» Feature Engineering For Time Series

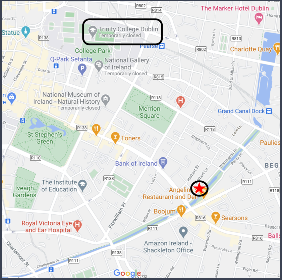
- A *series* is an ordered sequence of values e.g. words in a sentence, daily weather, daily stock prices
- A *time series* is a series where each term has a timestamp.
- Note that real measurements always have a timestamp, even if we sometimes choose to ignore it because it seems irrelevant.
- As usual, our task is prediction.

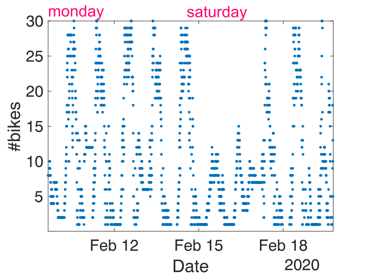
» Feature Engineering For Time Series

* Predicting real-valued quantities (regression) e.g.
  * Energy demand, stock prices, time before battery in electric car discharged
* Predicting discrete-valued quantities (classification). Distinguish two cases:
  1. Underlying task is to predict a real-valued quantity. Then map to discrete quantity using e.g. sign() function.
     * Classification
     * Anomaly detection → calculate difference between predicted and observed outputs and flag anomaly if greater than some threshold.
  2. Genuinely discrete task e.g.
     * Predictive text - given sequence of words/characters predict next one
     * Chatbot - given sequence of sentences, predict what to say next
     * Machine translation - given sequence of text in English predict sequence in French
     * We'll avoid these. A key problem is forming a useful measure is distance between values. E.g. if predict next word to be *great* or *good* they're both much the same so error is small but if *predict* purple then that's a lot worse. Word2Vec etc try to address this.

» Example: Dublin Bikes

Dublin bikes data[1]: #bikes available at Herbert Place bike stand vs time in Feb 2020. We'll use this as a running example.





- This bike stand is on a commuter route, its regularly full and empty during week days
- Observe the regular pattern on weekdays. Also a regular difference between weekdays and weekends → *seasonality*
- Observe short-term correlation i.e. if #bikes is high at time k then its also likely to be high at nearby times → *trends*
- *Its this structure in the data that lets us make predictions*

[1] https://data.gov.ie/dataset/dublinbikes-api

» Time Series Features

* Bike time series is a sequence of pairs (timestamp, #bikes) i.e.

$$(t^{(k)}, y^{(k)}),\ k = 1, 2, \dots$$

  where $t^{(k)}$ is the time of the k'th measurement and $y^{(k)}$ is the k measurement of #bikes
* Measurements are taken at regular intervals (every 5 mins) i.e. the *sampling interval* is constant
  * For k'th measurement $t^{(k)} = k \times 5$mins
  * So $t^{(k)}$ value is redundant, can simplify our data to the sequence of individual values:

$$y^{(k)},\ k = 1, 2, \dots$$

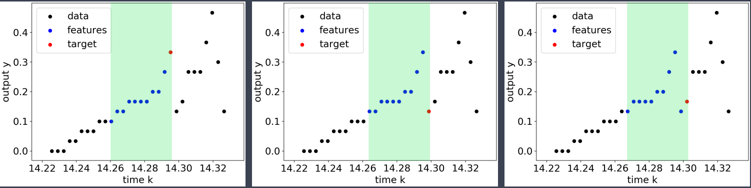
* Notice that we don't have an input and output, just a sequence of $y^{(k)}$ values, so how to map this to our machine learning framework?

