Day 4 ¶

https://www.analyticsvidhya.com/blog/2018/09/non-stationary-time-series-python/ (https://www.analyticsvidhya.com/blog/2018/09/non-stationary-time-series-python/)

In [1]:
```
%pylab inline

from xgboost.sklearn import XGBClassifier
from typing import Tuple
import xgboost as xgb
import pandas as pd
import json
import sklearn
from funcy import *
from typing import List
```

```
Populating the interactive namespace from numpy and matplotlib
```

In [2]:
```
day0 = pd.read_csv('data/longitudinal/0.csv')
```

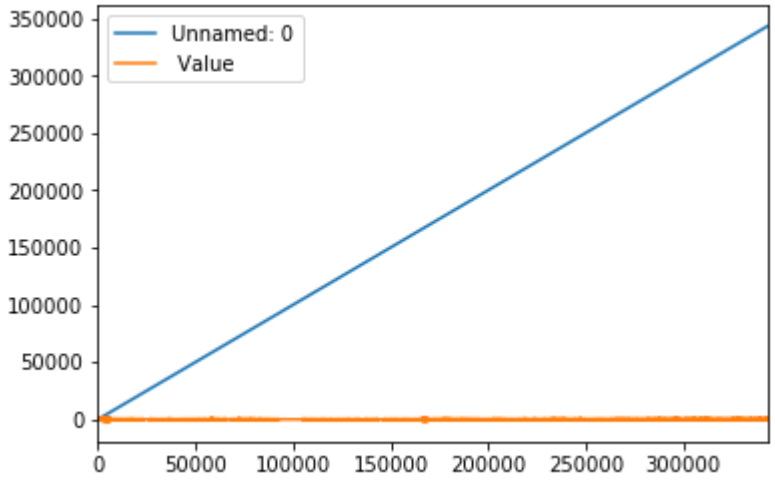
In [3]:
```
day0.head()
```

Out[3]:

	Unnamed: 0	Time	Value	event name
0	0	15:49:43	11.0	blinkInstruction
1	1	15:49:43	13.0	blinkInstruction
2	2	15:49:43	16.0	blinkInstruction
3	3	15:49:43	18.0	blinkInstruction
4	4	15:49:43	19.0	blinkInstruction

In [4]:
```
day0.plot()
```

Out[4]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x118c9dac8>
```



In [5]:
```
day0['event name'].unique()
```

Out[5]:
```
array(['blinkInstruction', 'blink1', 'blink2', 'blink3', 'blink4',
       'blink5', 'relaxInstruction', 'relax', 'mathInstruction', 'math1',
       'math2', 'math3', 'math4', 'math5', 'math6', 'math7', 'math8',
       'math9', 'math10', 'math11', 'math12', 'musicInstruction', 'music',
       'videoInstruction', 'video-ver2', 'thinkOfItemsInstruction-ver2',
       'thinkOfItems-ver2', 'colorInstruction1', 'colorInstruction2',
       'readyRound1', 'colorRound1-1', 'colorRound1-2', 'colorRound1-3',
       'colorRound1-4', 'colorRound1-5', 'colorRound1-6', 'readyRound2',
       'colorRound2-1', 'colorRound2-2', 'colorRound2-3', 'colorRound2-4',
       'colorRound2-5', 'colorRound2-6', 'readyRound3', 'colorRound3-1',
       'colorRound3-2', 'colorRound3-3', 'colorRound3-4', 'colorRound3-5',
       'colorRound3-6', 'readyRound4', 'colorRound4-1', 'colorRound4-2',
       'colorRound4-3', 'colorRound4-4', 'colorRound4-5', 'colorRound4-6',
       'readyRound5', 'colorRound5-1', 'colorRound5-2', 'colorRound5-3',
       'colorRound5-4', 'colorRound5-5', 'unlabeled'], dtype=object)
```

In [6]:
```
def chunk (df: pd.DataFrame) -> List:
    return list(partition(256, df[' Value']))
```

In [7]:
```
relax_chunks = chunk(day0[day0['event name'].str.contains('colorRound')])
math_chunks = chunk(day0[day0['event name'].str.contains('math')])
```

In [8]:
```
np.array(relax_chunks).shape, np.array(math_chunks).shape
```

Out[8]:
```
((148, 256), (70, 256))
```

Now, we'll turn these data into our feature/label matrices `X` and `y`.

In [9]:
```
X = np.concatenate([relax_chunks, math_chunks])
```

In [10]:
```
X.shape
```

Out[10]:
```
(218, 256)
```

In [11]:
```
y = np.array([ 0 for x in relax_chunks] + [ 1 for x in math_chunks])
assert X.shape[0] == y.shape[0]
```

Let's work together to make a function that takes a given `day` of readings, and makes a corpus of data `X` and labels `y` for two tasks.

In [12]:
```
day = []
def getXy(
    num:int,
    task1:str,
    task2:str
         ):
    numstr = str(num)
    day = pd.read_csv('data/longitudinal/'+numstr+'.csv')
    day.head()
    day['event name'].unique()
    relax_chunks = chunk(day[day['event name'].str.contains(task1)])
    math_chunks = chunk(day[day['event name'].str.contains(task2)])
    X = np.concatenate([relax_chunks, math_chunks])
    y = np.array([ 0 for x in relax_chunks] + [ 1 for x in math_chunks])
    assert X.shape[0] == y.shape[0]
    return X,y
```

In [13]:
```
X,y = getXy(0,'colorRound','math')
X.shape, y.shape
```

Out[13]:
```
((218, 256), (218,))
```

Note that we are assigning `0` to "positive" examples, and `1` to "negative" examples. That means `0` will mean "ACCEPT" and `1` will mean "REJECT."

TODO

Now, train and test a classifier! Estimate your classifier's accuracy. Produce a classifier named `clf` trained on day0's data -- `'math'` gainst `'blink'`. Use the function we wrote together.

In [14]:
```
# get day0's data -- 'math' gainst 'blink'.
X0,y0 = getXy(0,'blink','math')
```

In [15]:
```
X0.shape, y0.shape
```

Out[15]:
```
((93, 256), (93,))
```

In [24]:
```
def fresh_clf () -> XGBClassifier:
    return XGBClassifier(
        # Don't worry about those parameters for now,
        # though feel free to look them up if you're interested.
        reg_alpha=0.8,
        objective= 'binary:logistic',
        seed=27)

clf = fresh_clf()
```

In [25]:
```
def xgb_cross_validate (
    X: np.array,
    y: np.array,
    nfold: int=7
) -> Tuple[XGBClassifier, pd.DataFrame]:
    # eval_metrics:
    # http://xgboost.readthedocs.io/en/latest//parameter.html
    metrics = ['error@0.1', 'auc']
#     metrics = [ 'auc' ]
    # we use the @ syntax to override the default of 0.5 as the threshold for 0 / 1 classification
    # the intent here to to minimize FAR at the expense of FRR
    alg = fresh_clf()

    xgtrain = xgb.DMatrix(X,y)
    param = alg.get_xgb_params()
    cvresults = xgb.cv(param,
                       xgtrain,
                       num_boost_round=alg.get_params()['n_estimators'],
                       nfold=nfold,
                       metrics=metrics,
                       early_stopping_rounds=100
                       )
    alg.set_params(n_estimators=cvresults.shape[0])
    alg.fit(X,y,eval_metric=metrics)
    return alg, cvresults
```

In [26]:
```
X0_train, X0_validate, y0_train, y0_validate = sklearn.model_selection.train_test_split(
    X0, y0,
    test_size=0.33,
    random_state=42)

clf0, cvres0 = xgb_cross_validate(X0_train, y0_train)
```

In [27]:
```
clf0.score(X0_train, y0_train), clf0.score(X0_validate, y0_validate)
```

Out[27]:
```
(1.0, 0.7419354838709677)
```

Now, these results might be good.

But our classifier's accuracy could be misleading.

Can you see why?

Nonstationarity

We are training, and testing, using data recorded over a single session. As we know, EEG changes over time, a property known as *nonstationarity*. Will our great results still hold a few weeks later?

In [20]:
```
# day52 = pd.read_csv('data/longitudinal/52.csv')

X52, y52 = getXy(52, "math", "blink")

clf0.score(X52, y52)
```

Out[20]:
```
0.21739130434782608
```

TODO

Create a systematic way of understanding how classification accuracy changes as a function of time.

Some questions to spur investigation:

- How does classifier accuracy for day 0's classifier perform on day 1? Day 5? Day 20?

- What features of readings cause a classifier that works on earlier recordings fail on later ones?
- What features remain the same? Are there any?
- What might be the source of these changing features? Changing placement in the EEG device? Changing properties of the brain?

Please note below all work you do, and any notes you make along the way. Ideally, your work should read like a story - words (and questions!) interspersed with code. Good luck, and have fun!

Q1: How does classifier accuracy for day 0's classifier perform on day 1? Day 5? Day 20?

In [21]:
```
X1, y1 = getXy(1, "math", "blink")
X5, y5 = getXy(5, "math", "blink")
X20, y20 = getXy(20, "math", "blink")
clf0.score(X0_validate, y0_validate), clf0.score(X1, y1), clf0.score(X5, y5), clf0.score(X20, y20)
```

Out[21]:
```
(0.7419354838709677,
 0.20212765957446807,
 0.21739130434782608,
 0.21505376344086022)
```

Q2: What features of readings cause a classifier that works on earlier recordings fail on later ones?

The nonstationarity of readings cause a classifier that works on earlier recordings fail on later ones

Use more data to train the model

In [22]:
```
def daysXy(days:int):
    X = []
    y = []
    for index in range(days):
        apX, apy = getXy(index, "math", "blink")
        X.extend(apX)
        y.extend(apy)
    X = np.array(X)
    y = np.array(y)
    print(X.shape, y.shape)
    return X, y
```

In [23]:
```
X0_1, y0_1 = daysXy(2)
```

```
(187, 256) (187,)
```

In [24]:
```
X0_1_train, X0_1_validate, y0_1_train, y0_1_validate = sklearn.model_selection.train_test_split(
    X0_1, y0_1,
    test_size=0.33,
    random_state=42)

clf0_1, cvres0_1 = xgb_cross_validate(X0_1_train, y0_1_train)
clf0_1.score(X52,y52)
```

Out[24]:
```
0.7608695652173914
```

In [25]:
```
def clfXy(days:int):
    Xdays, ydays = daysXy(days)
    Xdays_train, Xdays_validate, ydays_train, ydays_validate = sklearn.model_selection.train_test_split(
    Xdays, ydays,
    test_size=0.33,
    random_state=42)
    clfdays, cvresdays = xgb_cross_validate(Xdays_train, ydays_train)
    print("The accuracy on X52, y52 of the model using",days,"days data is:",clfdays.score(X52, y52))
    return clfdays.score(X52, y52)
```

In [26]:
```
print(clfXy(2)==clf0_1.score(X52,y52))
```

```
(187, 256) (187,)
The accuracy on X52, y52 of the model using 2 days data is: 0.7608695652173914
True
```

In [27]:
```
clfXy(2), clfXy(11), clfXy(21), clfXy(30)
```

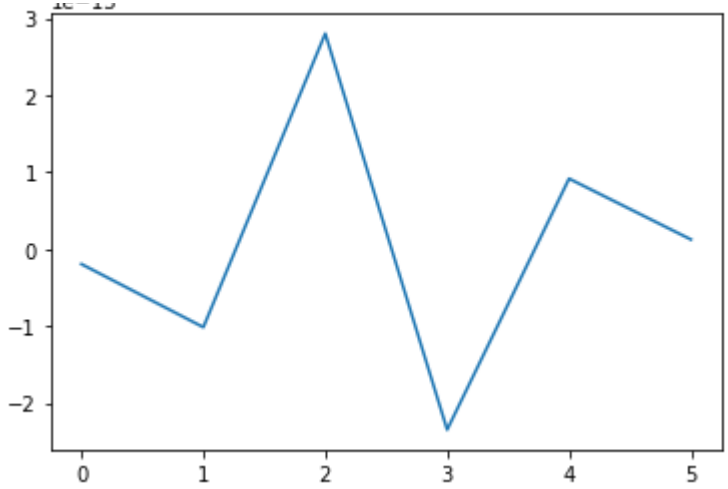
```
(187, 256) (187,)
The accuracy on X52, y52 of the model using 2 days data is: 0.7608695652173914
(1019, 256) (1019,)
The accuracy on X52, y52 of the model using 11 days data is: 0.8152173913043478
(1946, 256) (1946,)
The accuracy on X52, y52 of the model using 21 days data is: 0.8043478260869565
(2779, 256) (2779,)
The accuracy on X52, y52 of the model using 30 days data is: 0.7934782608695652
```

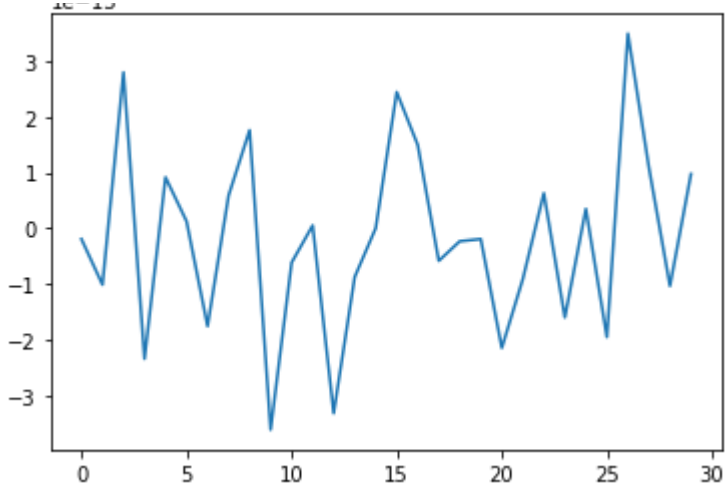
Out[27]:
```
(0.7608695652173914,
 0.8152173913043478,
 0.8043478260869565,
 0.7934782608695652)
```

The classifier accuracy for day 0's classifier on day 52 is only 0.21739130434782608. By contrast, the classifier accuracy for mutiple days' classifier on day 52 is significantly higher, which shows the nonstationarity of neural signals.

In [28]:
```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

estimator1 = PCA(n_components=1)   # Initialization -> compress the data to be 1-dimensional

X = list(range(0,30))
y = list(range(0,30))
pca_X = list(range(0,30))
meanX = list(range(0,30))

for n in range(0,30):
    X[n], y[n] = getXy(n, "math", "blink")
    pca_X[n] = estimator1.fit_transform(X[n])
    meanX[n] = np.mean(pca_X[n])
spca = meanX
sday = list(range(0,30))

X0, y0 = getXy(0, "math", "blink")
X1, y1 = getXy(1, "math", "blink")
X2, y2 = getXy(2, "math", "blink")
X3, y3 = getXy(3, "math", "blink")
X4, y4 = getXy(4, "math", "blink")
X5, y5 = getXy(5, "math", "blink")


pca_X0 = estimator1.fit_transform(X0)
pca_X1 = estimator1.fit_transform(X1)
pca_X2 = estimator1.fit_transform(X2)
pca_X3 = estimator1.fit_transform(X3)
pca_X4 = estimator1.fit_transform(X4)
pca_X5 = estimator1.fit_transform(X5)
pca = [np.mean(pca_X0),np.mean(pca_X1),np.mean(pca_X2),np.mean(pca_X3),np.mean(pca_X4),np.mean(pca_X5)]
day = [0,1,2,3,4,5]
plt.plot(day,pca)
plt.show()

plt.plot(sday,spca)
plt.show()

dataframe = pd.DataFrame({'day':sday,'1dX':spca})
```

1e−13


3
2
1
0
−1
−2
0
1
2
3
4
5



ADF (Augmented Dickey Fuller) Test [https://www.analyticsvidhya.com/blog/2018/09/non-stationary-time-series-python/ (https://www.analyticsvidhya.com/blog/2018/09/non-stationary-time-series-python/)](https://www.analyticsvidhya.com/blog/2018/09/non-stationary-time-series-python/)

Null Hypothesis: The series has a unit root (value of a =1)

Alternate Hypothesis: The series has no unit root.

If we fail to reject the null hypothesis, we can say that the series is non-stationary. This means that the series can be linear or difference stationary (we will understand more about difference stationary in the next section).

In [29]:
```
#define function for ADF test
from statsmodels.tsa.stattools import adfuller
def adf_test(timeseries):
    #Perform Dickey-Fuller test:
    print ('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Us
    for key,value in dftest[4].items():
       dfoutput['Critical Value (%s)'%key] = value
    print (dfoutput)

#apply adf test on the series
adf_test(dataframe['1dX'])
```

```
Results of Dickey-Fuller Test:
Test Statistic                -6.354569e+00
p-value                        2.561588e-08
#Lags Used                     0.000000e+00
Number of Observations Used    2.900000e+01
Critical Value (1%)           -3.679060e+00
Critical Value (10%)          -2.623158e+00
Critical Value (5%)           -2.967882e+00
dtype: float64
```

the test statistic > critical value, which implies that the series is not stationary.

KPSS (Kwiatkowski-Phillips-Schmidt-Shin) Test https://www.analyticsvidhya.com/blog/2018/09/non-stationary-time-series-python/ (https://www.analyticsvidhya.com/blog/2018/09/non-stationary-time-series-python/)

Null Hypothesis: The process is trend stationary.

Alternate Hypothesis: The series has a unit root (series is not stationary).

In [30]:
```
#define function for kpss test
from statsmodels.tsa.stattools import kpss
#define KPSS
def kpss_test(timeseries):
    print ('Results of KPSS Test:')
    kpsstest = kpss(timeseries, regression='c')
    kpss_output = pd.Series(kpsstest[0:3], index=['Test Statistic','p-value','Lags Used'])
    for key,value in kpsstest[3].items():
        kpss_output['Critical Value (%s)'%key] = value
    print(kpss_output)
kpss_test(dataframe['1dX'])
```

```
Results of KPSS Test:
Test Statistic           0.230631
p-value                  0.100000
Lags Used                9.000000
Critical Value (1%)      0.739000
Critical Value (10%)     0.347000
Critical Value (5%)      0.463000
Critical Value (2.5%)    0.574000
dtype: float64

/Users/yangzeyu/anaconda3/envs/tensorflow/lib/python3.5/site-packages/statsmodels/tsa/stattools.py:1685: Futu
reWarning: The behavior of using lags=None will change in the next release. Currently lags=None is the same a
s lags='legacy', and so a sample-size lag length is used. After the next release, the default will change to
be the same as lags='auto' which uses an automatic lag length selection method. To silence this warning, eith
er use 'auto' or 'legacy'
  warn(msg, FutureWarning)
/Users/yangzeyu/anaconda3/envs/tensorflow/lib/python3.5/site-packages/statsmodels/tsa/stattools.py:1710: Inte
rpolationWarning: p-value is greater than the indicated p-value
  warn("p-value is greater than the indicated p-value", InterpolationWarning)
```

The test statistic < the critical value, we fail to reject the null hypothesis (series is stationary).

So in summary, the ADF test has an alternate hypothesis of linear or difference stationary, while the KPSS test identifies trend-stationarity in a series.

The series is not linear-stationarity but trend-stationarity.

Q3: What features remain the same? Are there any?

Noisy

Without PCA

In [31]:
```
from sklearn.metrics import classification_report
y0_1_pred = clf0_1.predict(X0_1_validate)
print(classification_report(y0_1_validate, y0_1_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.94      0.92        51
           1       0.67      0.55      0.60        11

    accuracy                           0.87        62
   macro avg       0.79      0.74      0.76        62
weighted avg       0.86      0.87      0.87        62
```

With PCA

In [32]:
```
from sklearn.decomposition import PCA
estimator = PCA(n_components=20)   # Initialization -> compress the data to be 20-dimensional
pca_X0_1_train = estimator.fit_transform(X0_1_train)
pca_X0_1_train.shape
```

Out[32]: (125, 20)

In [33]:
```
pca_X0_1_validate = estimator.transform(X0_1_validate)
pca_X0_1_validate.shape
```

Out[33]: (62, 20)

In [34]:
```
clf = fresh_clf()
clf.fit(pca_X0_1_train,y0_1_train)
y0_1_predict = clf.predict(pca_X0_1_validate)
print(classification_report(y0_1_validate, y0_1_predict))
```

```
              precision    recall  f1-score   support

           0       0.89      0.92      0.90        51
           1       0.56      0.45      0.50        11

    accuracy                           0.84        62
   macro avg       0.72      0.69      0.70        62
weighted avg       0.83      0.84      0.83        62
```

PCA will lose a little bit of predictive accuracy (about 0.03), because in the process of dimensionality reduction, although noise is avoided, some useful nformation is also avoided.

But the dimensional compression not only saves a lot of model training time, but also reduces the training difficulty of the model, It is a effective choice for high-dimensional samples.

Nonlinear

To illustrate this a linear regression model is used.

In [35]:
```
import numpy as np
from sklearn.linear_model import LinearRegression
```

In [36]:
```
model = LinearRegression()
```

In [37]:
```
model.fit(X0_1_train, y0_1_train)
```

Out[37]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [38]:
```
model.score(X0_1_validate, y0_1_validate)
```

Out[38]:
```
-15.063133260670124
```

The result is not satisfactory. It shows neural signals are nonlinear.

Q4: What might be the source of these changing features? Changing placement in the EEG device? Changing properties of the brain?

Both of them could be the source of the changing features. The source of the changing features could be multiple.

However, I guess it is mainly because the the brain memorizes the stimulus that is given before. The brain learns the feature of previous stimulus, which causes a more significant neural signal when a a similar stimulus is given again. Using the control variable method to eliminate the other factors, the relationship between the different days of stimulation and the neural signal can be studied.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: