Predicting the Weather Without Physics

by Mark D. Fruman

Capstone Course for Certificate in Data Analytics, Big Data, and Predictive Analytics

Ryerson University, Toronto, ON, Canada

Advisor: Dr. Bora Çağlayan

Abstract

Methods of predictive analytics, including multiple linear regression, principal component analysis and k-means clustering, are applied to the classic problem of weather forecasting. Daily aggregate statistics are computed from eight years (2005-2012) of hourly observations from a network of 29 airport weather stations and used to train and test a hierarchy of regression models.

Models using only data from the station for which the forecast is to be obtained only slightly outperform the baseline estimate that the value on the forecast day does not change from the prediction day. Models using data from all stations do considerably better, at the cost of much higher complexity. Models using simultaneous observations at all stations also tend to suffer from strong *multicollinearity*, and are therefore difficult to interpret and prone to overfitting. Models built using principal component analysis can achieve similar performance using as few as 10% as many features as the direct multi-station models.

k-Means clustering is used to divide the training and testing sets into clusters having similar properties, and separate regression models are trained for each cluster. While not providing a significant improvement in overall performance over the regular (single-cluster) models, they do explain why the weather on certain days is more predictable than on others.

A suite of python modules was developed to accumulate, archive, and process the data and to build and apply the regression models. **Complete source code is available at github.com/majorgowan/wpwp**

Introduction

Short-range forecasting amounts to a deterministic initial-value problem. Given accurate and complete enough data about the state of the atmosphere today and details of any forcing processes (such as solar heating and lunar tides), a system of partial differential equations known as the *Primitive Equations*, derived from fluid mechanics and thermodynamics, can, in principle, be solved to predict the state of the atmosphere tomorrow, the next day and so on. This is, of course, much more easily said than done for several reasons:

- The equations are nonlinear and chaotic, so small changes in initial conditions lead to completely different forecasts
- The initial data is sparse and of varying quality, so it must be filled in by interpolation and modeling
- The Primitive Equations support motion on multiple time-scales, from very fast sound waves (seconds) to "gravity waves" associated with buoyant oscillations of fluid parcels in the atmosphere (minutes to hours) to the slow "synoptic" *scale* motions we associate with weather (hours to days). The initial conditions must be carefully filtered so that they do not initiate fast motions that would completely overshadow and contaminate the forecast of the slow motions
- Several important physical processes take place on spatial scales much finer than the observation network can resolve, for example the formation and evolution of clouds. Their effects must be *parameterized*, i.e. represented in the equations by a parameter or simplified formula.

These complications notwithstanding, modern weather forecasting is done numerically, using supercomputer clusters running computer models that solve a discrete form of the Primitive Equations complemented by the necessary parameterizations. A worldwide network of ground-based observation stations, balloon-borne *radiosonde* probes and data from satellites orbiting the earth are used to initialize and constrain the computer models.

Even though we understand to a great extent the physical processes underlying the evolution of the weather, the models used to make forecasts need not necessarily be based on that physical understanding, especially since many aspects of the models represent the physics very crudely. The purpose of this project is to predict the weather using station data from previous days and some of the basic tools of predictive analytics, including principal component analysis, multiple linear regression, and cluster analysis, and to compare the resulting models to simple baseline models (such as the "persistence" assumption that the weather tomorrow will be the same as today or the "climatological" assumption that it will be the same as the average value for that day of the year) as well as to published professional forecasts.

The report is organized as follows: After a review of some of the relevant literature, a detailed description of the data is presented. Next, each of the regression models is described, and the results compared and discussed. A list of IPython notebooks and a documentation for the suite of python modules are in appendices.

Literature Review

See Lynch (2008) for a review of the first attempt at a numerical weather forecast, by L. F. Richardson in 1922 (Richardson wrote, “Perhaps some day in the dim future it will be possible to advance the computations faster than the weather advances . . .. But that is a dream.”).

Since traditional weather prediction, including numerical weather prediction, is such a well established industry with tested, tuned and optimized models and methods, there does not seem to have been much effort to start from square one with abstract data analytics methods. On the other hand, methods like regression, classification, clustering, and principal component analysis are used extensively in interpretation of models and data as well as in aspects of forecasting like data assimilation (the process of incorporating observations in a running numerical model), and downscaling (inferring likely values of variables on a finer mesh than is available from observation). For accessible introductions to the physics behind weather, see for example, Shonk (2013), Ahrens (2005) and Gordon et al. (1998).

There are a few studies that use basic machine learning schemes for predicting weather. Paras and Mathur (2012) describe a multiple linear regression model for predicting meteorological variables using past values over various time intervals at the same station. The method was tested on data from Pantnagar in India. Göçken et al. (2015) use multiple linear regression along with two methods based on neural networks (Artificial Neural Networks, ANN, and Adaptive Neuro-Fuzzy Inference System, ANFIS) to predict the daily mean temperature in Gaziantep, Turkey using other meteorological data. It is not clear whether the predictions are for future temperature (i.e. forecasting) or for temperature on the same day as the observations of the other variables. Tektas (2010) uses ANFIS and Auto Regressive Moving Average (ARIMA) to forecast meteorological variables in Istanbul, Turkey. Sawale and Gupta (2013) describe using ANN on a global network of weather stations to predict future weather (they do not give results; the article seems to be a statement of concept as opposed to a finished study).

The R package rattle contains the dataset weather containing data from a network of weather stations in Australia. Mitsa (2015) uses the dataset to build a logistic regression model to predict whether or not it will rain.

Although not for the purpose of forecasting, Clifton and Lundquist (2012) use k-means clustering on wind speed and direction data to classify wind conditions at a single location. The occurrence of the clusters identified is found to be correlated with climate oscillations such as El Nino.

For a detailed description of the METAR convention for recording meteorological data, see U.S. Dept. of Commerce and NOAA (2005).

Data Description

Retrieval

Data is retrieved by “scraping” web pages on the Weather Underground website www.wunderground.com using the python requests package. Historical quasi-hourly data from airport weather stations is available in comma-separated-value (CSV) format under the URL schema

http://www.wunderground.com/history/airport/CODE/YYYY/M/D/DailyHistory.html?reqdb.magic=1&reqdb.wmo=99999&format=1

where *CODE* is the International Air Transport Association (IATA) code for the desired station and YYYY/M/D is the date.

Functions in the wUnderground module take a station name and date (range), construct the necessary URL, download the data from the website and save it as a *list* of python *dictionaries* (with keys given by the headers in the CSV data). Each month of data from a single station is stored as a list of days, where a day consists of a list of dictionaries, and is stored in a text file in JSON format as an array of arrays of documents. In can be written to and read from in python using the json package.

Figure 1: IATA codes and locations of stations used (plotted using basemap package).

Overview

The data used for this project was taken from a network of 29 Canadian and US cities selected somewhat randomly (see Figure 1). Data were obtained for the eight years 2005-2012. A single day's worth of data consists of a list of observations (hourly plus occasional additional observations, roughly 30 per day). The variables included in the data are listed in Table 1.

Local Time	*Time in one of EDT, EST, CDT, CST*
Universal Time Code	*Standard time at 0 degrees longitude (cf. Greenwich Mean Time)*
Temperature (deg C)	*Instantaneous temperature*
Dew Point Temperature (deg C)	*Temperature at which the air would be saturated with water vapour (depends on temperature and humidity)*
Relative Humidity (%)	*Ratio of partial pressure of water vapour in the air to the partial pressure at which the air would be fully saturated*
Sea Level Pressure (hPa)	*Atmospheric pressure that would be observed at sea level given station altitude, local pressure and temperature*
Visibility (km)	*Measure of distance at which objects can be easily recognized*
Wind Direction (N, NNE, NE, …)	*Direction from which wind is blowing*
Wind Direction (degrees)	*Wind direction in degrees clockwise from north*
Wind Speed (km/h)	*2-minute-average wind speed at 10 m altitude*
Wind Gust Speed (km/h)	*Maximum instantaneous wind speed*
Precipitation (mm)	*Precipitation observed in past hour*
Events	*Rain, snow, …*
Conditions	*Clear, overcast, …*

Table 1: Variables included in raw hourly data and their meaning

Missing Values

In general, missing values are indicated in the METAR data by either a blank record (‘’), the letters ‘N/A’, or the value ‘-9999’ (or occasionally ‘9999’ and ‘-9999.0’). In addition, values far out of the normal range for the corresponding variable, which were assumed to be corrupted values or typos, were treated as missing. Values that appeared to be entered in the wrong units

(e.g. pressure in kilopascals instead of hectopascals) were also treated as missing so as not to require manual corrections of the raw data. Precipitation data was not available at Canadian stations so was not used for any stations.

Visualizations of hourly data

The following are examples of quasi-hourly data from November 2007 at station CYYZ (Toronto Pearson Airport). The data is read using the function wUnderground.getJSON(). See the IPython notebook wU_Working_with_Hourly_Data for the source code used to create the figures (including the cleaning of missing values).

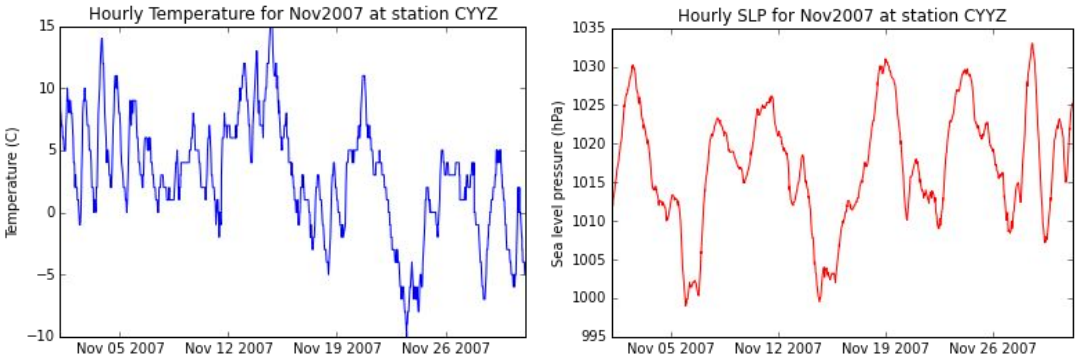

Figure 2: One month of hourly temperature (left) and sea-level pressure (right) at Toronto

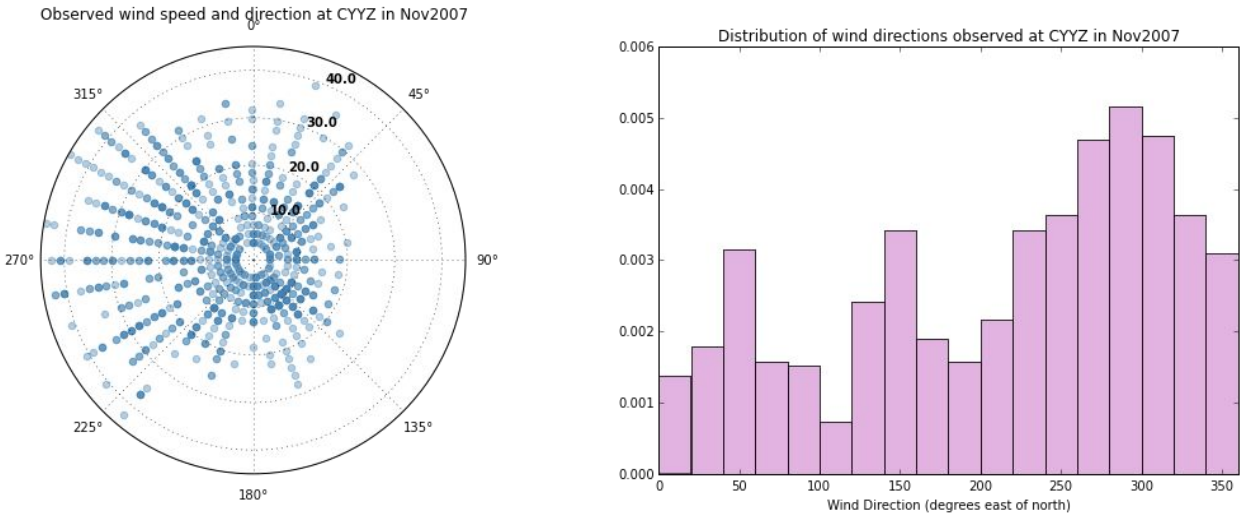

Figure 3: Observations of wind speed in km/h and direction (left) and distribution of observed wind directions (right) at Toronto over one month.

Daily aggregate statistics (stored in CSV)

Because it is not uniform in timing and frequency from station to station and because it contains missing values, the (quasi-)hourly data is not used directly for building predictive models in this project. Rather, the following daily aggregates of the hourly data are precomputed and stored to serve as the dependent and independent variables for the predictive models:

`TempMean, TempMax, TempMin, PressMean, PressMax, PressMin, HumidityMean, VisibilityMean`	*Daily means, maxima and minima of Temperature, Pressure, Humidity and Visibility*
`TempMaxTime, TempMinTime, PressMinTime, PressMaxTime`	*Times of daily maxima and minima of Temperature and Pressure*
`TotalPrecip`	*Daily totals*
`WindMeanX, WindMeanY`	*Daily mean wind components*
`WindMaxSpd, WindMaxDir, WindMaxTime`	*Speed, direction and time of daily maximum wind*

Table 2: Daily aggregate statistics stored in CSV files

Since the observations are not perfectly regular in time, mean quantities are computed using the *rectangle method* of numerical integration:

$$\bar{X} = \frac{1}{T}\int_0^T X(t)\,dt \;\approx \frac{1}{T}\sum_{i=0}^{N-1} X_i \Delta t_i$$

where N is the number of non-missing values on that day, Δt_i is the time interval between adjacent non-missing values X_i of the variable X, and $T = \sum_{i=0}^{N-1} \Delta t_i$ is the time interval spanned by the non-missing data (usually one day).

The components of the mean wind are computed from the wind speed and direction data via

$$U_{xi} = -|U_i| \sin\theta_i\,, \qquad U_{yi} = -|U_i| \cos\theta_i$$

where U_i and θ_i are the instantaneous wind speed and direction (angle) of observation i.

For the time-of-mininum (maximum) variables, if the minimum (maximum) value occurs several times in the same day, one of those times is selected at random and recorded.

The source code for processing the hourly data into daily summaries is in the module wUStats.

Derived statistics

Finally, the variables listed in Table 3 may be computed from the stored daily summary values. The functions for computing the derived binary statistics for a set of daily records are in the module wUDerived.

`dailyTempRange, dailyPressRange`	*Difference between daily maximum and daily minimum; negative if minimum occurs earlier in day than maximum*
`isFoggy`	*Boolean: true if daily mean visibility is less than 5 km*
`windQuadrant`	*Direction of daily mean wind (discrete: 0 = NE, 1 = SE, 2 = SW, 3 = NW)*
`isEasterly, isWesterly, isNortherly, isSoutherly`	*Boolean: true if daily mean wind is from named direction*
`isMorningMinTemp, isMorningMaxTemp, isMorningMaxWind`	*Boolean: true if corresponding minimum (maximum) occurs before 12:00 local time*
`isDaytimeMinPress, isDaytimeMaxPress`	*Boolean: true if minimum (maximum) pressure occurs between 8:00 and 20:00 local time*

Table 3: Derived daily statistics (not stored in CSV)

Visualizations of daily aggregate data

Figures 4-7 are examples of visualizations of daily aggregate data. The data is read using the functions wUUtils.loadDailyVariable() for simultaneous values of the variable at several stations, and wUUtils.loadDailyVariableRange() for a series of values at a single station. See the IPython notebooks wU_Working_with_Daily_Summaries and wU_Visualize_Data_on_Map for the source code used to create the figures.

Note that the contour plots over map backgrounds are made using interpolation of very sparse data so should be considered illustrative only.

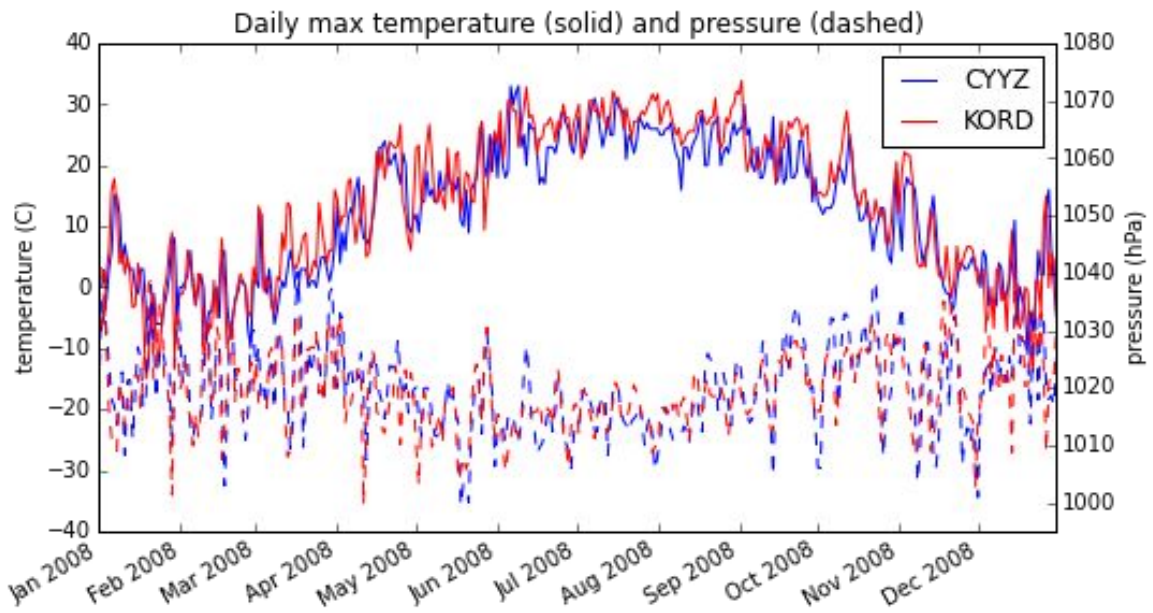

Figure 4: Daily maximum temperature `TempMax` (solid lines) and sea-level pressure `PressMax` (dashed lines) for the year 2008 at CYYZ (Toronto) and KORD (Chicago)

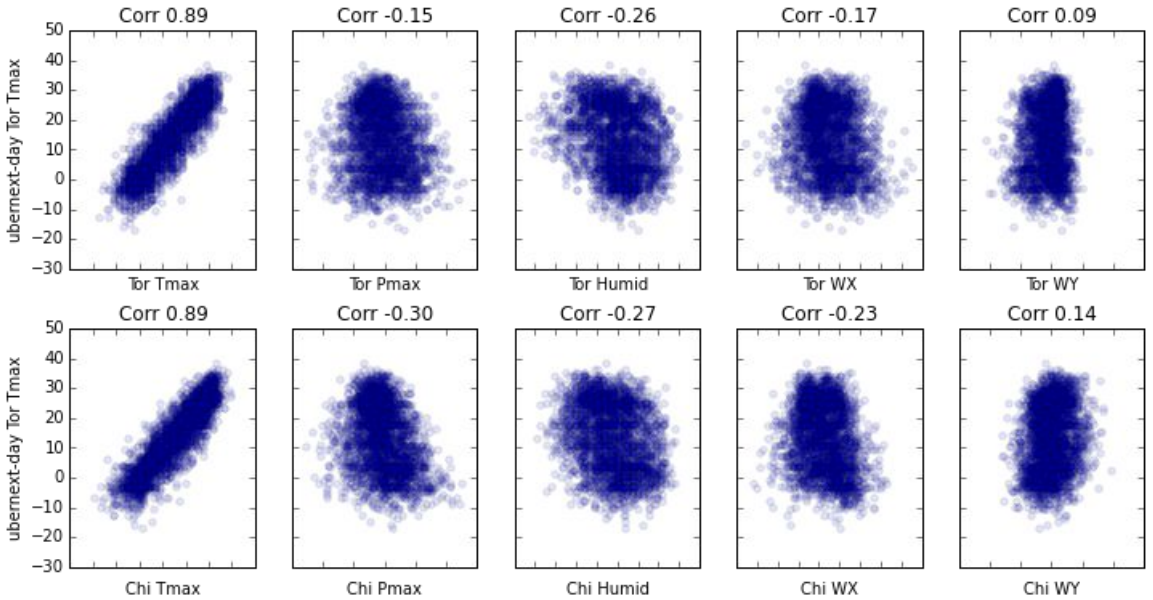

Figure 5: Scatter plots of the daily maximum temperature at CYYZ versus various daily aggregate variables (`TempMax, PressMax, HumidityMean, WindMeanX, WindMeanY`) at CYYZ and KORD two days earlier. Correlation coefficients are noted in the subfigure titles. Note that correlations are slightly stronger for the earlier observations at Chicago.

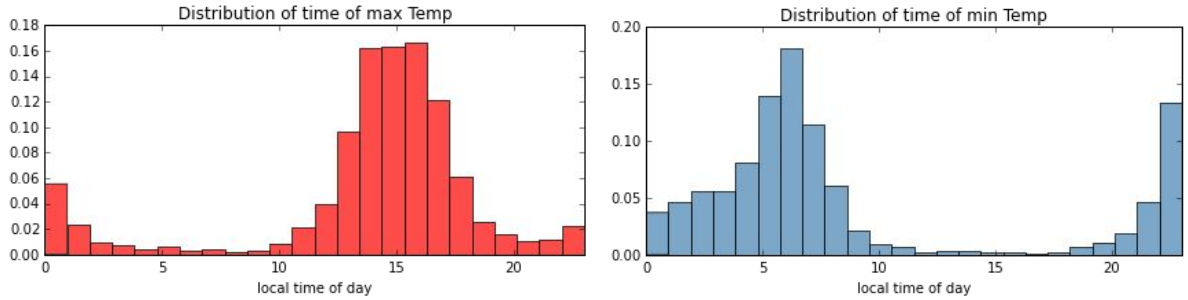

Figure 6: Distributions of `TempMaxTime` (left) and `TempMinTime` (right) at CYYZ in 2008.

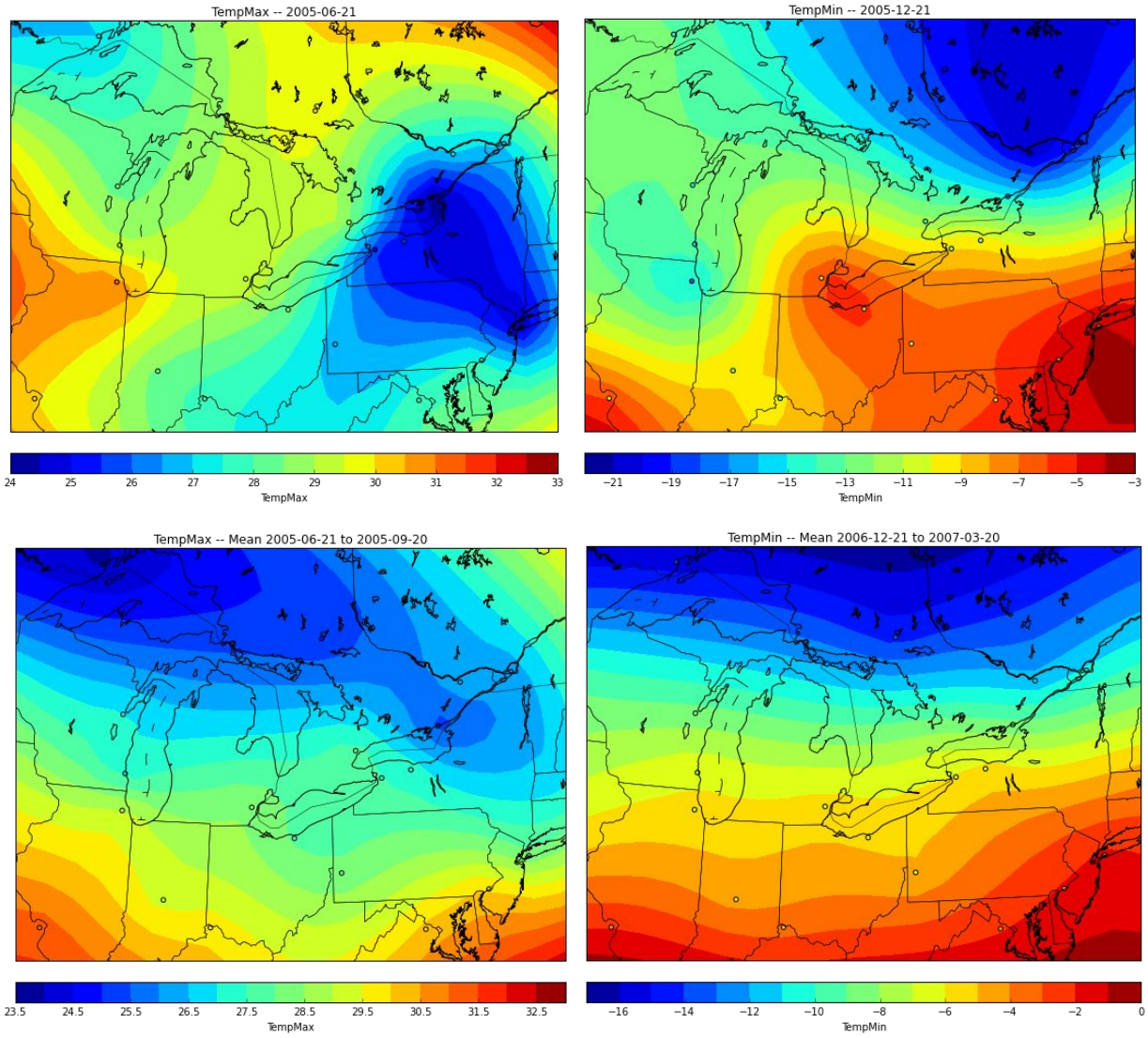

Figure 7: Top row: Daily maximum temperature on the first day of summer 2005 (left) and daily minimum temperature on the first day of winter 2005 (right); Bottom row: Mean daily maximum temperature for summer 2005 (left) and mean daily minimum for winter 2005-2006 (right).

Principal component analysis

Simultaneous observations from multiple stations are usually strongly correlated. Adding data from more and more stations to a regression model therefore results in increasing model complexity but diminishing returns in terms of model performance. Principal component analysis (PCA) is used to identify the modes -- linear combinations of the values from all stations -- that capture the most variability in the data set. The modes are mutually orthogonal and uncorrelated.

While it is entirely possible to apply PCA to the entire dataset, mixing in one set of modes variables of different types, say a combination of `TempMax`, `dailyPressRange`, `HumidityMean` and the binary variable `isWesterly` indicating westerly wind, here we construct separate sets of principal component (PC) modes for each variable. The set of values of the same variable at all stations on the same day is transformed into a sum of PC. Since the leading PC capture most of the variability in the training data set, the 29 values required to specify one day of observations can often be reduced to as few as three or four PC coefficients without losing much valuable information.

Figure 8 shows the time series of the leading three PC based on the daily-maximum temperature, daily-mean humidity and daily-maximum pressure observations from all stations for the years 2005-2010. The annual cycle is clearly visible in the leading PC of each variable, but it explains a much higher fraction of the variance of temperature than it does in the cases of the other two.

Figure 9 shows the spatial structures of the three leading PC of daily-maximum temperature. The leading PC (PC0), which explains almost 90% of the variance, has a very homogeneous structure (note the range of values on the colour-bar axis). This is because the whole region warms and cools together with the seasons. The second PC (PC1) has an east-west antisymmetry, presumably reflecting the fact that the amplitude of the seasonal cycle is stronger in the interior (hotter summers and colder winters) than on the coast. The third PC (PC2) has a north-south antisymmetry. When PC2 is positive, the north-south temperature gradient (the south is generally warmer than the north) is weaker than usual or perhaps even reversed, and when it is negative, the gradient is stronger than usual. The first three pressure PC have similar structures in terms of east-west and north-south symmetry (not shown). This can be useful in interpreting the predictions of the regression models.

See the IPython notebook wU_PCA_Analysis for the source code for these figures.

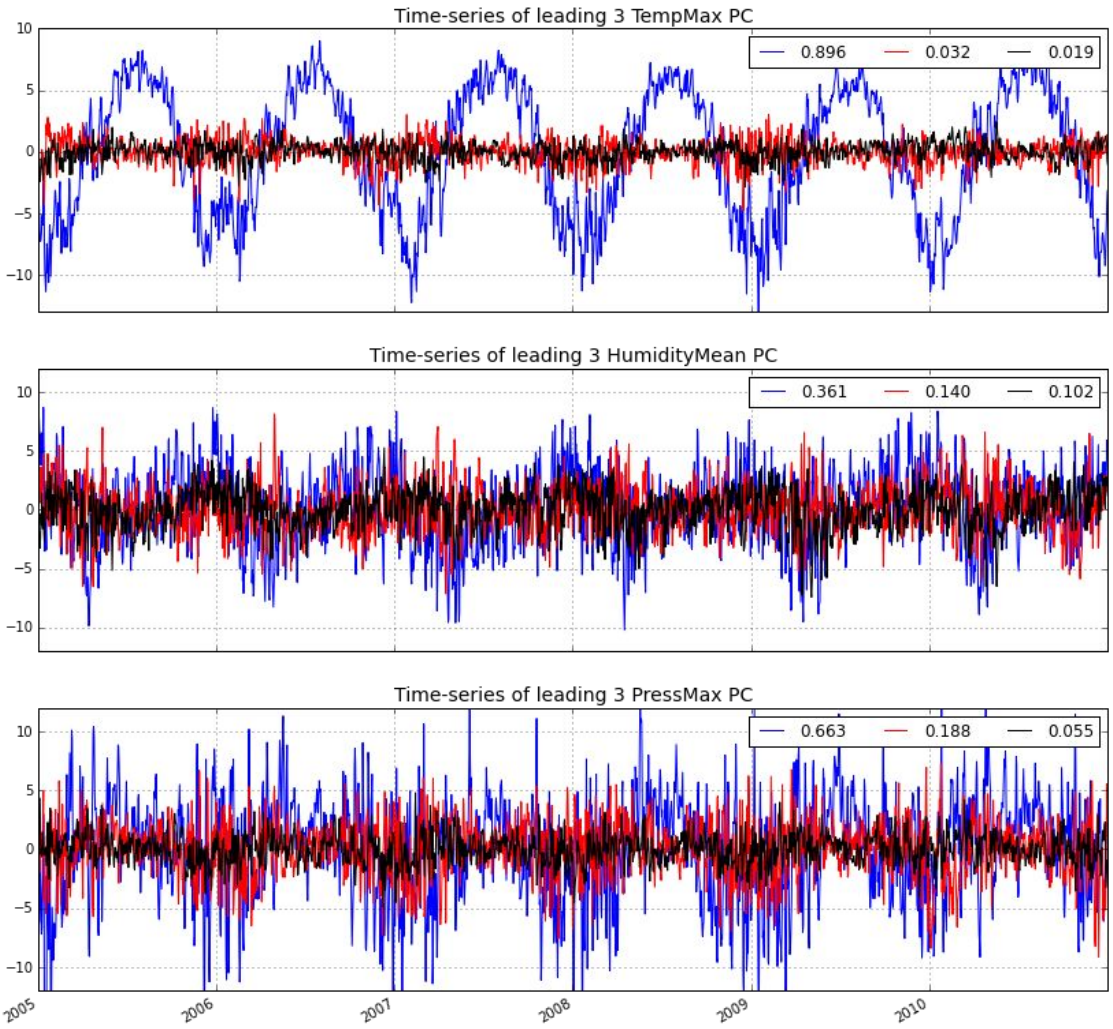

Figure 8: Time series of leading 3 PC of each of daily maximum temperature `TempMax`, daily mean humidity `HumidityMean` and daily maximum sea-level pressure `PressMax`;
The fractions of the total variance explained by each mode are listed in the legends.

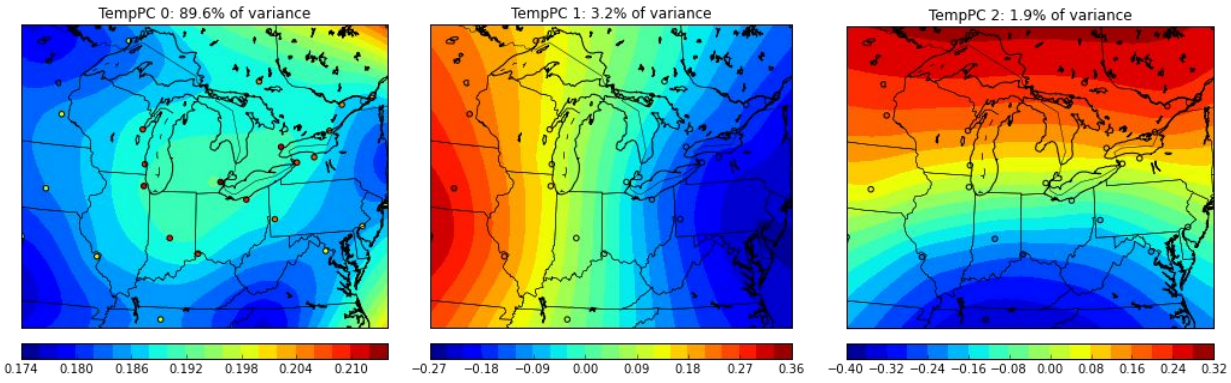

Figure 9: Structures of the three leading PC of daily-maximum temperature for 2005-2010.

Regression Models

Multiple-linear regression models of varying complexity were constructed. The simplest uses data from only the station for which predictions are to be made and from only the day on which the prediction is to be made (henceforth the **prediction day**). The most complex uses multiple days' worth of data from all available stations, uses principal-component analysis and clustering, and incorporates interactions between features. The python modules wURegression and wUClusterRegression contain the main functions used to construct the models and use them for making predictions. The models are listed below.

One-station Models

wURegression.oneCityModel()
wURegression.oneCityPredict()
Linear regression is used to forecast a single quantitative variable (such as daily maximum temperature) on a future day (the **forecast day**) using data for several variables from the prediction day. No steps are taken to account for multicollinearity of the independent variables.

wURegression.oneCityTaylorModel()
wURegression.oneCityTaylorPredict()
Data from the prediction day as well as the preceding day(s) are used. The independent regression variables are the values of the predictive features on the prediction day, their changes since the previous day (approximately the first partial time derivative), the changes in the changes since the previous day (the second derivative), etc. This method is preferred to using the values from preceding days because the latter would be very strongly correlated with the values on the prediction day. These models are referred to as *Taylor models* since the dependent variable Y on the forecast day may be estimated by the Taylor series expansion of Y about the prediction day:

$$Y(t+\Delta t) \approx Y(t) + \frac{\partial Y}{\partial t}\Delta t + \frac{1}{2}\frac{\partial^2 Y}{\partial t^2} + \ldots$$

Multi-station Models

wURegression.multiCityTaylorModel()
wURegression.multiCityTaylorPredict()
Data from a specified list of stations on the prediction day and arbitrarily many preceding days is used to predict a single variable at a single target station on a future day. This model inevitably suffers from strong multicollinearity between independent variables since observations of the same variable at nearby stations are strongly correlated. Inferring the importance of the different features and stations is therefore virtually impossible.

Advection Models

wURegression.advectionTaylorModel()
wURegression.advectionTaylorPredict()
Similar to the previous model, but instead of the values of the variables at the source stations (the stations other than the target station), the model uses only the difference between the values at the source stations and the value at the target station multiplied by the component of the daily mean wind in the direction from the source station to the target station. This is a representation of the approximation to the advection of that variable towards the target station, $\vec{U} \cdot \nabla X$ (the advection is usually a term in the fluid dynamics evolution equation for a typical property).

Model Incorporating Interaction (nonlinear) Terms

wURegression.multiCityInteractionModel()
wURegression.multiCityInteractionPredict()
Identical to the multi-station model but allows for interactions between independent variables, e.g. daily maximum temperature `TempMax` multiplied by the binary `isFoggy` variable, encoded as `TempMax:isFoggy`. Both variables involved in the interaction term are standardized (i.e. centred and normalized so that they have mean zero and standard deviation unity) before multiplication.

Principal Component Analysis Model

wURegression.pcaTaylorModel()
wURegression.pcaTaylorPredict()
In these models the values of the independent variables at the full set of stations are replaced with coefficients of a truncated set of PC. A separate set of PC are generated for each independent variable. The number of PC to use can be different for different variables.

Clustering Model

wUClusterRegression.clusterRegression()
wUClusterRegression.clusterRegressionPredict()

wUClusterRegression.pcaClusterModel()
wUClusterRegression.pcaClusterPredict()
k-Means clustering is applied to the data using a subset of the independent variables before a separate regression model is trained for each cluster. Two versions of this model are available, one using PC and one not..

Results and Discussion

We use the following performance measures to evaluate the various regression models. The **root-mean-square error** is defined as

$$RMSE \equiv \sqrt{\frac{SSE}{N}}$$

where $SSE \equiv \sum_{i}^{N}(Y_i - \hat{Y}_i)^2$ is the sum of squared errors, with Y_i the prediction and $\hat{Y}_i$ the true value of the dependent variable. The **coefficient of determination** of the regression fit is

$$R^2_{mean} \equiv (1 - \frac{SSE}{SSM})$$

where $SSM \equiv \sum_{i}^{N}(\bar{Y} - \hat{Y}_i)^2$ is the total sum of square, with $\bar{Y}$ the mean value of the dependent variable (in the training set). Finally, R^2_{base} is

$$R^2_{base} \equiv (1 - \frac{SSE}{SSE_{base}})$$

where SSE_{base} is the sum of squared errors in the baseline model, which we define to be the model that predicts the value on the forecast day to be the same as the value on the prediction day, otherwise known as the “persistence” forecast.

All models were trained on the six year period 2005-2010 and tested on the two year period 2011-2012.

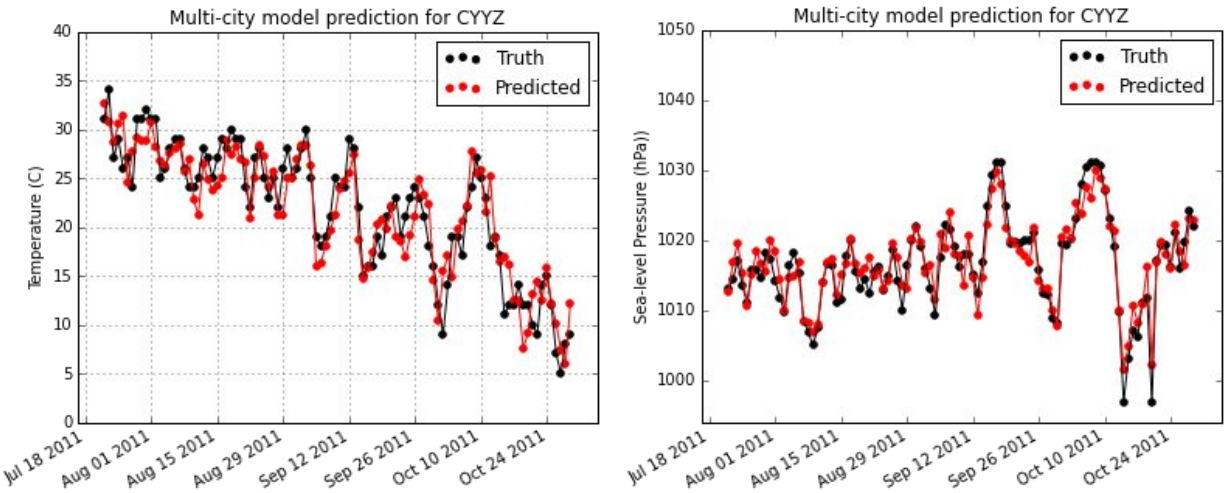

FIgure 10: Out-of-sample predictions of next-day daily maximum temperature `TempMax` (left) and sea-level pressure `PressMax` (right) at CYYZ using the multi-station two-day Taylor model.

Figure 10 shows next-day predictions of daily maximum temperature and daily maximum pressure at CYYZ for a representative 100 day interval during the testing period. The model uses two days worth of values of the variables `TempMax`, `dailyTempRange`, `PressMax`, `dailyPressRange`, `HumidityMean`, `isWesterly`, `isSoutherly`, and `isFoggy` at all 29 stations shown in Figure 1. The model performance statistics are listed in Table 4.

Target variable	`TempMax`	`PressMax`
R^2_mean (in-sample)	0.968	0.880
R^2_mean (out-of-sample)	0.943	0.773
R^2_base	0.639	0.686
RMSE	2.60 C	3.35 hPa

Table 4: Performance of multi-city two-day Taylor regression model for predicting next-day daily maximum temperature and pressure at CYYZ.

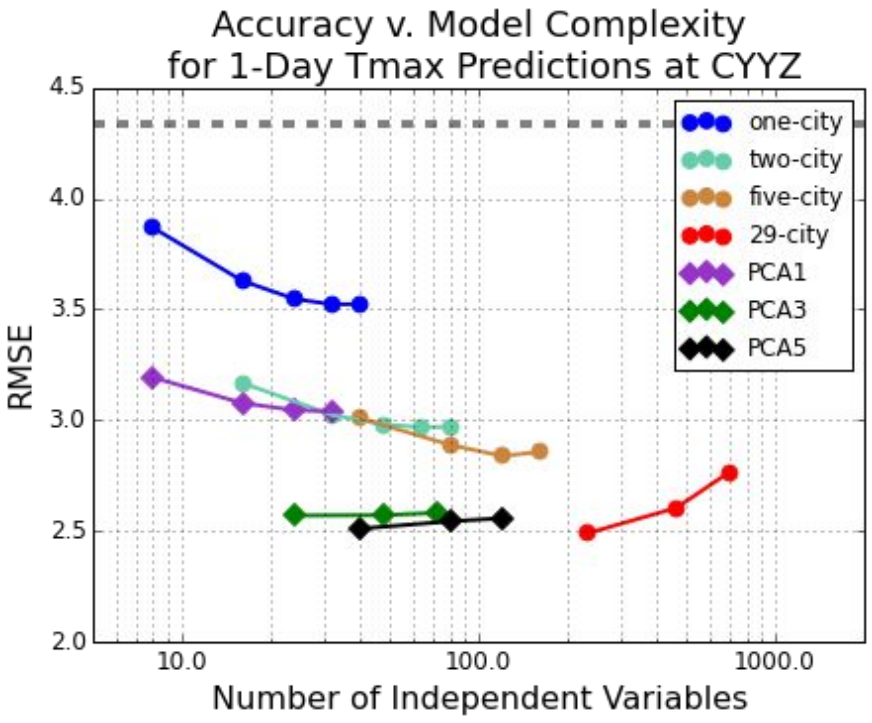

Figure 11: RMSE of several models as a function of the number of independent variables used for prediction. In addition to the target station CYYZ, the two-city model used station KORD, and the five-city model used stations KORD, KCLE, KDTW, and KBUF. Dashed greay line indicates the accuracy of the baseline model (no change since prediction day).

Figure 11 shows a comparison of the out-of-sample (i.e. testing set) RMSE of several models plotted against the total number of independent variables included in the model. The eight features TempMax, dailyTempRange, PressMax, dailyPressRange, HumidityMean, isWesterly, isSoutherly, and isFoggy were used as independent variables in each case, but at varying numbers of stations and on varying number of days before the prediction day. For example, the 29-station 3-day Taylor model uses 29 x 3 x 8 = 696 total independent variables. Notice that the out-of-sample RMSE increases with the number of variables in this range (with only six years -- about 2200 days -- of training data, such a model is likely to suffer from overfitting).

Of all the models tested, the one whose prediction of daily maximum temperature at CYYZ had the lowest out-of-sample RMSE (of 2.39 C) was an average of four models. The four models and the variables used for each are listed in Table 5. The distribution of errors made by the averaged model and by the baseline model are shown in Figure 13 together with best-fit normal curves. Notice that the errors are very close to normally distributed. Table 6 summarizes the performance of the model in extreme events. An n-sigma event is defined as a day when the day-over-day change is between n and n+1 standard deviations (here 4.33 C) away from zero.

One sees from Figure 11 that the models with the best compromise between efficiency -- in terms of number of features used and hence execution time -- and performance are the truncated principal component models. The model with only 3 PC of each feature (henceforth PCA-3) is competitive with the non-PCA model with over 200 total independent variables. The scikit-learn RFE module was used to isolate an optimum subset of PC. Starting with two-days worth of 5 PC of each of the available features TempMean, TempMin, TempMax, VisibilityMean, PressMean, PressMin, PressMax, HumidityMean, WindMaxSpd, WindMeanX, WindMeanY, dailyTempRange, dailyPressRange, isSoutherly, isWesterly, isFoggy, isMorningMinTemp, isMorningMaxTemp, isDaytimeMinPress, isDaytimeMaxPress, and isMorninMaxWind (205 total independent variables), subsets of different numbers of “best features” for the prediction of daily maximum temperature at CYYZ were extracted.

Figure 14 compares the in-sample and out-of-sample RMSE of each of these models. There is no benefit in terms of RMSE to using more than 20 best features. The in-sample error continues to decrease with more variables but not the out-of-sample error, a sign of overfitting Table 7 lists the features selected by the 20-best-feature model and the regression coefficient for each feature (since the features are standardized, the size of the coefficient is a measure of the importance of the feature). Not surprisingly the first 8 features and 15 of the 20 features overall are associated with temperature, 4 features are associated with pressure and one with the north-south wind component (which can plausibly influence changes in temperature).

Model	**Features** [number of PC]
Multi-station two-day with interactions	`'TempMax','dailyTempRange','TempMax:WindMeanX','TempMax:WindMeanY', 'TempMax:isFoggy','PressMax','dailyPressRange','HumidityMean'`
Multi-station two-day	`'TempMax','dailyTempRange','isWesterly','isSoutherly','isFoggy', 'PressMax','dailyPressRange','HumidityMean'`
PCA two-day	`'TempMax'[8],'dailyTempRange'[5],'PressMax'[8],'dailyPressRange'[5], 'HumidityMean'[5],'isSoutherly'[5],'isWesterly'[5],'isFoggy'[5]`
Advection two-day	`'TempMax', 'PressMax'`

Table 5: Parameters of the four models used for two-day composite-model prediction.

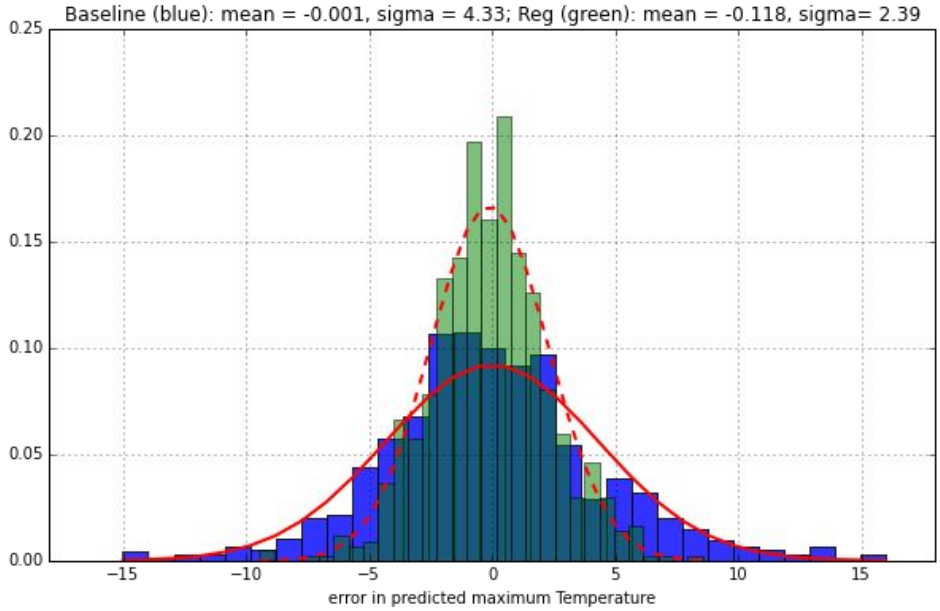

Figure 13: Distributions of out-of-sample errors of four-model composite model (green; see Table 4) and baseline model (blue) for predictions of next-day maximum temperature at CYYZ.

Event	number	Baseline RMSE	Model RMSE	|Err| < |Baseline|
0-sigma	460	2.39 C	1.96 C	67.8%
1-sigma	151	6.13 C	3.07 C	97.4%
2-sigma	31	10.15 C	3.47 C	100%
3-sigma	11	14.05 C	5.61 C	100%

Table 6: Statistics of errors by averaged model (see Table 5) and baseline in extreme events.

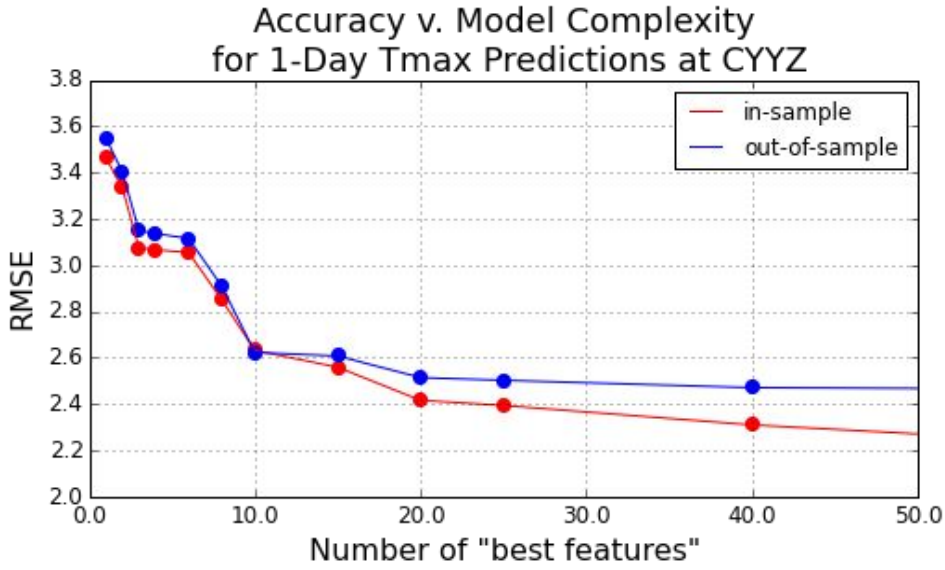

Figure 14: In- and out-of-sample RMSE for PCA model as function of number of best features selected with the sklearn.feature_selection.RFE package.

intercept	13.20		
TempMean_PC4	2.87	TempMin_PC0_D	0.61
TempMean_PC0	2.71	TempMin_PC1	0.58
TempMax_PC2	1.22	TempMax_PC3	-0.56
TempMin_PC4	1.19	PressMin_PC2_D	-0.41
TempMax_PC4	-1.17	dailyPressRange_PC3	-0.38
TempMin_PC0	-1.01	TempMax_PC0	0.38
TempMin_PC3	0.98	WindMeanY_PC1	-0.35
TempMean_PC4_D	-0.69	dailyTempRange_PC4	-0.35
PressMax_PC2_D	0.66	dailyTempRange_PC0_D	0.33
PressMean_PC2_D	-0.63	isMorningMinTemp_PC3	-0.10

Table 7: Regression coefficients from best-20-features two-day PCA model, sorted by absolute value; _PCn indicates the PC number and _D indicates a difference term.

Clustering Analysis

The future value of a meteorological variable (such as temperature) at a station depends on the current conditions at that station and the current conditions in the surrounding region. For example, if there is a large-scale westerly wind, conditions to the west of the station will have more predictive value than conditions to the east. More to the point, the regression coefficients corresponding to conditions at a station to the west will differ depending on whether the wind is westerly or not. In general, different weather patterns should require different regression models.

This reasoning suggests that a clustering analysis that separates days based on large-scale weather patterns, with a different regression model trained for each cluster, might lead to improved forecasts. Recall that the principal components of temperature and pressure are indicative of different patterns (see Figure 9) -- for example, when the second PC of temperature is positive, there is an east-to-west temperature gradient (perhaps a hot summer day) and when it is negative, the gradient is reversed (as it might be on a cold winter day). After some experimentation it was found that dividing the data into 4 clusters based on the third PC (PC2) of `TempMax` and the third PC of `PressMax` resulted in a modest decrease in overall out-of-sample RMSE. Most other choices of variables for clustering actually increased the overall RMSE (surprisingly).

Figure 15 is a scatter plot of the values of the two clustering variables colour-coded by cluster assignment. Notice that the average forecast error is different for the different clusters. Days with positive values of `TempMax-PC2` have lower errors than negative values. An explanation for this might be that a weakened north-south temperature gradient, implying a more homogeneous temperature distribution, means smaller temperature changes and generally less volatile weather. The left panel of Figure 16 shows the occurrence of the four clusters over the two year testing period. There is no very obvious pattern, other than cluster 1 (red markers) occurring predominantly in spring and cluster 2 (green) corresponding to mild winter days.

The right panel of Figure 16 shows cluster occurrence over time for clustering on `TempMax-PC0` only, which divides the data into summer, winter and spring/fall. Evidently, predictability is worse than average (RMSE is higher) in spring and fall and better than average (lower RMSE) in summer and winter.

Predictability by forecast day

Naturally the mean prediction error is larger when the forecast day is further in the future. Figure 17 shows the RMSE and R^2_{base} for forecasts of `TempMax` at CYYZ and KJFK one through seven days in the future using both the one-station and PCA-3 models. The same eight

features were used as for Figure 11. The PCA-3 model is more accurate but its error grows more or less as quickly with forecast day as does the one-station model.

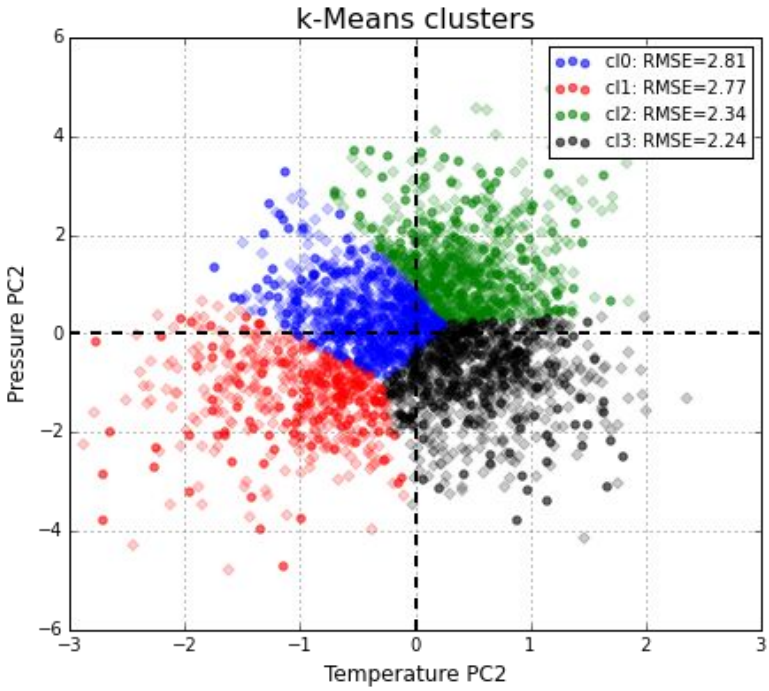

Figure 15: k-Means cluster allocation of training (pale markers) and testing (dark markers) data. RMSE of PCA3-model predictions of `TempMax` at CYYZ for each cluster indicated in the legend.

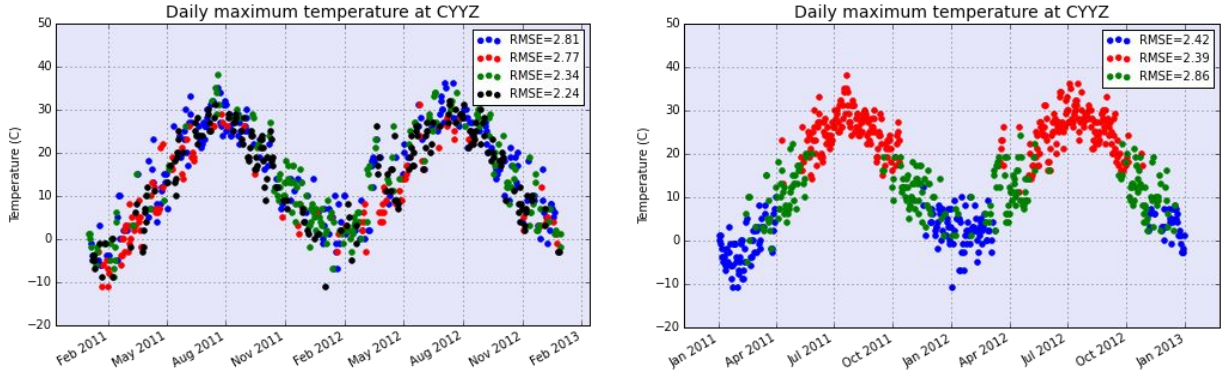

Figure 16: PCA3-Cluster model out-of-sample predictions versus time for clustering on [`TempMax-PC2`, `PressMax-PC2`] (left) and on `TempMax-PC0` (right). Cluster assignments are indicated by marker colour and RMSE by cluster in the legend.

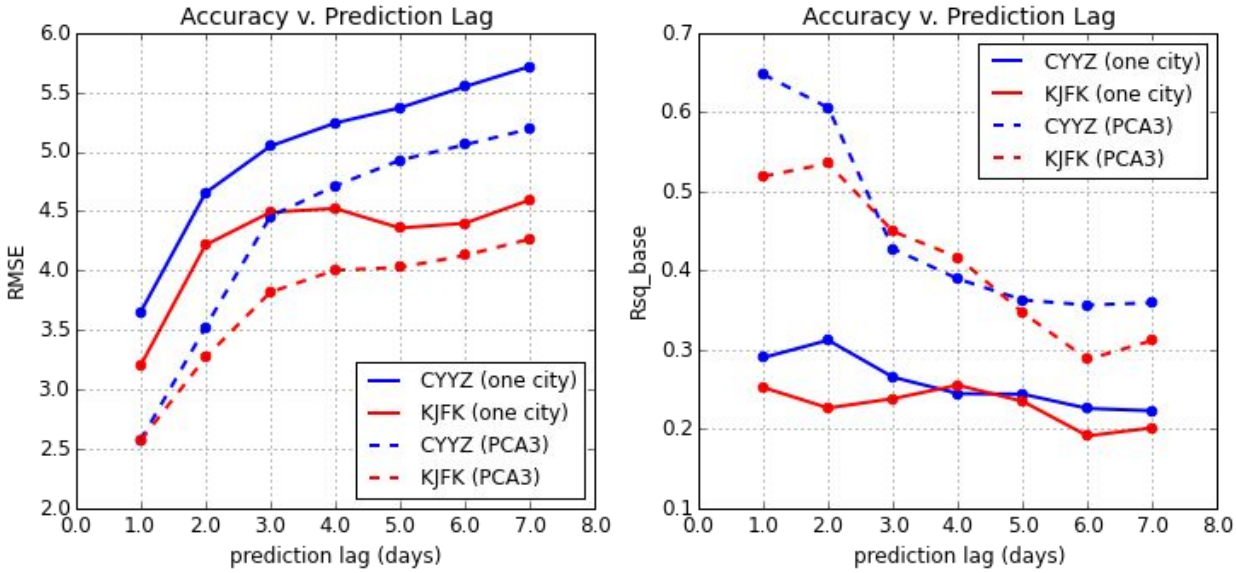

Figure 17: Prediction accuracy as function of number of days between prediction and forecast days: RMSE (left) and R^2_{base}(right).

Since R^2_{base} for the PCA-3 model decreases with forecast day, the relative predictive value of the regression model is reduced for longer forecast times. The one-station model does approximately as well compared to the persistence baseline model for long and short forecast times. The RMSE for KJFK is always lower than for CYYZ but the R^2_{base} is smaller (worse) suggesting that the persistence model is "better" for KJFK (in other words the temperature is less variable at KJFK, probably due to the moderating influence of the ocean).

In his discussion of the accuracy of weather forecasting, Silver (2012) reports that the accuracy of professional forecasts of `TempMax` increases approximately linearly with forecast day. Forecasts longer than about 9 days do not even do as well on average as predicting the "climatological" average for that station on that day of the year.

Predictability by station location

Both one-station and PCA-3 models were used to predict next-day `TempMax` at each of the 29 stations in Figure 1.

Figure 18 shows contour plots of the out-of-sample RMSE and R^2_{base}. The one-station predictions for stations on the Atlantic coast have lower than average RMSE but not necessarily higher than average R^2_{base}, probably reflecting the lower variability of temperature due to the moderating effect of the ocean (water having a higher heat capacity than land).

With the PCA-3 models, RMSE decreased from northwest to southeast. Since weather systems in North America tends to move from west to east, stations further east within the network benefit more from information from other stations than do stations further west.

The stations for which R^2_{base} was highest with both types of model were around the eastern Great Lakes.

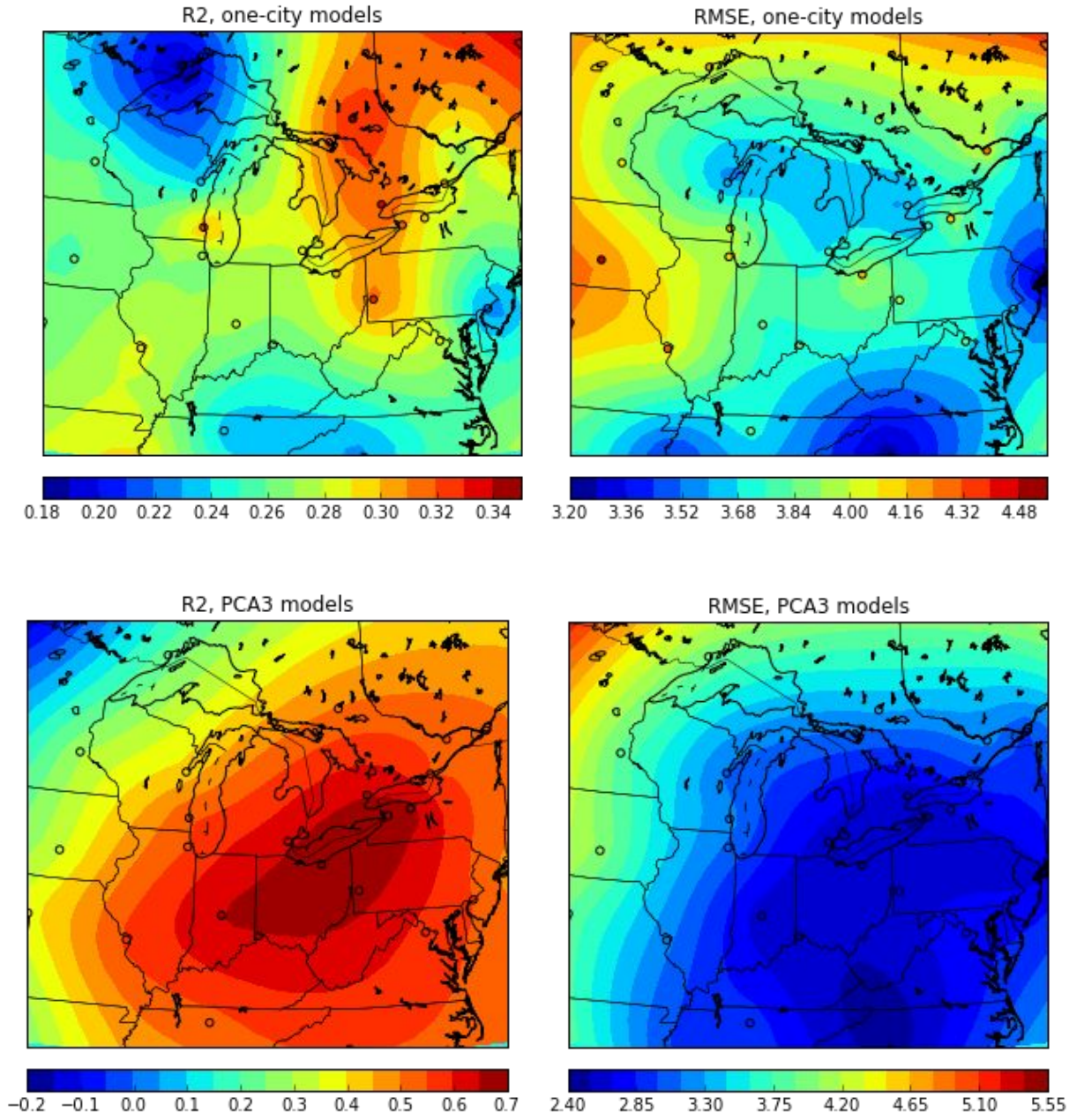

Figure 18: R^2_{base} (left column) and RMSE (right column) for next-day predictions of `TempMax` using one-station two-day model (top row) and PCA-3 one-day model (bottom row).

Comparison with published forecasts

One-, two- and three-day ahead forecasts of `TempMax` and `TempMin` at CYYZ for the period November 11-23, 2015 published at www.wunderground.com were noted. The left panel of FIgure 19 compares the published predictions of `TempMax` with those of the PCA-3 regression model (trained as usual on the years 2005-2010) and with the observed values. The regression model did not capture the cooling between Nov 15 and 17 but was otherwise competitive with the published forecast. Table 8 compares the RMSE of the published predictions with those of the regression model. Based on this admittedly very small sample size, the regression model errors are generally higher and increase more with forecast day. Presumably the decrease in RMSE with forecast day of the published predictions is anomalous and due to chance.

The values of `TempMax-PC2` and `PressMax-PC2` of the test points are plotted in the right panel of Figure 19. Most of the points fell in the blue and green clusters, but the respective errors in those clusters do not reflect the pattern in the large 2011-2012 test set (compare Figure 15). Again, one should not draw any general conclusions from such a small sample.

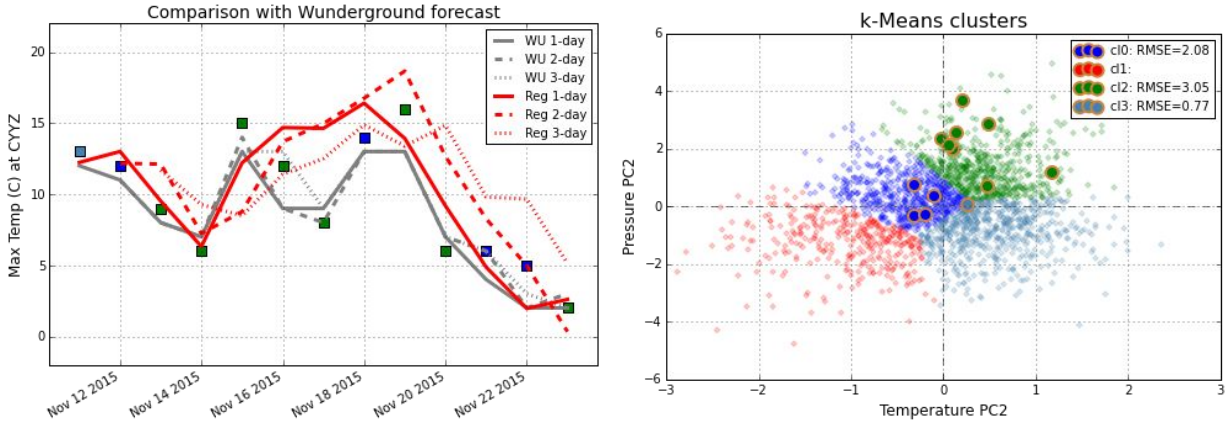

Figure 19: Forecast of daily maximum temperature at CYYZ for the period Nov 11-23, 2015 using the PCA-3 model (red), forecast from wunderground.com (grey) and observations (square markers, colour-coded by cluster) (left); Cluster assignments of test points (cf. Figure 15) (right).

Prediction Day	wunderground		PCA3	
-1 day	2.65 C	1.80 C	1.92 C	2.18 C
-2 days	2.61 C	1.68 C	2.60 C	3.94 C
-3 days	2.95 C	1.45 C	3.22 C	4.16 C

Table 8: RMSE from wunderground.com and PCA-3 (3 PC per feature) regression model forecasts of maximum and minimum temperature at CYYZ for Nov 11-23, 2015.

Summary and Outlook

Methods of predictive analytics, including multiple linear regression, principal component analysis and k-means clustering, were applied to the classic problem of weather forecasting. Daily aggregate statistics computed from eight years (2005-2012) of hourly observations from a network of 29 airport weather stations in the eastern part of North America were used to train and test a hierarchy of regression models. Models using only data from the station for which the forecast is to be obtained only slightly outperformed the baseline "persistence" estimate that the value on the forecast day does not change from the prediction day. Models using data from all stations do considerably better, at the cost of higher complexity and strong multicollinearity (and they are prone to overfitting). Models built using principal component analysis (PCA) achieve similar performance using as few as 10% as many features as the direct multi-station models.

Observations of several meteorological quantities, including temperature, atmospheric pressure, humidity, and wind speed and direction were used. For prediction of future temperature, the most useful variable was, naturally, temperature, with pressure and wind speed also of value. The out-of-sample RMSE for next-day predictions of daily maximum temperature was approximately 2.5 C using the best of the regression models.

Information from days prior to the prediction day improved forecasts using the one-station model but not significantly for the many-station and PCA models (see Figure 11), partly due to overfitting, since information from previous days multiplies the number of features used, and partly because information from multiple stations already provides some information about the tendency of the temperature (since weather systems move through space).

k-Means clustering did not significantly reduce the prediction errors, but does seem to be a useful tool in interpreting results. The performance of the regression models varies for different clusters, which represent different weather patterns.

This project has represented a somewhat crude first step in developing a practical forecasting tool from "brute force" predictive analytics methods. The detailed quasi-hourly data was not used to its full potential -- perhaps sophisticated pattern-matching of the time series on each day could be used to generate new summary variables with predictive values. Also, many times the number of stations could easily be included. Ultimately, it is not hard to imagine that in the future, deep learning could be used to predict future weather from supplied data using models developed entirely by machine. These models will still use combinations of temperature, pressure, wind speed, humidity, etc. at different locations and times, but they will bear little resemblance to the differential equations of fluid mechanics and be incomprehensible to human users. Yet the new models will (almost) certainly outperform physics-based models.

Appendix A - Weather Underground Forecasts at Toronto Pearson Airport November 10-23, 2015

The following forecasts of daily minimum and maximum temperature for the station CYYZ (Toronto Pearson Airport) published on www.wunderground.com were noted on the days November 10-23, 2015 for comparison with predictions of regression models and observed values.

Forecast Date	Forecast (deg C)						***Obs*** *(deg C).*	
	-1 Day		***-2 Day***		***-3 Day***			
2015-11-11	6	*12*					**7**	**13**
2015-11-12	6	*11*	*6*	*11*			**7**	**12**
2015-11-13	1	*8*	*1*	*8*	*2*	*8*	**3**	**9**
2015-11-14	2	*7*	*2*	*7*	*2*	*7*	**1**	**6**
2015-11-15	3	*13*	*3*	*14*	*4*	*13*	**2**	**15**
2015-11-16	2	*9*	*4*	*9*	*6*	*13*	**2**	**12**
2015-11-17	7	*9*	*6*	*8*	*7*	*9*	**2**	**8**
2015-11-18	11	*13*	*11*	*13*	*9*	*13*	**7**	**14**
2015-11-19	1	*13*	*2*	*13*	*1*	*13*	**5**	**16**
2015-11-20	-1	*7*	*0*	*7*	*-1*	*7*	**2**	**6**
2015-11-21	0	*4*	*2*	*6*	*-1*	*6*	**1**	**6**
2015-11-22	-4	*2*	*-4*	*2*	*-4*	*3*	**-4**	**5**
2015-11-23	-2	*2*	*-1*	*3*	*-2*	*2*	**-6**	**2**
2015-11-24	-2	*6*	*-2*	*6*	*-1*	*6*	**-1**	**6**

Table 9: Minimum (blue) and maximum (red) daily temperature forecasts from the Weather Underground website and corresponding observed values.

Appendix B - IPython Notebooks

The IPython notebooks listed in Table 10 are available on the project github page in the folder https://github.com/majorgowan/wpwp/tree/master/Notebooks/

wU_Data_retrieval	*Downloading station data from the Weather Underground website and archiving to JSON files*
wU_Working_with_Hourly_Data	*Reading, cleaning and visualizing the quasi-hourly data stored in a JSON file*
wU_Working_with_Daily_Summaries	*Reading, analyzing and visualizing daily summary data stored in CSV files; multiple linear regression using the scikit-learn and statsmodels packages*
wU_Visualize_Data_on_Map	*Plots of station locations and contours of feature data on a map projection background (uses basemap package)*
wU_PCA_Analysis	*Applying Principal Component Analysis (PCA) to the daily summary data; visualizing principal components on map background*
wU_Clustering	*k-Means clustering applied to daily summary data*
wU_Regression_models_comparison_nvars	*Comparison of regression model performance as function of number of independent variables*
wU_Regression_models_comparison_lag	*Dependence of predictability on prediction day*
wU_Regression_feature_selection.ipynb	*Use scikit-learn RFE module to select subset of variables to build optimum regression model*
wU_Regression_models **(main notebook!!)**	*Apply the various flavours of multiple linear regression to the problem of forecasting weather variables (concentrating on daily maximum temperature)*

Table 10: List of IPython notebooks used in this project.

Appendix C - List of Python Modules and Functions

The following python modules are available on the project github page in the folder https://github.com/majorgowan/wpwp/tree/master/Python/

wUnderground.py	
readStationDay(stationCode, year, month, day, echo=False) read one day of observations of wunderground.com	
input	stationCode (string): IATA code of station year (integer), month (integer), day (integer), echo (boolean)
returns	string [text of webpage]
daysInMonth(year, month) compute the number of days in a given month (accounting for leap years)	
input	year (integer), month (integer)
returns	integer [number of days in month]
readDay(data) process string of lines containing CSV data	
input	data (string)
returns	list of dictionaries [one day of observations]
readInterval(stationCode, start, end, verbose=True) retrieve several months of data from a single station	
input	stationCode (string), start (string; 'YYYY-MM'), end (string; 'YYYY-MM'), verbose (boolean)
returns	list of datetime objects [dates of days in data], list of lists of dictionaries [observation data], list of tuples [information on failed retrievals]
putJSON(dates, data, station, year, month) store a month of data in JSON format	
input	dates (list of datetime objects), data (list of list of dictionaries), station (string), year (integer), month (integer)
returns	None

getJSON(station, year, month) load a month of data from JSON file	
input	station (string), year (integer), month (integer)
returns	list of datetime objects [dates of days in file], list of list of dictionaries [observation data]
retrieveStationYear(station, year) retrieve a full year of data from a single station; store as JSON files	
input	station (string), year (integer)
returns	None
getJSONFolder(station) load all stored data from a folder of JSON files	
input	station (string)
returns	list of datetime objects [dates of days in file], list of list of dictionaries [observation data]

wUUtils.py	
getStationList() scan JSON_DATA folder to find stations present	
returns	list of strings [station IATA codes]
getStationLonLat(stations) query airports data file to find coordinates of stations	
input	stations (list of strings)
returns	list of floats [longitudes of stations], list of floats [latitudes of stations]
loadDailyVariable(stations, outdate, variable) load observations of one variable for a given date from a list of stations	
input	stations (list of strings), outdate (string; 'YYYY-MM-DD'), variable (string)
returns	list of floats [one value per stations]
loadDailyVariableRange(station, startDate, endDate, variable, castFloat=False) load observations of one variable for a given station for a range of dates	

input	station (string), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD'), variable (string), castFloat (boolean; True: return floats, False: strings)
returns	list [of floats or strings]
dateList(startDate, endDate) generate a list of datetime objects	
input	startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD')
returns	list of datetime objects
smooth(data, window=3) apply a running-mean "boxcar" filter to a list of floats	
input	data (list of floats), window (integer)
returns	list of floats [smoothed list]

wUStats.py	
collectAllStats(stationList = None) process JSON files of hourly data and store daily summaries as CSV	
input	stationList (list of strings)
returns	None
getTimeZoneOffset(data) detect timezone of list of days of JSON data and return list of offsets	
input	data (list of lists of dictionaries)
returns	list of integers [timezone offset for each day]
isMissing(valString) determine if a given record is a missing value	
input	valString (string)
returns	boolean [True: is missing, False: not missing]
addWindVectors(data) calculate eastward and northward components of daily mean wind for a set of days	
input	data (list of lists of dictionaries)

returns	None [adds two keys and values to each dictionary]
toSecondsValues(utc, y) removing missing values; convert set of pairs of (UTC string, value) to pairs (seconds since first UTC string, value)	
input	utc (list of strings; UTC time codes), y (list)
returns	list of floats [time since first UTC code], list [cleaned values]
trapezoid(utc, y) compute time integral numerically using trapezoid method	
input	utc (list of strings; UTC time codes), y (list)
returns	float [time spanned by UTC codes in seconds], float [value of integral]
rectangle(utc, y) compute time integral numerically using rectangle method	
input	utc (list of strings; UTC time codes), y (list)
returns	float [time spanned by UTC codes in seconds], float [value of integral]
dailyMean(data, variable, method='rectangle') compute daily means of a list of days of hourly data	
input	data (list of list of dictionaries), variable (string), method (string; integration method to use)
returns	list of floats [daily means of variable]
dailySum(data, variable) compute daily sums of a list of days of hourly data	
input	data (list of list of dictionaries), variable (string)
returns	list of floats [daily sums of variable]
dailyMax(data, variable, minmax=1) compute daily maxima or minima of a list of days of hourly data	
input	data (list of list of dictionaries), variable (string), minmax (integer; 1=compute max, -1=compute min)
returns	list of floats [daily maxima or minima of variable], list of strings [UTC codes of times at which extrema occur]
dailyMaxWind(data)	

	compute daily maximum wind speeds from a list of days of hourly data
input	data (list of list of dictionaries)
returns	list of floats [daily maximum wind speeds], list of strings [UTC codes of times at which maxima occur]

wUDerived.py	
dailyTempRange(station, startDate, endDate)	daily range of temperature (maximum minus minimum)
input	station (string), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD')
returns	list of floats [negative if maximum precedes minimum]
dailyPressRange(station, startDate, endDate)	daily range of sea-level pressure (maximum minus minimum)
input	station (string), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD')
returns	list of floats [negative if maximum precedes minimum]
isWesterly/isEasterly/isNortherly/isSoutherly(station, startDate, endDate)	determine if wind is from the west (east, north, south)
input	station (string), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD')
returns	list of booleans
isFoggy(station, startDate, endDate)	determine if daily mean visibility is less than 5 km
input	station (string), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD')
returns	list of booleans
isMorningMinTemp/isMorningMaxTemp(station, startDate, endDate)	determine if daily minimum/maximum temperature occurs before noon
input	station (string), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD')
returns	list of booleans

isDaytimeMinPress/isDaytimeMaxPress(station, startDate, endDate) determine if daily minimum/maximum pressure occurs between 8:00 and 20:00	
input	station (string), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD')
returns	list of booleans
isMorningMaxWind(station, startDate, endDate) determine if daily maximum wind speed occurs before noon	
input	station (string), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD')
returns	list of booleans

wUAdvection.py	
unitVector(lon, lat) compute unit vector directed from one point to another	
input	lon (list of two floats), lat (list of two floats)
returns	list of two floats
dDeriv(station1, station2, variable, startDate, endDate) compute directional derivative of variable in directionfrom one station to another	
input	station1 (string), station2 (string), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD')
returns	list of floats [directional derivatives for all days in interval], list of two floats [unit vector between two stations]

wUPCA.py	
pcaComp(data, ncomp = None) compute standardization and principal component transforms for a set of vectors	
input	data (numpy array), ncomp (integer; number of PC to compute)
returns	PCA object, StandardScaler object
pcaTransform(pc, scaler, data) apply standardization and PC transforms to a set of vectors	

input	pc (PCA object), scaler (StandardScaler object), data (numpy array)
returns	numpy array [columns in PC space]

pcaInverseTransform(pc, scaler, pcData)
apply inverse PC and standardization transforms to a set of vectors

input	pc (PCA object), scaler (StandardScaler object), pcData (numpy array)
returns	numpy array [columns in original space]

pcaConvert(stations, features, startDate, endDate, ncomp=None)
retrieve set of features for set of stations over an interval of dates and compute PC

input	stations (list of strings), features (list of strings), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD'), ncomp (list of integers)
returns	list of lists of floats [PC data for each feature], dictionary {'features': list of strings, 'stations': list of strings, 'ncomp': list of integers, 'scalers': list of StandardScaler objects, 'pcas': list of PCA objects}

pcaPredict(transform_params, startDate, endDate)
retrieve set of features for set of stations over an interval of dates and compute PC

input	transform_params (dictionary; see pcaConvert), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD')
returns	list of lists of floats [PC data for each feature]

wUCluster.py

computeClusters(data, nclusters, ranseed = 666, scale=True)
compute k-Means clusters for a set of vectors

input	data (numpy array), nclusters (integer; number of clusters), ranseed (integer; seed for random number generator)
returns	StandardScaler object, KMeans object

assignClusters(scaler, clusterer, data)

	assign clusters to set of data
input	scaler (StandardScaler object), clusterer (KMeans object), data (numpy array)
returns	list of integers [indices of cluster for each row in data]
clusterFeatureData(featureData, stations, features, clusterFeatures, nclusters, ranseed=666) partition a set of data containing several features into clusters based on values of a subset of the features	
input	featureData (numpy array), stations (list of strings), features (list of strings), clusterFeatures (list of strings), nclusters (integer), ranseed (integer)
returns	list of integers [cluster assignments], dictionary {'scaler': StandardScaler object, 'clusterer': KMeans object, 'nclusters': integer, 'features': list of strings, 'nstations': integer, 'clusterFeatures': list of strings}
assignClustersAllFeatures(featureData, clusterParams) partition a set of data into precomputed clusters	
input	featureData (numpy array), clusterParams (dictionary; see clusterFeatureData)
returns	list of integers [indices of cluster for each row in data], list of numpy arrays [featureData partitioned into clusters]

wUMapPlots.py	
plotStationsOnMap(showIt=True) plot all stations on map	
input	showIt (boolean)
returns	None
plotWindVectorsOnMap(date, showIt=True) plot daily mean wind vectors for a given day	
input	date (string; 'YYYY-MM-DD'), showIt (boolean)
returns	None

plotMeanWindVectorsOnMap(startDate, endDate, showIt=True) plot wind vectors averaged over an interval of days	
input	startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD'), showIt (boolean)
returns	None
contourPlotStationsOnMap(stations, data, title='data', width_fac = 16, height_fac = 12) create a contour plot of data defined at several stations	
input	stations (list of strings), data (list of floats), width_fac (integer), height_fac (integer)
returns	None
contourPlotVarOnMap(variable, date, npts = 20, ncntrs = 10, width_fac = 16, height_fac = 12) contour plot of a daily variable on a given day	
input	variable (string), date (string; 'YYYY-MM-DD'), npts (integer), ncntrs (integer), width_fac (integer), height_fac (integer)
returns	None
contourPlotMeanVarOnMap(variable, startDate, endDate, npts = 20, ncntrs = 10, width_fac = 16, height_fac = 12) contour plot of the time average of a variable over an interval of days	
input	variable (string), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD'), npts (integer), ncntrs (integer), width_fac (integer), height_fac (integer)
returns	None

wURegression.py	
oneCityModel(station, startDate, endDate, features, targetVar='TempMax', lag=1, scale=False) construct regression model using data from target station on prediction day only	
input	station (string), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD'), features (list of strings), targetVar (string), lag (integer), scale (boolean)
returns	numpy array [independent variable data],

	list of floats [dependent (target) variable data], dictionary = {'station': string, 'startDate': string, 'endDate': endDate, 'targetVar': string, 'features': list of strings, 'regr': LinearRegression object, 'lag': integer, 'scale': boolean, 'scaler': StandardScaler object}
oneCityPredict(model_params, startDate, endDate, actual=True) make predictions with one-city model	
input	model_params (dictionary; see oneCityModel), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD'), actual (boolean)
returns	list of datetime objects, list of floats [predictions for target variable], list of floats [actual target variable data (if available)], dictionary = {'R2_mean': float, 'R2_base': float, 'RMSE': float}
oneCityTaylorModel(station, startDate, endDate, features, targetVar='TempMax', lag=1, order=0, verbose=True, scale=False) construct regression model using data from target station on prediction day and earlier	
input	station (string), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD'), features (list of strings), targetVar (string), lag (integer), order (integer), verbose (boolean), scale (boolean)
returns	numpy array [independent variable data], list of floats [dependent (target) variable data], dictionary = {'station': string, 'startDate': string, 'endDate': endDate, 'targetVar': string, 'features': list of strings, 'regr': LinearRegression object, 'lag': integer, 'order': integer, 'scale': boolean, 'scaler': StandardScaler object}
oneCityTaylorPredict(model_params, startDate, endDate, actual=True) make predictions with one-city Taylor model	
input	model_params (dictionary; see oneCityTaylorModel), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD'), actual (boolean)
returns	list of datetime objects,

	list of floats [predictions for target variable], list of floats [actual target variable data (if available)], dictionary = {'R2_mean': float, 'R2_base': float, 'RMSE': float}
multiCityTaylorModel(stations, startDate, endDate, features, targetVar='TempMax', lag=1, order=0, verbose=True, scale=False) construct regression model using data from a list of stations on prediction day and earlier	
input	stations (list of strings), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD'), features (list of strings), targetVar (string), lag (integer), order (integer), verbose (boolean), scale (boolean)
returns	numpy array [independent variable data], list of floats [dependent (target) variable data], dictionary = {'stations': list of strings, 'startDate': string, 'endDate': endDate, 'targetVar': string, 'features': list of strings, 'regr': LinearRegression object, 'lag': integer, 'order': integer, 'scale': boolean, 'scaler': StandardScaler object}
multiCityTaylorPredict(model_params, startDate, endDate, actual=True) make predictions with multi-city Taylor model	
input	model_params (dictionary; see multiCityTaylorModel), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD'), actual (boolean)
returns	list of datetime objects, list of floats [predictions for target variable], list of floats [actual target variable data (if available)], dictionary = {'R2_mean': float, 'R2_base': float, 'RMSE': float}
multiCityInteractionModel(stations, startDate, endDate, features, targetVar='TempMax', lag=1, order=0, verbose=True, scale=False) construct regression model using data from a list of stations on prediction day and earlier; allow for interaction terms	
input	stations (list of strings), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD'), features (list of strings), targetVar (string), lag (integer), order (integer), verbose (boolean), scale (boolean)
returns	numpy array [independent variable data], list of floats [dependent (target) variable data], dictionary = {'stations': list of strings,

	'startDate': string, 'endDate': endDate, 'targetVar': string, 'features': list of strings, 'regr': LinearRegression object, 'lag': integer, 'order': integer, 'scale': boolean, 'scaler': StandardScaler object, 'prescalers': list of StandardScaler objects}
multiCityInteractionPredict(model_params, startDate, endDate, actual=True) make predictions with multi-city Taylor model with interactions	
input	model_params (dictionary; see multiCityInteractionModel), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD'), actual (boolean)
returns	list of datetime objects, list of floats [predictions for target variable], list of floats [actual target variable data (if available)], dictionary = {'R2_mean': float, 'R2_base': float, 'RMSE': float}
advectionTaylorModel(stations, startDate, endDate, features, targetVar='TempMax', lag=1, order=0, verbose=False) construct regression model using data from a list of stations on prediction day and earlier; use only "advection" towards target city	
input	stations (list of strings), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD'), features (list of strings), targetVar (string), lag (integer), order (integer), verbose (boolean)
returns	numpy array [independent variable data], list of floats [dependent (target) variable data], dictionary = {'stations': list of strings, 'startDate': string, 'endDate': endDate, 'targetVar': string, 'features': list of strings, 'regr': LinearRegression object, 'lag': integer, 'order': integer}
advectionTaylorPredict(model_params, startDate, endDate, actual=True) make predictions with "advection" model	
input	model_params (dictionary; see advectionTaylorModel), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD'), actual (boolean)
returns	list of datetime objects, list of floats [predictions for target variable], list of floats [actual target variable data (if available)],

	dictionary = {'R2_mean': float, 'R2_base': float, 'RMSE': float}
pcaTaylorModel(stations, startDate, endDate, features, ncomp=None, targetVar='TempMax', lag=1, order=0, smooth_window=0, verbose=False) construct regression model using data from a list of stations on prediction day and earlier; transform feature data using PCA	
input	stations (list of strings), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD'), features (list of strings), ncomp (list of integers), targetVar (string), lag (integer), order (integer), smooth_window (integer), verbose (boolean)
returns	numpy array [independent variable data], list of floats [dependent (target) variable data], dictionary = {'stations': list of strings, 'startDate': startDate, 'endDate': endDate, 'targetVar': string, 'features': list of strings, 'regr': LinearRegression object, 'lag': integer, 'order': integer, 'smooth_window': integer, 'transform_params': dictionary (see wUPCA.pcaConvert)}
pcaTaylorPredict(model_params, startDate, endDate, actual=True) make predictions with PCA regression model	
input	model_params (dictionary; see pcaTaylorModel), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD'), actual (boolean)
returns	list of datetime objects, list of floats [predictions for target variable], list of floats [actual target variable data (if available)], dictionary = {'R2_mean': float, 'R2_base': float, 'RMSE': float}
compareStationsOneCity(stations, startTrain, endTrain, startTest, endTest, features, targetVar='TempMax', lag=1, order=0) construct and test one-city regression models for each station in list	
input	stations (list of strings), startTrain (string; 'YYYY-MM-DD'), endTrain (string; 'YYYY-MM-DD'), startTest (string; 'YYYY-MM-DD'), endTest (string; 'YYYY-MM-DD'), features (list of strings), targetVar (string), lag (integer), order (integer)
returns	list of dictionaries (cf oneCityPredict) {'station': string, 'R2_mean': float, 'R2_base': float, 'RMSE': float}
compareStationsMultiCity(stations, startTrain, endTrain, startTest, endTest, features, targetVar='TempMax', lag=1, order=0)	

construct and test multi-city regression models for each (target) station in list	
input	stations (list of strings), startTrain (string; 'YYYY-MM-DD'), endTrain (string; 'YYYY-MM-DD'), startTest (string; 'YYYY-MM-DD'), endTest (string; 'YYYY-MM-DD'), features (list of strings), targetVar (string), lag (integer), order (integer)
returns	list of dictionaries (cf oneCityPredict) {'station': string, 'R2_mean': float, 'R2_base': float, 'RMSE': float}

compareStationsPCA(stations, startTrain, endTrain, startTest, endTest, features, ncomp, targetVar='TempMax', lag=1, order=0) construct and test multi-city regression models for each (target) station in list	
input	stations (list of strings), startTrain (string; 'YYYY-MM-DD'), endTrain (string; 'YYYY-MM-DD'), startTest (string; 'YYYY-MM-DD'), endTest (string; 'YYYY-MM-DD'), features (list of strings), ncomp (list of integers), targetVar (string), lag (integer), order (integer)
returns	list of dictionaries (cf oneCityPredict) {'station': string, 'R2_mean': float, 'R2_base': float, 'RMSE': float}

plotModelPred(date_list, pred, target, startIndex=0, endIndex=0, showIt=True) plot predicted and actual values of target variable on same axes	
input	date_list (list of datetime objects), pred (list of floats), target (list of floats), startIndex (integer), endIndex (integer), showIt (boolean)
returns	Axes object

wUClusterRegression.py	
clusterRegression(stations, startDate, endDate, features, clusterFeatures=None, nclusters=1, ranseed=666, targetVar='TempMax', lag=1, order=0, scale=False, verbose=False) compute k-Means clusters and construct regression model for each	
input	stations (set of strings), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD'), features (list of strings), clusterFeatures (list of strings), nclusters (integer), ranseed (integer), targetVar (string), lag (integer), order (integer), scale (boolean), verbose (boolean)
returns	numpy array [independent variable data],

	list of floats [dependent (target) variable data], dictionary = {'stations': list of strings, 'startDate': string, 'endDate': string, 'targetVar': string, 'features': list of strings, 'regrs': list of LinearRegression objects, 'clusterParams': dictionary (see wUCluster.clusterFeatureData) 'classes': list of integers, 'lag': integer, 'order': integer, 'scale': boolean, 'scalers': list of StardardScaler objects, 'prescalers': list of StandardScaler ojects}
clusterRegressionPredict(model_params, startDate, endDate, actual=True) make predictions with cluster regression model	
input	model_params (dictionary; see clusterRegression), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD'), actual (boolean)
returns	list of datetime objects, list of floats [predictions for target variable], list of floats [actual target variable data (if available)], numpy array [independent variable data of test set] list of integers [cluster assignments of test data] dictionary = {'R2_mean': float, 'R2_base': float, 'RMSE': float}
pcaClusterModel(stations, startDate, endDate, features, ncomp=None, clusterVars=[], nclusters=1, targetVar='TempMax', lag=1, order=0, ranseed=666, verbose=False) apply PCA, compute k-Means clusters and construct regression model for each cluster	
input	stations (set of strings), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD'), features (list of strings), ncomp (list of integers), clusterVars (list of strings), nclusters (integer), targetVar (string), lag (integer), order (integer), ranseed (integer), verbose (boolean)
returns	numpy array [independent variable data], list of floats [dependent (target) variable data], dictionary = {'stations': list of strings, 'startDate': string, 'endDate': string, 'targetVar': string, 'features': list of strings, 'clusterVars': list of 2-tuples, 'clusterParams': dictionary (see wUCluster.clusterFeatureData) 'classes': list of integers, 'regrs': list of LinearRegression objects, 'lag': integer,

	'order': integer, 'transformParams': dictionary (see wUPCA.pcaConvert)}
pcaClusterPredict(modelParams, startDate, endDate, actual=True) make predictions with PCA cluster regression model	
input	modelParams (dictionary; see pcaClusterModel), startDate (string; 'YYYY-MM-DD'), endDate (string; 'YYYY-MM-DD'), actual (boolean)
returns	list of datetime objects, list of floats [predictions for target variable], list of floats [actual target variable data (if available)], numpy array [independent variable data of test set] list of integers [cluster assignments of test data] dictionary = {'R2_mean': float, 'R2_base': float, 'RMSE': float}

Appendix C - Data storage format

Quasi-hourly data stored in JSON format filed by station and month:

Python/JSON_DATA/CODE /CODE_YYYY_MM.json

where CODE is the IATA code of the station, YYYY is the year and MM the month number. Data organized as arrays of arrays of documents


```
[
 [
            {
            "Sea Level PressurehPa": "1018.2",
            "DateUTC": "2005-01-01 05:00:00",
            "TemperatureC": "6.0",
            "TimeEST": "12:00 AM",
            "Gust SpeedKm/h": "37.0",
            "Wind Direction": "WNW",
            "Humidity": "70",
            "Wind SpeedKm/h": "25.9",
            "VisibilityKm": "19.3",
            "Precipitationmm": "N/A",
            "WindDirDegrees": "300",
            "Dew PointC": "1.0",
            "Conditions": "Partly Cloudy",
            "Events": ""
            },
            {
            "Sea Level PressurehPa": "1020",
            "DateUTC": "2005-01-01 06:00:00",
            "TemperatureC": "4",
            "TimeEST": "1:00 AM",
            "Gust SpeedKm/h": "",
            "Wind Direction": "NW",
            "Humidity": "55",
            "Wind SpeedKm/h": "33.3",
            "VisibilityKm": "24",
            "Precipitationmm": "",
            "WindDirDegrees": "310",
            "Dew PointC": "-1",
            "Conditions": "Partly Cloudy",
            "Events": ""
            },
            ...
 ],
…
]
```


Listing 1 - File JSON_DATA/CYYZ/CYYZ_2005_01.json

Daily summaries are stored in comma-separated value (CSV) format filed by station (one file per station for all months):

Python/CSV_DATA/CODE.csv

The data can be loaded directly into R, pandas, etc.

```
date, TimeZone, TempMean, TempMin, TempMinTime, TempMax, TempMaxTime, TotalPrecip, VisibilityMean, PressMean,
PressMin, PressMinTime, PressMax, PressMaxTime, HumidityMean, WindMaxSpd, WindMaxDir, WindMaxTime, WindMeanX,
WindMeanY
2005-01-01, -5, 1.17, -1.0, 2005-01-02 00:00:00, 6.0, 2005-01-01 05:00:00, N/A, 23.9, 1030, 1018.2, 2005-01-01 05:00:00, 1036.9,
2005-01-02 01:00:00, 68, 37.0, 310, 2005-01-01 06:00:00, 8.35, -12.51
2005-01-02, -5, 1.75, -1.0, 2005-01-02 05:00:00, 6.0, 2005-01-03 00:00:00, N/A, 12.7, 1027, 1020.7, 2005-01-02 20:00:00, 1035.6,
2005-01-02 05:00:00, 91, 35.2, 110, 2005-01-02 10:00:00, -9.32, 3.27
2005-01-03, -5, 2.50, 1.0, 2005-01-03 12:00:00, 4.0, 2005-01-03 05:00:00, N/A, 23.1, 1025, 1023.6, 2005-01-04 02:00:00, 1027.5,
2005-01-03 16:00:00, 88, 20.4, 310, 2005-01-03 05:00:00, 1.51, -5.09
2005-01-04, -5, 0.12, -1.0, 2005-01-05 03:00:00, 1.0, 2005-01-04 08:00:00, N/A, 20.7, 1026, 1023.8, 2005-01-04 06:00:00, 1028.3,
2005-01-04 16:00:00, 80, 24.1, 360, 2005-01-04 12:00:00, 4.96, -11.05
2005-01-05, -5, -3.71, -6.0, 2005-01-05 21:00:00, -1.0, 2005-01-05 11:00:00, N/A, 23.3, 1025, 1021.9, 2005-01-06 04:00:00, 1027.7,
2005-01-05 16:00:00, 70, 25.9, 30, 2005-01-05 12:00:00, -5.11, -15.24
2005-01-06, -5, -5.00, -8.0, 2005-01-06 11:00:00, -1.0, 2005-01-07 02:00:00, N/A, 15.3, 1010, 1001.8, 2005-01-06 19:00:00, 1021.0,
2005-01-06 05:00:00, 78, 51.9, 270, 2005-01-07 02:31:00, -2.49, -4.29
...
```

Listing 2 - File CSV_DATA/CYYZ.csv

Airport station data is stored in the single CSV file AIRPORT_DATA/airports.csv, obtained from http://ourairports.com/data/ .

```
"id","ident","type","name","latitude_deg","longitude_deg","elevation_ft","continent","iso_country","iso_region","municipality","schedule
d_service","gps_code","iata_code","local_code","home_link","wikipedia_link","keywords"
6523,"00A","heliport","Total Rf
Heliport",40.07080078125,-74.93360137939453,11,"NA","US","US-PA","Bensalem","no","00A",,"00A",,,,
6524,"00AK","small_airport","Lowell Field",59.94919968,-151.695999146,450,"NA","US","US-AK","Anchor
Point","no","00AK",,"00AK",,,,
6525,"00AL","small_airport","Epps
Airpark",34.86479949951172,-86.77030181884766,820,"NA","US","US-AL","Harvest","no","00AL",,"00AL",,,,
…
```

Listing 3 - File Python/AIRPORT_DATA/airports.csv

Acknowledgements

Thanks to Bora Çağlayan, Carl Barrelet, and Cesar Osorio for constructive discussions during the preparation of this project and to Shannon Dyck for setting up and making available the computer lab in the Victoria Building.

References

Ahrens, C. D. (2005): *Essentials of Meteorology: An Invitation to the Atmosphere* (Belmont, CA, USA: Thomson), 473pp.

Clifton, A. and J. K. Lundquist (2012): Data Clustering Reveals Climate Impacts on Local Wind Phenomena. *Journal of Applied Meteorology and Climatology*, **51**, 1547–1557.

Göçken, M., A. Boru, A. T. Dosdoğru and N. Berber (2015): Application of Soft Computing Models to Daily Average Temperature Analysis. *International Journal of Engineering Technologies*, **1**, 56-64.

Gordon, A., W. Grace, P. Schwerdtfeger and R. Byron-Scott (1998): *Dynamic Meteorology: A Basic Course* (New York, USA: Arnold), 325pp.

Klausmann, T (2015): *PyMETAR*, version 0.20, http://www.schwarzvogel.de/software/pymetar.html.

Lynch, P. (2008): The origins of computer weather prediction and climate modeling. *Journal of Computational Physics*, **227**, 3431–3444

Mitsa, T. (2015): “An Example of R Logistic Regression for Weather Prediction (corrected)”, Posted July 22, 2015, http://www.theophanomitsa.com/blog/an-example-of-r-logistic-regression-for-weather-prediction/

Paras and S. Mathur (2012): A Simple Weather Forecasting Model Using Mathematical Regression. *Indian Research Journal of Extension Education Special Issue*, **1**, 161-168.

Sawale, G. J. and Gupta, S. R. (2013): Use of Artificial Neural Network in Data Mining For Weather Forecasting. *International Journal Of Computer Science And Applications*, **2**, 383-387.

Shonk, J. (2013): *Introducing Meteorology: A Guide to Weather* (Edinburgh, UK: Dunedin), 150pp.

Silver, N. (2012): *The Signal and the Noise* (New York, USA: Penguin), 534pp.

Tektas, M. (2010): Weather Forecasting Using ANFIS and ARIMA MODELS. A Case Study for Istanbul. *Environmental Research, Engineering and Management*, **51**, 5-10.

U.S. Dept. of Commerce and National Oceanic and Atmospheric Administration (2005): *Federal Meteorological Handbook No. 1 Surface Weather Observations and Reports* (Washington D.C.), 104pp.