DATA traffic analysis

This notebook contains analysis about the data traffic generated by three different kinds of source:

```
* Download a page from as.com (www.as.com)
* Download a video from youtube (https://www.youtube.com/watch?v=ruabyha_mGI)
* Simulate voIP packets (
    * Bits per second
    CODEC=64000
    * Voice payload seconds per packet
    PACKETVOICE=0.02
    * 8 bits are one byte
    BIT_TO_BYTE=8
    IPHEADER=40 # BYTES
    ETHERNETHEADER=18 # BYTES
    VOIP_PACKET_SIZE=$(echo "($CODEC * $PACKETVOICE + $IPHEADER + $ETHERNETHEADER) * (1/$BIT_TO_BYT
E)" | bc -l | awk '{print int($1+0.5)}') )
```

In [1]:

```
import pandas as pd
import os
import shutil
from matplotlib import pyplot as plt
import seaborn as sns
import numpy as np
%matplotlib inline
```

In [2]:

```
data_name = 'data_traffic'
path_to_data = '../Data/'
sep = '#'
file = path_to_data + data_name
TFM_all_data = pd.read_csv(file, sep=sep)
```

In [3]:
```
TFM_all_data.iloc[0:4]
```

Out[3]:

	is_youtube	IP_FiveTuple	IP_SrcIP	proto	timeStamp	coord_1	dpiPktNum	IP_UpLink	IP_TotLen	IP_DstII
0	0	17;192.168.1.135/0;2.16.8.43/0;80;46710	192.168.1.135	tcp	20:13:13.499324	10	1-7	1	0	2.16.8.4:
1	0	17;192.168.1.135/0;2.16.8.43/0;80;46710	2.16.8.43	tcp	20:13:13.513781	71	0-8	0	0	192.168.1.13:
2	0	17;192.168.1.135/0;2.16.8.43/0;80;46710	192.168.1.135	tcp	20:13:13.513812	72	1-9	1	0	2.16.8.4:
3	0	17;192.168.1.135/0;2.16.8.43/0;80;46710	192.168.1.135	tcp	20:13:13.513914	22	1-10	1	197	2.16.8.4:

We will mix traffic labels into one column. First, we will set label name instead of 1 and 0 of each label. Then we remove each labe column.

In [4]:
```
# Mix labels
video_label = 'is_youtube'
voip_label = 'voIP'
brow_label = 'browsing'
TFM_data_df = TFM_all_data.copy()
TFM_data_df.loc[TFM_data_df[video_label] == 1, 'label'] = video_label
TFM_data_df.loc[TFM_data_df[voip_label] == 1, 'label'] = voip_label
TFM_data_df.loc[TFM_data_df[brow_label] == 1, 'label'] = brow_label

# Remove each label column
column_names = TFM_data_df.columns.tolist()
column_names.remove(video_label)
column_names.remove(voip_label)
column_names.remove(brow_label)
TFM_data_df = TFM_data_df[column_names]

TFM_data_df.iloc[0:4]
```

Out[4]:

	IP_FiveTuple	IP_SrcIP	proto	timeStamp	coord_1	dpiPktNum	IP_UpLink	IP_TotLen	IP_DstIP	IP_Versior
0	17;192.168.1.135/0;2.16.8.43/0;80;46710	192.168.1.135	tcp	20:13:13.499324	10	1-7	1	0	2.16.8.43	4
1	17;192.168.1.135/0;2.16.8.43/0;80;46710	2.16.8.43	tcp	20:13:13.513781	71	0-8	0	0	192.168.1.135	4
2	17;192.168.1.135/0;2.16.8.43/0;80;46710	192.168.1.135	tcp	20:13:13.513812	72	1-9	1	0	2.16.8.43	4
3	17;192.168.1.135/0;2.16.8.43/0;80;46710	192.168.1.135	tcp	20:13:13.513914	22	1-10	1	197	2.16.8.43	4

Flows

Following some read papers, main data of a flow is contained in the first 4.5 seconds of a flow of packets. According with this data, we are going to calculate (helped by timeStamp column) the number of packets in 4.5 seconds for each label.

In [5]:
```
vid_df = TFM_data_df.loc[TFM_data_df['label'] == video_label].reset_index(drop=True)
bro_df = TFM_data_df.loc[TFM_data_df['label'] == brow_label].reset_index(drop=True)
voip_df = TFM_data_df.loc[TFM_data_df['label'] == voip_label].reset_index(drop=True)
```

In [6]:
```
init_vid = vid_df['timeStamp'][0]
init_bro = bro_df['timeStamp'][0]
init_voip = voip_df['timeStamp'][0]

{'video_init_hour': init_vid, 'voip_init_hour': init_voip, 'bro_init_hour': init_bro}
```

Out[6]:
```
{'bro_init_hour': '20:13:13.499324',
 'video_init_hour': '20:34:12.451696',
 'voip_init_hour': '21:22:07.297557'}
```

In [7]:
```
vid_num_packets = 2100
print('HORA INICIO VIDEO: ' + str(init_vid) + 'HORA CALCULADA: ' + str(vid_df.iloc[vid_num_packets][3]))
```

```
HORA INICIO VIDEO: 20:34:12.451696HORA CALCULADA: 20:34:16.305427
```

In [8]:
```
bro_num_packets = 350
print('HORA INICIO BROWSING: ' + str(init_bro) + 'HORA CALCULADA: ' + str(bro_df.iloc[bro_num_packets][3]))
```

```
HORA INICIO BROWSING: 20:13:13.499324HORA CALCULADA: 20:13:17.887830
```

In [9]:
```
voip_num_packets = 400
print('HORA INICIO VOIP: ' + str(init_voip) + 'HORA CALCULADA: ' + str(voip_df.iloc[voip_num_packets][3]))
```

```
HORA INICIO VOIP: 21:22:07.297557HORA CALCULADA: 21:22:11.385689
```

In [10]:
```
(voip_num_packets + bro_num_packets + vid_num_packets) / 3
```

Out[10]:
```
950.0
```

Important INFO

Now, for the creation of the model, we will remove some columns that depend on where the demo is being running. The goal is create a demo thata can be showed in any place. Due to that, we will remove:

```
* TimeStamp: Only needed for the visualization in Kibana.
* coord_1 and coord_2: rigth now are created in random way so are not important.
* IP_Version: alwys version 4
* dpiPktNum: number of packet generated. Is a number set by the generator so not important here.
```

IP_SrcIP, IP_DstIP and IP_FiveTuple contains info about IPs and ports of the communication. It could be relevant information but that tie us to have a concrete IP in the device where the data traffic is being analyzed.

In [11]:
```
columns_to_remove = ['IP_FiveTuple', 'IP_SrcIP', 'timeStamp', 'coord_1',
                     'dpiPktNum','IP_UpLink', 'IP_DstIP', 'IP_Version', 'coord_2']
for col in columns_to_remove:
    column_names.remove(col)

TFM_data_df = TFM_data_df[column_names]
TFM_data_df = TFM_data_df[['label','proto', 'IP_TotLen']]
df_rows = TFM_data_df.shape[0]
TFM_data_df.iloc[0:4]
```

Out[11]:

	label	proto	IP_TotLen
0	browsing	tcp	0
1	browsing	tcp	0
2	browsing	tcp	0
3	browsing	tcp	197

In [12]:
```
TFM_data_df.groupby('label').count()
```

Out[12]:

	proto	IP_TotLen
label		
browsing	38112	38112
is_youtube	56235	56235
voIP	52183	52183

In [13]:
```
TFM_data_df.groupby('label').describe()
```

Out[13]:

	IP_TotLen							
	count	mean	std	min	25%	50%	75%	max
label								
browsing	38112.0	2429.369674	3497.168700	0.0	0.0	133.0	4344.0	27512.0
is_youtube	56235.0	1757.337619	1989.861707	0.0	0.0	1448.0	2896.0	17376.0
voIP	52183.0	175.132629	27.367425	0.0	175.0	175.0	175.0	4155.0

In [14]:
```
TFM_data_df.groupby('proto').count()
```

Out[14]:

	label	IP_TotLen
proto		
ICMP	51956	51956
UDP	1508	1508
tcp	93066	93066

In [15]:
```
TFM_data_df.groupby('proto').describe()
```

Out[15]:

	IP_TotLen							
	count	mean	std	min	25%	50%	75%	max
proto								
ICMP	51956.0	175.000000	0.000000	175.0	175.0	175.0	175.0	175.0
UDP	1508.0	83.465517	85.238964	24.0	28.0	35.0	133.0	451.0
tcp	93066.0	2055.882900	2730.843771	0.0	0.0	1448.0	2896.0	27512.0

In [16]:
```
TFM_data_df.groupby(['label', 'proto']).count()
```

Out[16]:

		IP_TotLen
label	**proto**	
browsing	**UDP**	1293
	tcp	36819
is_youtube	**UDP**	69
	tcp	56166
voIP	**ICMP**	51956
	UDP	146
	tcp	81

In [17]:
```
TFM_data_df.groupby(['label', 'proto']).describe()
```

Out[17]:

		IP_TotLen							
		count	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**
label	**proto**								
browsing	**UDP**	1293.0	82.337974	87.083690	24.0	28.0	56.0	133.0	451.0
	tcp	36819.0	2511.792118	3529.756939	0.0	0.0	226.0	4344.0	27512.0
is_youtube	**UDP**	69.0	124.275362	84.941136	33.0	42.0	83.0	213.0	246.0
	tcp	56166.0	1759.343838	1990.257499	0.0	0.0	1448.0	2896.0	17376.0
voIP	**ICMP**	51956.0	175.000000	0.000000	175.0	175.0	175.0	175.0	175.0
	UDP	146.0	74.164384	60.549214	31.0	35.0	35.0	154.0	300.0
	tcp	81.0	442.197531	625.349131	0.0	0.0	257.0	1033.0	4155.0

Dataframe transformations

In order to plot and to prepare data for fitting, we will apply some transformations

In [18]:
```
TFM_data_model = TFM_data_df.copy()
proto_keys = {'tcp': 1, 'UDP': 2, 'ICMP': 3}
label_keys = {video_label: 1, voip_label: 2, brow_label: 3}

TFM_data_model.loc[TFM_data_model['proto'] == 'tcp', 'proto'] = proto_keys['tcp']
TFM_data_model.loc[TFM_data_model['proto'] == 'UDP', 'proto'] = proto_keys['UDP']
TFM_data_model.loc[TFM_data_model['proto'] == 'ICMP', 'proto'] = proto_keys['ICMP']

TFM_data_model.loc[TFM_data_model['label'] == video_label, 'label'] = label_keys[video_label]
TFM_data_model.loc[TFM_data_model['label'] == voip_label, 'label'] = label_keys[voip_label]
TFM_data_model.loc[TFM_data_model['label'] == brow_label, 'label'] = label_keys[brow_label]
```

In [19]:
```
def to_csv(path_tocsv, file_name_tocsv, df, sep = '#', header = True):
    data_path_tocsv = path_tocsv + file_name_tocsv
    df.to_csv(path_or_buf= data_path_tocsv, sep=sep, header = True, index=False)
```

In [20]:
```
path_tocsv = '../Data/'
file_name_tocsv = 'data_model'
df = TFM_data_model
to_csv(path_tocsv, file_name_tocsv, df)
```

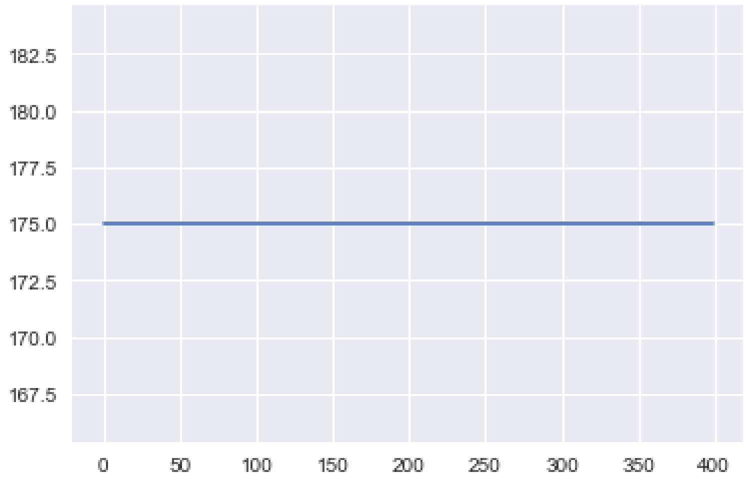
PLOTS

According to some read papers, only with packet length, the source of the data traffic could be guessed. Due to that, we will make some plots. We plot the flow of datatraffic for each label and for each protocol

In [21]:
```
vid_df = TFM_data_df.loc[TFM_data_df['label'] == video_label, 'IP_TotLen'].tolist()[0:vid_num_packets]
bro_df = TFM_data_df.loc[TFM_data_df['label'] == brow_label, 'IP_TotLen'].tolist()[0:bro_num_packets]
voip_df = TFM_data_df.loc[TFM_data_df['label'] == voip_label, 'IP_TotLen'].tolist()[0:voip_num_packets]
```

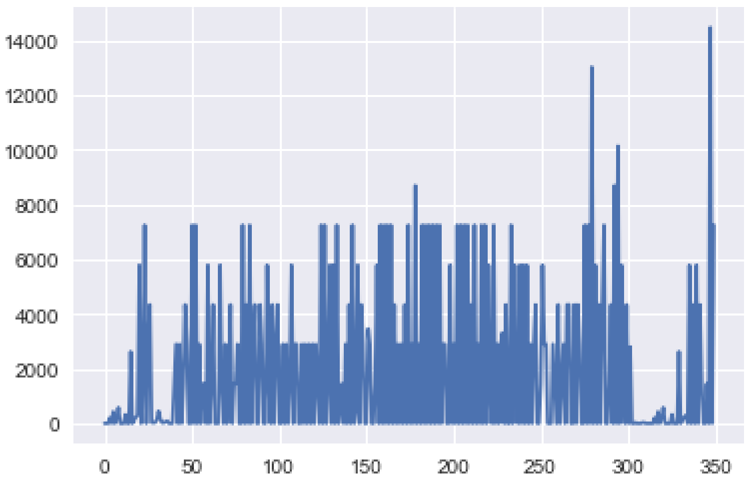
In [22]:
```
plt.plot(voip_df)
```

Out[22]: [<matplotlib.lines.Line2D at 0x7fe66e6ba898>]



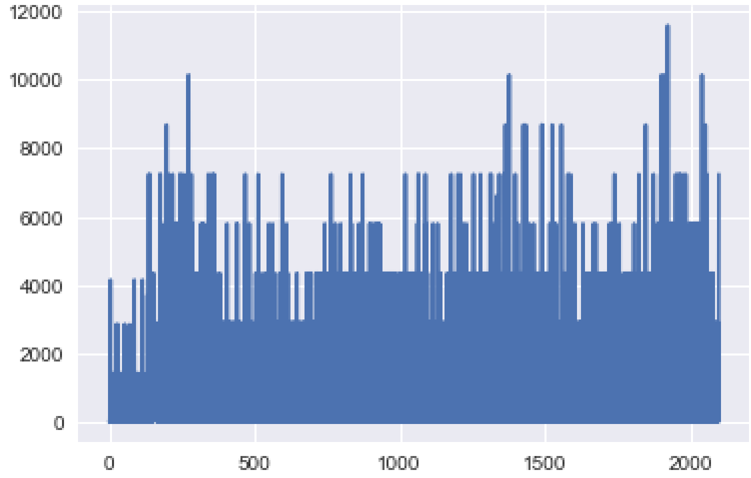
In [23]:
```
plt.plot(bro_df)
```

Out[23]: [<matplotlib.lines.Line2D at 0x7fe671562278>]



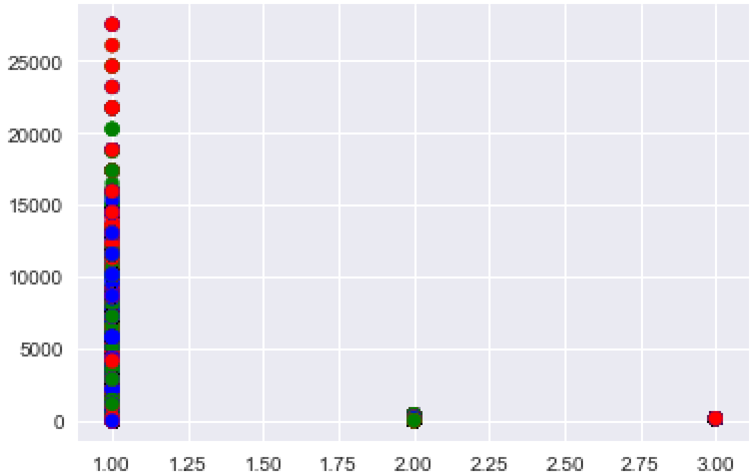
In [24]:
```
plt.plot(vid_df)
```

Out[24]: [<matplotlib.lines.Line2D at 0x7fe670dafc88>]



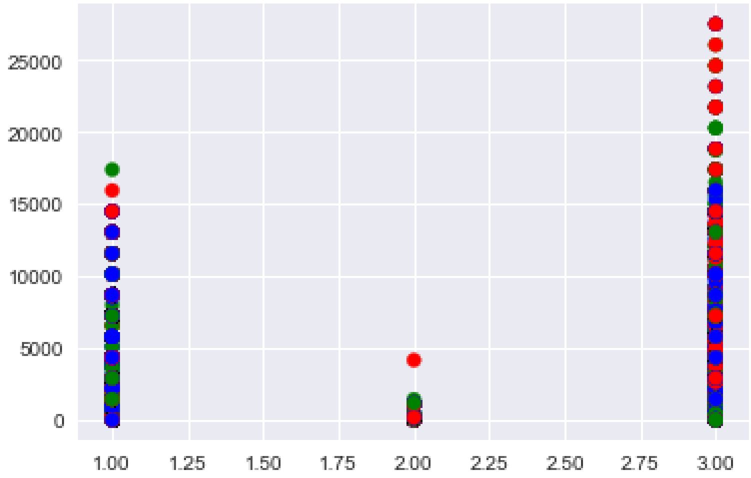
In [25]:
```
# PLot grouping by protocol
plt.scatter(TFM_data_model['proto'].tolist(),
            TFM_data_model['IP_TotLen'].tolist(),
            color=['red','green','blue'])

plt.show()
```



In [26]:
```
# PLot grouping by label
plt.scatter(TFM_data_model['label'].tolist(),
            TFM_data_model['IP_TotLen'].tolist(),
            color=['red','green','blue'])

plt.show()
```



Fitting

In [1]:

```
# Spark
import findspark
spark_path = "/home/nacho/spark"
findspark.init(spark_path)
import pyspark
from pyspark.sql import SparkSession
spark = (SparkSession.builder
    .master("local[*]")
    .config("spark.driver.cores", 1)
    .appName("understanding_sparksession")
    .getOrCreate() )
sc = spark.sparkContext

print(spark)
print(sc)
from pyspark.sql.types import *
from pyspark.sql.functions import *
from pyspark.sql.functions import col, when
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer
from pyspark.ml import Pipeline
# RANDOM FOREST
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.feature import IndexToString, StringIndexer, VectorIndexer
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.classification import RandomForestClassificationModel
# GRADIENT BOOSTED TREE
from pyspark.ml.classification import GBTClassifier
from pyspark.ml.feature import StringIndexer, VectorIndexer
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
# MULTILAYER PERCEPTRON
from pyspark.ml.classification import MultilayerPerceptronClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

```
<pyspark.sql.session.SparkSession object at 0x7f177765da20>
<pyspark.context.SparkContext object at 0x7f17a7720400>
```

From CSV to LIBSVM format (only for numerical data)

For all models in spark is needed that training files are in *libsvm* format. Due to that, we will need to use the next functions

In [28]:
```
import sys
import csv
from collections import defaultdict

# Source: https://github.com/zygmuntz/phraug/blob/master/csv2libsvm.py

def construct_line( label, line ):
    new_line = []
    if float( label ) == 0.0:
        label = "0"
    new_line.append( label )

    for i, item in enumerate( line ):
        if item == '':# or float( item ) == 0.0:
            continue
        new_item = "%s:%s" % ( i + 1, item )
        new_line.append( new_item )
    new_line = " ".join( new_line )
    new_line += "\n"
    return new_line

def csv2libsvm(inputfile, outoutfile, labels=0,headers=0):
    """
    Convert CSV file to libsvm format. Works only with numeric variables.
    Put -1 as label index (label) if there are no labels in your file.
    Expecting no headers. If present, headers can be skipped with headers == 1.

    Convert CSV to the LIBSVM format. If there are no labels in the input file,
    specify label index = -1. If there are headers in the input file, specify skip headers = 1.
    """

    input_file = inputfile
    output_file = outoutfile

    try:
        label_index = int( labels )
    except IndexError:
        label_index = 0

    try:
        skip_headers = headers
```

```
    except IndexError:
        skip_headers = 0

    i = open( input_file, 'r' )
    o = open( output_file, 'w' )

    reader = csv.reader( i , delimiter = '#')

    if skip_headers:
        headers = next(reader)

    for line in reader:
        if label_index == -1:
            label = '1'
        else:
            label = line.pop( label_index )

        new_line = construct_line( label, line )

        o.write( new_line )
```

We transform the desired data into the correct format

In [29]:
```
read_path = '../Data/'
input_file = 'data_model'
output_file = 'data_model_libsvm.csv'

csv_input = read_path + input_file
csv_ouput = read_path + output_file
csv2libsvm(csv_input, csv_ouput, labels=0,headers=1)
```

Algorithms

Gradient-boosted tree classifier

In [32]:
```python
from pyspark.ml import Pipeline
from pyspark.ml.classification import GBTClassifier
from pyspark.ml.feature import StringIndexer, VectorIndexer
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

GBTmodel_path = "../Model/Gradient_boosted_tree"

def gradient_boosted_tree(data, model_path = GBTmodel_path):

    # Index labels, adding metadata to the label column.
    # Fit on whole dataset to include all labels in index.
    labelIndexer = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(data)
    # Automatically identify categorical features, and index them.
    # Set maxCategories so features with > 4 distinct values are treated as continuous.
    featureIndexer =\
        VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=4).fit(data)

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

    # Train a GBT model.
    gbt = GBTClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures", maxIter=10)

    # Chain indexers and GBT in a Pipeline
    pipeline = Pipeline(stages=[labelIndexer, featureIndexer, gbt])

    # Train model.  This also runs the indexers.
    model = pipeline.fit(trainingData)

    # Save the model
    model.write().overwrite().save(model_path)

    ###################### Make predictions with test Data
    print('TEST DATA')

    # Make predictions.
    predictions = model.transform(testData)

    # Select example rows to display.
    predictions.select("prediction", "indexedLabel", "features").show(5)
```

```
    # Select (prediction, true label) and compute test error
    evaluator = MulticlassClassificationEvaluator(
        labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
    accuracy = evaluator.evaluate(predictions)
    print("Test Error = %g" % (1.0 - accuracy))

    gbtModel = model.stages[2]
    print(gbtModel)  # summary only

    ####################### Make predictions with training Data
    print('TRAINING DATA')

    # Make predictions.
    predictions = model.transform(trainingData)

    # Select example rows to display.
    predictions.select("prediction", "indexedLabel", "features").show(5)

    # Select (prediction, true label) and compute test error
    evaluator = MulticlassClassificationEvaluator(
        labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
    accuracy = evaluator.evaluate(predictions)
    print("Test Error = %g" % (1.0 - accuracy))

    gbtModel = model.stages[2]
    print(gbtModel)  # summary only

    return testData, trainingData

read_path = '../Data/'
file_name = 'data_model_libsvm.csv'
libsvm_filename = read_path + file_name
data = spark.read.format("libsvm").load(libsvm_filename)
test, train = gradient_boosted_tree(data)
```

```
    112                 else:  # must be an Estimator
--> 113                     model = stage.fit(dataset)
    114                     transformers.append(model)
    115                     if i < indexOfLastEstimator:

/home/nacho/spark/python/pyspark/ml/base.py in fit(self, dataset, params)
     62                 return self.copy(params)._fit(dataset)
     63             else:
```

```
---> 64                 return self._fit(dataset)
     65         else:
     66             raise ValueError("Params must be either a param map or a list/tuple of param maps, "

/home/nacho/spark/python/pyspark/ml/wrapper.py in _fit(self, dataset)
    211 
    212     def _fit(self, dataset):
--> 213         java_model = self._fit_java(dataset)
    214         return self._create_model(java_model)
    215 

/home/nacho/spark/python/pyspark/ml/wrapper.py in _fit_java(self, dataset)
```

Multilayer perceptron classifier

In [3]:
```python
from pyspark.ml.classification import MultilayerPerceptronClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

MPCmodel_path = "../Model/Multi_percep"

def multilayer_perceptron(data, layers, model_path = MPCmodel_path):


    # Split the data into train and test
    splits = data.randomSplit([0.6, 0.4], 1234)
    train = splits[0]
    test = splits[1]

    # create the trainer and set its parameters
    trainer = MultilayerPerceptronClassifier(maxIter=100, layers=layers, blockSize=128, seed=1234)

    # train the model
    model = trainer.fit(train)

    # Save the model
    model.write().overwrite().save(model_path)

    ###################### Make predictions with test Data
    print('TEST DATA')
    # compute accuracy on the test set
    result = model.transform(test)
    predictionAndLabels = result.select("prediction", "label")
    evaluator = MulticlassClassificationEvaluator(metricName="accuracy")
    print("Test set accuracy = " + str(evaluator.evaluate(predictionAndLabels)))

    ###################### Make predictions with test Data
    print('TRAINING DATA')
    # compute accuracy on the test set
    result = model.transform(train)
    predictionAndLabels = result.select("prediction", "label")
    evaluator = MulticlassClassificationEvaluator(metricName="accuracy")
    print("Training set accuracy = " + str(evaluator.evaluate(predictionAndLabels)))

    return test, train
```

In [4]:
```
read_path = '../Data/'
libsvm_filename =  read_path + 'data_model_libsvm_multi.csv'

data = spark.read.format("libsvm").load(libsvm_filename)

# specify layers for the neural network:
# input layer of size 2 (features), two intermediate of size 5 and 4
# and output of size 3 (classes)
layers = [2, 5, 4, 3]

test, train = multilayer_perceptron(data, layers)
```

```
TEST DATA
Test set accuracy = 0.5251414795944537
TRAINING DATA
Training set accuracy = 0.5261753046875888
```

In [5]:
```
read_path = '../Data/'
libsvm_filename =  read_path + 'data_model_libsvm_multi.csv'

data = spark.read.format("libsvm").load(libsvm_filename)

# specify layers for the neural network:
# input layer of size 2 (features), several intermediate
# and output of size 3 (classes)
layers = [2, 5, 4, 4, 7, 3]

test, train = multilayer_perceptron(data, layers)
```

```
TEST DATA
Test set accuracy = 0.5177212809246183
TRAINING DATA
Training set accuracy = 0.5191331311548029
```

Random Forest

General example with all data

In [30]:
```python
RFmodel_path = "../Model/RandomForest_Batch"
def randomforest(data, model_path = RFmodel_path):
    # Index labels, adding metadata to the label column.
    # Fit on whole dataset to include all labels in index.
    labelIndexer = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(data)

    # Index labels, adding metadata to the label column.
    # Fit on whole dataset to include all labels in index.
#     labelIndexer = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(data)

    # Automatically identify categorical features, and index them.
    # Set maxCategories so features with > 4 distinct values are treated as continuous.
    featureIndexer =\
        VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=4).fit(data)

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

    # Train a RandomForest model.
    rf = RandomForestClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures", numTrees=100)

    # Convert indexed labels back to original labels.
    labelConverter = IndexToString(inputCol="prediction", outputCol="predictedLabel",
                                   labels=labelIndexer.labels)

    # Chain indexers and forest in a Pipeline
    pipeline = Pipeline(stages=[labelIndexer, featureIndexer, rf, labelConverter])

    # Train model.  This also runs the indexers.
    model = pipeline.fit(trainingData)

    # Save the model
    model.write().overwrite().save(model_path)

    ####################### Make predictions with test Data
    print('TEST DATA')
    predictions = model.transform(testData)

    # Select example rows to display.
    predictions.select("predictedLabel", "label", "features").show(5)
```

```
    # Select (prediction, true label) and compute test error
    evaluator = MulticlassClassificationEvaluator(
        labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
    accuracy = evaluator.evaluate(predictions)
    print("Test Error = %g" % (1.0 - accuracy))

    rfModel = model.stages[2]
    print(rfModel)  # summary only
    print('\n\n\n')

    ####################### Make predictions with train Data
    print('TRAINING DATA')
    predictions = model.transform(trainingData)

    # Select example rows to display.
    predictions.select("predictedLabel", "label", "features").show(5)

    # Select (prediction, true label) and compute test error
    evaluator = MulticlassClassificationEvaluator(
        labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
    accuracy = evaluator.evaluate(predictions)
    print("Test Error = %g" % (1.0 - accuracy))

    rfModel = model.stages[2]
    print(rfModel)  # summary only
    return testData, trainingData

read_path = '../Data/'
file_name = 'data_model_libsvm.csv'
libsvm_filename = read_path + file_name
data = spark.read.format("libsvm").load(libsvm_filename)
test, train = randomforest(data)
```

```
TEST DATA
+--------------+-----+-------------+
|predictedLabel|label|     features|
+--------------+-----+-------------+
|           1.0|  1.0|(2,[0],[1.0])|
|           1.0|  1.0|(2,[0],[1.0])|
|           1.0|  1.0|(2,[0],[1.0])|
|           1.0|  1.0|(2,[0],[1.0])|
|           1.0|  1.0|(2,[0],[1.0])|
```

```
+--------------+-----+-------------+
only showing top 5 rows

Test Error = 0.215483
RandomForestClassificationModel (uid=rfc_053eced0c34b) with 100 trees




TRAINING DATA
+--------------+-----+-------------+
|predictedLabel|label|     features|
+--------------+-----+-------------+
|           1.0|  1.0|(2,[0],[1.0])|
|           1.0|  1.0|(2,[0],[1.0])|
|           1.0|  1.0|(2,[0],[1.0])|
|           1.0|  1.0|(2,[0],[1.0])|
|           1.0|  1.0|(2,[0],[1.0])|
+--------------+-----+-------------+
only showing top 5 rows

Test Error = 0.211873
RandomForestClassificationModel (uid=rfc_053eced0c34b) with 100 trees
```

Predicting only one value

In [9]:
```
from pyspark.mllib.tree import RandomForest, RandomForestModel
from pyspark.mllib.util import MLUtils

RF_streaming_path = '../Model/RandomForest_Streaming'

NUM_TREES = 100

read_path = '../Data/'
file_name = 'data_model_libsvm.csv'
libsvm_filename = read_path + file_name

# Load and parse the data file into an RDD of LabeledPoint.
data = MLUtils.loadLibSVMFile(sc, libsvm_filename)
# Split the data into training and test sets (30% held out for testing)
(trainingData, testData) = data.randomSplit([0.7, 0.3])

# Train a RandomForest model.
#  Empty categoricalFeaturesInfo indicates all features are continuous.
#  Note: Use larger numTrees in practice.
#  Setting featureSubsetStrategy="auto" lets the algorithm choose.
model = RandomForest.trainClassifier(trainingData, numClasses=4, categoricalFeaturesInfo={},
                                     numTrees=NUM_TREES, featureSubsetStrategy="auto",
                                     impurity='gini', maxDepth=4, maxBins=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(predictions)
testErr = labelsAndPredictions.filter(
    lambda lp: lp[0] != lp[1]).count() / float(testData.count())

print('Test Error = ' + str(testErr))
print('Learned classification forest model:')
print(model.toDebugString())

# Save and load model
RF_streaming_path = '../Model/RandomForest_Streaming'
if os.path.isdir(RF_streaming_path):
    # If exists
    shutil.rmtree(RF_streaming_path)
```

```python
# Save
model.save(sc, RF_streaming_path)
# Load
sameModel = RandomForestModel.load(sc, RF_streaming_path)
```

```
Test Error = 0.21548003089645146
Learned classification forest model:
TreeEnsembleModel classifier with 100 trees

  Tree 0:
    If (feature 0 <= 2.0)
     If (feature 1 <= 4344.0)
      If (feature 1 <= 1418.0)
       If (feature 1 <= 0.0)
        Predict: 1.0
       Else (feature 1 > 0.0)
        Predict: 3.0
      Else (feature 1 > 1418.0)
       If (feature 1 <= 2896.0)
        Predict: 1.0
       Else (feature 1 > 2896.0)
        Predict: 1.0
     Else (feature 1 > 4344.0)
      If (feature 1 <= 7240.0)
       If (feature 1 <= 5791.0)
```

In [ ]:
```python
label_keys
```

In [ ]:
```python
proto_keys
```

In [ ]:
```python
value = [1, 1090]
sameModel.predict(value)
```

In [ ]:
```python
# For predicting just running this chunk
from pyspark.mllib.tree import RandomForest, RandomForestModel
from pyspark.mllib.util import MLUtils
sameModel = RandomForestModel.load(sc, RF_streaming_path)
value = [3, 175]
sameModel.predict(value)
```

In [8]:

```

```

In [ ]:

```

```