Unit-3

In this chapter,

Transforming data into the time series format - Slicing time series data - operating on time series data - Extracting statistics from time series data - Building HMM for sequential data - Building conditional Random fields for sequential text data - Analyzing stock market data using HMM.

Introduction:

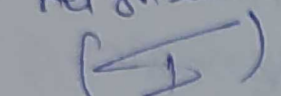
→ Time series data is basically a sequence of measurements that are collected over time.

→ these measurements are taken with respect to a predetermined variable at regular time intervals.

→ one of the main characteristics of time series data is that the ordering matters.

→ In order to visualize time series data, we tend to plot line charts or bar graphs.

→ the models that we build ?r ?n ?ime series and sequential data analysis should take in to accant the ordering of data & extract relationships between neighbors.



① transforming data into the time series format.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt.
```

② def convert_data_to_timeseries (input_file, column, verbose=False):

③ # Load the input file
data = np.loadtxt (input_file, delimiter=',')

④
```
# Extract the start and end dates
start_date = str (int (data [0, 0]),
    ) + '-' + str (int (data, [0, 1]))
end_date = str (int (data [-1, 0] + 1))
    + '_' + str (int (data
    [-1 1] % 12 + 1)
```

⑤. if verbose:
print "\nstart data =", start_date
print "End date =", end_date.

(6) create a pandas variable,

```
dates = pd.date_range(start_date, end_date,
                      freq = 'M')
```

(7) #convert the data into time series data

```
data_timeseries = pd.series(data[:,column],
                            index = dates)
```

(8) if verbose

```
print "\n time series data:\n", data_
                                timeseries[:10]
```

(9) Return time-indexed.

```
return data_timeseries
```

(10) Define main

```
if __name__ == '__main__':
```

(11) # input file containing data

```
input_file = 'data_timeseries.txt'
```

(12) #Load input data

```
column_num = 2
data_time_series = convert_data_to_timeseries
                   (i/p file, colum_num)
```

(12) # plot the time series data

data_timeseries.plot()

plt.title('Input data')

plt.show()

(13)

(14) Slicing time series data

Code:

(1) import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from convert_to_timeseries

import convert_data_to_timeseries

(2) input_file = 'data_timeseries.txt'

(3) # Load data

column_num = 2

data_timeseries = convert_data_to_timeseries(input_file, column_num)

(3)

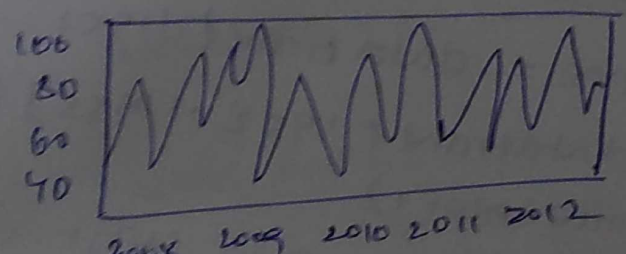
(4) start = '2002'
end = '2015'

(5) plt.figure()
data_timeseries[start:end].plot()
plt.title('Data from' + start + 'to' + end)

(6) Slice data based on a certain range of months.
# plot within a certain range of dates
start = '2007-2'
end = '2007-11'

(7) plot.
plt.figure()
data_timeseries[start:end].plot()
plt.title('Data from' + start + 'to' + end)
plt.show()

(8) plot.



[illegible]

(1) [illegible]

(2) [illegible]

(3) [illegible]

(4) [illegible]

(5) [illegible]
[illegible] ['1952':'1955'] [illegible]
[illegible] ([illegible] on top on each other')

(6) [illegible]
[illegible]
[illegible] = [illegible] ['1952':'1955'] ['first']
[illegible] ['1952':'1955'] ['Second']
[illegible]
[illegible] ('Difference (first - second)')

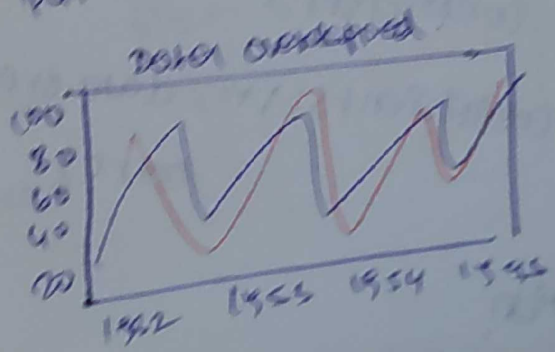
(7) # where 'first' is greater than a certain threshold
and 'second' is smaller than a certain threshold

dataframe[(dataframe['first'] > 60) & (dataframe['second'] < 20)].plot()

plt.title('first > 60 and second < 20')

plt.show().

(8)



(9) Extracting statistics from time series data

(1) import numpy as np, pandas, plt.

(2) input_file = 'data_timeseries.txt'

(3) data = function (input, 2)
data = function (input, 3).

(4) dataframe = pd.Dataframe ({'first': data,
'second': data2})

(5) Stats now,
print 'In maximum: \n', dataframe.max()

(6) #print mean
print '\n mean raw-wise: \n', dataframe
.mean(1)[:10]

(7) #plot rolling mean
pd.rolling_mean(dataframe, window=24).plot()

(8) Correlation coefficients
#print correlation coefficients
print '\n correlation coefficients: \n', dataframe.
corr()

(9) #plot rolling correlation
plt.figure()
pd.rolling_corr(dataframe['first'],
dataframe['second'], window=60).
plot()

plt.show()

(10) output:

Maximum:
first 99.82
second 99.97
dtype: float64

minimum:
first 0.07
second 0.00
dtype: float64

Mean
first
51.264
second:
49.695
float64

(12)

[illegible]	[illegible]
1940-01-31	[illegible]
1940-02-29	[illegible]
1940-03-31	[illegible]

[illegible] by

[illegible] [illegible].

	first	second
first	1.00	0.67
second	0.67	1.00

3 [illegible].

(5) Training Hidden Markov Models on Sequential data.

→ HMMs are [illegible] [illegible] [illegible] to sequential data.

code:

```
import datetime
import numpy as np
import matplotlib.pyplot as plt
from hmmlearn.hmm import GaussianHMM
from convert_to_timeseries import convert_data_
to_timeseries
```

(2) #Load data

```
data = np.loadtxt('I/P.file', delimeter=',')
```

(3) #Arrange data for training

```
X = np.column_stack([data[:, 2]])
```

(4) create & train HMM using 4 components.

```
# create and train Gaussian HMM
print("\n training HMM...")
num_components = 4
model = GaussianHMM(n_components=num_components, covariance_type='diag', n_iter=1000)

model.fit(X)
```

(5) Run the predictor to get the hidden statements

```
#predict the hidden states of HMM
hidden_states = model.predict(X)
```

(6)

```
print("\n Means & variances of hidden states:
for i in range(model.n_components):
    print("\n Hidden state", i+1
```

(6) Building CRF for sequential data.

Code:

(1) import os
import argparse
import cPickle as pickle
import numpy as np
import matplotlib.pyplot as plt
from pystruct.datasets import load_letters
from pystruct.models import ChainCRF
from pystruct.learners import FrankWolfeSSVM

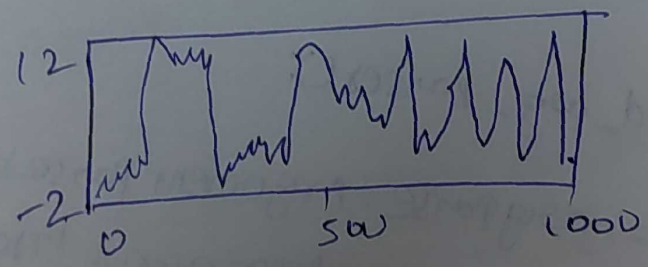
(2) def build_arg_parser():
parser = argparse.ArgumentParser
(description = 'Trains the
CRF classifier')
parser.add_argument ("--c-value", dest
='C_value', required
= False, type = float,
default = 1.0, help = 'The C value
could be used for training')
return parser.

```
print "mean =", round(model.means_[i][0],3)
print "variance =", round(np.diag(model.covars_[i][0],3)
```

(7)

```
#Generate data using model.
num_samples = 1000
samples, _ = model.sample(num_samples)
plt.plot(np.arange(num_samples),
         samples[:,0], c='black')
plt.title('Number of components = ' + str(num_components))
plt.show()
```

(8) O/P



Training HMM...

Means & variances of hidden states:

Hidden state 1

Mean = 5.092

variance = 0.677

Hidden state 2

Mean = 0.6

variance = 0.254

(7) Loads the letters dataset.

```
def load_data(self):
    letters = load_letters()
```

(8) Load the data/labels into their respective variables:

```
X, Y, Folds = letters['data'], letters
                ['labels'],
              letters['folds']

X, Y = np.array(X), np.array(Y)

return X, Y, folds
```

(9) # X is a numpy array of samples where each # sample has the shape (n_letters, n-features)

```
def train(self, X_train, Y_train):
    self.clf.fit(X_train, Y_train)
```

(10) Define a method evaluate the performance of the model.

```
def evaluate(self, X_test, Y_test):
    return self.clf.score(X_test, Y_test)
```

(3) class to handle all CRF-related processy.

class CRFTrainer (object):

(4) Define an (init) function to initialize the vals.

def init_ (self, c_value, classifier_name
= 'chain CRF'):

self.c_value = c_value

self.classifier_name = classifier_name

(5) if self.classifier_name == 'chain CRF':

model = chain CRF()

(6) Define the classifier that we will use with our CRF model.

self.clf = FrankWolfeSSVM (model = model,
C=self.c_value, max_iter = 50)

else:

raise TypeError ('Invalid classifier
type').

(15) Load the letters data:

```
X, y, folds = crf.load_data()
```

(16)
```
X_train, X_test = X[folds == 1], X[folds != 1]
y_train, y_test = y[folds == 1], y[folds != 1]
```

(17) Train the CRF model,

```
crf.train(X_train, y_train)
```

(18) Evaluate the performance of CRF model:

```
score = crf.evaluate(X_test, y_test)
print("\nAccuracy score =", str(round(score*100,2)) + "%")
```

(19)
```
Print "\n True label =", decoder(y_test[0])
predicted_output = crf.classifier(X_test[0])
print "predicted output =", decoder(predicted_output)
```

(20) Output:

```
Training the CRF model...
Accuracy score = 78.05%
True label = ommanding
predicted output = ommanfig.
```

(11) classifying new data

Print the classification on input data

```
def classify(sess, input_data):
    return sess.run(predict(input_data))[0]
```

(12)

```
def decode(args):
    alphabets = 'abcdefghijklmnopqrstuvwxyz'
    output = ''
    for i in args:
        output += alphabets[i]
    return output
```

(13) Initialize main()

```
if __name__ == '__main__':
    args = build_arg_parser().parse_args()
    c_value = args.c_value
```

(14)

```
cls = CKFTrainer(c_value)
```

(4) X = np.column_stack([diff_percentage, volume_of_shares])

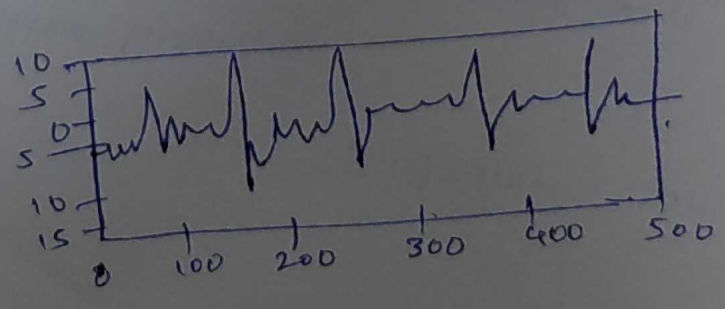
(5) train the HMM using five components:

```
#create And train Gaussian HMM.
print "\n Training HMM..."
model = GaussianHMM(n_components = 5, covariance_type
                                    = "diag", n_iter=1000)

model.fit(X)
```

(6) Generate 500 samples using the trained HMM and plot this, as follows:

```
# Generate data using model
num_samples = 500
samples, _ = model.sample(num_samples)
plot.plot(np.arrange(num_samples), samples[:, 0],
                                    c = 'black')

plt.show()
```

(7) Output:



7. Analyzing Stock market data using Hidden Markov Model.

Code:

1.
```
import datetime
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.finance import quotes_historical_
                                yahoo_ochl

from hmmlearn.hmm import GaussianHMM
```

2.
```
# Extract the required values
dates = np.array([quote[0] for quote in quote],
                 dtype=np.int)
closing_values = np.array([quote[2] for quote in
                           quotes])
volume_of_shares = np.array([quote[5] for quote
                  in quotes])[1:]
```

3.
```
# Take diff of closing values and computing rate of
change
diff_percentage = 100.0 * np.diff(closing_values)
                  / closing_values[:-1]

dates = dates[1:]
```