Day 3: Passthoughts

In [1]:
```
%pylab inline
import json
from xgboost.sklearn import XGBClassifier
import xgboost as xgb
import sklearn
import pandas as pd
from typing import Tuple
from typing import List

dataset = pd.read_csv('data/eeg-data.csv',
                      parse_dates=['indra_time'],
                      index_col='indra_time')
# convert to arrays from strings
dataset.raw_values = dataset.raw_values.map(json.loads)
```

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

What if you could simply *think your password*? That's the premise behind *passthoughts*. We'll discuss passthoughts in more depth in lecture 3, but for now, we'll lay this out as a classification problem:

> Given a reading, and a person, is that person who they claim to be?

Build a passthought authenticator for participant 1. Pretend they use the "colorRound" task as their passthought.

Here, I'll get you started.

In [2]:
```
# All the readings during the "relax" task
color1 = dataset[(dataset.label.str.contains('colorRound')) &
                 (dataset.id == 1) ]
# all the readings from subjects aside from subj 1
other = dataset[(dataset.id != 1) ].sample(100)
```

In [11]:

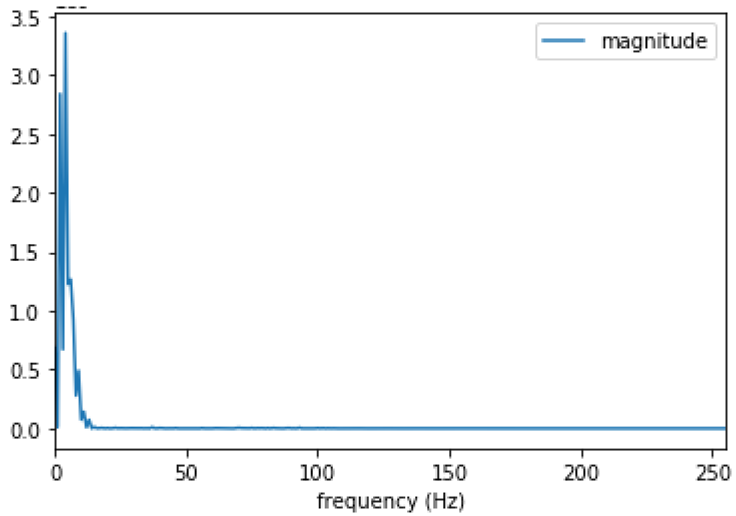
```python
def to_power_spectrum (
    raw_readings: np.array,
    sampling_rate: int = 512,
) -> pd.Series:
    '''
    Take raw voltages,
    transform into frequency domain,
    return a power spectrum.
    '''
    # FFT the raw readings
    fftd = np.fft.fft(raw_readings)
    # take absolute value
    # producing a symmetrical power spectrum
    ps = np.abs(fftd)**2
    # since the power spectrum is symmetrical,
    # take half
    half_len = len(ps)//2
    ps = ps[:half_len]
#     # we'll calculate the frequencies
    window_size = len(raw_readings)
    freqs = numpy.fft.fftfreq(window_size, d=1/sampling_rate)
#     # splitting that in half to match
    freqs = freqs[:half_len]
    power_spectrum = pd.DataFrame({
        'frequency (Hz)': freqs,
        'magnitude': ps,
    })
    return power_spectrum

ps = to_power_spectrum(dataset.raw_values[1000])
ps.plot(x='frequency (Hz)')
```

Out[11]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x131ded8d0>
```

1e9



In [4]:
```
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.
        objective= 'binary:logistic',
        seed=27)

clf = fresh_clf()
```

In [5]:
```
def to_features (
    df: pd.DataFrame
) -> np.array:
    power_specs = df.raw_values.apply(to_power_spectrum)
    return np.array([row.magnitude.values for row in power_specs])
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

For authentication, what we want even more than "accuracy" here are two metrics:

- False Acceptance Rate (FAR): The percentage of readings *not* from subject A incorrectly classified "ACCEPT."
- False Rejection Rate (FRR): The percentage of readings *from* subject A incorrectly classified 'REJECT."

For authentication /security/, we want FAR to be as low as possible (so nobody can break in). For authentication /usability/, we want FRR to be low (so user's don't get frustrated constantly re-trying their passthought).

Question 1

How well does the authenticator perform against participants other than participant 1? What is its false acceptance rate (FAR)? What is its false rejection rate (FRR)?

In [6]:
```
par1color1 = dataset[(dataset.label.str.contains('colorRound')) &
                     (dataset.id == 1)]
other1color1 = dataset[(dataset.label.str.contains('colorRound')) &
                       (dataset.id != 1)]
len(par1color1), len(other1color1)
```

Out[6]:
```
(79, 2326)
```

In [7]:
```
par1color1_features = to_features(par1color1)
other1color1_features = to_features(other1color1)
features = np.concatenate([par1color1_features, other1color1_features])

assert np.all( [ len(feat) == 256 for feat in features ] )
```

In [8]:
```
labels = np.array([ 0 for feature in par1color1_features  ] \
                + [ 1 for feature in other1color1_features ])

# list of labels should be the same
# as the number of features
assert len(labels) == len(features)
# first label in the list should be 0
assert labels[0] == 0
# last label in the list should be 1
assert labels[-1] == 1

labels[:5], labels[-5:]
```

Out[8]:
```
(array([0, 0, 0, 0, 0]), array([1, 1, 1, 1, 1]))
```

In [9]:
```
assert features.shape[0] == labels.shape[0]

features.shape, labels.shape
```

Out[9]:
```
((2405, 256), (2405,))
```

In [10]:
```
X = features
y = labels
```

In [11]:
```
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 [12]:
```
X_train, X_validate, y_train, y_validate = sklearn.model_selection.train_test_split(
    X, y,
    test_size=0.33,
    random_state=42)

clf, cvres = xgb_cross_validate(X_train, y_train)
```

In [13]:
```
cvres.tail()
```

Out[13]:

	test-auc-mean	test-auc-std	test-error@0.1-mean	test-error@0.1-std	train-auc-mean	train-auc-std	train-error@0.1-mean	train-error@0.1-std
95	0.811342	0.091243	0.033527	0.011567	1.0	0.0	0.027519	0.001820
96	0.811587	0.089553	0.033527	0.011567	1.0	0.0	0.027209	0.002084
97	0.810331	0.089167	0.033527	0.011567	1.0	0.0	0.027312	0.002042
98	0.811681	0.088968	0.033527	0.011567	1.0	0.0	0.026485	0.002154
99	0.810817	0.089406	0.033527	0.011567	1.0	0.0	0.025657	0.002011

In [14]:
```
clf.score(X_train, y_train)
```

Out[14]: 1.0

In [17]:
```
clf.score(X_validate, y_validate)
```

Out[17]: 0.9659949622166247

In [15]:
```
y_pred = clf.predict(X_validate)
```

In [16]:
```
from sklearn.metrics import classification_report
print(classification_report(y_validate, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        25
           1       0.97      1.00      0.98       769

    accuracy                           0.97       794
   macro avg       0.48      0.50      0.49       794
weighted avg       0.94      0.97      0.95       794
```

In [ ]:
```
# Nick
def far_frr (y_h, y_t):
    '''Find false acceptance rate and false rejection rate for predicted (y_h) against the actual (y_t)'''
    false_accept = 0
    false_reject = 0
    for predicted, actual in zip(y_h,y_t):
        # if we predicted it was subj 1, but it was actually not
        if predicted == 0 and actual == 1:
            # that's a false accept
            false_accept+=1
        # if we predicted it was not subject 1, but it actually was
        if predicted == 1 and actual == 0:
            # that's a false reject
            false_reject +=1
    # TODO
    # We're not sure
    # if it's better to compute the false accepts and rejects
    # with all predictions as denominator,
    # or with only true accept / true reject attempts as denominator.
    far = false_accept/len(y_h)
    frr = false_reject/len(y_h)
    return far, frr
```

(Bonus) Question 2

How well can the authenticator distinguish between participant 1 relaxing, and participant 1's other thoughts? Again, what is the FAR/FRR?

In [29]:
```
par1relax = dataset[(dataset.label.str.contains('colorRound')) &
                    (dataset.id == 1)]
par1other = dataset[~(dataset.label.str.contains('colorRound')) &
                       (dataset.id == 1)]
len(par1relax), len(par1other)
```

Out[29]:
```
(79, 861)
```

In [30]:
```
par1relax_features = to_features(par1relax)
par1other_features = to_features(par1other)

X = np.concatenate([par1relax_features, par1other_features])
y = np.array([ 0 for feature in par1relax_features ]\
             + [ 1 for feature in par1other_features ])

X_train, X_validate, y_train, y_validate = sklearn.model_selection.train_test_split(
    X,y,test_size=0.33,random_state=42)
clf, cvres = xgb_cross_validate(X_train, y_train)
clf.score(X_validate, y_validate)
```

Out[30]:
```
0.9131832797427653
```

In [31]:
```
y_pred = clf.predict(X_validate)
print(classification_report(y_validate, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        27
           1       0.91      1.00      0.95       284

    accuracy                           0.91       311
   macro avg       0.46      0.50      0.48       311
weighted avg       0.83      0.91      0.87       311


/Users/yangzeyu/anaconda3/envs/tensorflow/lib/python3.5/site-packages/sklearn/metrics/classification.py:1437:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicte
d samples.
  'precision', 'predicted', average, warn_for)
```

In [ ]:

(Bonus) Question 3

Notice how we structured our positive and negative examples:

- *Positive examples*: The right person thinking the right task.

- *Negative examples*: The wrong person thinking any task (whether it is right or wrong).

In the context of passthoughts, consider other possibilites for selecting positive and negative features. Here, (1) pick one configuration of positive and negative examples, aside from the ones listed, and (2) discuss their possible consequences (pros/cons). Explain how you might evaluate this selection (with data, with user experiments, etc - your choice).

In [21]:
```
par1math = dataset[(dataset.label.str.match('math\d')) &
                   (dataset.id == 1) ]
others = dataset[dataset.id != 1]
len(par1math), len(others)
```

Out[21]:
```
(30, 29073)
```

In [22]:
```
par1math_features = to_features(par1math)
others_features = to_features(others)
features = np.concatenate([par1math_features, others_features])

assert np.all( [ len(feat) == 256 for feat in features ] )
```

In [23]:
```
labels = np.array([ 0 for feature in par1math_features  ] \
                + [ 1 for feature in others_features ])

# list of labels should be the same
# as the number of features
assert len(labels) == len(features)
# first label in the list should be 0
assert labels[0] == 0
# last label in the list should be 1
assert labels[-1] == 1

labels[:5], labels[-5:]
```

Out[23]:
```
(array([0, 0, 0, 0, 0]), array([1, 1, 1, 1, 1]))
```

In [24]:
```
assert features.shape[0] == labels.shape[0]

features.shape, labels.shape
```

Out[24]:
```
((29103, 256), (29103,))
```

In [25]:
```
X = features
y = labels
```

In [26]:
```
X_train, X_validate, y_train, y_validate = sklearn.model_selection.train_test_split(
    X, y,
    test_size=0.33,
    random_state=42)

clf, cvres = xgb_cross_validate(X_train, y_train)
```

In [27]:
```
clf.score(X_validate, y_validate)
```

Out[27]:
```
0.9993752603082049
```

In [28]:
```
y_pred = clf.predict(X_validate)
print(classification_report(y_validate, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         6
           1       1.00      1.00      1.00      9598

    accuracy                           1.00      9604
   macro avg       0.50      0.50      0.50      9604
weighted avg       1.00      1.00      1.00      9604


/Users/yangzeyu/anaconda3/envs/tensorflow/lib/python3.5/site-packages/sklearn/metrics/classification.py:1437:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicte
d samples.
  'precision', 'predicted', average, warn_for)
```

In [ ]: