Using SPark + matplotlib + ipython mining pcap file

Tasks

- read and parse raw log files
- read and reshape csv-files
- spark + pandas
- spark-mllibs
  - linear regression
  - clustering algorithm
  - decision tree

raw data

In [1]:
```
sc # SparkContext
```

Out[1]:
```
<pyspark.context.SparkContext at 0x4514a90>
```

In [2]:
```
# creating an RDD (Resilient Distributed dataset)
lines= sc.textFile("../dataLogs/learnDataSet.log")
lines
```

Out[2]:
```
../dataLogs/learnDataSet.log MappedRDD[1] at textFile at NativeMethodA
ccessorImpl.java:-2
```

In [3]:
```
#Take a peek at the data.
lines.take(1)   # returns an array
```

Out[3]:
```
[u'5789:5:error,0:~8:response,4930:11:httpversion,8:1:1#1:1#]13:timest
amp_end,17:1413247021.321856^3:msg,2:OK,15:timestamp_start,16:14132470
21.31447^7:headers,1035:33:15:X-Frame-Options,10:SAMEORIGIN,]37:16:X-X
ss-Protection,13:1; mode=block,]36:22:X-Content-Type-Options,7:nosniff
,]30:15:X-Ua-Compatible,8:chrome=1,]30:22:X-Xhr-Current-Location,1:/,]
44:12:Content-Type,24:text/html; charset=utf-8,]45:4:Etag,34:"ea927ca9
2e0c7a85b168d210be0a5df4",]56:13:Cache-Control,35:max-age=0, private,
must-revalidate,]56:12:X-Request-Id,36:1c96e7f2-2bd3-4070-b2b9-73ad728
c111f,]23:9:X-Runtime,8:0.017533,]50:6:Server,37:WEBrick/1.3.1 (Ruby/2
.1.2/2014-05-08),]40:4:Date,29:Tue, 14 Oct 2014 00:36:07 GMT,]25:14:Co
ntent-Length,4:3726,]28:10:Connection,10:Keep-Alive,]44:10:Set-Cookie,
26:request_method=GET; path=/,]393:10:Set-Cookie,374:_sample_app_sessi
on=NlljR05mSytQNzJ4N0gxQzA1VkJBQ3lDOUpEOWpwWFJnaGVOaUdFU0ZDek82RE5VK0I
5V2NpY0RrRFcrSjZhT21oOTRZQW5UUzBhTDlIMUJ1QjNnZUZva3VCSEUzZ2FZeWsxMU5xc
nE4S093aGpwaVBRbUUxeTFkSENINGFsdzMzUER6RVhHMkVyVCtmUTdzeS91S2lVc3hPU2V
4Y1VOKy9GMzg0QUNNZFFzMFBUdWVUY1FvNmdiTnVRUGtQbzlULS1jWGEzRS9VcEtoWDc4b
WtBTlNSajN3PT0%3D--a87e267085a6649f66c98a671fa8dac9629f7bd9; path=/; H
ttpOnly,]]7:content,3726:<!DOCTYPE html>']
```

In [4]:
```
# lines count
lines.count()  #
```

Out[4]:
```
19111
```

In [9]:
```
# filter short lines
longlines = lines.filter(lambda line: len(line.split(":"))>10).cache()
longlines.count()
```

Out[9]:
```
35
```

In [10]:
```
longlines.take(1)
```

Out[10]:
```
[u'5789:5:error,0:~8:response,4930:11:httpversion,8:1:1#1:1#]13:timest
amp_end,17:1413247021.321856^3:msg,2:OK,15:timestamp_start,16:14132470
21.31447^7:headers,1035:33:15:X-Frame-Options,10:SAMEORIGIN,]37:16:X-X
ss-Protection,13:1; mode=block,]36:22:X-Content-Type-Options,7:nosniff
,]30:15:X-Ua-Compatible,8:chrome=1,]30:22:X-Xhr-Current-Location,1:/,]
44:12:Content-Type,24:text/html; charset=utf-8,]45:4:Etag,34:"ea927ca9
2e0c7a85b168d210be0a5df4",]56:13:Cache-Control,35:max-age=0, private,
must-revalidate,]56:12:X-Request-Id,36:1c96e7f2-2bd3-4070-b2b9-73ad728
c111f,]23:9:X-Runtime,8:0.017533,]50:6:Server,37:WEBrick/1.3.1 (Ruby/2
.1.2/2014-05-08),]40:4:Date,29:Tue, 14 Oct 2014 00:36:07 GMT,]25:14:Co
ntent-Length,4:3726,]28:10:Connection,10:Keep-Alive,]44:10:Set-Cookie,
26:request_method=GET; path=/,]393:10:Set-Cookie,374:_sample_app_sessi
on=NlljR05mSytQNzJ4N0gxQzA1VkJBQ3lDOUpEOWpwWFJnaGVOaUdFU0ZDek82RE5VK0I
5V2NpY0RrRFcrSjZhT21oOTRZQW5UUzBhTDlIMUJ1QjNnZUZva3VCSEUzZ2FZeWsxMU5xc
nE4S093aGpwaVBRbUUxeTFkSENINGFsdzMzUER6RVhHMkVyVCtmUTdzeS91S2lVc3hPU2V
4Y1VOKy9GMzg0QUNNZFFzMFBUdWVUY1FvNmdiTnVRUGtQbzlULS1jWGEzRS9VcEtoWDc4b
WtBTlNSajN3PT0%3D--a87e267085a6649f66c98a671fa8dac9629f7bd9; path=/; H
ttpOnly,]]7:content,3726:<!DOCTYPE html>']
```

In [15]:
```
#
pageTupe = longlines.flatMap(lambda line: line.split(":")).filter(lamb
da x: x.find(",")>-1)  #.collect()
pageTupe.take(10)
```

Out[15]:
```
[u'error,0',
 u'response,4930',
 u'httpversion,8',
 u'timestamp_end,17',
 u'msg,2',
 u'OK,15',
 u'timestamp_start,16',
 u'headers,1035',
 u'X-Frame-Options,10',
 u'SAMEORIGIN,]37']
```

In [33]:
```
# Maximum and Minimum response size
responseSize = pageTupe.filter(lambda x: x.split(",")[0]=="response").
map(lambda w:int(w.split(",")[1])).cache()

print "responseSize max, min, count, mean"
responseSize.max(),responseSize.min(),responseSize.count(), int(respon
seSize.mean())
```

```
responseSize max, min, count, mean
```

Out[33]:
```
(273816, 421, 420, 1787)
```

In [36]:
```
# frequently words in this log files
#
import re
def mapper(line):
    words = re.split(",", line)
    return [(w.lower(), 1)for w in words if w.isalpha()]

#lines = sc.textFile("output3_df.txt")
word_freqs = lines.flatMap(mapper).reduceByKey(lambda a,b: a+b).collec
t()
word_freqs = sorted(word_freqs, key= lambda x: ~x[1])[:10]
word_freqs
```

Out[36]:
```
[(u'deflate', 420),
 (u'sdch', 419),
 (u'button', 6),
 (u'select', 6),
 (u'input', 6),
 (u'textarea', 4),
 (u'var', 2),
 (u'label', 2),
 (u'canvas', 1),
 (u'figure', 1)]
```

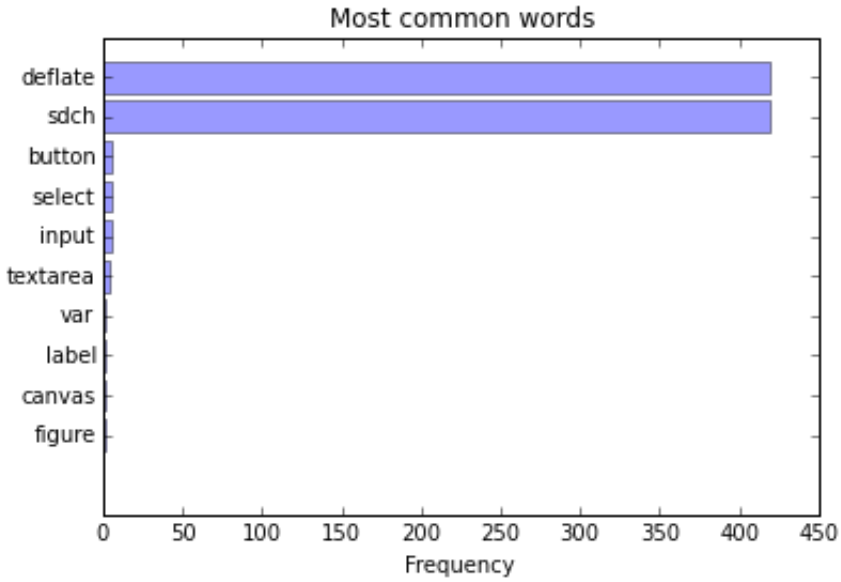
In [38]:
```
%pylab inline
import matplotlib.pyplot as plt

words = [w[0] for w in word_freqs]
y_pos = range(len(word_freqs))
frequency = [w[1] for w in word_freqs]

plt.barh(y_pos, frequency[::-1], align="center", alpha=0.4)
plt.yticks(y_pos, words[::-1])
plt.xlabel("Frequency")
plt.title("Most common words")
#plt.show()
```

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

Out[38]:
```
<matplotlib.text.Text at 0x4fd2850>
```



In []:
```
```

read pandas- dataframe

In [349]:
```
lines = sc.textFile("../dataLogs/output3_df.txt")

lines.distinct().count()
```

Out[349]:
```
421
```

In [350]:
```
labels = lines.take(1)[0].split(",")
lst=[labels.index("response"),labels.index("X-Runtime"),labels.index("
timeStampStartEnd")]
lst
```

Out[350]:
```
[37, 17, 41]
```

In [331]:
```
lines=lines.filter(lambda line: line.find("response")<0)
words=lines.map(lambda line: line.replace('"', ""))\
           .map(lambda line: line.replace(", ", "  "))\
           .map(lambda line: line.replace(" ,", " "))\
           .map(lambda line: line.split(" "))
```

In [354]:
```
words.count()
#lines.take(1)[0].find("response")
```

Out[354]: 420

In [355]:
```
words.take(1)
```

Out[355]:
```
[[u'0',
  u'gzip,deflate,sdch',
  u'',
  u'en-US,en;q=0.8,ms;q=0.6,id;q=0.4',
  u'',
  u'max-age=0',
  u'private',
  u'must-revalidate',
  u'keep-alive,3726,text/html;',
  u'charset=utf-8',
  u'Tue',
  u'14',
  u'Oct',
  u'2014',
  u'00',
  u'',
  u'',
  u'',
  u'ea927ca92e0c7a85b168d210be0a5df4',
  u'',
  u'',
  u',2,,WEBrick/1.3.1',
  u'(Ruby/2.1.2/2014-05-08)',
  u"[[{u'request_method':",
  u"u'GET'}",
  u"{u'",
  u"path':",
  u"u'/'}]",
  u"[{u'_sample_app_session':",
  u"u'NlljR05mSytQNzJ4N0gxQzA1VkJBQ3lDOUpEOWpwWFJnaGVOaUdFU0ZDek82RE5V
K0I5V2NpY0RrRFcrSjZhT21oOTRZQW5UUzBhTDlIMUJ1QjNnZUZva3VCSEUzZ2FZeWsxMU
5xcnE4S093aGpwaVBRbUUxeTFkSENINGFsdzMzUER6RVhHMkVyVCtmUTdzeS91S2lVc3hP
U2V4Y1VOKy9GMzg0QUNNZFFzMFBUdWVUY1FvNmdiTnVRUGtQbzlULS1jWGEzRS9VcEtoWD
c4bWtBTlNSajN3PT0%3D--a87e267085a6649f66c98a671fa8dac9629f7bd9'}",
  u"{u'",
  u"path':",
  u"u'/'}",
  u"{u'",
  u"HttpOnly':",
  u'None}]]',
  u'nosniff,SAMEORIGIN,1c96e7f2-2bd3-4070-b2b9-73ad728c111f,0.017533,c

hrome=1,/,1;',
  u'mode=block',
  u'50.59.22.130,5',
  u',200,0,0,399,54.165.254.99,path,1#1,54.165.254.99,GET,OK,/,80,789,
1,4930,http,1413247021.055149,1413247021.052226,0.0029230117797851562'
]]
```

In [356]:
```
len(lines.take(1)[0].split(","))
```

Out[356]:
```
42
```

In [356]:

In []:

pandas dataframe to spark

In [123]:
```
import pandas as pd
dFile = "../dataLogs/output3_df.txt"
DF = pd.DataFrame(pd.read_csv(dFile, header = 0)) #hea


# only continoues columns
labels=[key for key in dict(DF.dtypes) if dict(DF.dtypes)[key] in ['fl
oat64', 'int64']]

print labels
for l in ["Content-Length", "Unnamed: 0", "content","headers","cert","
response"]:
    labels.remove(l)

#data1=data.iloc[:,[37,17,41]]
data1 = DF.loc[:,["response"]+labels]
data1[:5]
```

```
['Content-Length', 'code', 'requestcount', 'port', 'Unnamed: 0', 'cont
ent', 'X-Runtime', 'Num-Cookie', 'error', 'response', 'request', 'head
ers', 'cert', 'timestamp_start', 'timestamp_end', 'timeStampStartEnd']
```

Out[123]:

	response	code	requestcount	port	X-Runtime	Num-Cookie	error	request	timestamp_sta
0	4930	200	1	80	0.017533	2	0	789	1.413247e+09
1	1108	200	2	80	0.003280	0	0	1179	1.413247e+09
2	16091	200	3	80	0.001283	0	0	1162	1.413247e+09
3	591	200	1	80	0.001312	0	0	1198	1.413247e+09
4	591	200	1	80	0.005190	0	0	1195	1.413247e+09

In [124]:
```
#data1.drop('cert', axis=1, inplace=True)
#data1.drop("Unnamed: 0", axis=1, inplace=True)
#data1.drop(["Content-Length", "content","header","cert","Unnamed: 0"]
,axis =1)
data1[:5]
```

Out[124]:

	response	code	requestcount	port	X-Runtime	Num-Cookie	error	request	timestamp_sta
0	4930	200	1	80	0.017533	2	0	789	1.413247e+09
1	1108	200	2	80	0.003280	0	0	1179	1.413247e+09
2	16091	200	3	80	0.001283	0	0	1162	1.413247e+09
3	591	200	1	80	0.001312	0	0	1198	1.413247e+09
4	591	200	1	80	0.005190	0	0	1195	1.413247e+09

In [125]:
```
#for d in data1.iloc[:,[1,2]]:    print d
data1.tail()
```

Out[125]:

	response	code	requestcount	port	X-Runtime	Num-Cookie	error	request	timestamp_s
415	426	304	76	80	0.001339	0	0	1366	1.413247e+0
416	426	304	64	80	0.001294	0	0	1354	1.413247e+0
417	424	304	65	80	0.003148	0	0	1357	1.413247e+0
418	426	304	65	80	0.001717	0	0	1357	1.413247e+0
419	426	304	73	80	0.002074	0	0	1359	1.413247e+0

In [126]:
```
# fill missing value
data1=data1.fillna(0.001)
```

In [159]:
```
#data1["code"]
print data1.drop_duplicates(subset = 'code', inplace = False)
data1.count() #[:5]
```

```
    response  code  requestcount  port  X-Runtime  Num-Cookie  error
request  \
0       4930   200             1    80   0.017533           2      0
    789
29      1092   304            10    80   0.017735           1      0
   1311
56      1388   302            12    80   0.086584           3      0
   1542

    timestamp_start  timestamp_end  timeStampStartEnd
0      1.413247e+09   1.413247e+09           0.002923
29     1.413247e+09   1.413247e+09           0.007055
56     1.413247e+09   1.413247e+09           0.020036
```

Out[159]:
```
response             420
code                 420
requestcount         420
port                 420
X-Runtime            420
Num-Cookie           420
error                420
request              420
timestamp_start      420
timestamp_end        420
timeStampStartEnd    420
dtype: int64
```

In [178]:
```
print ["response"]+labels
ll =["response"]+labels
ll[2:3]+ll[4:6]+ll[7:8]+ll[10:]
```

```
['response', 'code', 'requestcount', 'port', 'X-Runtime', 'Num-Cookie'
, 'error', 'request', 'timestamp_start', 'timestamp_end', 'timeStampSt
artEnd']
```

Out[178]:
```
['requestcount', 'X-Runtime', 'Num-Cookie', 'request', 'timeStampStart
End']
```

DataFrame subset, only continous features

In [131]:
```
#for d in data1.ix[:,:]:    print d[:,:]
#for index, row in data1.iterrows():     print row.values[0:3] #, row[
1],row[2]  #row['X-Runtime']

data1.to_csv("test.csv",index=False)
```

In [111]:
```
#data1=data.iloc[:,[37,17,41]]
dataRDD = sc.parallelize(data1)

#parsedData = dataDF.filter(lambda x: x.find("response")!=0).map(parse
Point)
#parsedData.take(2)
dataRDD
```

Out[111]:
```
ParallelCollectionRDD[117] at parallelize at PythonRDD.scala:364
```

In [375]:
```
data1[:1]
```

Out[375]:

	response	code	requestcount	port	X-Runtime	Num-Cookie	error	request	timestamp_sta
0	4930	200	1	80	0.017533	2	0	789	1.413247e+09

In [391]:
```
from pyspark.mllib.regression import LabeledPoint, LinearRegressionWit
hSGD
from numpy import array

data = sc.textFile("test.csv")

# Load and parse the data
def parsePoint(line):
    values = array([float(x) for x in line.split(",")]) #replace(",",
" ").split(" ")]
    values2=  array([values[5],values[4], values[10]]) # [values[2], v
alues[4], values[5],values[7],values[10]])
    return LabeledPoint(values[0], values2) # values[1:])

parsedData = data.filter(lambda x: x.find("response")<0).map(parsePoin
t)
```

In [392]:
```
parsedData.take(2)
```

Out[392]:
```
[LabeledPoint(4930.0, [2.0,0.017533,0.00292301177979]),
 LabeledPoint(1108.0, [0.0,0.00328,0.00594902038574])]
```

In [393]:
```
# Build the model
model = LinearRegressionWithSGD.train(parsedData)

# Evaluate the model on training data
valuesAndPreds = parsedData.map(lambda p: (p.label, model.predict(p.fe
atures)))
MSE = valuesAndPreds.map(lambda (v, p): (v - p)**2).reduce(lambda x, y
: x + y) / valuesAndPreds.count()
print("Mean Squared Error = " + str(MSE))
```

```
Mean Squared Error = 223860714.617
```

In [396]:
```
for d in valuesAndPreds.take(10):
    print d
```

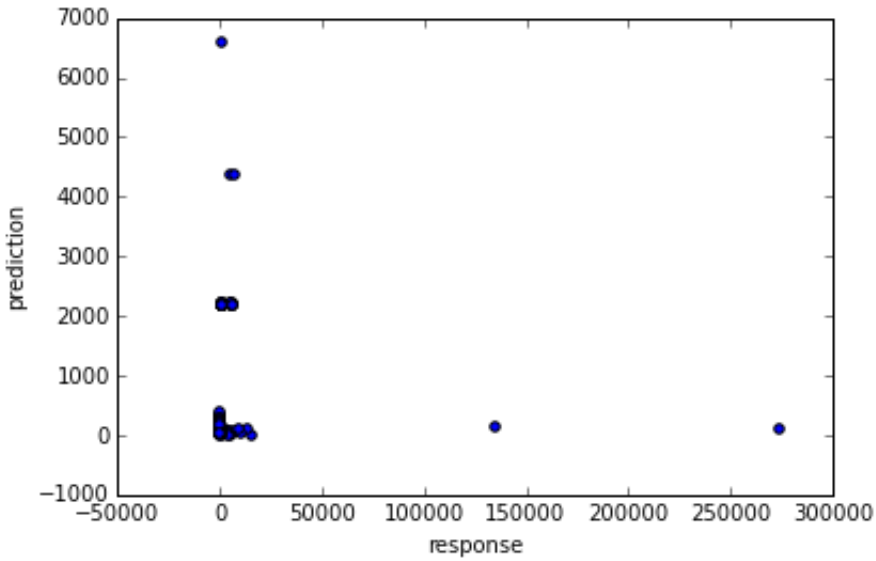
```
(4930.0, 4361.5654843222574)
(1108.0, 14.641110597012293)
(16091.0, 13.924851610082133)
(591.0, 87.155277352778924)
(591.0, 128.36164494481065)
(591.0, 56.170882563710627)
(134336.0, 135.60905976331748)
(273816.0, 90.210775745072809)
(13268.0, 90.891562187377005)
(4097.0, 77.149254091160259)
```

In [410]:
```
zip(*valuesAndPreds.take(5))
```

Out[410]:
```
[(4930.0, 1108.0, 16091.0, 591.0, 591.0),
 (4361.5654843222574,
  14.641110597012293,
  13.924851610082133,
  87.155277352778924,
  128.36164494481065)]
```

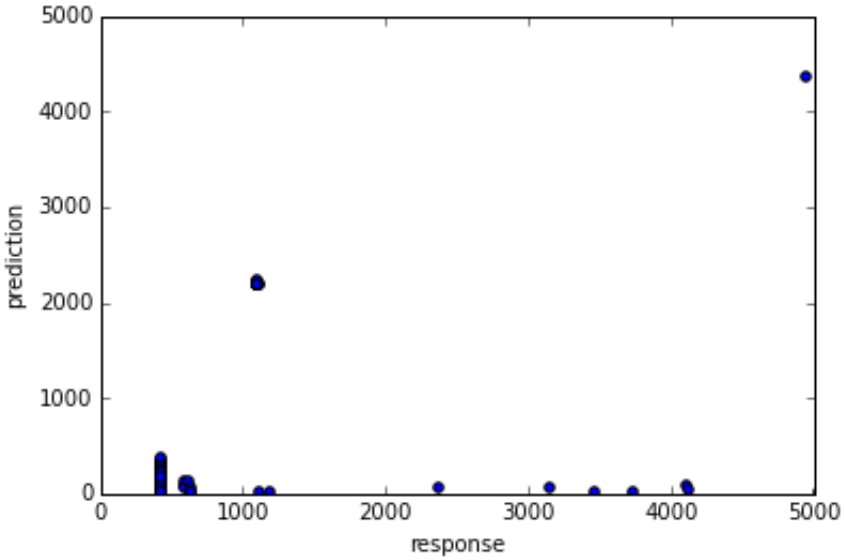
In [416]:
```
# plot
plt.xlabel("response")
plt.ylabel("prediction")
plt.scatter(*zip(*valuesAndPreds.collect()))
```

Out[416]:
```
<matplotlib.collections.PathCollection at 0x88eb550>
```



In [419]:
```
plt.xlabel("response")
plt.ylabel("prediction")

fig = plt.subplot(111)
fig.scatter(*zip(*valuesAndPreds.collect()))  #plot(target, prediction
, 'bs', target,target, 'r--')
xlim=5000
fig.set_xlim([0,xlim])
fig.set_ylim([0,xlim])
plt.show()
```



Unsupervised Clustering -- Kmean

In [450]:
```
from pyspark.mllib.clustering import KMeans
data = sc.textFile("test.csv")
parsedData = data.filter(lambda x: x.find("response")!=0)\
                 .map(lambda line: array([float(x) for x in line.split
(",")]))

# Build the model (cluster the data)
clusters = KMeans.train(parsedData, 2, maxIterations=10,
        runs=10, initializationMode="random")

# Evaluate clustering by computing Within Set Sum of Squared Errors
def error(point):
    center = clusters.centers[clusters.predict(point)]
    return sqrt(sum([x**2 for x in (point - center)]))

WSSSE = parsedData.map(lambda point: error(point)).reduce(lambda x, y:
 x + y)
print("Within Set Sum of Squared Error = " + str(WSSSE))

prediction = parsedData.map(lambda point: clusters.predict(point))
```

```
Within Set Sum of Squared Error = 431596.749847
```

In [452]:
```
#[clusters.predict(p) for p in parsedData.take(10)],
prediction.take(10)
```

Out[452]:
```
([1, 1, 1, 1, 1, 1, 0, 0, 1, 1], [1, 1, 1, 1, 1, 1, 0, 0, 1, 1])
```

In [438]:
```
# Visilization
```

In []:
```

```

decision tree

convert csv to libsvm format $ python csv2libsvm.py test.csv test_libsvm.data 0 1

In [420]:
```
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.tree import DecisionTree
from pyspark.mllib.util import MLUtils

# Load and parse the data file into an RDD of LabeledPoint.
data = MLUtils.loadLibSVMFile(sc,"test_libsvm.data")

# Split the data into training and test sets (30% held out for testing
)
(trainingData, testData) = data.randomSplit([0.7, 0.3])


# Train a DecisionTree model.
#  Empty categoricalFeaturesInfo indicates all features are continuous
.
model = DecisionTree.trainRegressor(trainingData, categoricalFeaturesI
nfo={},
                                    impurity="variance", maxDepth=5, m
axBins=32)

# Evaluate model on test instances and compute test error
predictions = model.predict(testData.map(lambda x: x.features))
labelsAndPredictions = testData.map(lambda lp: lp.label).zip(predictio
ns)
testMSE = labelsAndPredictions.map(lambda (v, p): (v - p) * (v - p)).s
um() / float(testData.count())
print('Test Mean Squared Error = ' + str(testMSE))
print('Learned regression tree model:')
print(model.toDebugString())
```

```
Test Mean Squared Error = 686472586.338
Learned regression tree model:
DecisionTreeModel regressor of depth 5 with 37 nodes
  If (feature 1 <= 2.0)
   If (feature 9 <= 0.0567169189453125)
    If (feature 6 <= 1168.0)
     If (feature 3 <= 0.00133)
      Predict: 4097.0
     Else (feature 3 > 0.00133)
      Predict: 13268.0
```

```
    Else (feature 6 > 1168.0)
     If (feature 3 <= 0.001273)
      Predict: 5026.0
     Else (feature 3 > 0.001273)
      If (feature 3 <= 0.00161)
       Predict: 1480.5
      Else (feature 3 > 0.00161)
       Predict: 591.0
   Else (feature 9 > 0.0567169189453125)
    Predict: 134336.0
  Else (feature 1 > 2.0)
   If (feature 0 <= 200.0)
    If (feature 3 <= 0.0013390000000000001)
     If (feature 1 <= 7.0)
      Predict: 639.0
     Else (feature 1 > 7.0)
      Predict: 424.0
    Else (feature 3 > 0.0013390000000000001)
     If (feature 1 <= 4.0)
      If (feature 3 <= 0.00161)
       Predict: 5829.75
      Else (feature 3 > 0.00161)
       Predict: 2410.3333333333335
     Else (feature 1 > 4.0)
      If (feature 6 <= 1168.0)
       Predict: 10523.0
      Else (feature 6 > 1168.0)
       Predict: 6332.888888888889
   Else (feature 0 > 200.0)
    If (feature 4 <= 0.0)
     If (feature 9 <= 0.007092952728271484)
      If (feature 6 <= 1353.0)
       Predict: 424.5
      Else (feature 6 > 1353.0)
       Predict: 425.6
     Else (feature 9 > 0.007092952728271484)
      If (feature 6 <= 1356.0)
       Predict: 425.9428571428571
      Else (feature 6 > 1356.0)
       Predict: 425.74178403755866
    Else (feature 4 > 0.0)
     If (feature 0 <= 302.0)
      Predict: 1388.0
     Else (feature 0 > 302.0)
      If (feature 6 <= 1353.0)
       Predict: 1092.8
      Else (feature 6 > 1353.0)
       Predict: 1104.6666666666667
```

In [421]:
```
#?MLUtils.loadLibSVMFile
```

In [422]:
```
data.take(1)
```

Out[422]:
```
[LabeledPoint(4930.0, (10,[0,1,2,3,4,6,7,8,9],[200.0,1.0,80.0,0.017533
,2.0,789.0,1413247021.05,1413247021.06,0.00292301177979]))]
```

In [423]:
```
labelsAndPredictions.take(1)
```

Out[423]:
```
[(4930.0, 13268.0)]
```

In [424]:
```
type(labelsAndPredictions.take(1)[0])
```

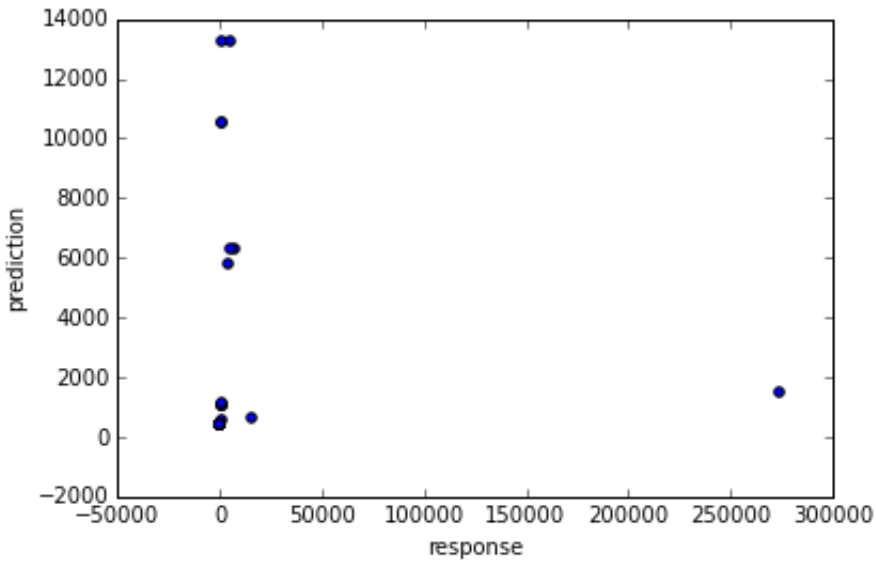
Out[424]:
```
tuple
```

In [425]:
```
l2=list(labelsAndPredictions.collect())
l2[1][1]
```

Out[425]:
```
591.0
```

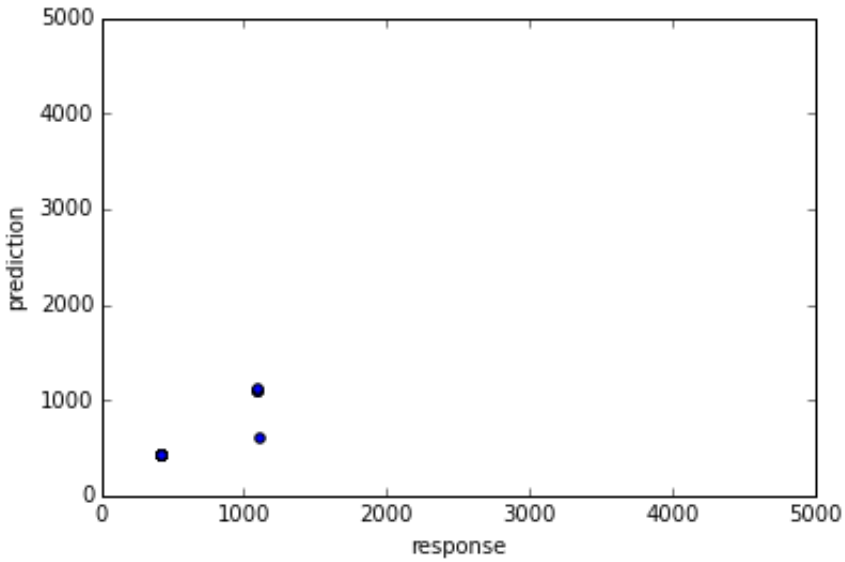
In [426]:
```
#
plt.xlabel("response")
plt.ylabel("prediction")
plt.scatter(*zip(*labelsAndPredictions.collect()))
```

Out[426]:
```
<matplotlib.collections.PathCollection at 0x88d3a10>
```



In [427]:
```
plt.xlabel("response")
plt.ylabel("prediction")

fig = plt.subplot(111)
fig.scatter(*zip(*labelsAndPredictions.collect()))  #plot(target, pred
iction, 'bs', target,target, 'r--')
xlim=5000
fig.set_xlim([0,xlim])
fig.set_ylim([0,xlim])
plt.show()
```



K-mean cluster

In [428]:
```
import numpy as np
#from pyspark import SparkContext
from pyspark.mllib.clustering import KMeans


lines = sc.textFile("test.csv")
lines = lines.filter(lambda x: x.find("response")<0)


parsedData = lines.filter(lambda x: x.find("response")<0)\
                  .map(lambda line: array([float(x) for x in line.spli
t(',')]))


# Build the model (cluster the data)
clusters = KMeans.train(parsedData, 2, maxIterations=10,
        runs=10, initializationMode="random")

# Evaluate clustering by computing Within Set Sum of Squared Errors
def error(point):
    center = clusters.centers[clusters.predict(point)]
    return sqrt(sum([x**2 for x in (point - center)]))


WSSSE = parsedData.map(lambda point: error(point)).reduce(lambda x, y:
 x + y)
print("Within Set Sum of Squared Error = " + str(WSSSE))


#lines.take(2),parsedData.take(1)
```

```
Within Set Sum of Squared Error = 431596.749847
```

In [453]:
```
parsedData.take(1),clusters.centers
```

Out[453]:
```
([array([  4.93000000e+03,   2.00000000e+02,   1.00000000e+00,
           8.00000000e+01,   1.75330000e-02,   2.00000000e+00,
           0.00000000e+00,   7.89000000e+02,   1.41324702e+09,
           1.41324702e+09,   2.92301178e-03])],
 [array([  2.04076000e+05,   2.00000000e+02,   1.00000000e+00,
           8.00000000e+01,   3.01000000e-03,   0.00000000e+00,
           0.00000000e+00,   1.18400000e+03,   1.41324702e+09,
           1.41324702e+09,   4.74305153e-02]),
  array([  8.19755981e+02,   2.95287081e+02,   3.58803828e+01,
           8.00000000e+01,   3.84713636e-03,   7.89473684e-02,
           0.00000000e+00,   1.34978230e+03,   1.41324705e+09,
           1.41324705e+09,   4.77754722e-02])])
```

In [497]:
```
prediction = parsedData.map(lambda point: clusters.predict(point))
prediction.take(10)
```

Out[497]:
```
[1, 1, 1, 1, 1, 1, 0, 0, 1, 1]
```

In [517]:
```
x2=parsedData.map(lambda x: [x[0]]+[x[4]]).collect()
type(x2)
```

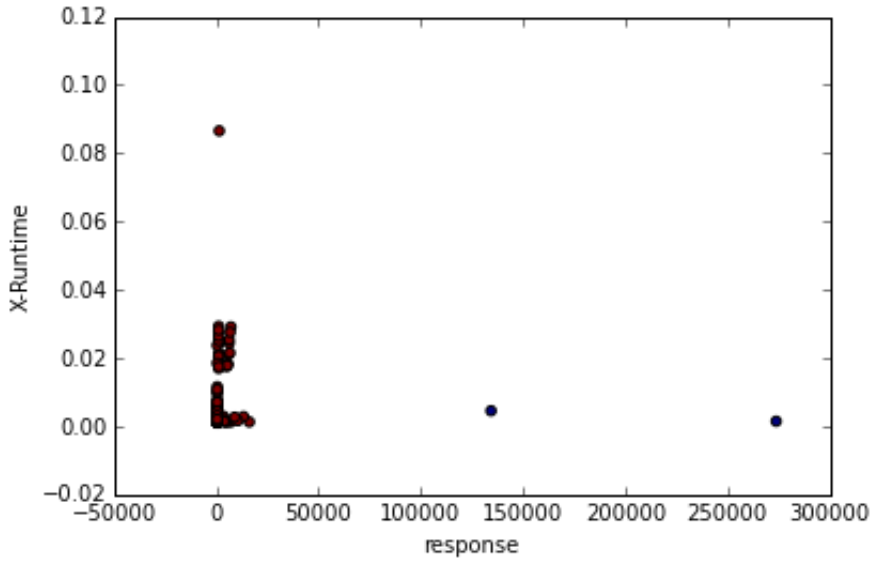
Out[517]: list

In [511]:
```
x2[:][1]
```

Out[511]: [1108.0, 0.0032799999999999999]

In [518]:
```
plt.xlabel("response")
plt.ylabel("X-Runtime")
plt.scatter(*zip(*x2),c=prediction.collect())
```

Out[518]: <matplotlib.collections.PathCollection at 0x95fc5d0>



In [512]:

In [430]:
```
def testKMean(lines, k=2):

#    lines = sc.textFile(sys.argv[1])
    data = lines.map(parseVector)
    model = KMeans.train(data, k)
    print "Final centers: " + str(model.clusterCenters)

#testKMean(lines, 2)
#data1= lines.map(parseVector)
#data1.take(1)
lines.take(2)


def mapper(line):
    """
    Mapper that converts an input line to a feature vector
    """
    feats = line.strip().split(",")
    # labels must be at the beginning for LRSGD, it's in the end in ou
r data, so
    # putting it in the right place
    label = feats[len(feats) - 1]
    feats = feats[: len(feats) - 1]
    feats.insert(0,label)
    features = [ float(feature) for feature in feats ] # need floats
    return np.array(features)

#parsedData = lines.map(mapper)
```

In []: