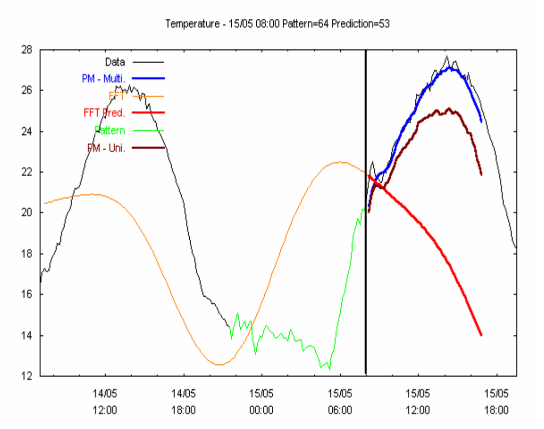

Figure 2. Temperature prediction – Pattern length=64, prediction length=53.

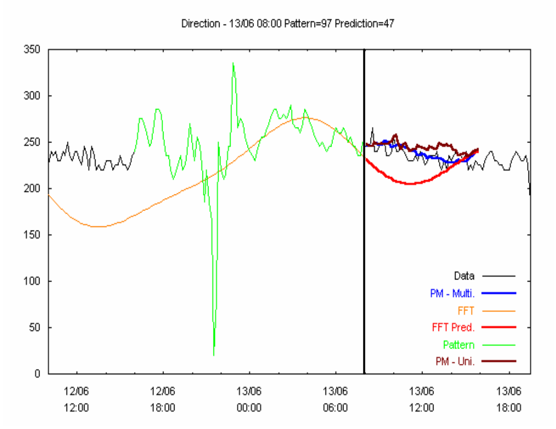

Figure 3. Direction prediction – Pattern length=97, prediction length=43.

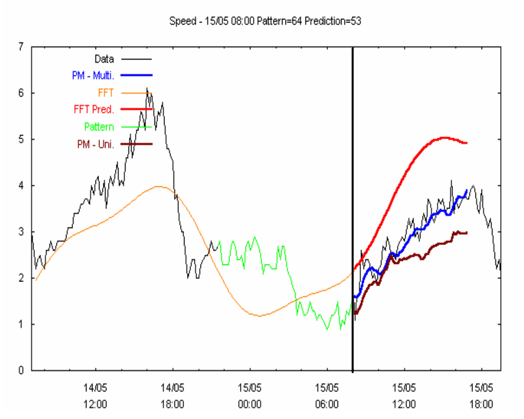

Figure 4. Speed prediction – Pattern length=64, prediction length=53.

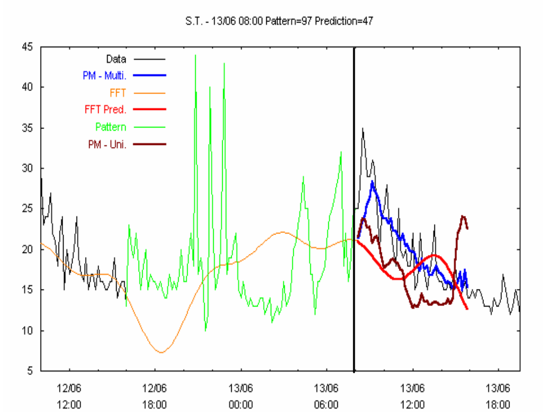

Figure 5. σ_θ prediction – Pattern length=97, prediction length=47.

Examples for the quality of predictions are given in Figures 2 to 5, corresponding to temperature, wind direction, wind speed and σ_θ, respectively. We performed a very large number of tests, varying the different parameters over large ranges. The four figures above were chosen as representative sample, illustrating the relative performance of the various algorithms for each of the meteorological parameters. In each figure there is a graphical illustration of the historical data of the given variable (to the left of the vertical line) and the prediction results (to its right). The query pattern that was chosen for the prediction is at the end of the history and is colored in green. The blue line represents the projection of the specific variable from a multivariate prediction, and the brown line the univariate prediction made for this variable. The orange line gives the FFT approximation (after shifting) and the red line is the FFT prediction. All the predictions follow immediately the query pattern. In these figures one can see that the projection of a specific variable from a multivariate prediction is congruent and sometimes even overlaps with the real measured data (the grey line). One might also observe that the univariate prediction is close to the multivariate one, but it is less accurate. Moreover, it is evident that FFT approximations are not as accurate as predictions produced by the suggested algorithm and Figure 2 also demonstrates that the FFT approximation revealed the global trend but failed in doing so locally, even after the adjustment, because of the delay problem mentioned above.

As to processing speed, we found that using pruning based on aggregated distance can accelerate the search for similar patterns up to 5.6 times with the D_∞ distance and up to 3.9 times with Euclidean distance. The effectiveness of pruning increased with the size of the database and also with the length of the query. However, for D_2 the effectiveness starts to decrease when the query becomes longer than 25 points, while for D_∞, there is no such decrease. The reason for this behavior is that the Euclidean distance between two subsequences is based on the distance between all the points, and when we use D_∞, only one pair of points determines the distance.

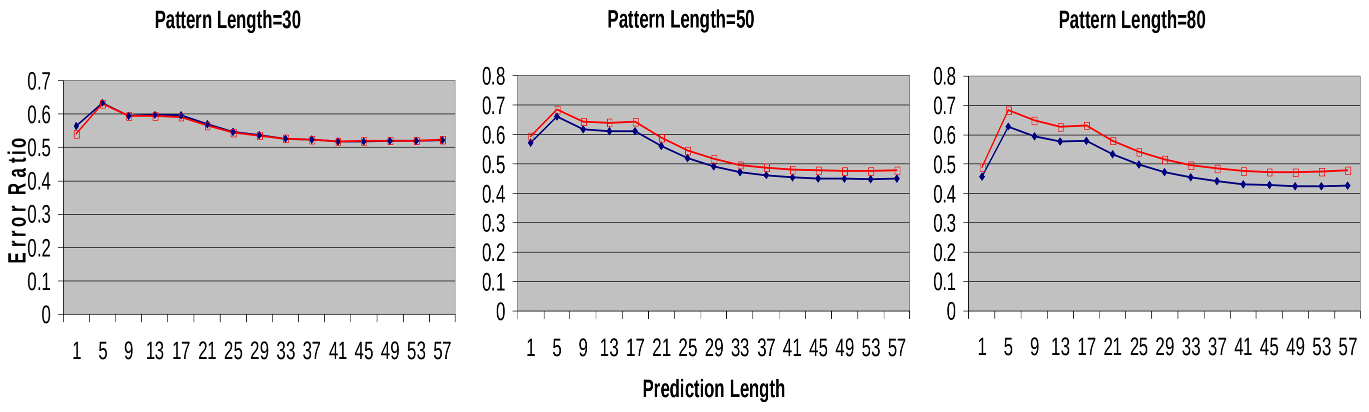

Figure 6: Comparing exponential and quadratic weights

Figure 6 compares the use, in equation 7, of exponential weights $w_i = 2^{-D(\vec{x}_t, \vec{x}_{t_i})}$ (blue lines with diamonds) with that of quadratic weights $w_i = D(\vec{x}_t, \vec{x}_{t_i})^{-2}$ (red lines with rectangles). We found that for predictions based on short patterns there was no difference at all (Figure 6 – pattern length = 30). However, for prediction based on longer patterns, there was an observable improvement for the exponential weights (pattern length = 50, 80). The reason for this is that we used aggregate distance between patterns, and therefore long patterns led to a larger distance, which affected the results (the influence of distant neighbors on the prediction result became negligible).

6. Concluding Remarks

We examined the possibility to predict local meteorological data with an algorithm based on pattern matching. The algorithm refers to the sequence of values immediately preceding the point from which the prediction is needed as a query pattern. Then, a search for similar patterns in the given history is performed in order to extrapolate a prediction based on the continuations of similar patterns.

Contrarily to standard IR systems, for which relevance judgment is often based on user decisions, a more objective evaluation is possible in our case. All we need is choosing the “current point” somewhere in the history, so that true data is available against which the “future” predictions may be compared. The results were encouraging and showed that it is possible to produce high quality predictions using the suggested method. Moreover, a comparison to predictions produced by one analytical method (FFT approximation) indicated that the predictions made by our algorithm are far better. It remains to see if our algorithm does also yield improved results relative to other analytical approaches, such as general regression neural networks [6].

Acknowledgement:
The authors wish to thank an anonymous referee for his helpful comments.

References

[1] T. Koskela, M. Varsta, J. Heikkonen and K. Kaski, Time series prediction using recurrent SOM with local linear models. *Int. J. of Knowledge-Based Intelligent Engineering Systems, 2*(1), 1998, 60-68.

[2] N.H. Packard, J.P. Crutchfield, J.D. Farmer and R.S. Shaw, Geometry from a time series, *Physical Review Letters, 45*(9), 1980, 712-716.

[3] F. Takens, Detecting strange attractors in turbulence, in D.A. Rand and L.S. Young (Eds.) *Dynamical Systems and Turbulence*, Warwick 1980, *Lecture Notes in Mathematics 898*, Springer Verlag, 1981, 366-381.

[4] J.D. Farmer and J.J. Sidorowich, Predicting chaotic time-series. *Physical Review Letters, 59*(8), 1987, 845.

[5] S. Singh, Multiple Forecasting using local approximation. *Pattern Recognition*, *34*(2), 2001, 443-455.

[6] N.M. Islam, S.Y. Liong, K.K. Phoon and C.Y. Liaw, Forecasting of river flow data with general regression neural network, *Proc. International Symposium on Integrated Water Resources Management, 9-12 April 2000, University of California, Davis*, IAHS red book pub. No. *272*, 2001, 285-290.

[7] T.H. Cormen, C.E. Leiserson and R.L. Rivest, *Introduction to Algorithms*, MIT Press, 1990, 853-885.

[8] D.Q. Goldin and P.C. Kanellakis, On similarity queries for time-series data: constraint specification and implementation. *Proc. of the First International Conference on Principles and Practice of Constraint Programming*, Cassis, France, 1995, 137–153.

[9] D.R. Wilson and T. Martinez, Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, *6*, 1997, 1-34.

[10] S.L. Lee, S.J. Chun, D.H. Kim, J.H. Lee and C.W. Chung, Similarity search for multidimensional data sequences. *Proc. 16th IEEE Conf. on Data Engineering*, 599–608, 2000.

[11] C. Faloutsos, M. Ranganathan and Y. Manolopoulos, Fast subsequence matching in time-series databases. *Proc. of the 1994 ACM SIGMOD International Conference on Management of Data*, 1994, 419–429.

[12] R. Weber, H.J. Schek and S. Blott, A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces, *Proc. of 24th International Conference on Very Large Data Bases*, 1998, 194-205.

[13] M.L. Hetland, A survey of recent methods for efficient retrieval of similar time sequences, in M. Last, A. Kandel and H. Bunke (Eds.), *Data Mining in Time Series Databases*, World Scientific, Singapore, 2004.

[14] C. Faloutsos, H.V. Jagadish, A.O. Mendelzon and T.A. Milo, Signature technique for similarity-based queries, *Proc. Compression and Complexity of Sequences '97 (SEQUENCES '97)*, Italy, June 1997.

[15] K.K. Phoon, M.N. Islam, C.Y. Liaw and S.Y. Liong, A practical inverse approach for forecasting nonlinear hydrological time series, *Journal of Hydrologic Engineering, ASCE*, *7*(2), March 2002, 116-128.

[16] V. Babovic and M. Keijzer, Forecasting of river discharges in the presence of chaos and noise, in J. Marsalek (Ed.), *Coping with Floods: Lessons Learned from Recent Experiences*, Kluwer, 1999.

[17] A.E. Mitchell, A comparison of short-term dispersion estimates resulting from various atmospheric stability classification methods, *Atmospheric Environment*, *16*(4), 1982, 765-773.