CODING

## Goals:

- engineer features that highlight the time dimesionality of the data

- test out four different algorithms

%matplotlib inline

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from warnings import filterwarnings

filterwarnings('ignore')

#load dataset

df=pd.read\_csv('dengue\_features\_train.csv')

labels = pd.read\_csv('dengue\_labels\_train.csv')

test = pd.read\_csv('dengue\_features\_test.csv')

#fill NaNs

df.fillna(method='ffill', inplace=True)

test.fillna(method='ffill', inplace=True)

#to datetime

df['week\_start\_date']=pd.to\_datetime(df['week\_start\_date'])

test['week\_start\_date']=pd.to\_datetime(test['week\_start\_date'])

#extract month to new column

df['month']=df.week\_start\_date.dt.month

test['month']=test.week\_start\_date.dt.month

df=pd.merge(df, labels, on=['city', 'year', 'weekofyear'])

### preprocessing - just experimenting here to figure out what features will help increase predictive performance

lag columns, take rolling averages, historical dengue counts

#idea historical average dengue counts and climate for that week in the year

#for each city, on that week, what is the avg num cases over the years?

df=df.join(df.groupby(['city','weekofyear'])['total\_cases'].mean(), on=['city','weekofyear'], rsuffix='\_avg')

test=test.join(df.groupby(['city','weekofyear'])['total\_cases'].mean(), on=['city','weekofyear'], rsuffix='\_avg')

#quick column fix

test.rename(columns={'total\_cases': 'total\_cases\_avg'}, inplace=True)

test.head()

# #plan to lag these columns 3 weeks

# #this lag did not greatly improve model performance, the rolling avg was better

# cols\_to\_lag=[

# 'precipitation\_amt\_mm',

# 'reanalysis\_air\_temp\_k',

# 'reanalysis\_avg\_temp\_k',

# 'reanalysis\_dew\_point\_temp\_k',

# 'reanalysis\_max\_air\_temp\_k',

# 'reanalysis\_min\_air\_temp\_k',

# 'reanalysis\_precip\_amt\_kg\_per\_m2',

# 'reanalysis\_relative\_humidity\_percent',

# 'reanalysis\_sat\_precip\_amt\_mm',

# 'reanalysis\_specific\_humidity\_g\_per\_kg',

# 'reanalysis\_tdtr\_k',

# 'station\_precip\_mm',

# ]

# for col in cols\_to\_lag:

# df['lagged\_'+col] = df[col].shift(2)

# for col in cols\_to\_lag:

# test['lagged\_'+col] = test[col].shift(2)

# lagged\_cols=[

# 'lagged\_precipitation\_amt\_mm',

# 'lagged\_reanalysis\_air\_temp\_k',

# 'lagged\_reanalysis\_avg\_temp\_k',

# 'lagged\_reanalysis\_dew\_point\_temp\_k',

# 'lagged\_reanalysis\_max\_air\_temp\_k',

# 'lagged\_reanalysis\_min\_air\_temp\_k',

# 'lagged\_reanalysis\_precip\_amt\_kg\_per\_m2',

# 'lagged\_reanalysis\_relative\_humidity\_percent',

# 'lagged\_reanalysis\_sat\_precip\_amt\_mm',

# 'lagged\_reanalysis\_specific\_humidity\_g\_per\_kg',

# 'lagged\_reanalysis\_tdtr\_k',

# 'lagged\_station\_precip\_mm',

# ]

# #fill NaNs after the lag

# df=df.fillna(df.groupby("month").transform(lambda x: x.fillna(x.mean())))

# test=test.fillna(test.groupby("month").transform(lambda x: x.fillna(x.mean())))

rolling\_cols\_sum=[

'precipitation\_amt\_mm',

'reanalysis\_sat\_precip\_amt\_mm',

'station\_precip\_mm'

]

rolling\_cols\_avg=[

'ndvi\_ne',

'ndvi\_nw',

'ndvi\_se',

'ndvi\_sw',

'reanalysis\_air\_temp\_k',

'reanalysis\_avg\_temp\_k',

'reanalysis\_dew\_point\_temp\_k',

'reanalysis\_max\_air\_temp\_k',

'reanalysis\_min\_air\_temp\_k',

'reanalysis\_precip\_amt\_kg\_per\_m2',

'reanalysis\_relative\_humidity\_percent',

'reanalysis\_specific\_humidity\_g\_per\_kg',

'reanalysis\_tdtr\_k',

'station\_avg\_temp\_c',

'station\_diur\_temp\_rng\_c',

'station\_max\_temp\_c',

'station\_min\_temp\_c'

]

#loop to make the columns with rolling averages on independent vars

#takes the avg of prior 3 or 4weeks

for col in rolling\_cols\_sum:

df['rolling\_sum\_'+col] = pd.rolling\_sum(df[col], 3)

test['rolling\_sum\_'+col] = pd.rolling\_sum(test[col], 3)

#loop to make the columns with rolling averages on independent vars

#takes the avg of prior 3 weeks

for col in rolling\_cols\_avg:

df['rolling\_avg\_'+col] = pd.rolling\_mean(df[col], 3)

test['rolling\_avg\_'+col] = pd.rolling\_mean(test[col], 3)

# # #engineer column to identify worst months for dengue in each location

# #this didn't help much either so am commenting it out here

# def bad\_mo\_sj (x):

# if x == 10: return 1

# if x == 11: return 1

# if x == 9: return 1

# return 0

# def bad\_mo\_iq (x):

# if x == 1: return 1

# if x == 12: return 1

# if x == 2: return 1

# return 0

# #create the new column

# sj['key\_months'] = sj.month.apply(bad\_mo\_sj)

# iq['key\_months'] = iq.month.apply(bad\_mo\_iq)

# sj\_test['key\_months'] = sj\_test.month.apply(bad\_mo\_sj)

# iq\_test['key\_months'] = iq\_test.month.apply(bad\_mo\_iq)

### I tried out many, many combinations before selecting the features below

features=[

'total\_cases\_avg',

'rolling\_avg\_reanalysis\_specific\_humidity\_g\_per\_kg',

'rolling\_avg\_station\_avg\_temp\_c',

'rolling\_avg\_reanalysis\_dew\_point\_temp\_k',

'rolling\_avg\_station\_min\_temp\_c',

'rolling\_avg\_station\_max\_temp\_c',

'rolling\_avg\_reanalysis\_min\_air\_temp\_k',

'rolling\_avg\_reanalysis\_max\_air\_temp\_k',

'rolling\_avg\_reanalysis\_air\_temp\_k',

'rolling\_avg\_reanalysis\_avg\_temp\_k',

'reanalysis\_specific\_humidity\_g\_per\_kg',

'reanalysis\_dew\_point\_temp\_k',

'reanalysis\_min\_air\_temp\_k',

'station\_min\_temp\_c'

]

#fill resulting NaNs from the lag functions

df.fillna(method='bfill', inplace=True)

test.fillna(method='bfill', inplace=True)

df.to\_csv('train\_edited.csv')

# separate san juan and iquitos

sj = df[df['city']=='sj']

iq = df[df['city']=='iq']

sj\_test=test[test['city']=='sj']

iq\_test=test[test['city']=='iq']

## Algorithms for comparison

from sklearn import model\_selection

from sklearn import linear\_model

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.svm import SVR

sj\_train\_subtrain = sj.head(800)

sj\_train\_subtest = sj.tail(sj.shape[0] - 800)

iq\_train\_subtrain = iq.head(400)

iq\_train\_subtest = iq.tail(iq.shape[0] - 400)

#code reference: Machine Learning Mastery - http://machinelearningmastery.com/

#set x and y

X= sj[features]

Y = sj['total\_cases']

# prepare configuration for cross validation test harness

seed = 7

# prepare models

models = []

models.append(('LR', linear\_model.LinearRegression()))

models.append(('DT', DecisionTreeRegressor()))

models.append(('RFR', RandomForestRegressor()))

models.append(('SVR', SVR()))

# evaluate each model in turn

results = []

names = []

scoring = 'neg\_mean\_absolute\_error'

for name, model in models:

kfold = model\_selection.KFold(n\_splits=10, random\_state=seed)

cv\_results = model\_selection.cross\_val\_score(model, X, Y, cv=kfold, scoring='neg\_mean\_absolute\_error')

results.append(cv\_results)

names.append(name)

msg = "%s: %f (%f)" % (name, cv\_results.mean(), cv\_results.std())

print(msg)

# boxplot algorithm comparison

fig = plt.figure()

fig.suptitle('Algorithm Comparison')

ax = fig.add\_subplot(111)

plt.boxplot(results)

ax.set\_xticklabels(names)

plt.show()

## IQUITOS

X= iq[features]

Y = iq['total\_cases']

# prepare configuration for cross validation test harness

seed = 7

# prepare models

models = []

models.append(('LR', linear\_model.LinearRegression()))

models.append(('DT', DecisionTreeRegressor()))

models.append(('RFR', RandomForestRegressor()))

models.append(('SVR', SVR()))

# evaluate each model in turn

results = []

names = []

scoring = 'neg\_mean\_absolute\_error'

for name, model in models:

kfold = model\_selection.KFold(n\_splits=10, random\_state=seed)

cv\_results = model\_selection.cross\_val\_score(model, X, Y, cv=kfold, scoring='neg\_mean\_absolute\_error')

results.append(cv\_results)

names.append(name)

msg = "%s: %f (%f)" % (name, cv\_results.mean(), cv\_results.std())

print(msg)

# boxplot algorithm comparison

fig = plt.figure()

fig.suptitle('Algorithm Comparison')

ax = fig.add\_subplot(111)

plt.boxplot(results)

ax.set\_xticklabels(names)

plt.show()

### visualize the importances of selected features

X= sj[features]

Y = sj['total\_cases']

model =DecisionTreeRegressor()

model.fit(X,Y)

model.predict(sj\_test[features])

importances=model.feature\_importances\_

indices = np.argsort(importances)[::-1]

# Print the feature ranking

print("Feature ranking:")

for f in range(X.shape[1]):

print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))

feature\_names = X.columns

f, ax = plt.subplots(figsize=(7, 4))

plt.title("Feature ranking", fontsize = 14)

plt.bar(range(X.shape[1]), importances[indices],

color="b",

align="center")

plt.xticks(range(X.shape[1]), feature\_names, rotation=90 )

plt.xlim([-1, X.shape[1]])

plt.ylabel("importance", fontsize = 10)

plt.xlabel("index of the feature", fontsize = 10)

X= iq[features]

Y = iq['total\_cases']

model =DecisionTreeRegressor()

model.fit(X,Y)

model.predict(iq\_test[features])

importances=model.feature\_importances\_

indices = np.argsort(importances)[::-1]

# Print the feature ranking

print("Feature ranking:")

for f in range(X.shape[1]):

print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))

feature\_names = X.columns

f, ax = plt.subplots(figsize=(7, 4))

plt.title("Feature ranking", fontsize = 14)

plt.bar(range(X.shape[1]), importances[indices],

color="b",

align="center")

plt.xticks(range(X.shape[1]), feature\_names, rotation=90 )

plt.xlim([-1, X.shape[1]])

plt.ylabel("importance", fontsize = 10)

plt.xlabel("index of the feature", fontsize = 10)

### perform gridsearch on support vector regressor

from sklearn.model\_selection import GridSearchCV

import time as time

X\_sj= sj[features]

Y\_sj = sj['total\_cases']

X\_iq= iq[features]

Y\_iq = iq['total\_cases']

train\_size = 100

svr = GridSearchCV(SVR(kernel='rbf', gamma=0.1), cv=5,

param\_grid={"C": [1e0, 1e1, 1e2, 1e3],

"gamma": np.logspace(-2, 2, 5)})

t0 = time.time()

svr.fit(X\_sj,Y\_sj)

svr\_fit = time.time() - t0

print("SVR complexity and bandwidth selected and model fitted in %.3f s"

