Projects

3-4 person groups preferred

Deliverables: Poster & Report & main code (plus proposal, midterm slide)

Topics your own or chose form suggested topics. Some **physics/engineering inspired**.

April 26 groups due to TA (if you don't have a group, ask in piazza we can help). TAs will construct groups after that.

May 5 proposal due. TAs and Peter can approve.

Proposal: One page: Title, a large paragraph, data, weblinks, references.

May 20 Midterm slide presentation. Presented to a subgroup of class.

June 5 final poster. Uploaded June 3

Report and code due **Saturday 15 June.**

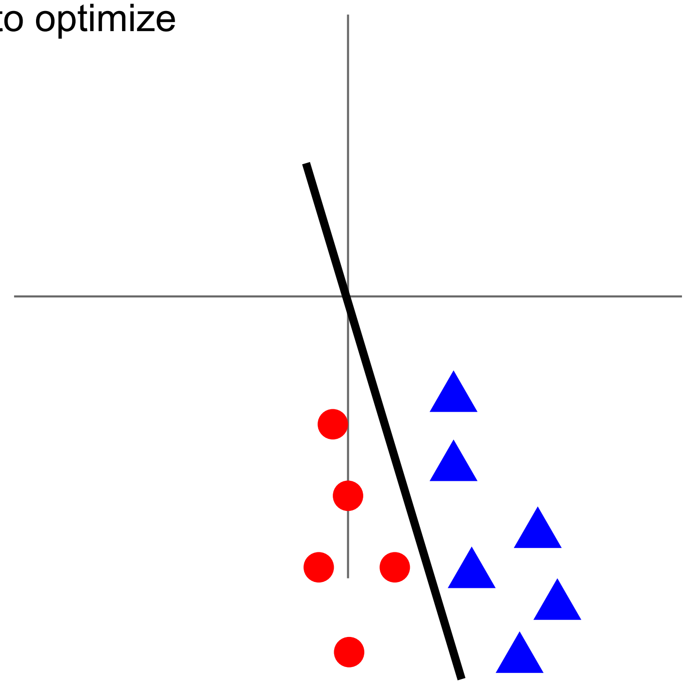
Q: Can the final project be shared with another class?

If the other class allows it it should be fine. You cannot turn in an identical project for both classes, but you can share common infrastructure/code base/datasets across the two classes.

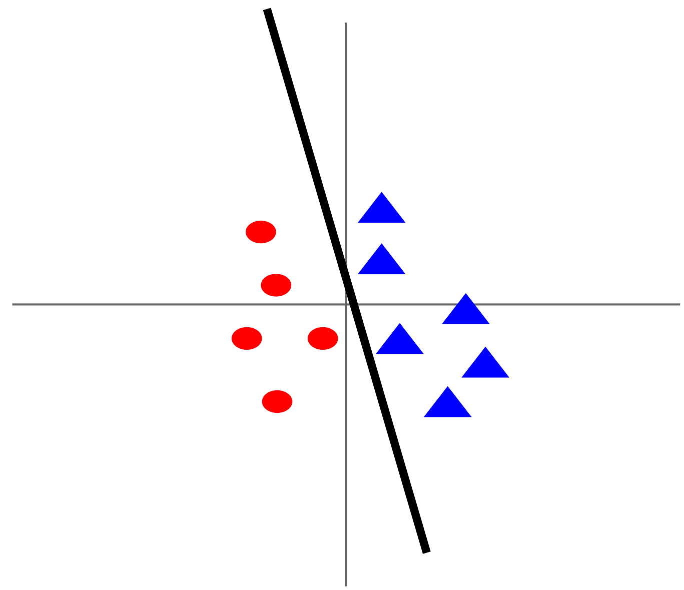
No cut and paste from other sources without making clear that this part is a copy. This applies to other reports or things from internet. Citations are important.

Last time: Data Preprocessing

Before normalization: classification loss very sensitive to changes in weight matrix; hard to optimize

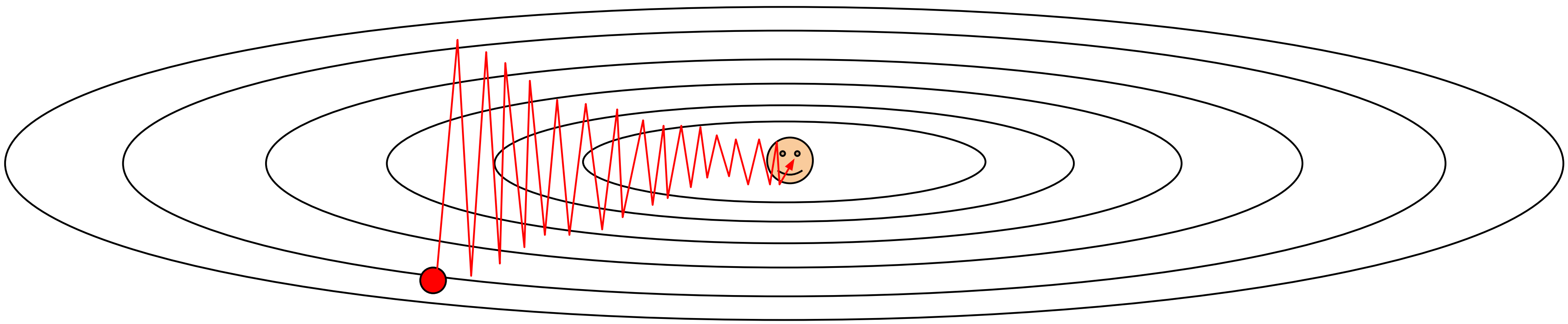


After normalization: less sensitive to small changes in weights; easier to optimize



Optimization: Problems with SGD

What if loss changes quickly in one direction and slowly in another?
What does gradient descent do?
Very slow progress along shallow dimension, jitter along steep direction

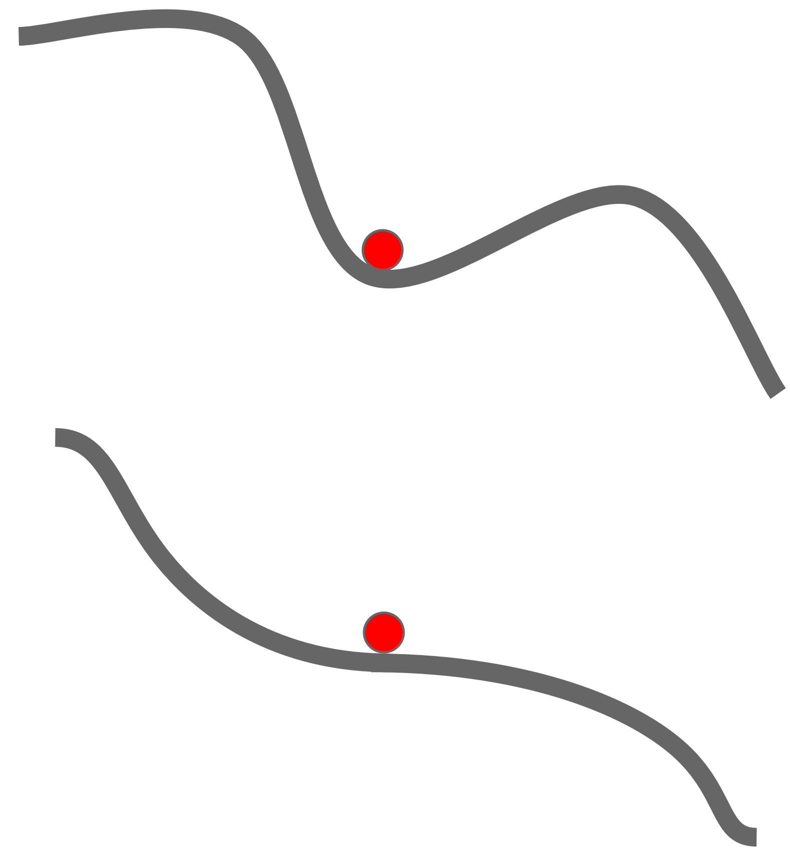


Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large

Optimization: Problems with SGD

What if the loss function has a **local minima** or **saddle point**?

Zero gradient, gradient descent gets stuck



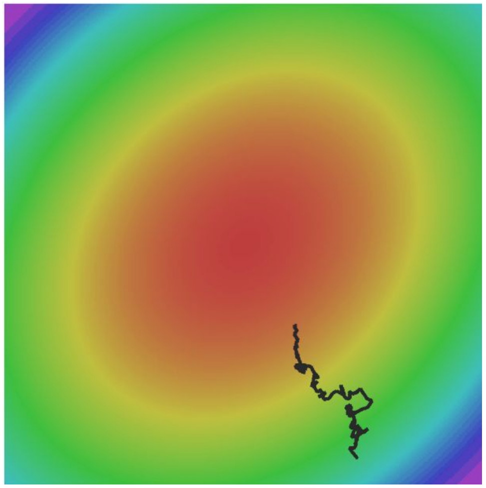
Saddle points much more common in high dimension

Optimization: Problems with SGD

Our gradients come from minibatches so they can be noisy!