» Time Series Features

- *One-step ahead prediction*. Given data up to time $k-1$ our task is to predict $y^{(k)}$.
  - Input is $[y^{(1)}, y^{(1)}, \ldots, y^{(k-1)}]$
  - Output is prediction for $y^{(k)}$
- *q-step ahead prediction*. Given data up to time $k-1$ our task is to predict $y^{(k-1+q)}$.
  - Input is $[y^{(1)}, y^{(2)}, \ldots, y^{(k-1)}]$ (note that we don't know $y^{(k)}$, $y^{(k+1)}, \ldots, y^{(k-2+q)}$)
  - Output is prediction for $y^{(k-1+q)}$ (and perhaps also predictions for $y^{(k)}$, $y^{(k+1)}, \ldots, y^{(k-2+q)}$)
- When k is large the input $y^{(1)}, y^{(2)}, \ldots, y^{(k-1)}$ is large too. So we usually truncate this to just the last n values use:
  - Input is $[y^{(k-1)}, y^{(k-n-1)}, \ldots, y^{(k-1)}]$
  - E.g. for $n = 2$ then:
    at time $k = 100$ the input is $[y^{(98)}, y^{(99)}]$
    at time $k = 101$ the input is $[y^{(99)}, y^{(100)}]$
    at time $k = 102$ the input is $[y^{(100)}, y^{(101)}]$
  - Observe how input progressively slides through the sequence of data $\ldots, y^{(98)}, y^{(99)}, y^{(100)}, y^{(101)}, \ldots$

» Time Series Features

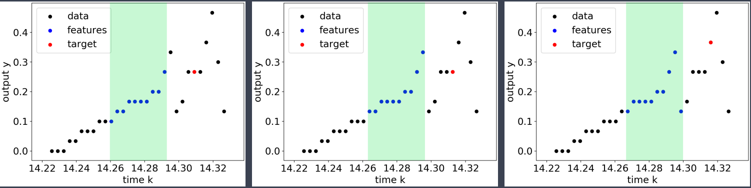
Example Dublin bikes training data for one-step ahead prediction:



- Target value $y^{(k)}$ (to be predicted) is marked in red. Values forming feature vector $x^{(k)}$ are marked in blue (and highlighted by shaded green area).
- Points forming $x^{(k)}, x^{(k+1)}, x^{(k+2)}, \dots$ can be thought of as a window that "slides" through the time series data.
- Correlations in time are an intrinsic aspect:
  - Structure (seasonality, trends) means that $y^{(k)}$ values are correlated i.e the elements $x_1^{(k)}, x_2^{(k)}, \dots$ of feature vector $x^{(k)}$ are correlated $\rightarrow$ *subsample/thin them out?*
  - Feature vectors at nearby times, e.g. $x^{(k)}$ and $x^{(k+1)}$, contain overlapping data so are not independent, our predictions $\hat{y}(x^{(k)})$ and $\hat{y}(x^{(k+1)})$ are therefore also not independent $\rightarrow$ *need to be careful when training model and evaluating performance*

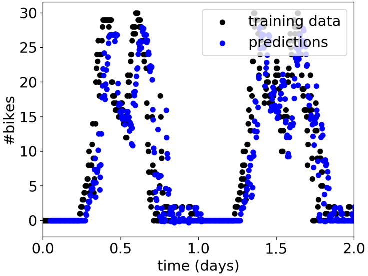
» Time Series Features

Example Dublin bikes training data for 5-step ahead prediction (i.e. 25 minutes ahead):



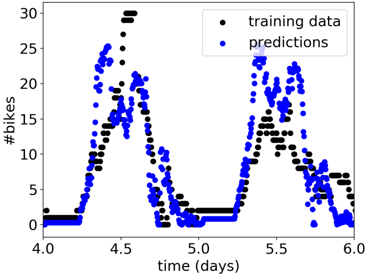
* As we predict further into the future, generally we can expect predictions to become less accurate
* For bike data we want to predict #bikes at station at start of journey, so perhaps 30 mins to 1 hour ahead.
    * Measurements are every 5 mins, so we're predicting between 6 and 12 points ahead.
* Do we need to predict the intermediate points i.e 1-step ahead, 2-step ahead, 3-step ahead etc?

» Using Trends For Prediction



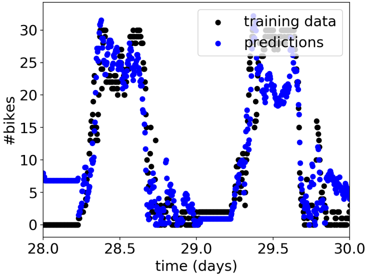
- $q = 10$-step ahead prediction (i.e. 50 minutes ahead). Feature vector is most recent three values $[y^{(k-3-q)}, y^{(k-2-q)}, y^{(k-1-q)}]$ $\rightarrow$ we're using short-term trends in data to make predictions
- Linear regression: $\hat{y} = \theta^T X$, mean square cost function
- $\theta = [0.14343904, -0.14428269, 0.92579867]$
  - Most of the weight is placed on the third element $y^{(k-1-q)}$ of X i.e. most recent observation
  - Model basically predicts #bikes 10 steps ahead to be same as current #bikes.
  - Tweaking features, using kNN model don't change things much

» Using Seasonal Behaviour For Prediction: Daily



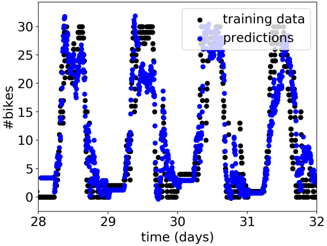
* $q = d$-step ahead prediction. Feature vector is values $[y^{(k-3d)}, y^{(k-2d)}, y^{(k-d)}]$ where $d = 288$ is number of measurements in 24 hours
  $\rightarrow$ we're using daily pattern in data to make one-day ahead predictions
* Linear regression: $\hat{y} = \theta^T X$, mean square cost function
* $\theta = [0.36519779, 0.18017767, 0.31751999]$
  * More weight applied to previous day and three days ago than to two days ago
  * Observe we don't have the shift between predictions and data that we saw using trend

» Using Seasonal Behaviour For Prediction: Weekly



- $q = w$-step ahead prediction. Feature vector is values $[y^{(k-3w)}, y^{(k-2w)}, y^{(k-w)}]$ where $w = 7 \times 288$ is number of measurements in 7 days
  $\rightarrow$ we're using weekly pattern in data to make one-week ahead predictions
- Linear regression: $\hat{y} = \theta^T X$, mean square cost function
- $\theta = [0.16243974, 0.56893161, 0.36624999]$
  - More weight applied to last two weeks, less to three weeks ago
  - Again we don't have the shift between predictions and data that we saw using trend

» Putting it together



* $q = 10$-step ahead prediction (i.e. 50 minutes ahead). Feature vector is values

$$[y^{(k-3w)}, y^{(k-2w)}, y^{(k-w)}, y^{(k-3d)}, y^{(k-2d)}, y^{(k-d)}, y^{(k-3-q)}, y^{(k-2-q)}, y^{(k-1-q)}]$$

$\rightarrow$ we're using weekly and daily patterns in data plus short-term trend to make predictions of $y^{(k+q)}$

* Linear regression: $\hat{y} = \theta^T X$, mean square cost function
* $\theta = [-0.019, 0.269, 0.216, -0.019, 0.484, 0.131, 0.029, -0.056, 0.0514]$
  * Most important terms are data two days ago and last two weeks
  * Should probably have distinguished weekends from weekdays

» Some Practicalities

- *Cross-validation*. Can (and should) use cross-validation as usual to select model hyperparameters
- *Evaluation*: When evaluating predictions need to test at points which are far enough apart that they are not too correlated (if test at two neighbouring points then will be over-optimistic about performance)
- *Training*: Feature vectors at nearby times can have many overlapping elements (due to sliding window) $\rightarrow$ when training it can be useful to use training points that are a bit apart so that overlap is reduced
- *Feature selection*. Elements within the same feature vector can be highly correlated e.g. due to trends in data. Makes numerics of optimisation harder. Can be helpful to:
  - Thin out features to reduce correlation e.g. use $[y^{(k)}, y^{(k-3)}, y^{(k-6)}, \dots]$ rather than $[y^{(k)}, y^{(k-1)}, y^{(k-2)}, \dots]$. In general, spacing between features should reflect the timescales of any structure present in time series e.g. daily, weekly, trends over a few minutes $\rightarrow$ we did this in bike example.
  - Use regularisation e.g. ridge regression has better numerics

» Python Code For Bike Example

```
import pandas as pd
import numpy as np
import math, sys
import matplotlib.pyplot as plt
plt.rc('font', size=18); plt.rcParams['figure.constrained_layout.use'] = True

# read data. column 1 is date/time, col 6 is #bikes
df = pd.read_csv("herbert.csv", usecols = [1,6], parse_dates=[1])
#print(df.head())

# 3rd Feb 2020 is a monday, 10th is following monday
start=pd.to_datetime("04-02-2020",format='%d-%m-%Y')
end=pd.to_datetime("14-03-2020",format='%d-%m-%Y')

# convert date/time to unix timestamp in sec
t_full=pd.array(pd.DatetimeIndex(df.iloc[:,0]).astype(np.int64))/1000000000
dt = t_full[1]-t_full[0]
print("data sampling interval is %d secs"%dt)

# extract data between start and end dates
t_start = pd.DatetimeIndex([start]).astype(np.int64)/1000000000
t_end = pd.DatetimeIndex([end]).astype(np.int64)/1000000000
t = np.extract([(t_full>=t_start) & (t_full<=t_end)], t_full)
t=(t-t[0])/60/60/24 # convert timestamp to days
y = np.extract([(t_full>=t_start) & (t_full<=t_end)], df.iloc[:,1]).astype(np.int64)
#plot extracted data
#plt.scatter(t,y, color='red', marker='.'); plt.show()
```

» Python Code For Bike Example (cont)

```
def test_preds(q,dd,lag,plot):
    #q−step ahead prediction
    stride=1
    XX=y[0:y.size−q−lag∗dd:stride]
    for i in range(1,lag):
        X=y[i∗dd:y.size−q−(lag−i)∗dd:stride]
        XX=np.column_stack((XX,X))
    yy=y[lag∗dd+q::stride]; tt=t[lag∗dd+q::stride]
    from sklearn.model_selection import train_test_split
    train, test = train_test_split(np.arange(0,yy.size),test_size=0.2)

    from sklearn.linear_model import Ridge
    model = Ridge(fit_intercept=False).fit(XX[train], yy[train])
    print(model.intercept_, model.coef_)
    if plot:
        y_pred = model.predict(XX)
        plt.scatter(t, y, color='black'); plt.scatter(tt, y_pred, color='blue')
        plt.xlabel("time (days)"); plt.ylabel("#bikes")
        plt.legend(["training data","predictions"],loc='upper right')
        day=math.floor(24∗60∗60/dt) # number of samples per day
        plt.xlim(((lag∗dd+q)/day,(lag∗dd+q)/day+2))
        plt.show()

# prediction using short−term trend
plot=True
test_preds(q=10,dd=1,lag=3,plot=plot)

# prediction using daily seasonality
d=math.floor(24∗60∗60/dt) # number of samples per day
test_preds(q=d,dd=d,lag=3,plot=plot)

# prediction using weekly seasonality
w=math.floor(7∗24∗60∗60/dt) # number of samples per day
test_preds(q=w,dd=w,lag=3,plot=plot)
```

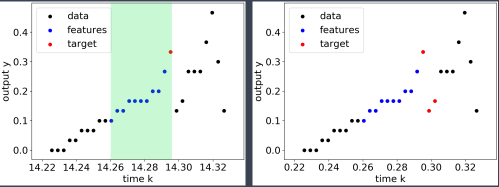
» Python Code For Bike Example (cont)

```
#putting it together
q=10
lag=3; stride=1
w=math.floor(7*24*60*60/dt) # number of samples per week
len = y.size-w-lag*w-q
XX=y[q:q+len:stride]
for i in range(1,lag):
        X=y[i*w+q:i*w+q+len:stride]
        XX=np.column_stack((XX,X))
d=math.floor(24*60*60/dt) # number of samples per day
for i in range(0,lag):
        X=y[i*d+q:i*d+q+len:stride]
        XX=np.column_stack((XX,X))
for i in range(0,lag):
        X=y[i:i+len:stride]
        XX=np.column_stack((XX,X))
yy=y[lag*w+w+q:lag*w+w+q+len:stride]
tt=t[lag*w+w+q:lag*w+w+q+len:stride]

from sklearn.model_selection import train_test_split
train, test = train_test_split(np.arange(0,yy.size),test_size=0.2)
#train = np.arange(0,yy.size)
from sklearn.linear_model import Ridge
model = Ridge(fit_intercept=False).fit(XX[train], yy[train])
print(model.intercept_, model.coef_)

if plot:
        y_pred = model.predict(XX)
        plt.scatter(t, y, color='black'); plt.scatter(tt, y_pred, color='blue')
        plt.xlabel("time (days)"); plt.ylabel("#bikes")
        plt.legend(["training data","predictions"],loc='upper right')
        day=math.floor(24*60*60/dt) # number of samples per day
        plt.xlim((4*7,4*7+4))
        plt.show()
```

» Multi-step prediction



* For prediction of $y^{(k)}$ feature vector is

$$x^{(k)} = [y^{(k-1)}, y^{(k-2)}, y^{(k-3)}, \dots]$$

* Suppose now also want to predict $y^{(k+1)}$
  * Feature vector we'd like is $x^{(k+1)} = [y^{(k)}, y^{(k-1)}, y^{(k-2)}, \dots]$ but we don't know $y^{(k)}$
  * Could build new model using feature vector $[y^{(k-1)}, y^{(k-2)}, \dots]$, but then have a separate model for every prediction
  * Altenative is to substitute in our prediction $\hat{y}(x^{(k)})$ for $y^{(k)}$ i.e. use feature vector

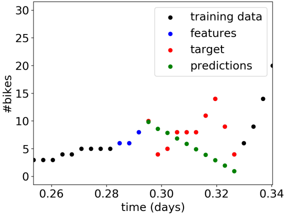
$$x^{(k+1)} = [\hat{y}(x^{(k)}), y^{(k-1)}, y^{(k-2)}, \dots]$$

  * And can repeat for $y^{(k+2)}$ etc i.e. use

$$x^{(k+1)} = [\hat{y}(x^{(k+1)}), \hat{y}(x^{(k)}), y^{(k-2)}, \dots]$$

and so on

» Multi-step prediction



Green dots show multi-step predictions, note decay over time

* Our prediction at step $k+q$ now depends on our predictions at steps k, $k+1$, $k+2, \dots, k+q-1$ ... we *feedback* back the model outputs to create predictions
* Contrast with *feedforward* models we've always used up until now (prediction is purely a function of input x)
* But need to be careful as feedback of prediction errors will cause build up of errors over time ... as we look further ahead we expect predictions to become less accurate
* Feedback creates *dynamics* $\rightarrow$ for another module
* Can train feedforward model and then use as a feedback model (as we did here), but usually better to take account of feedback when training model $\rightarrow$ for another module

» Summary

* Time series are ubiquitous $\rightarrow$ most data has a timestamp and ordering, even if we sometimes choose to ignore it.
* With time series we use the past data to form the feature vector
* Usually want feature vector to span multiple time scales, e.g. minutes, days, weeks $\rightarrow$ feature selection involves choosing which past data points to include
* *Feedforward models* (i.e. what we've looked at so far) are often fine for single predictions, all of our usual ideas and techniques can be applied
* *Feedback models*, where feature vector contains previous predictions, become more important for multi-step prediction. When feedback is used with a neural net model its called a *recurrent* neural net e.g. LSTM is a recurrent neural net that's currently popular for use with time-series data.