% svr\_fit)

model\_svr\_sj=svr.best\_estimator\_

model\_svr\_sj

train\_size = 100

svr = GridSearchCV(SVR(kernel='rbf', gamma=0.1), cv=5,

param\_grid={"C": [1e0, 1e1, 1e2, 1e3],

"gamma": np.logspace(-2, 2, 5)})

t0 = time.time()

svr.fit(X\_iq,Y\_iq)

svr\_fit = time.time() - t0

print("SVR complexity and bandwidth selected and model fitted in %.3f s"

% svr\_fit)

model\_svr\_iq=svr.best\_estimator\_

model\_svr\_iq

model\_svr\_sj.fit(X\_sj,Y\_sj)

model\_svr\_iq.fit(X\_iq,Y\_iq)

# test\_model=SVR(C=10.0, cache\_size=200, coef0=0.0, degree=3, epsilon=0.1,

# gamma=0.10000000000000001, kernel='rbf', max\_iter=-1, shrinking=True,

# tol=0.001, verbose=False)

# test\_model\_iq=SVR(C=10.0, cache\_size=200, coef0=0.0, degree=3, epsilon=0.1,

# gamma=0.10000000000000001, kernel='rbf', max\_iter=-1, shrinking=True,

# tol=0.001, verbose=False)

# test\_model.fit(sj\_train\_subtrain[features], sj\_train\_subtrain['total\_cases'])

# test\_model\_iq.fit(iq\_train\_subtrain[features], iq\_train\_subtrain['total\_cases'])

preds\_sj\_svr= model\_svr\_sj.predict(sj\_train\_subtest[features]).astype(int)

preds\_iq\_svr=model\_svr\_iq.predict(iq\_train\_subtest[features]).astype(int)

sj\_train\_subtest['fitted'] = preds\_sj\_svr

iq\_train\_subtest['fitted'] = preds\_iq\_svr

sj\_train\_subtest['fitted'] = preds\_sj\_svr

iq\_train\_subtest['fitted'] = preds\_iq\_svr

### reset axis

sj\_train\_subtest.index = sj\_train\_subtest['week\_start\_date']

iq\_train\_subtest.index = iq\_train\_subtest['week\_start\_date']

sj\_train\_subtest['fitted'] = preds\_sj\_svr

iq\_train\_subtest['fitted'] = preds\_iq\_svr

### reset axis

sj\_train\_subtest.index = sj\_train\_subtest['week\_start\_date']

iq\_train\_subtest.index = iq\_train\_subtest['week\_start\_date']

figs, axes = plt.subplots(nrows=2, ncols=1, figsize=(11, 9))

sj\_train\_subtest.total\_cases.plot(ax=axes[0], label="Actual")

sj\_train\_subtest.fitted.plot(ax=axes[0], label="Predictions")

iq\_train\_subtest.total\_cases.plot(ax=axes[1], label="Actual")

iq\_train\_subtest.fitted.plot(ax=axes[1], label="Predictions")

plt.suptitle("Dengue Predicted Cases vs. Actual Cases")

plt.legend()

### clearly the model is unable to predict the spikes in disease outbreak

Need to somehow account for the cases in previous weeks

Idea: use historical counts and/or predicted vals in the testing set where we have no counts and train the model using previous week counts as a feature

preds\_sj\_svr= model\_svr\_sj.predict(sj\_test[features]).astype(float)

preds\_iq\_svr=model\_svr\_iq.predict(iq\_test[features]).astype(float)

submission = pd.read\_csv("submission\_format.csv",

index\_col=[0, 1, 2])

submission.total\_cases = np.concatenate([preds\_sj\_svr, preds\_iq\_svr])

#add a total cases column to the test df

test['total\_cases'] = np.concatenate([preds\_sj\_svr, preds\_iq\_svr])

submission.to\_csv("svr.csv")

### create dummy predictions for the test set for the next model iteration

The reason I added in the dataset avg is because the predictions were too low. It seems like the model needs to know previous actual case counts before it can perform well.

test['total\_cases']=test['total\_cases']+test['total\_cases\_avg']

test['total\_cases']=test['total\_cases']+test['total\_cases\_avg']

test['random']=np.random.uniform(low=0.8, high=1.5, size=len(test))

#add some randomness

test['total\_cases']=test['total\_cases\_avg']\*test['random']

#save file

test.to\_csv("test\_preds\_added.csv")