$$L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(x_i, y_i, W)$$

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W)$$



SGD + Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
  dx = compute_gradient(x)
  x += learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x += learning_rate * vx
```

- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically rho=0.9 or 0.99

Adam (full form)

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

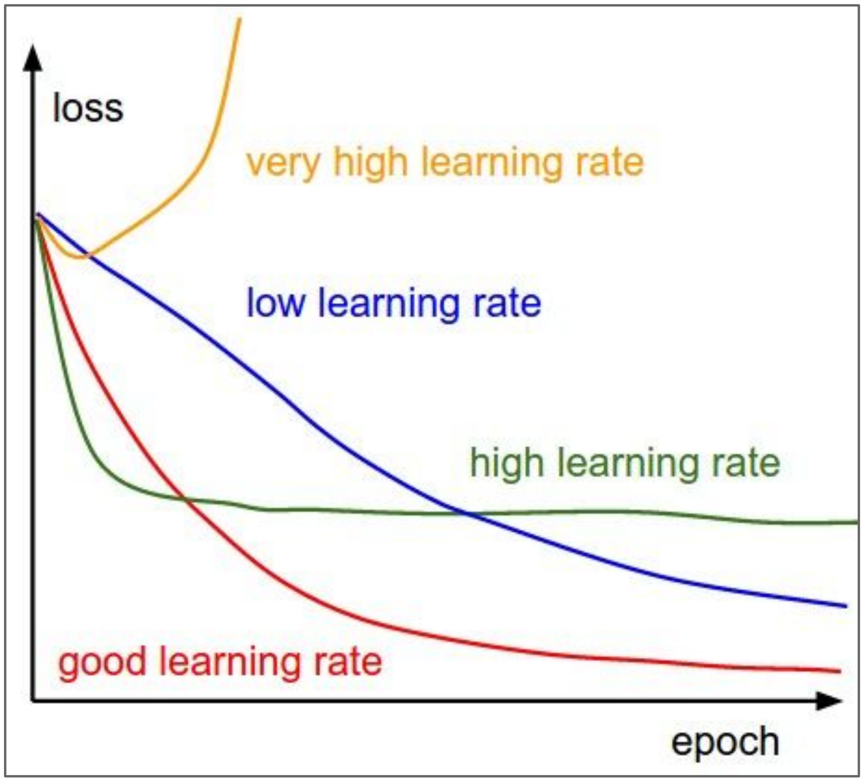
Bias correction

AdaGrad / RMSProp

Bias correction for the fact that first and second moment estimates start at zero

Adam with beta1 = 0.9, beta2 = 0.999, and learning_rate = 1e-3 or 5e-4 is a great starting point for many models!

SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter.



=> Learning rate decay over time!

step decay:
e.g. decay learning rate by half every few epochs.

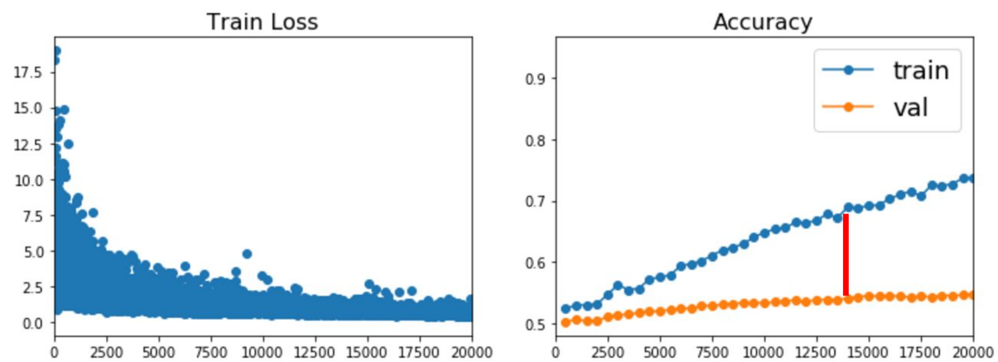
exponential decay:

$$\alpha = \alpha_0 e^{-kt}$$

1/t decay:

$$\alpha = \alpha_0 / (1 + kt)$$

How to improve single-model performance?



Regularization

Regularization: Add term to loss

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i;W)_j - f(x_i;W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

In common use:

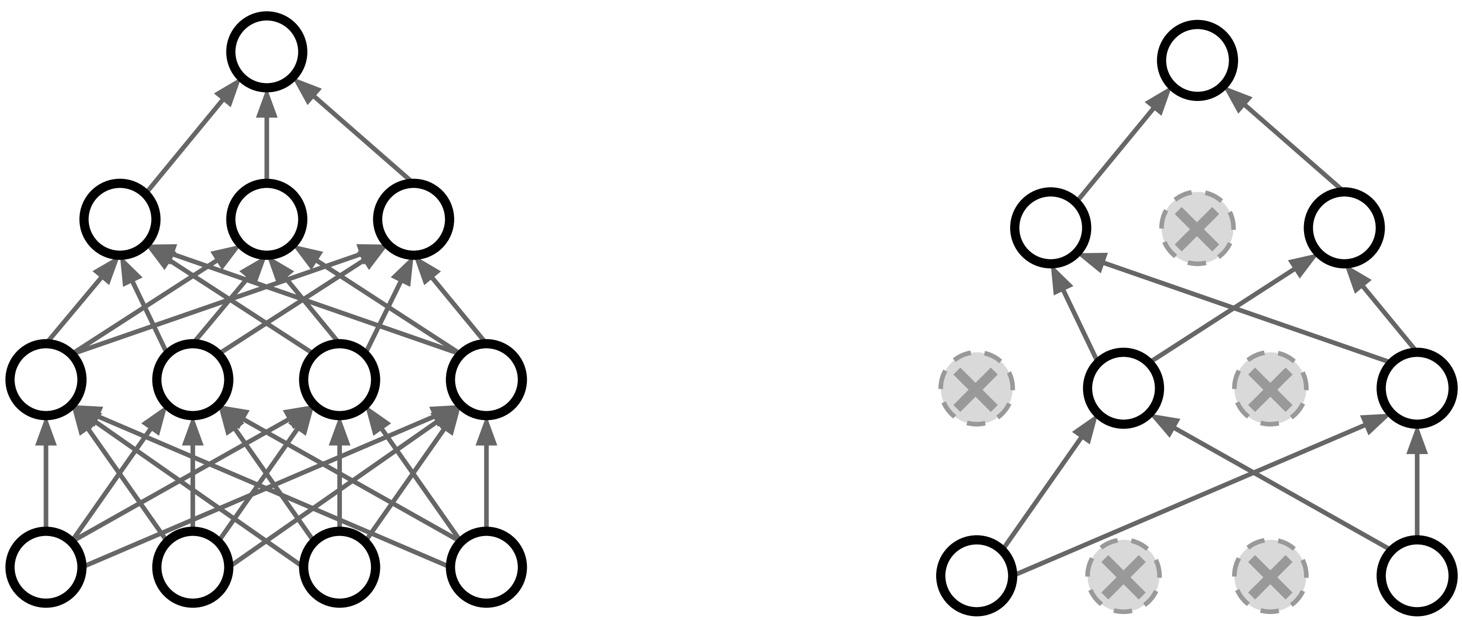
L2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$ (Weight decay)

L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Regularization: Dropout

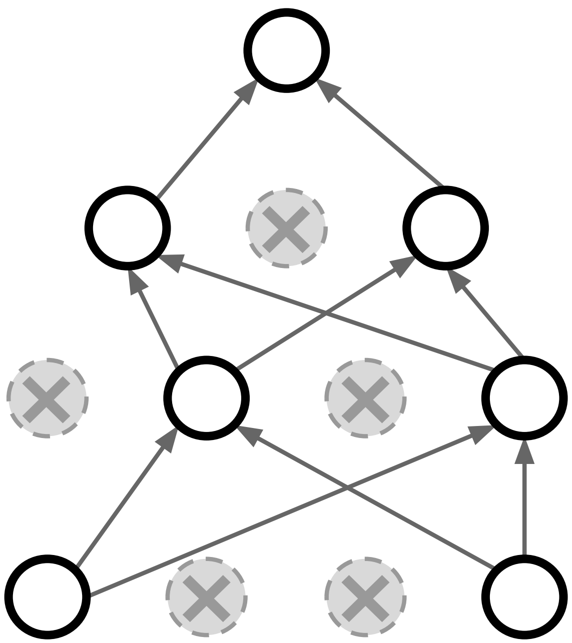
In each forward pass, randomly set some neurons to zero
Probability of dropping is a hyperparameter; 0.5 is common



Homework

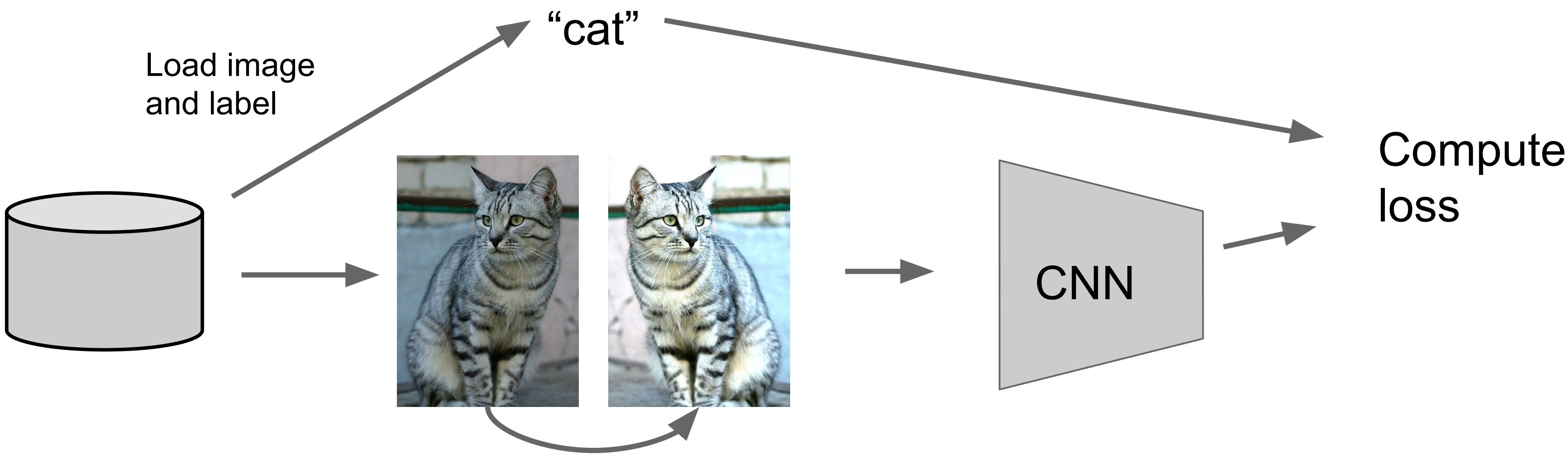
Regularization: Dropout

How can this possibly be a good idea?



Forces the network to have a redundant representation;
Prevents co-adaptation of features

Regularization: Data Augmentation



Data Augmentation

Get creative for your problem!

Random mix/combinations of :
- translation
- rotation
- stretching
- shearing,
- lens distortions, … (go crazy)

+simulated data using physical model.

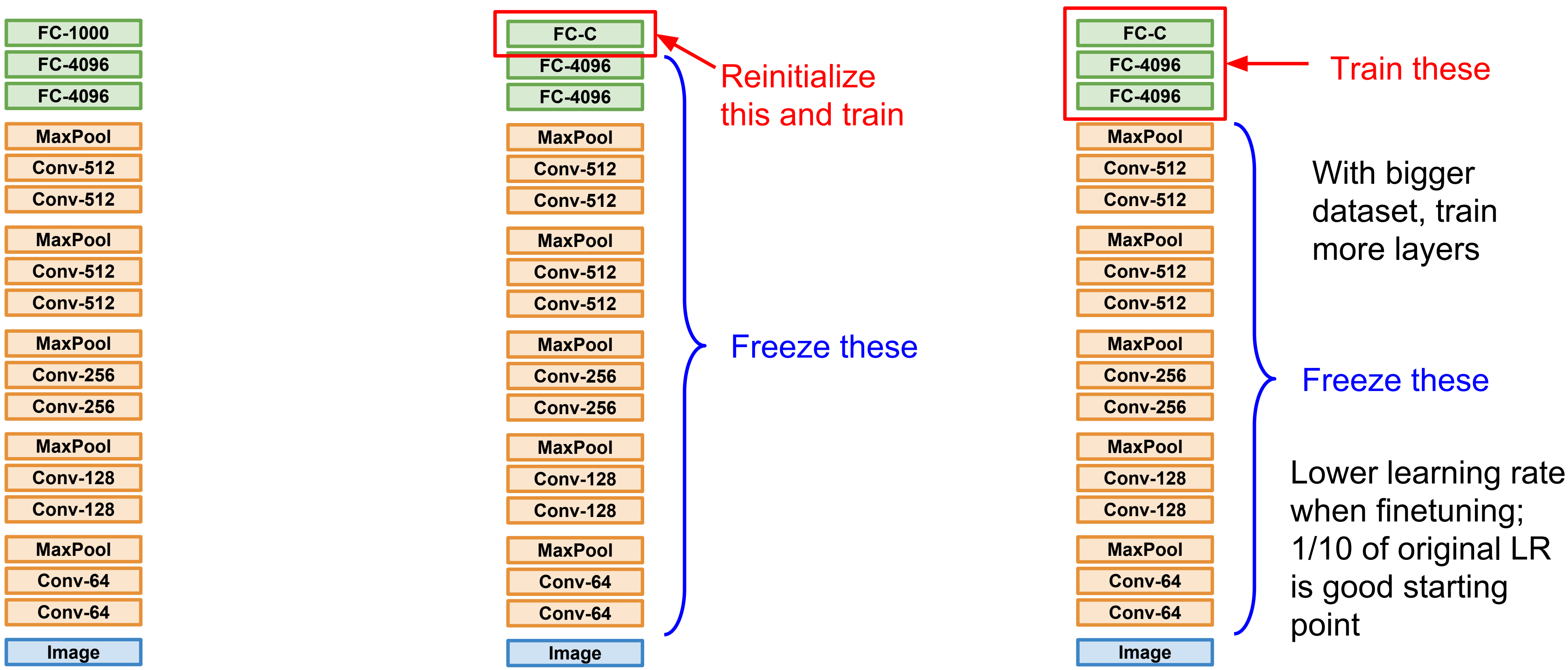
Transfer Learning with CNNs

Donahue et al, “DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition”, ICML 2014
Razavian et al, “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition”, CVPR Workshops 2014

1. Train on Imagenet

2. Small Dataset (C classes)

3. Bigger dataset



Predicting Weather with Machine Learning:

Intro to ARMA and Random Forest

Emma Ozanich
PhD Candidate,
Scripps Institution of Oceanography

Background

Shi et al NIPS 2015 –

- Predicting rain at different time lags
- Shows convolutional lstm vs nowcast models vs fully-connected lstm
- Used radar echo (image) inputs
  - Hong Kong, 2011-2013,
  - 240 frames/day
  - Selected top 97 rainy days
  - Note: <10% of data used!
- Preprocessing: k-means clustering to denoise
- ConvLSTM has better performance *and* lower false alarm (lower left)

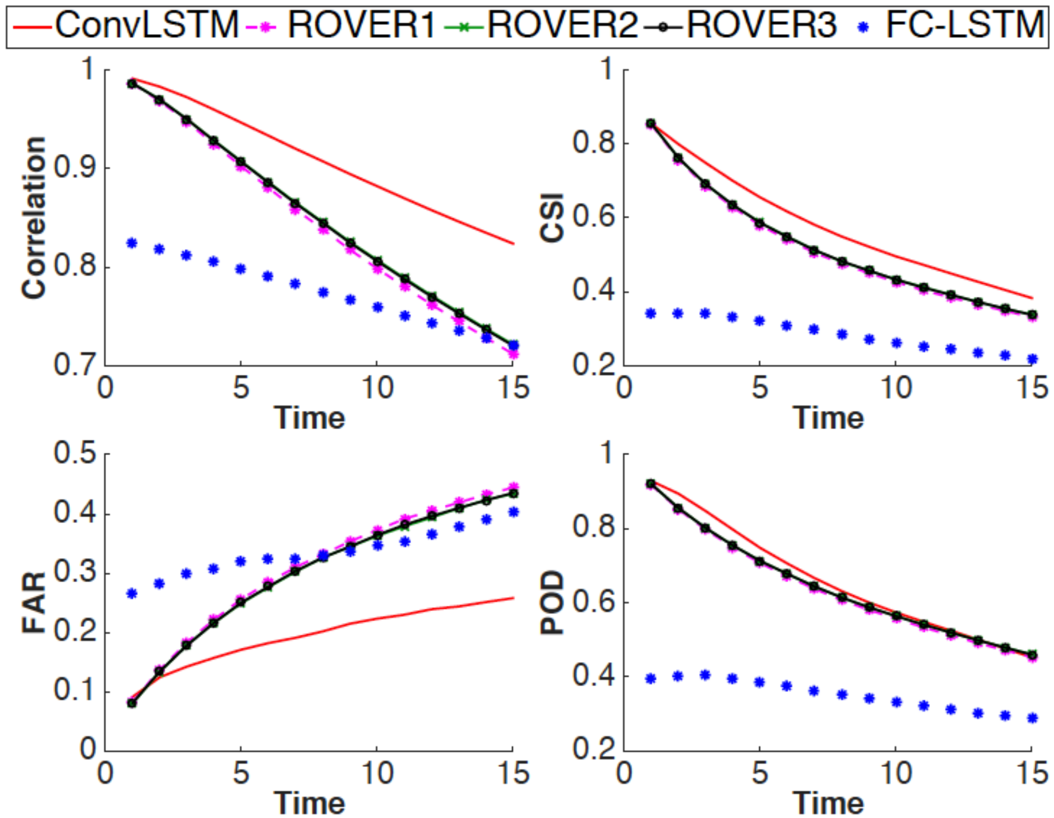


Figure 1: (**Larger Version**) Comparison of different models based on four precipitation nowcasting metrics over time.

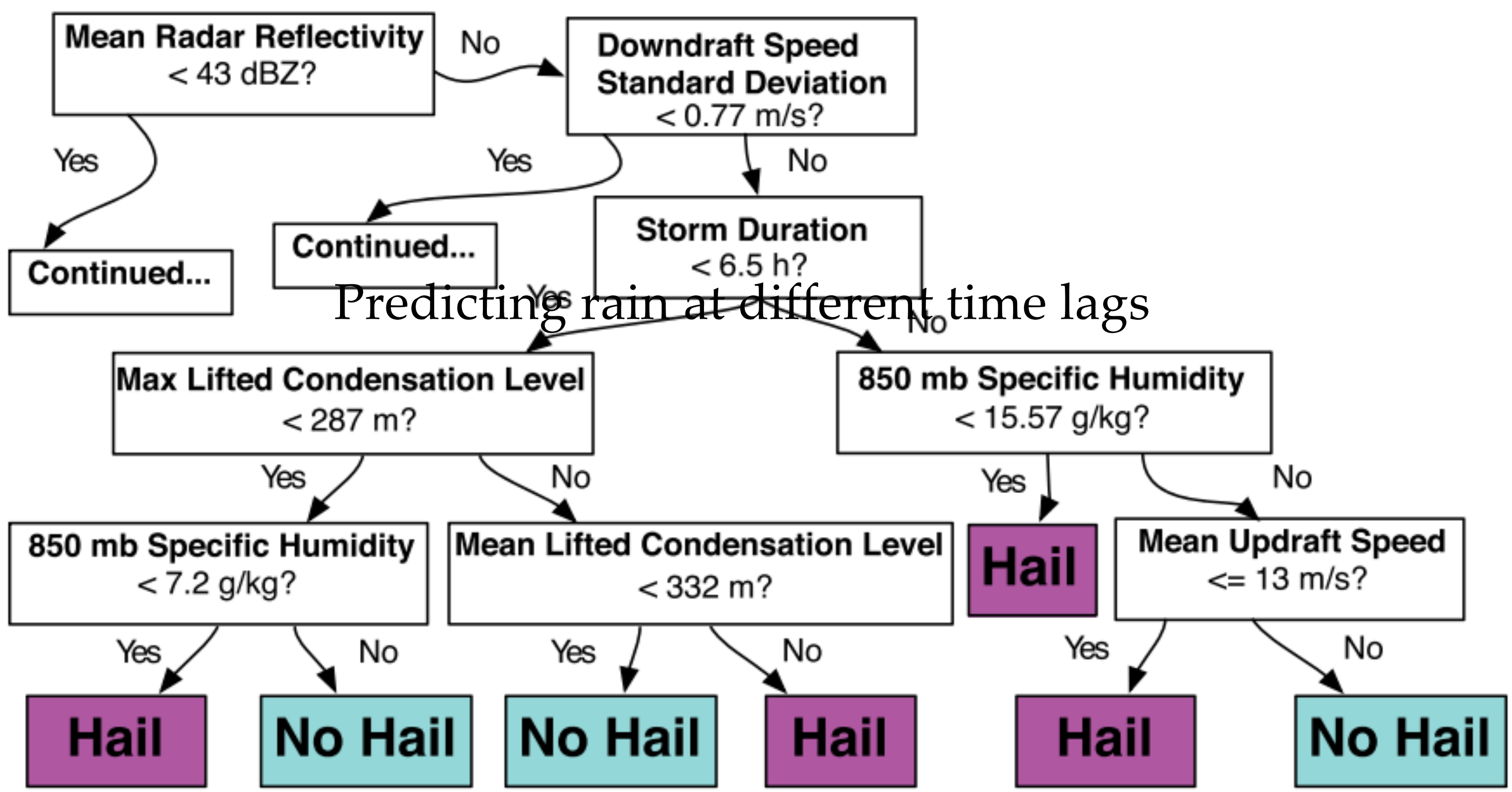
CSI: hits/(hits+misses+false) FAR: false/(hits+false) POD: hits/(hits+misses)

false = false alarm

Background

McGovern et al 2017 BAM –

- Decision trees used in meteorology since mid-1960s



Predicting rain at different time lags

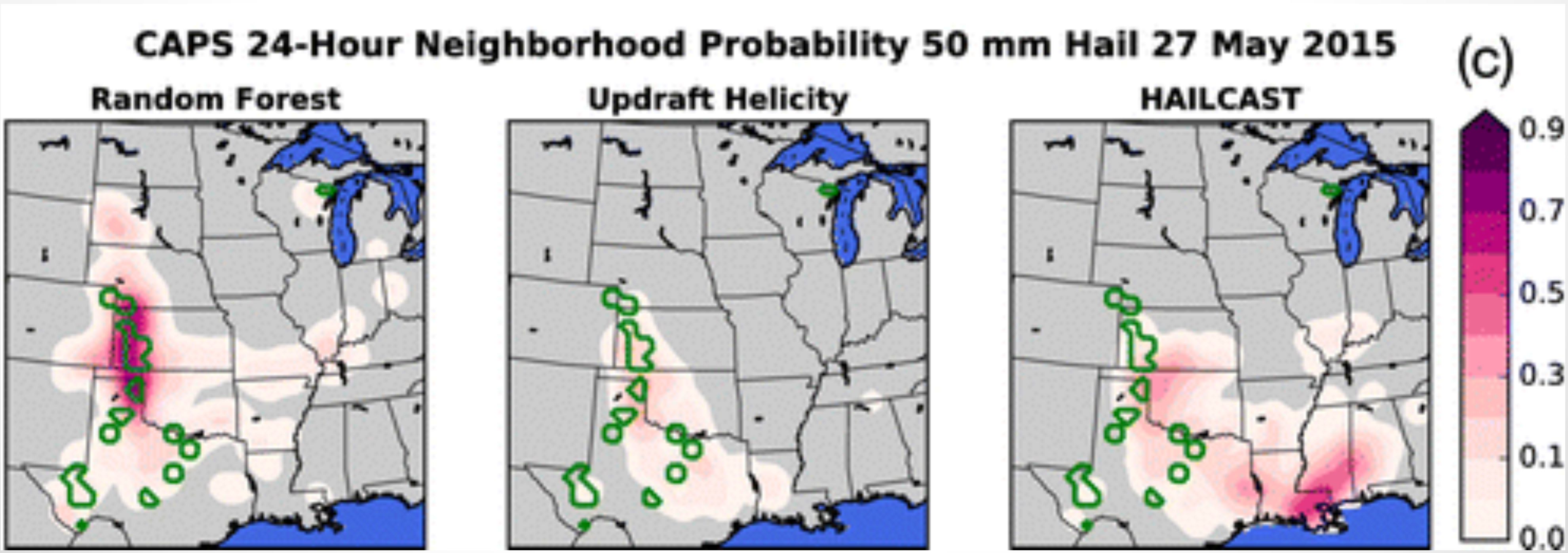
FIG. 1. An example of a decision tree for predicting if hail will occur. A version of this decision tree first appeared in Gagne (2016).

McGovern et al 2017, Bull. Amer. Meteor. Soc. 98:10, p. 2073-2090.

Background

McGovern et al 2017 BAM –

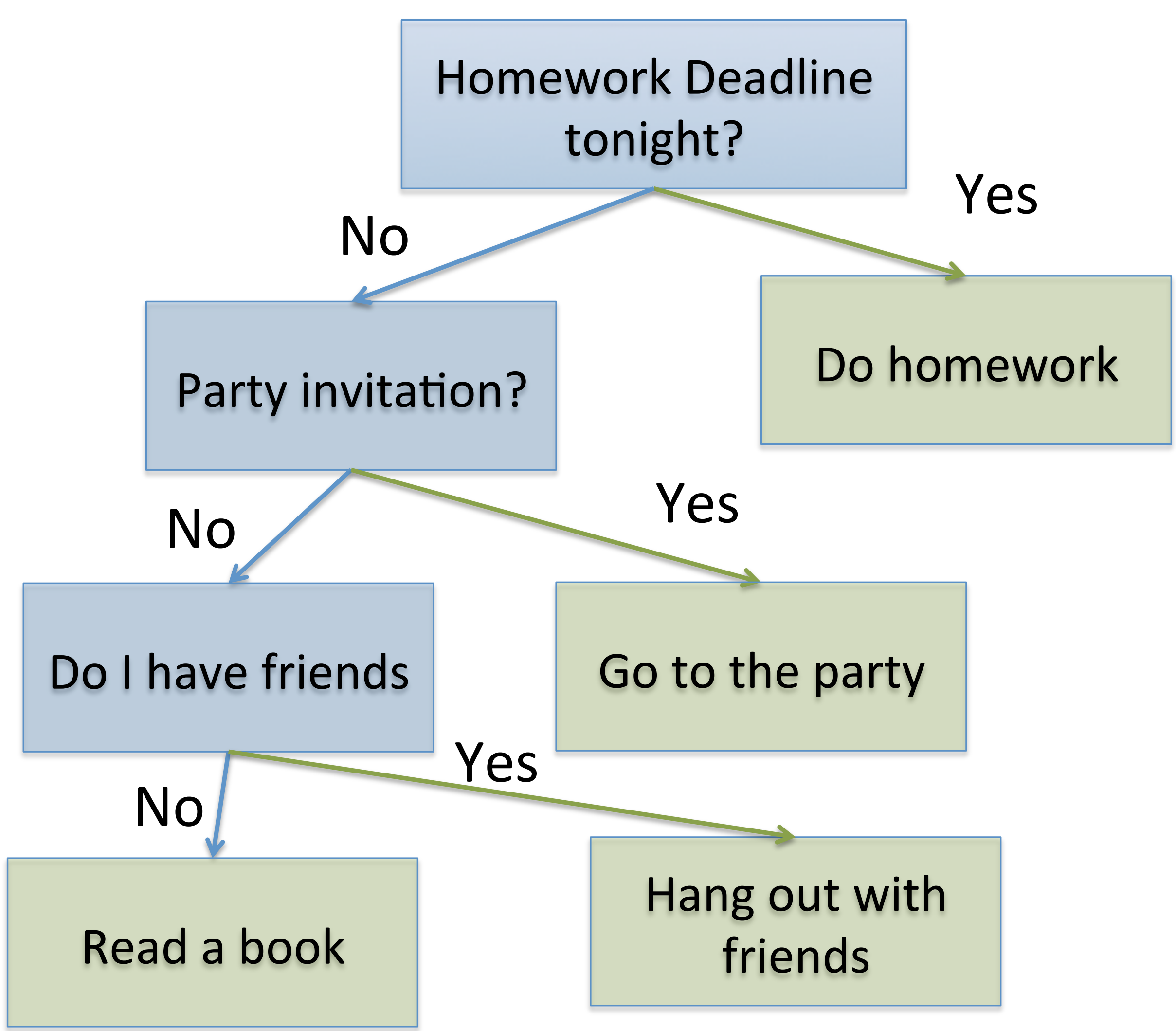
- Green contours = hail occurred (truth)
- Physics based method: Convection-allowing model (CAM)
  - Doesn't directly predict hail
- Random forest predicts hail size (Γ) distribution based on weather
- HAILCAST = diagnostic measure based on CAMs
- Updraft Helicity = surrogate variable from CAM



McGovern et al 2017, Bull. Amer. Meteor. Soc. 98:10, p. 2073-2090.

Decision Trees

- Algorithm made up of conditional control statements



Decision Trees

McGovern et al 2017 BAM –

- Decision trees used in meteorology since mid-1960s

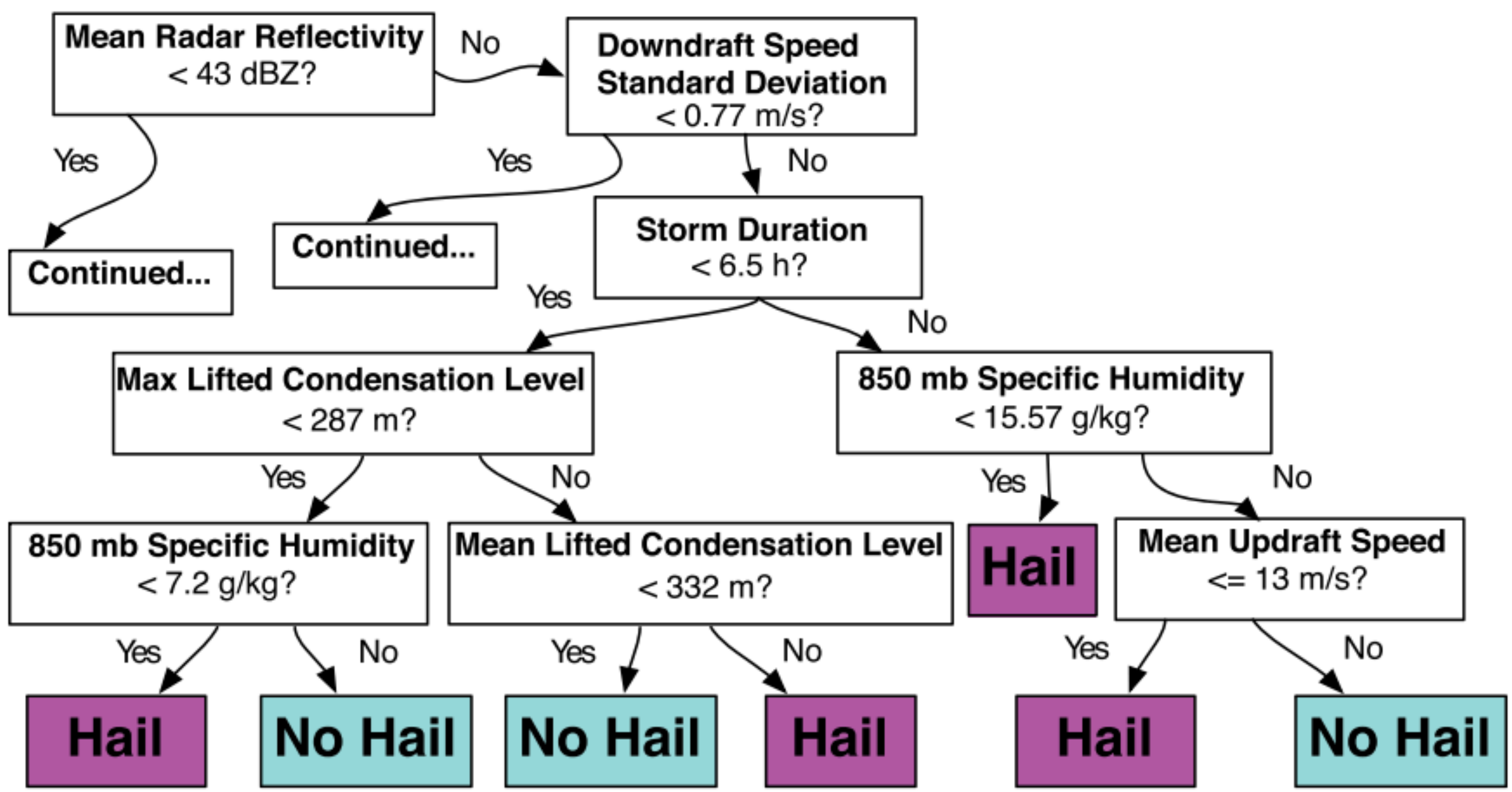
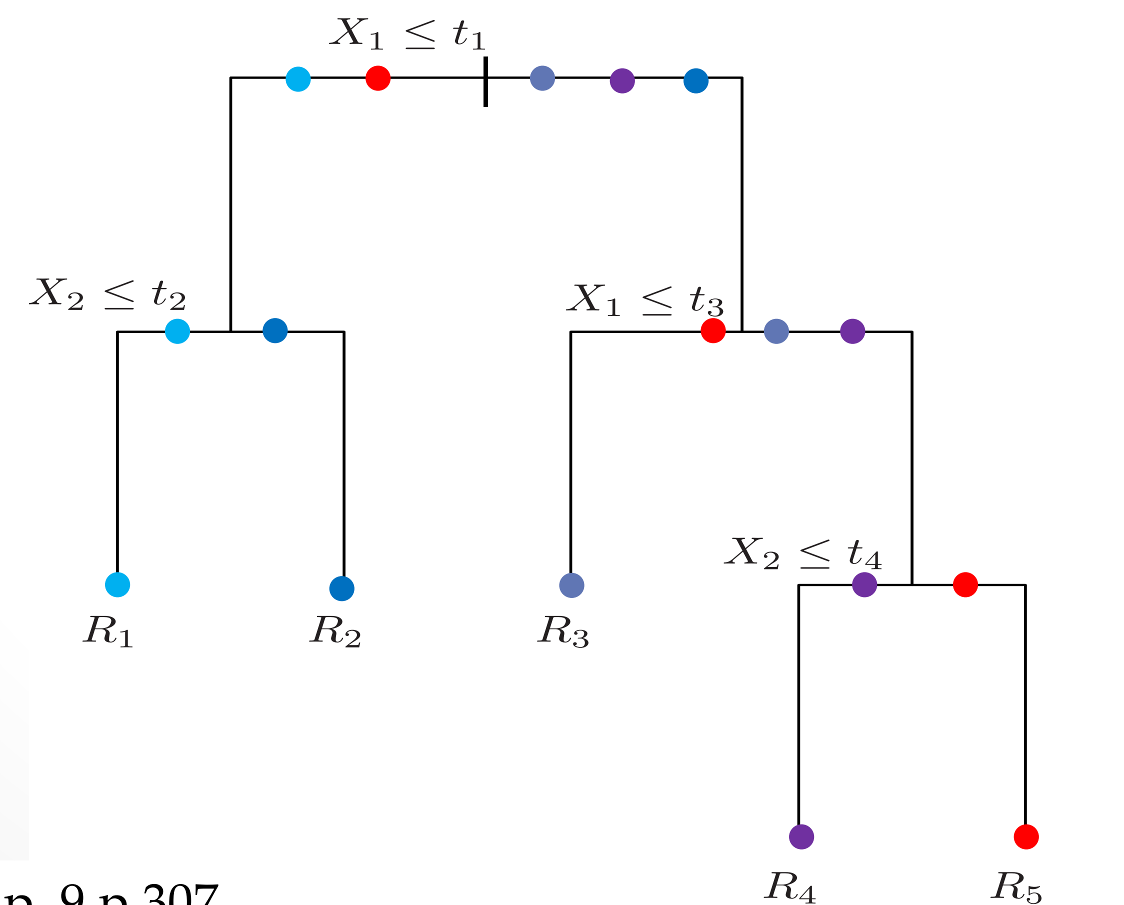


FIG. 1. An example of a decision tree for predicting if hail will occur. A version of this decision tree first appeared in Gagne (2016).

McGovern et al 2017, Bull. Amer. Meteor. Soc. 98:10, p. 2073-2090.

Regression Tree

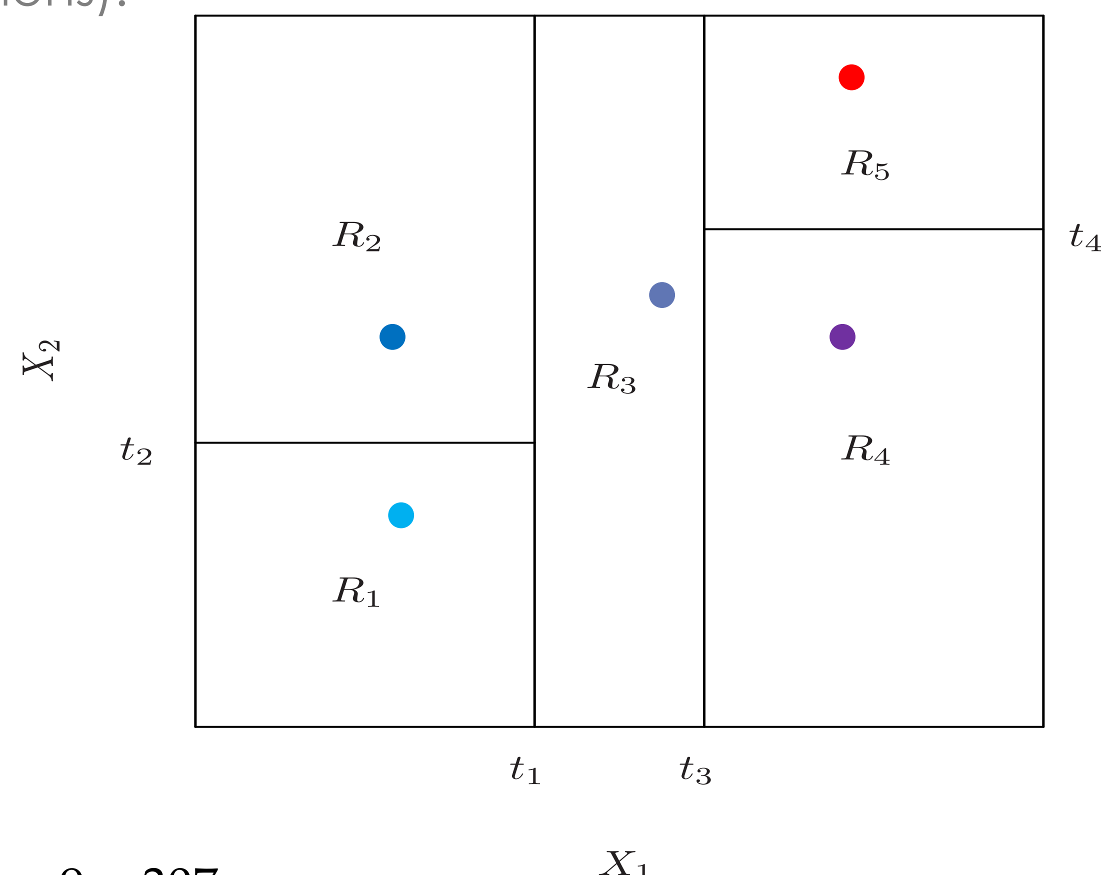
- Divide data into distinct, non-overlapping regions $R_1, \ldots, R_J$
- Below y_i = color = continuous target (<blue = 1 and >red = 0).
- x_i , i = 1,..,5 samples
- $x_i = [X_1, X_2]$, with P = 2 features.
- j = 1,..,5 (5 regions).



Hastie et al 2017, Chap. 9 p 307.

Tree-building

- Or, consecutively partition a region into non-overlapping rectangles
- y_i = color = continuous target (<blue = 1 and >red = 0).
- x_i , i = 1,…,5 samples
- $x_i = [X_1, X_2]$, with P = 2 features.
- j = 1,…,5 (5 regions).



Hastie et al 2017, Chap. 9 p 307.

Regression Tree

- How to optimize a regression tree?
- Randomly select t_1

$$R_1(j,\ t_1) = \{X \mid X_j \le t_1\}$$

$$R_2(j,\ t_1) = \{X \mid X_j > t_1\}\ ,\ j = 1$$

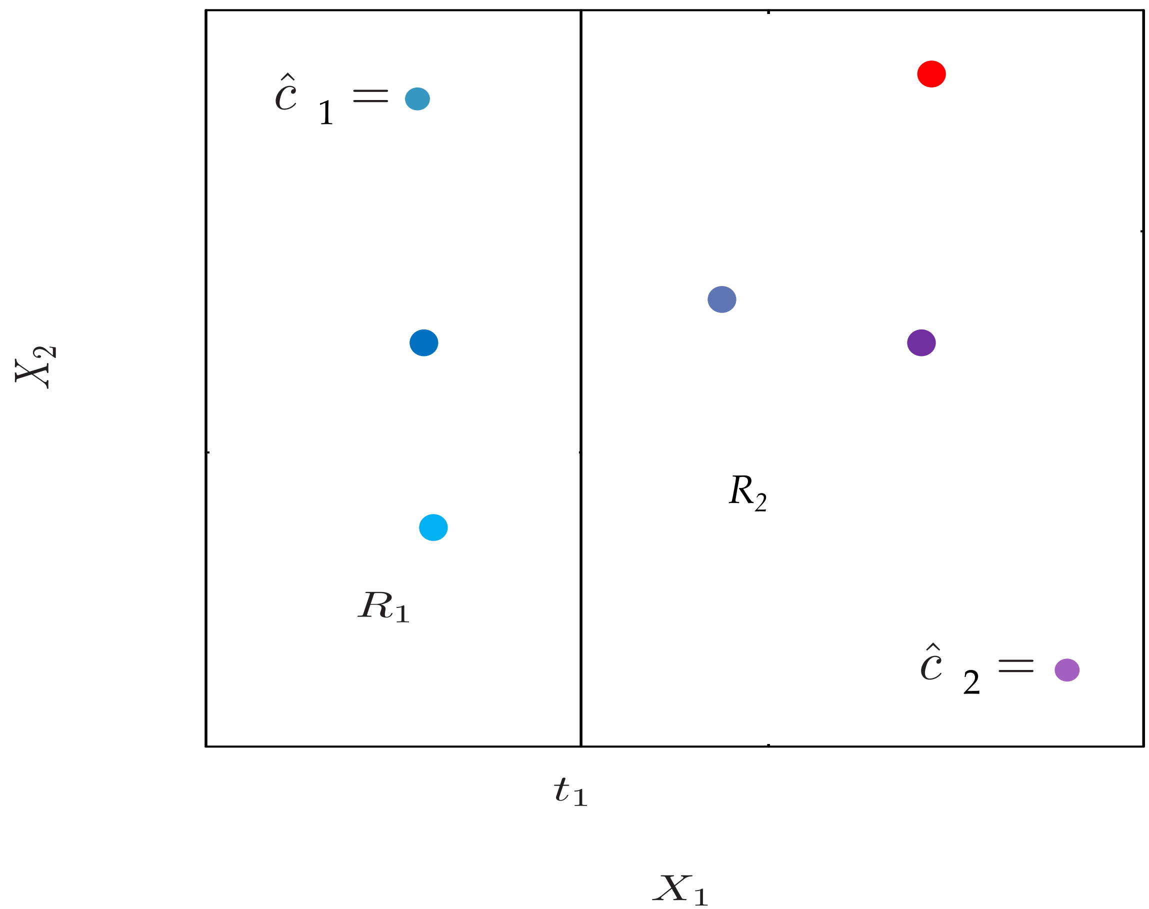
- Assign region labels:

$$\hat{c}_m = \text{ave}(y_i \mid x_i \in R_m).$$

  - Example-

    $\hat{c}_1 =$

    $\hat{c}_2 =$

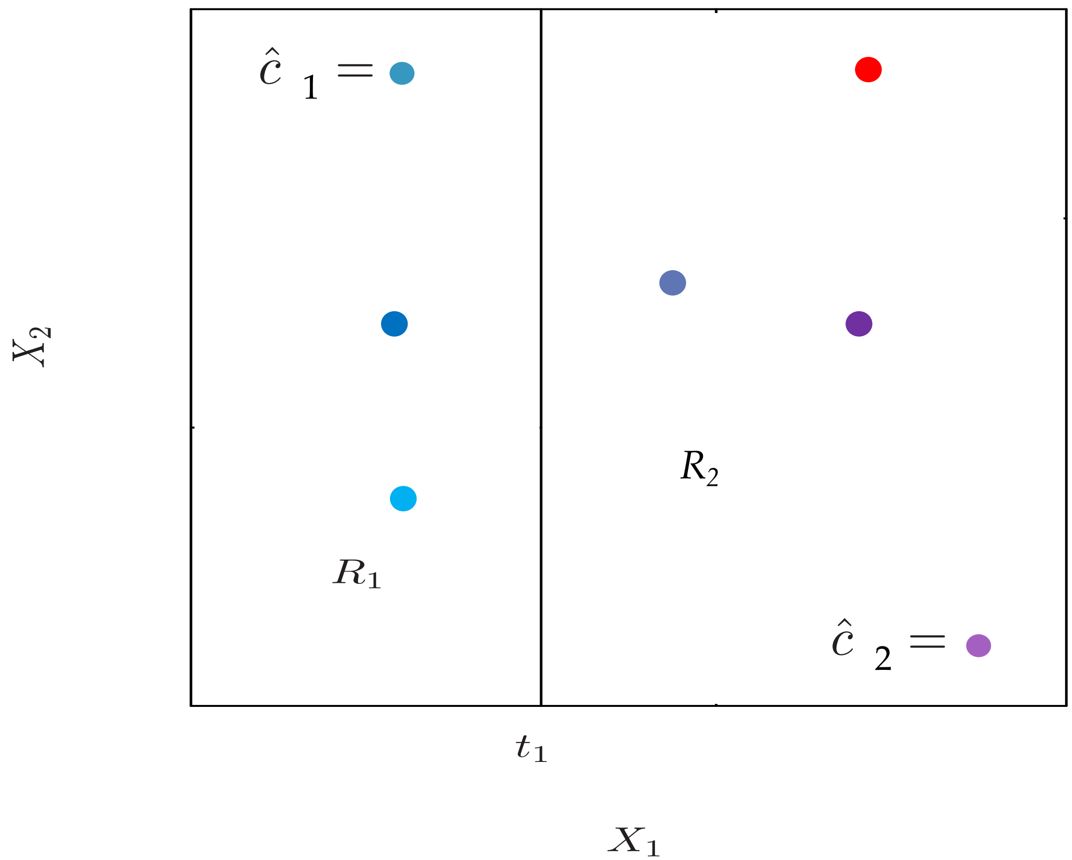


Regression Tree

- Compute the cost of the tree, $Q_m(T)$,
- Minimize $Q_m(T)$ by changing t_1

$$N_m = \#\{x_i \in R_m\},$$

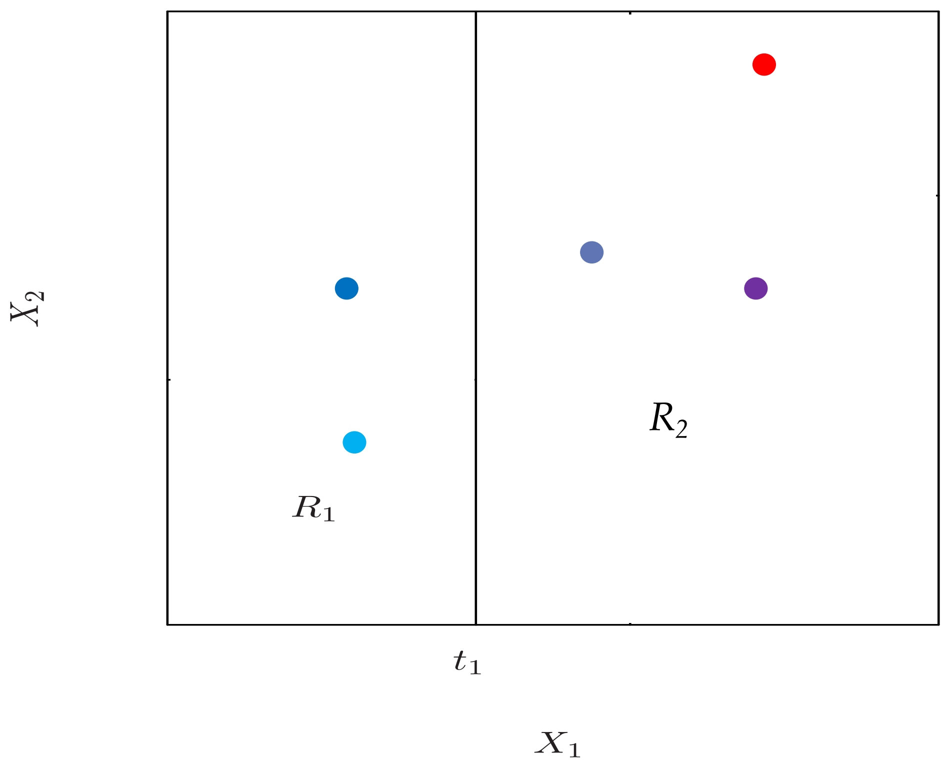
$$Q_m(T) = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2,$$



Regression Tree

- Algorithm to build tree T_b
- In our simple case, $m = 1$ *and* $p = 2$
- Daughter nodes are equivalent to regions

1. Select m variables at random from the p variables.
2. Pick the best variable/split-point among the m.
3. Split the node into two daughter nodes.



Bootstrap samples

- Select a subset of the total samples, (x^*_i, y^*_i), $i = 1,\ldots,N$
- Draw samples uniformly at random with replacement
- Example: If $I = 5$ originally, we could choose $N = 2$

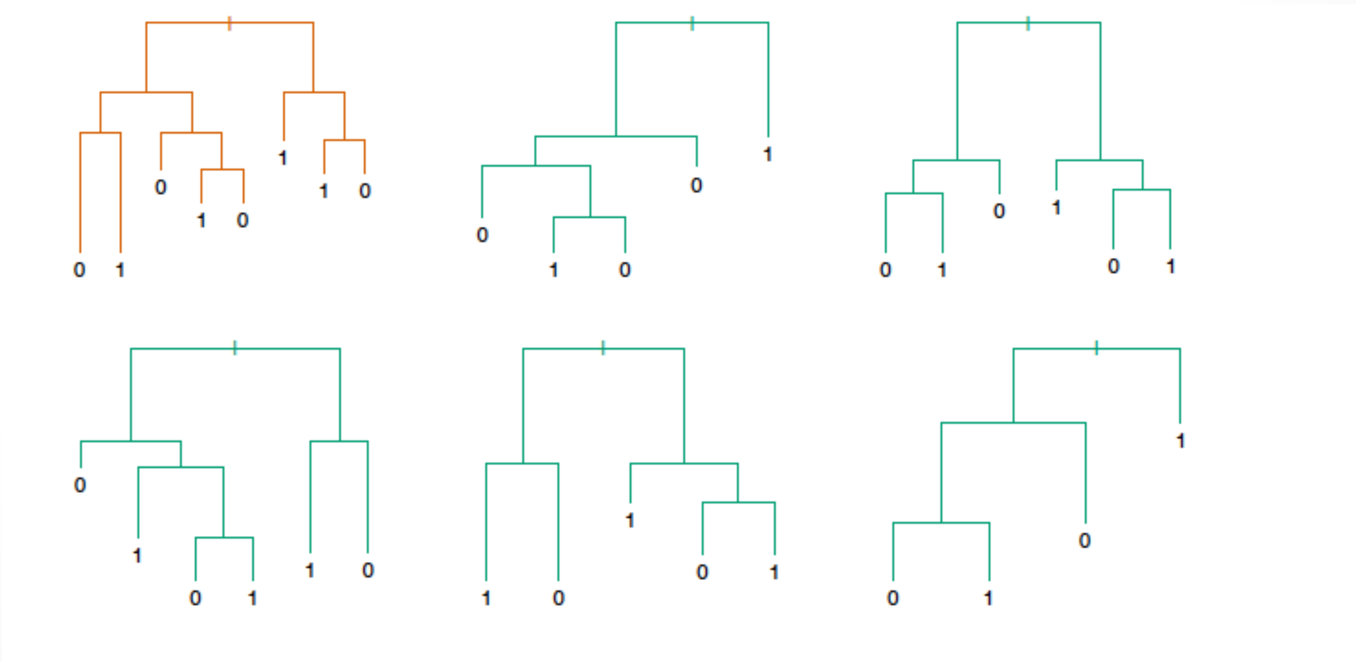
- Samples are drawn assuming equal probability:
  - If x_i, y_i is drawn more often, it is more likely

$$P_{\hat{\mathcal{F}}}\{(X,Y) = (x,y)\} = \begin{cases} \frac{1}{n} & \text{if } (x,y) = (x_i, y_i) \text{ for some } i \\ 0 & \text{otherwise} \end{cases}$$

  - (X,Y) are the expectations of the underlying distributions

Random Forest

- Example of binary classification tree from Hastie et al 2017
- *Orange*: trained on all data
- *Green*: trained from different bootstrap samples
- Then, average the (green) trees



Hastie et al 2017, Chap. 8 p. 284

Random Forest

- Bootstrap + bagging => more robust RF on future test data
- Train each tree T_b on bootstrap sampling

Algorithm 15.1 *Random Forest for Regression or Classification.*

1. For $b = 1$ to B:

   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.

   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.

      i. Select m variables at random from the p variables.

      ii. Pick the best variable/split-point among the m.

      iii. Split the node into two daughter nodes.

2. Output the ensemble of trees $\{T_b\}_1^B$.

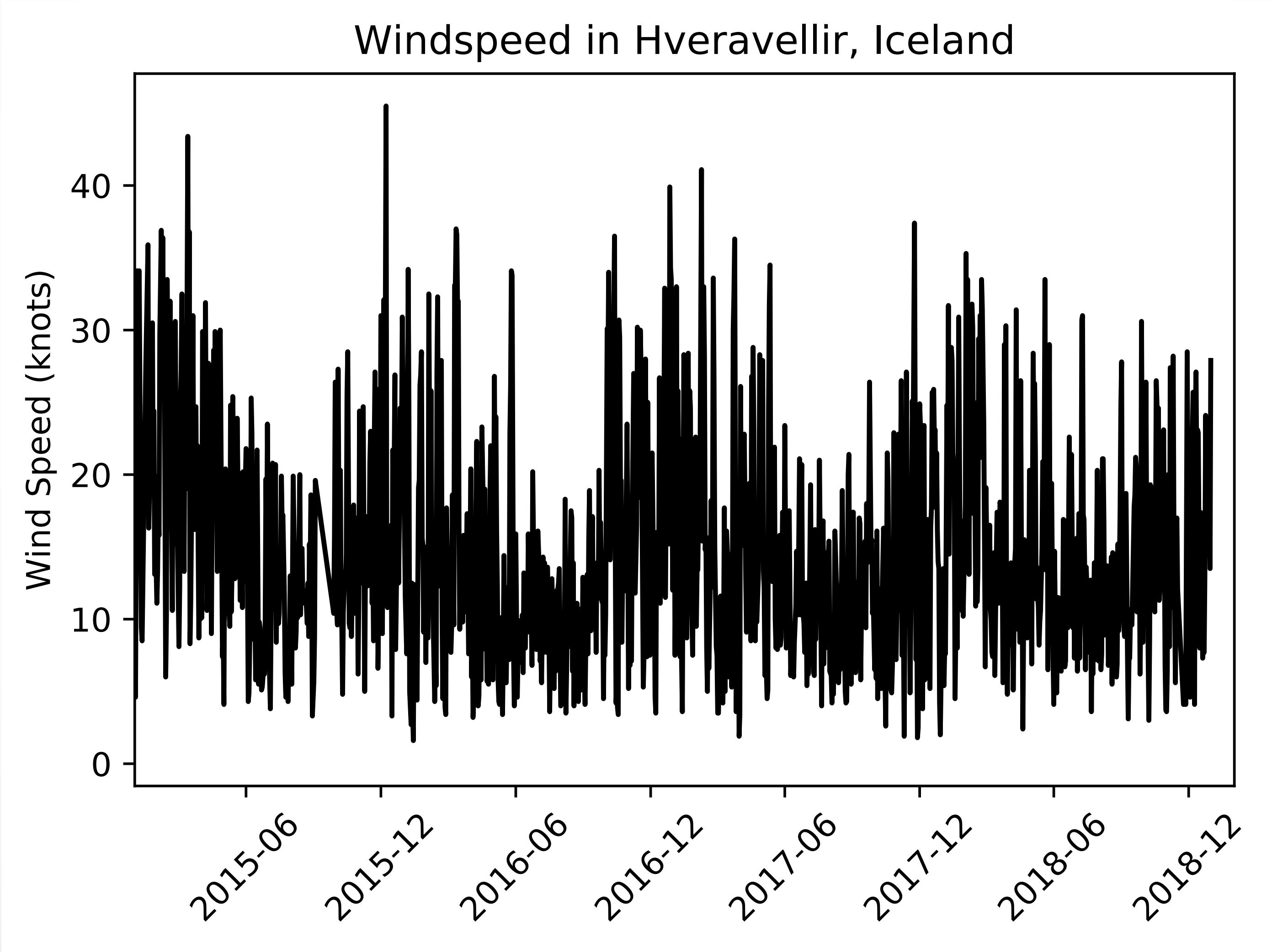
To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$.

Hastie et al 2017, Chap. 15 p. 588

Timeseries (TS)

- Timeseries: one or more variables sampled in the same location at successive time steps



ARMA

- Autoregressive moving-average :
  - (weakly) stationary stochastic process
  - Polynomials model process and errors as polynomial of prior values

- **Autogressive (order p)**
  - Linear model of past (lagged) and future values
  - *p* lags

$$X_t = c + \sum_{i=1}^{p} \varphi_i X_{t-i} + \varepsilon_t$$

  - φ_i are (weights) parameters
  - *c* is constant
  - ε_t is white noise (WGN)
  - Note, for stationary processes, $|\varphi_i| < 1$.

- **Moving-average (order q)**
  - Linear model of past errors
  - *q* lags
  - Below, assume $\langle X_t \rangle = 0$ (expectation is 0)

$$X_t = c + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} + \varepsilon_t$$

ARMA

- Autoregressive moving-average :
  - (weakly) stationary stochastic process
  - Linear model of prior values = expected value term + error term + WGN

- **ARMA: AR(p) + MA(q)**

$$X_t = c + \sum_{i=1}^{p} \varphi_i X_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} + \varepsilon_t$$

Data retrieval

Just a few public data sources for physical sciences…

- NOAA:
  - Reanalysis/model data, research cruises, station observations, gridded data products, atmospheric & ocean indices timeseries, heat budgets, satellite imagery
- NASA:
  - EOSDIS, gridded data products (atmospheric), satellite imagery, reanalysis/model data, meteorological stations, DAAC's in US
- IMOS:
  - ocean observing hosted by Australian Ocean Data Network
- USGS Earthquake Archives
- CPC/NCEI:
  - gridded and raw meteorological and oceanographic
- ECMWF
  - global-scale weather forecasts and assimilated data

…

Possible data formats:

- CSV
- NetCDF
- HDF5/HDF-EOS
- Binary
- JPEG/PNG
- ASCII text

- ……

Basic data cleaning

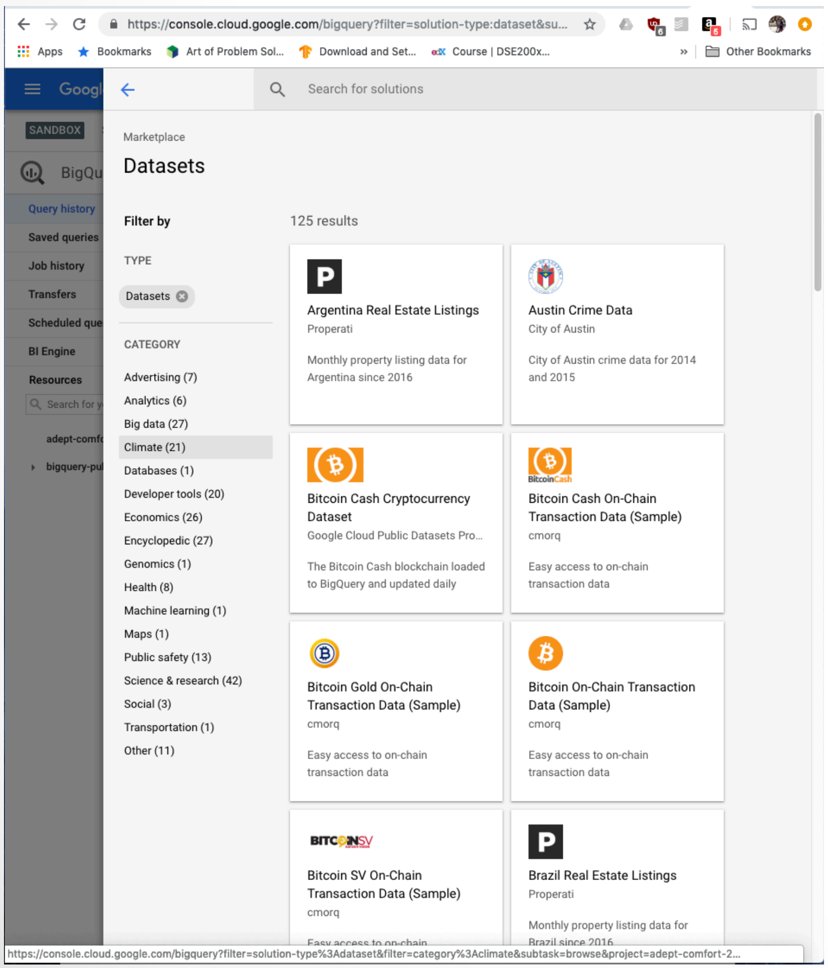
- "*[ML for physical sciences] is 80% cleaning and 20% models*" ~ paraphrased, Dr. Gerstoft
- Basic cleaning for NOAA GSOD to HW – necessary
  - Remove unwanted variables (big data is slow)
  - Replaced "9999" filler values with NaN
  - Converted strings to floats (i.e. for wind speed)
  - Created a DateTime index

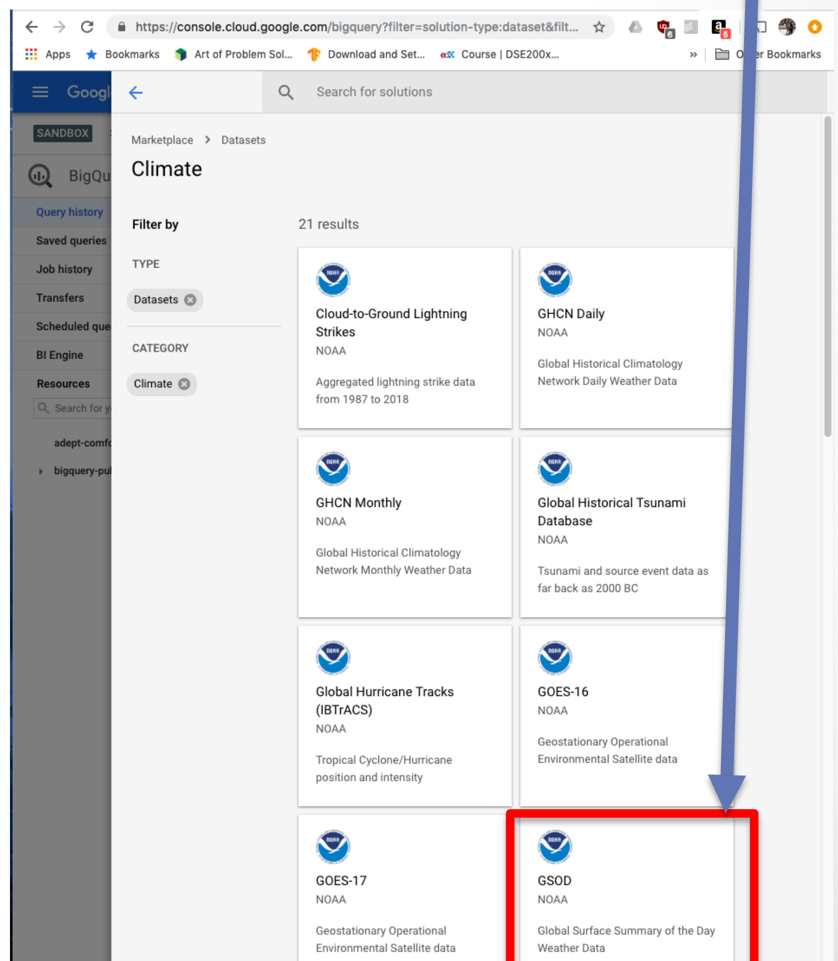
- Physical data needs cleaning, reorganizing
- Quality-controlled data *still* causes bugs
- Application-specific

Data for HW

- BigQuery:
  - Open-source database hosted by Google
  - Must have Google account
  - 1 TB data free/ month

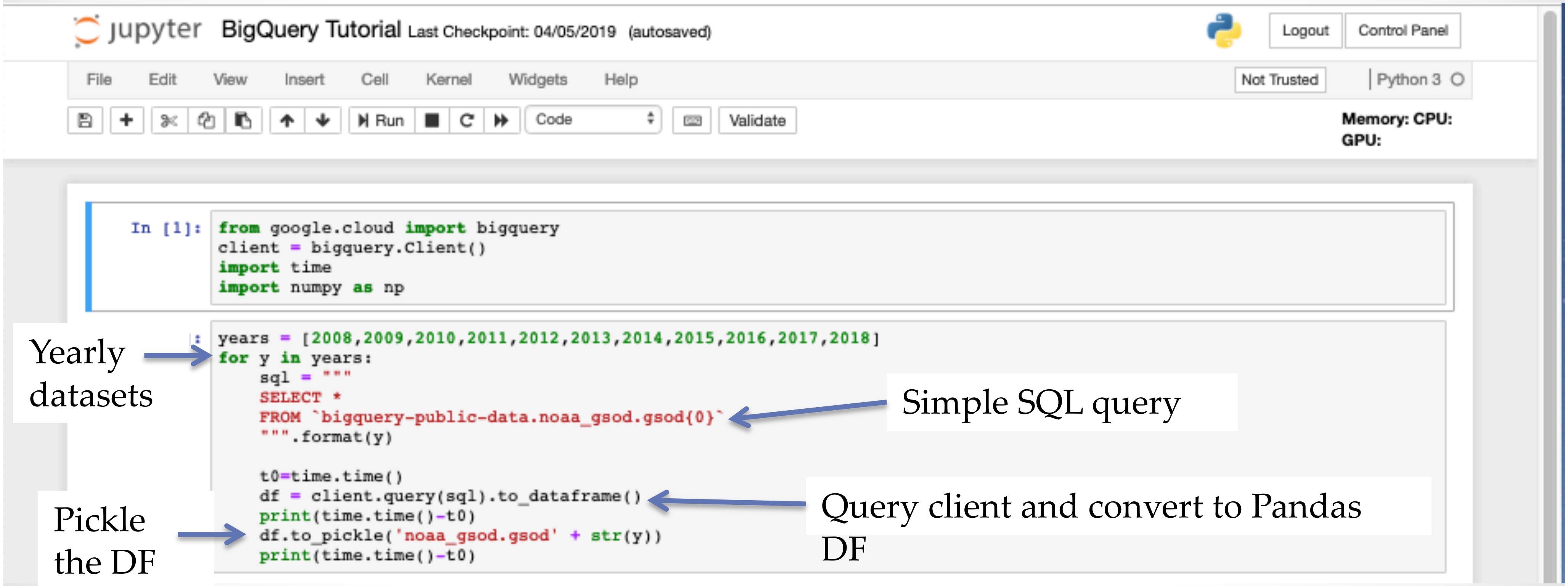
NOAA GSOD dataset





Data for HW

- How to get BigQuery data?
- bigquery package in Jupyter Notebook (SQL server)



- More complex queries may include dataframe joins, aggregations, or subsetting

Tutorial Notebook

- Open "In-Class Tutorial"
- We will do:
  1. Load preprocessed data
  2. Define timeseries index
  3. Look at data
  4. Visualize station
  5. Detrend data
  6. Smooth data
  7. Try ARMA model

Tutorial Notebook

- Load packages, (pre-processed) data with Pandas

```
In [60]: import pandas as pd
         import numpy as np
         from numpy.random import randint
         import matplotlib.pyplot as plt
         import matplotlib.dates as mdates
         import time
         from mpl_toolkits.basemap import Basemap
         import scipy.signal as sig
         from statsmodels.tsa.arima_model import ARMA
         from pandas.plotting import autocorrelation_plot
```

import packages

Load our pre-processed data.

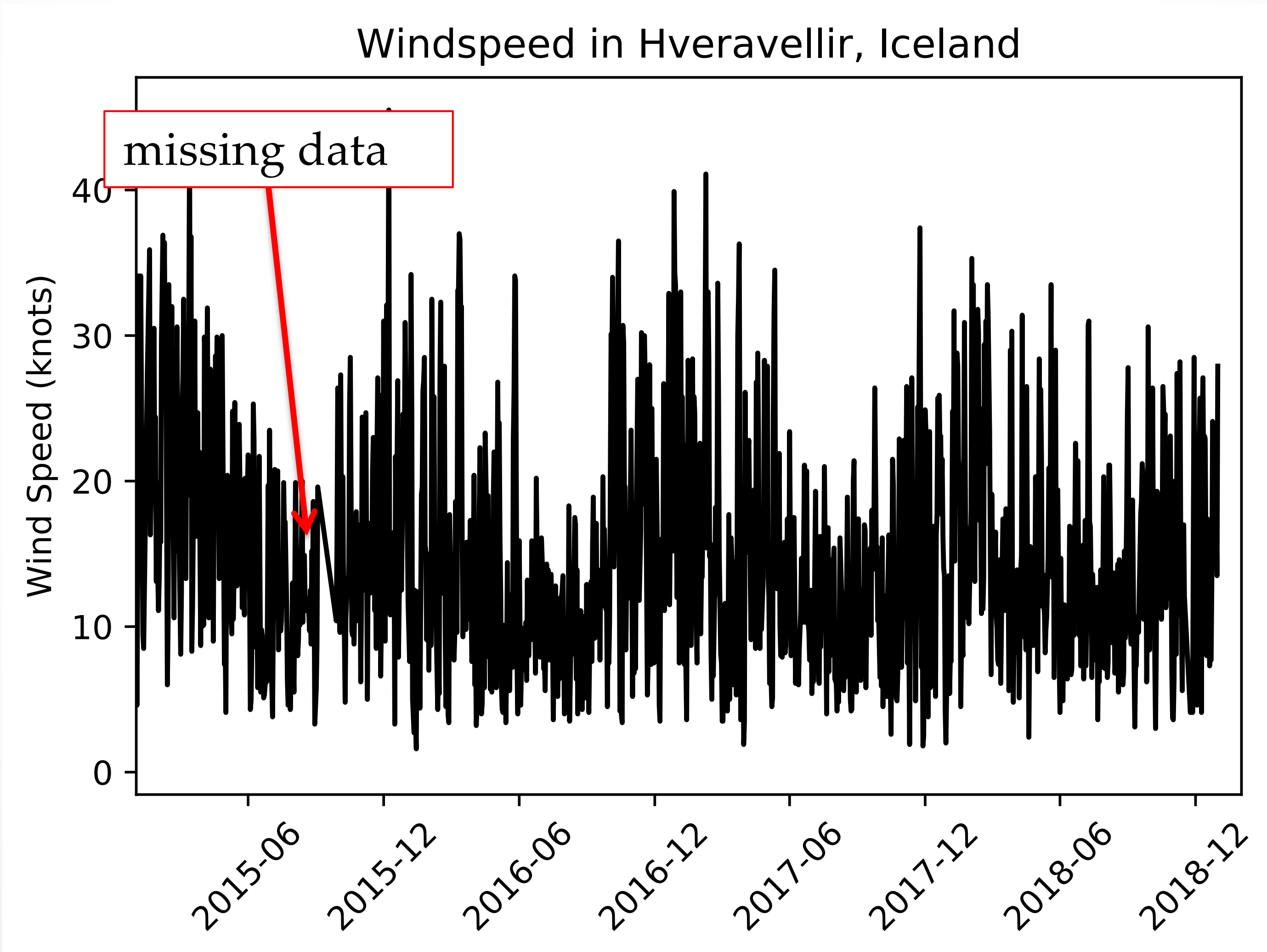
```
In [2]: t0 = time.time()
        dataset_path = '/datasets/NOAA_SST/'
        data = pd.read_pickle(dataset_path + 'noaa_gsod/Combined_noaa_gsod') # load weather data
        stations = pd.read_pickle(dataset_path + 'noaa_gsod.stations') # load station data

        # # USE ONLY 2008-2018 # #
        data = data.loc[data.index >= pd.Timestamp(2008, 1, 1)]
        data = data.drop(columns=['yr','year','da','mo']) # don't need these anymore
        print(time.time()-t0)

        22.46479368209839
```

load data

find where data is after 2008

Timeseries processing

- We may be missing data, but that's ok for now
- Replace with neighbor data, smooth, fill with mean



Tutorial Notebook

Where is my station located?

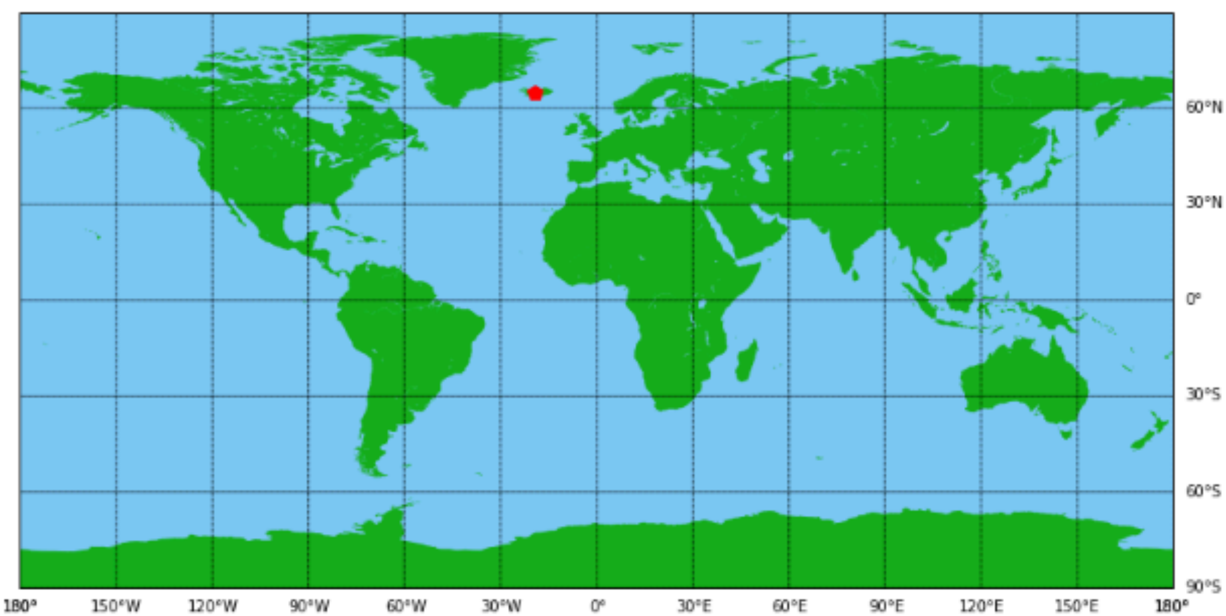
In [5]:
```
my_station = stations.loc[stations['usaf'] == my_station_number]
my_station.head()
```

Out[5]:

	usaf	wban	name	country	state	call	lat	lon	elev	begin	end
18673	041560	99999	HVERAVELLIR	IC	None	None	64.867	-19.567	+0641.0	20080115	20190401

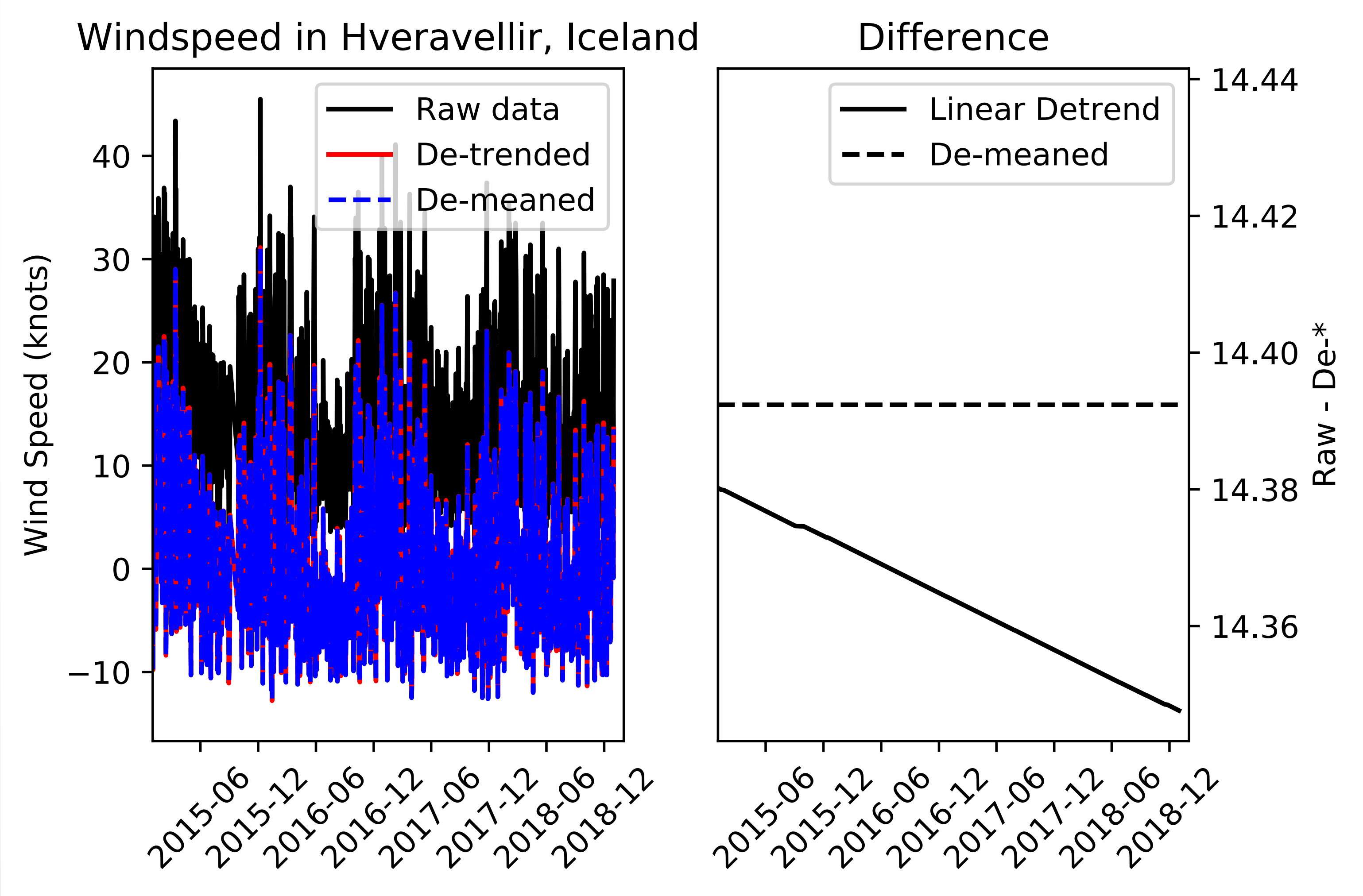
In [6]:
```
fig = plt.figure(figsize=(15,7))

#create a Basemap projection, cylindrical centered at 0
m = Basemap(projection='cyl',llcrnrlat=-90,urcrnrlat=90,\
            llcrnrlon=-180,urcrnrlon=180, resolution='l')

# draw the oceans and countries and lines
m.drawmapboundary(fill_color='xkcd:lightblue')
m.fillcontinents(color='xkcd:green',lake_color='xkcd:lightblue')
m.drawmeridians(np.arange(0.,350.,30.),labels=[True,False,False,True])
m.drawparallels(np.arange(-90.,90,30.),labels=[False,True,True,False])

# show my station
pl, = plt.plot(my_station['lon'],my_station['lat'],'rp', markersize=10)
plt.show()
```

Basemap is handy but some problems if running on your laptop



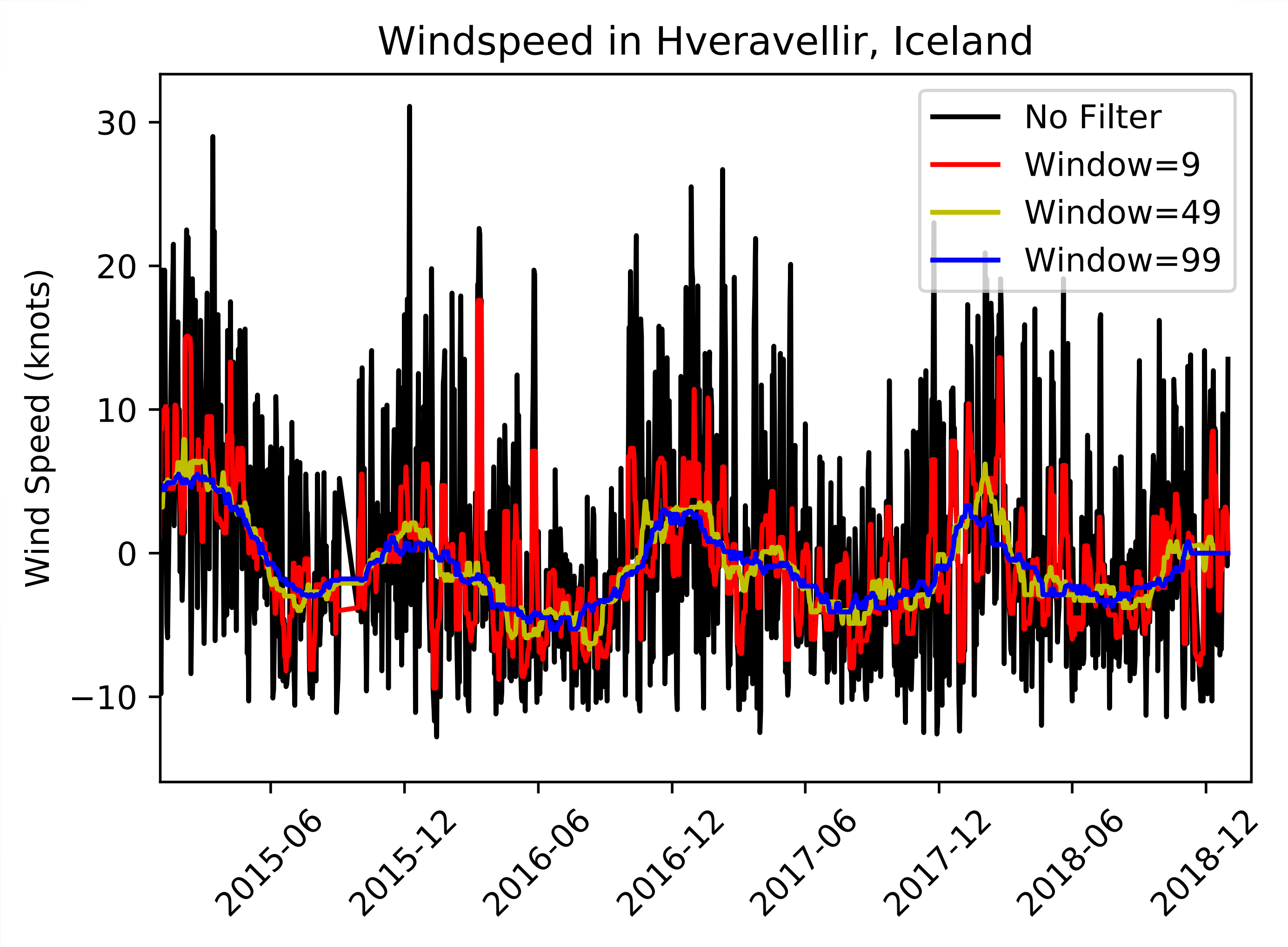
Timeseries processing

- Remove mean (slope=0) or linear (slope ≠ 0)? *(linear)*
- What can we learn from trend?



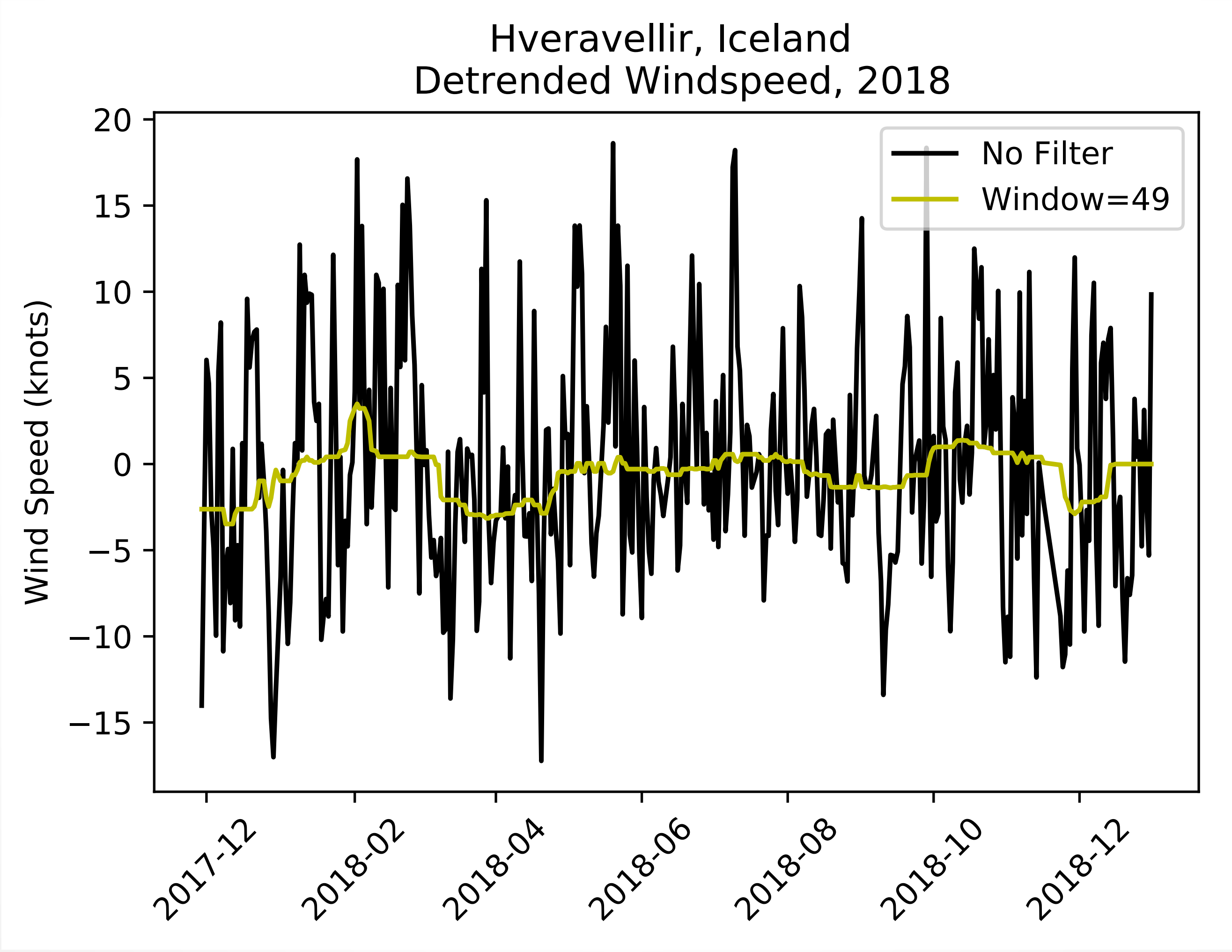
Timeseries processing

- Smoothing: median filter



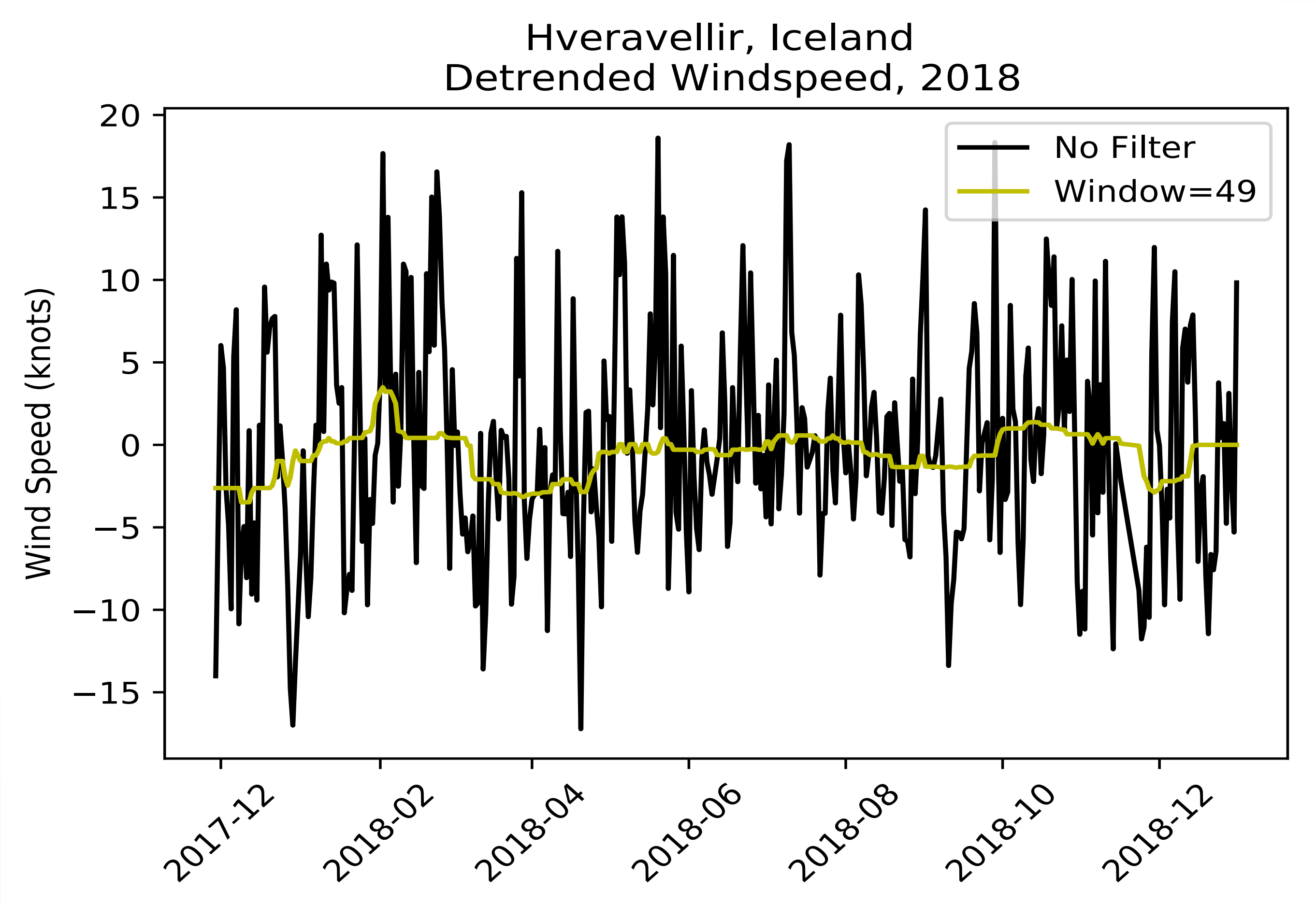
Tutorial Notebook

- Shortened timeseries – Y2018 (final 10%)
- ARMA most effective predicting one step at a time



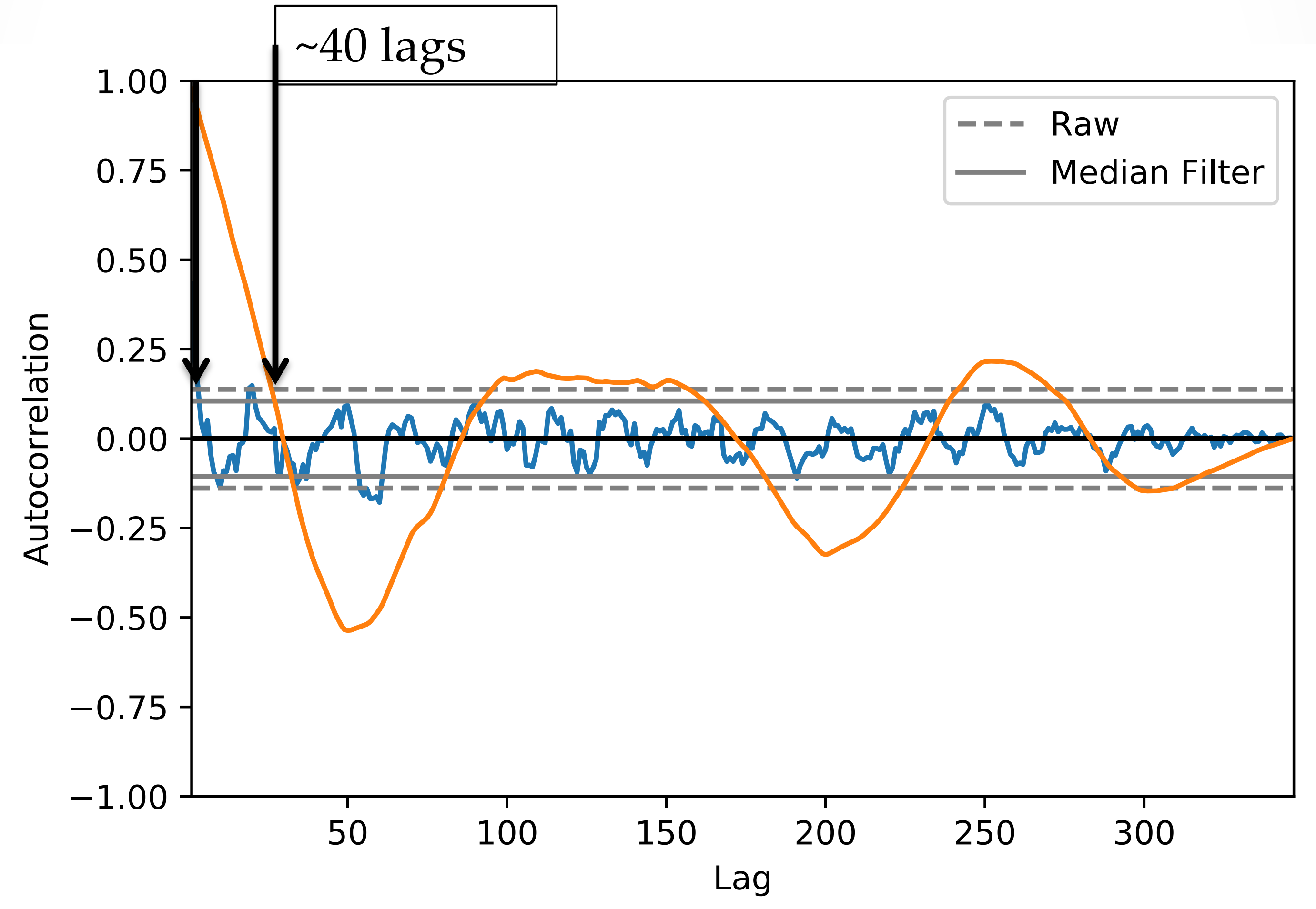
Tutorial Notebook

- Is ARMA a machine learning technique? (I think so..)
  - Filtering method (like Kalman filter)
  - Data-driven
  - Maximum likelihood
  - Conclusion: statistics-based



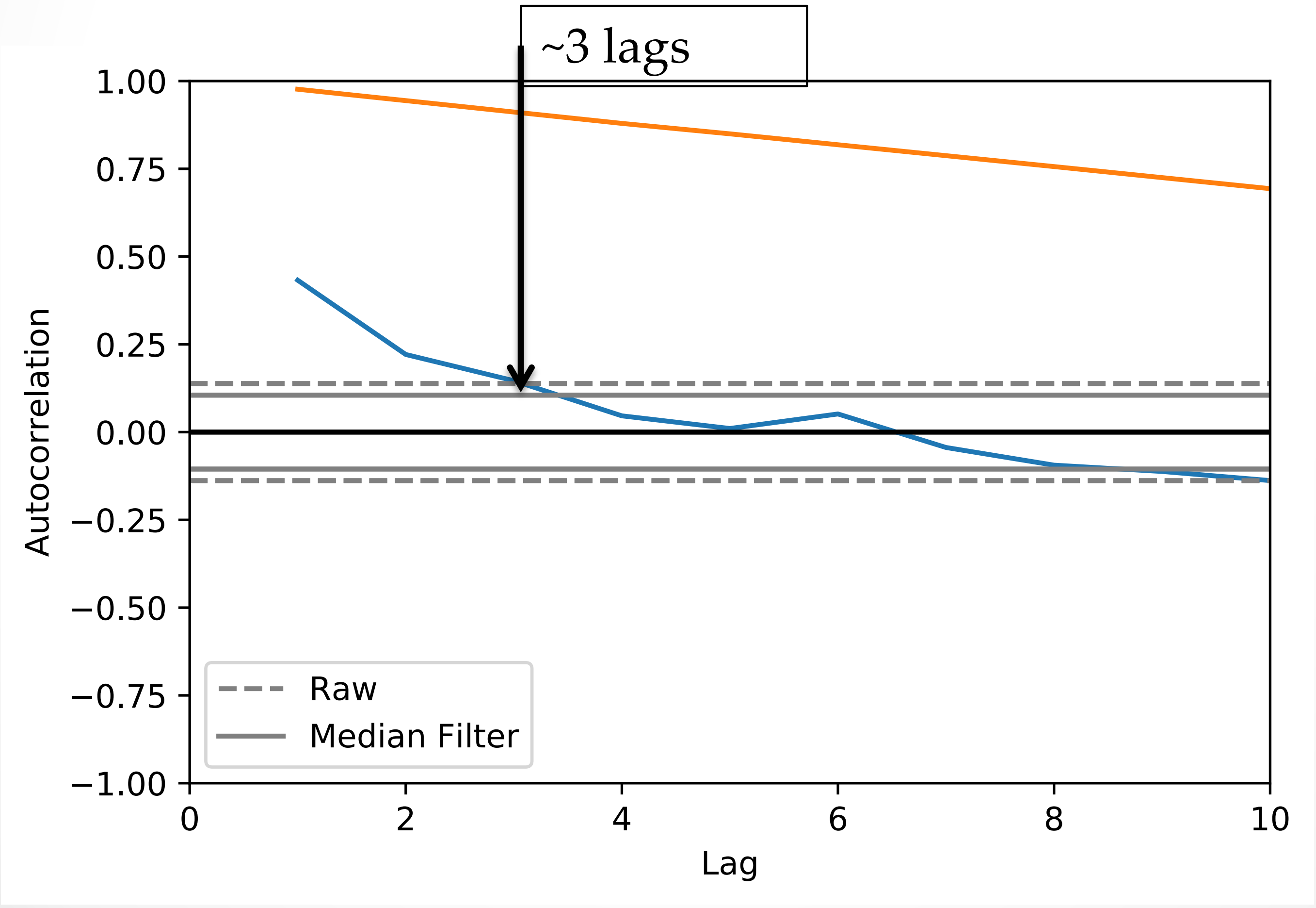
Tutorial Notebook

- Autocorrelation:
  - A statistical method to find temporal (or spatial) relations in data
  - When can reject the null hypothesis that the data is statistically similar?
  - E.g. How many time steps before the data is decorrelated



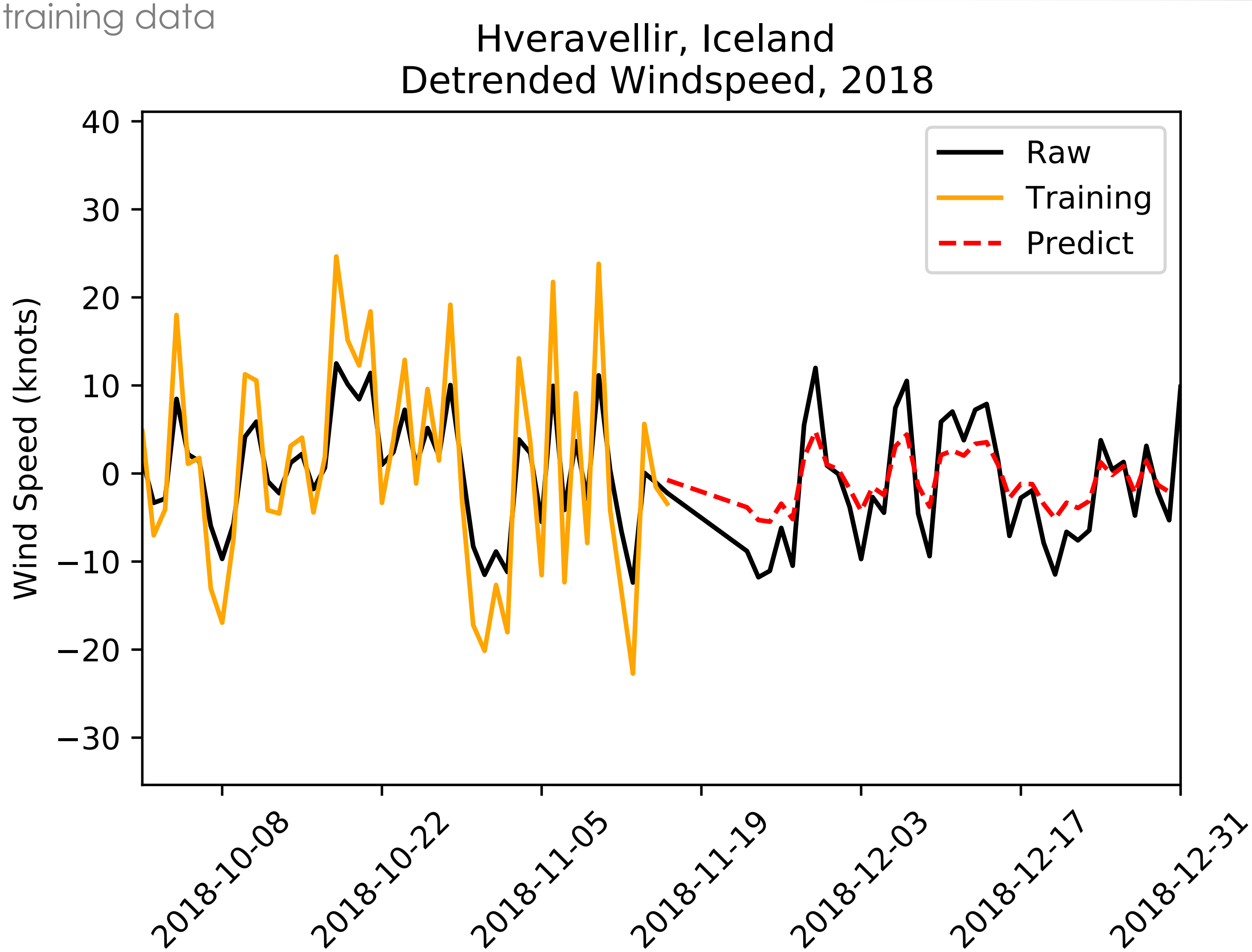
Tutorial Notebook

- Median filter increases decorrelation scale
    - By averaging neighbor samples
- Raw series is more random
- Use raw timeseries



Tutorial Notebook

- ARMA algorithm:
  1. Train on all previous data
  2. Predict one time step
  3. Add next value to training data
  4. Repeat



•

Homework

- How to load and preview data with Pandas

Predict rain based on other weather variables

This notebook will use time lags to train a machine learning model for predicting temprature.

First, we select a random station. The data is kept at daily resolution. Then, we generate a lagged feature matrix.

import packages

n [2]:

```
import pandas as pd
import numpy as np
from numpy.random import randint
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import time
import glob
from mpl_toolkits.basemap import Basemap
```

load data

find where data is after 2008

```
t0 = time.time()
data = pd.read_pickle('noaa_gsod/Combined_noaa_gsod') # load weather data
stations = pd.read_pickle('noaa_gsod.stations') # load station data

# # USE ONLY 2008-2018 # #
data = data.loc[data.index >= pd.Timestamp(2008, 1, 1)]
data = data.drop(columns=['yr','year','da','mo']) # don't need these anymore
print(time.time()-t0)
```

22.778506994247437

```
stations.head()
```

Out[4]:

	usaf	wban	name	country	state	call	lat	lon	elev	begin	end
0	007018	99999	WXPOD 7018	None	None	None	0.00	0.000	+7018.0	20110309	20130730
1	007026	99999	WXPOD 7026	AF	None	None	0.00	0.000	+7026.0	20120713	20170822
2	007070	99999	WXPOD 7070	AF	None	None	0.00	0.000	+7070.0	20140923	20150926
3	008268	99999	WXPOD8278	AF	None	None	32.95	65.567	+1156.7	20100519	20120323
4	008307	99999	WXPOD 8318	AF	None	None	0.00	0.000	+8318.0	20100421	20100421

-

Homework

- How to load and preview data with Pandas

In [5]:
```
data.head()
```

Out[5]:

Datetime	stn	temp	slp	wdsp	mxpsd	max	min	prcp
2008-08-22	619970	41.7	1032.1	7.2	14.0	46.6	37.8	0.00
2008-09-28	714170	54.9	1023.8	9.1	15.0	70.3	51.4	0.06
2008-02-14	041560	35.6	1032.8	28.5	38.9	36.5	34.5	0.00
2008-02-25	041560	4.5	997.1	4.3	7.0	10.4	-1.1	0.00
2008-10-06	041650	43.3	998.8	24.3	30.9	44.6	40.3	0.00

corresponding recording station

column of all temperature entries

Homework

- Randomly select a station
- Check if the station has enough data
  - May reduce "3650" to lower number, i.e. 1000, but be aware you may have nans in data – just look at it!

In [7]:

```python
# # SELECT RANDOM STATION # #
np.random.seed(3)
rs = np.unique(data['stn'].values) # find unique stations with data
rand_stat = rs[randint(len(rs))] # pick a random station

# # ideally we should check < len(np.unique(data.index)), but many are shorter
while (len(data.loc[data['stn'] == rand_stat]) < 3650): # If not enough data
    if len(stations.loc[stations['usaf'] == rand_stat]): # If station info available
        if (stations.loc[stations['usaf'] == rand_stat].iloc[0]['wban'] != '99999'): # If station number not unique
            rand_stat = rs[randint(len(rs))] # get a new station
    else:
        rand_stat = rs[randint(len(rs))] # get a new station

select_station = stations.loc[stations['usaf'] == rand_stat] # get location, etc, for random station

features = data.loc[data['stn'] == rand_stat] # pick weather at random station
features = features.drop(columns='stn')
features = features.drop(columns='max')
features = features.drop(columns='min')
features = features.sort_index()
select_station.head() # see where it is
```

select random station

find data that matches station

remove data related to temperature

Out[7]:

	usaf	wban	name	country	state	call	lat	lon	elev	begin	end
5946	712220	99999	DEASE LAKE (AUT)	CA	None	CWKX	58.433	-130.033	+0802.0	19930829	20190326

Homework

- Manually time-delay data
- Pandas "shift()"

Time-shift the data

In [8]:

```
columns = features.columns
for co in columns:
    # one day lag
    features[co + '_lag1'] = features[co].shift(periods=1)

    # two days lag
    features[co + '_lag2'] = features[co].shift(periods=2)

    # three days lag
    features[co + '_lag3'] = features[co].shift(periods=3)
features = features.iloc[3:]
print(str(len(features)) + ' samples, ' + str(len(features.columns)) + ' features.')
features.head()
```

Pandas shift()

remove first 3 entries

```
3926 samples, 20 features.
```

Out[8]:

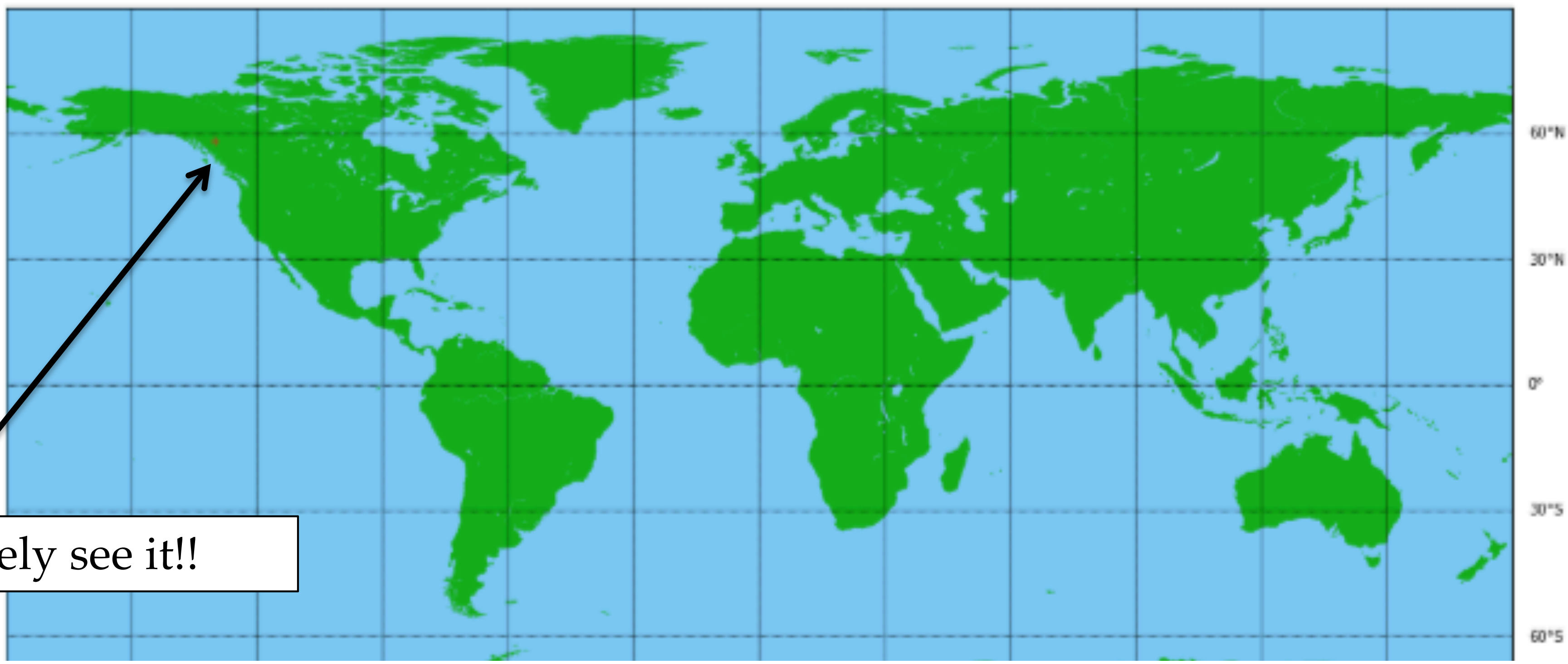
Datetime	temp	slp	wdsp	mxpsd	prcp	temp_lag1	temp_lag2	temp_lag3	slp_lag1	slp_lag2	slp_lag3	wdsp_lag1	wdsp_lag2	wdsp_la
2008-01-04	9.5	982.6	2.4	5.8	0.08	5.5	0.0	-1.7	991.0	998.5	1024.8	4.6	2.5	
2008-01-05	6.6	978.9	2.2	4.9	0.03	9.5	5.5	0.0	982.6	991.0	998.5	2.4	4.6	
2008-01-06	11.4	987.0	2.8	5.8	0.12	6.6	9.5	5.5	978.9	982.6	991.0	2.2	2.4	
2008-01-07	-7.3	1005.2	3.7	7.0	0.00	11.4	6.6	9.5	987.0	978.9	982.6	2.8	2.2	
2008-01-08	-8.9	1005.8	5.3	9.9	0.00	-7.3	11.4	6.6	1005.2	987.0	978.9	3.7	2.8	

Homework

- (Map is supposed to show red “X” for station)

In [13]:

```
# Show the position of the station

fig = plt.figure(figsize=(18.5, 10.5))
m = Basemap(projection='cyl',llcrnrlat=-90,urcrnrlat=90,\
            llcrnrlon=-180,urcrnrlon=180, resolution='l')
m.drawmapboundary(fill_color='xkcd:lightblue')
m.fillcontinents(color='xkcd:green',lake_color='xkcd:lightblue')
m.drawmeridians(np.arange(-180.,180.,30.),labels=[True,False,False,True])
m.drawparallels(np.arange(-90.,90,30.),labels=[False,True,True,False])
lon = select_station['lon'].tolist()
lat = select_station['lat'].tolist()
m.plot(lon, lat,'r+')
plt.show()
```



I can barely see it!!

Homework

- Snapshots from "timeseries_prediction_Temp.ipynb"

Create train/val/test

In [8]:

```
ylabel = features['temp'] # use today's temperature as the label
features = features.drop(columns='temp') # don't put it in training d

# Use 20% test split (80% training + validation)
ntrain = int(len(features)*0.8)
x_test = features.iloc[ntrain:,:]
y_test = ylabel[ntrain:]

# Split remaining 80% into training-validation sets (of original data)
x_train, x_val, y_train, y_val = train_test_split(features.iloc[0:ntrain,:], ylabel[0:ntrain], \
                                                  test_size=0.2, random_state=1)

# Scale features. Fit scaler on training only.
scaler = MinMaxScaler() #scale features between 0 and 1
x_train = scaler.fit_transform(x_train)
x_val = scaler.transform(x_val)
x_test = scaler.transform(x_test)
```

training label = temperature

split data into train/test with the help of Sklearn

scaling features improves learning

Homework

- Random forest model in a couple lines
- You may want to write a "plot.py" function

Predict with Random Forest

In [9]:

```
clf = RandomForestRegressor(n_estimators=100, max_depth=10, random_state=0) # define Random Forest object
clf.fit(x_train, y_train) # train Random Forest
y = clf.predict(x_test) # predict temperature

# plot predictions
plt.figure(figsize=(15,7))
plt.subplot(1,2,1)
plt.plot(features.iloc[ntrain:].index,y_test) # plot actual temperature
plt.plot(features.iloc[ntrain:].index, y) # plot predicted temperature
plt.xticks(features.iloc[ntrain:].index[::30], rotation = 45) # set xticks to monthly
myFmt = mdates.DateFormatter('%b-%d-%y')
plt.gca().xaxis.set_major_formatter(myFmt)
plt.ylabel('Daily Temperature (degree Fahrenheit)', fontsize=12)
plt.legend(('Temperature','Random Forest Prediction'), fontsize=12, loc=1)
#plt.show()

# # Plot the feature importances # #
nfeatures = 10
fi = clf.feature_importances_ # get feature importances
fi_sort = np.argsort(fi)[::-1] # sort importances most to least
plt.subplot(1,2,2)
plt.bar(range(nfeatures), fi[fi_sort[0:nfeatures]], width=1, \
        tick_label=features.columns.values[[fi_sort[0:nfeatures]]]) # plot features
plt.ylabel('Feature Importance (avg across trees)', fontsize=12)
plt.xticks(rotation = 45)
plt.show()
```

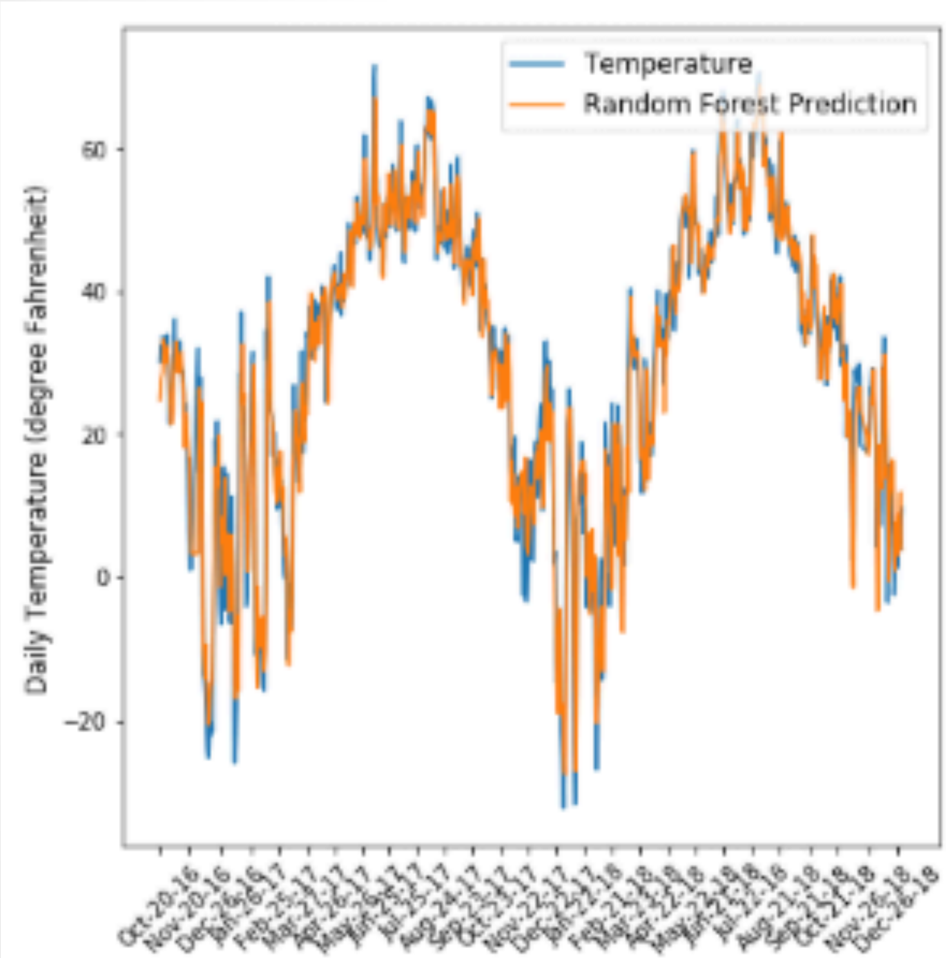
Define, train, and predict with random forest

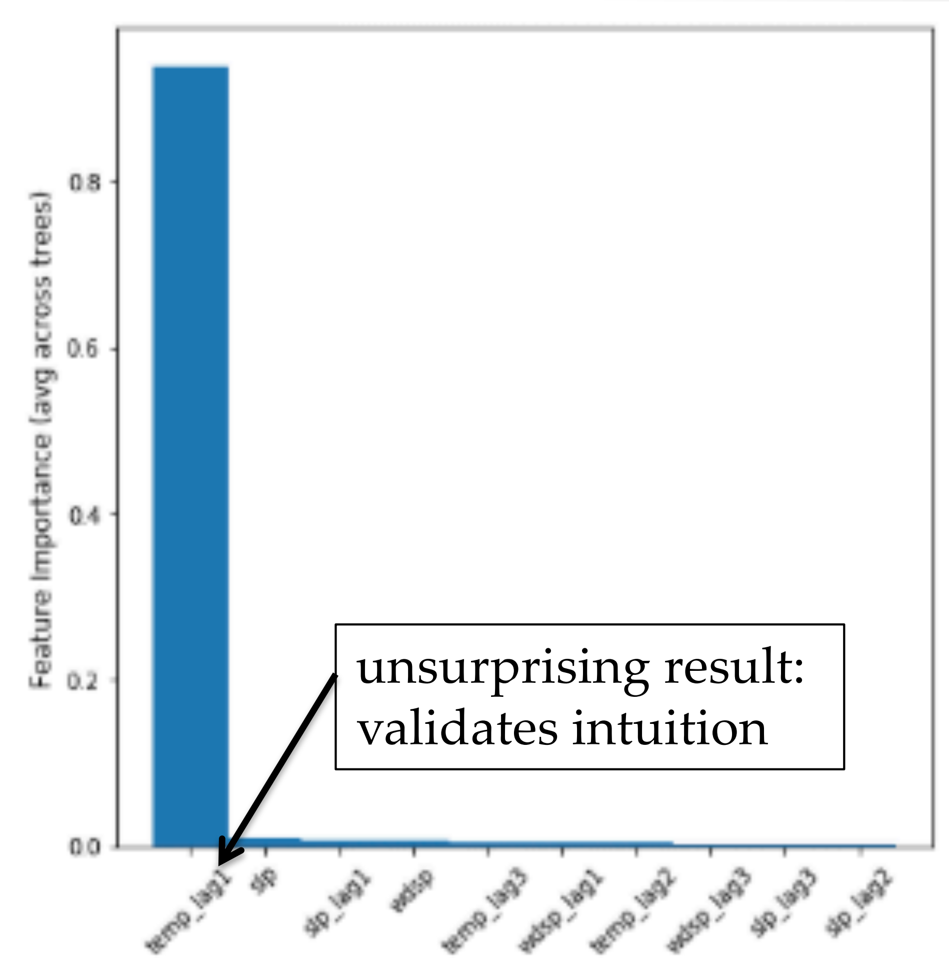
plotting true and predicted temperature

look at feature importances

Homework

- Congratulations!
- We showed that tomorrow's temperature is usually similar to today's (at this Canada station)





Takeaway for your projects and beyond:

Have some dataset of interest but it has < ~1M images?

1. Find a very large dataset that has similar data, train a big ConvNet there
2. Transfer learn to your dataset

Deep learning frameworks provide a “Model Zoo” of pretrained models so you don’t need to train your own

Caffe: https://github.com/BVLC/caffe/wiki/Model-Zoo
TensorFlow: https://github.com/tensorflow/models
PyTorch: https://github.com/pytorch/vision