gender_prediction_multiple

May 25, 2016

In [1]:
```
# Find Spark installation
import findspark
findspark.init()
```

In [2]:
```
# Add spark-csv package and jdbc driver
import os
os.environ['PYSPARK_SUBMIT_ARGS'] = '--packages com.databricks:spark-csv_2.10:1.1.0 pyspark-shel
os.environ['SPARK_CLASSPATH'] = '/home/h_tariq/email_pressure/lib/sqljdbc4.jar'
```

In [3]:
```
# Python imports
import time
import shutil
import json
import pymssql
import itertools as IT
from datetime import datetime
from json2html import json2html
from IPython.core.display import HTML, display
from numpy import array
from pandas import Series
import xgboost as xgb
import numpy
from collections import Counter, OrderedDict
from copy import deepcopy
```

In [4]:
```
# PySpark imports
from pyspark import SparkContext
from pyspark.sql import SQLContext
from pyspark.mllib.regression import LabeledPoint
from pyspark.mllib.tree import RandomForest
from pyspark.mllib.evaluation import BinaryClassificationMetrics
from pyspark.mllib.util import MLUtils
from pyspark.sql.functions import countDistinct
```

In [5]:
```
# PySpark initialization
sc = SparkContext(appName='GenderPrediction')

# SQLContext initializatino
sql = SQLContext(sc)
```

In [6]:
```
df = sql.read.format('com.databricks.spark.csv') \
    .option('header', 'true') \
    .option('inferschema', 'true') \
    .load('export_gender')
```

```
In [7]: train = df.filter(df.GENDER.isin(['0', '1']))
        test = df.subtract(train)

In [8]: _ = train.cache()
        _ = test.cache()

In [9]: print train.count()
        print test.count()

3134257
3160042

In [10]: top_n = 1000
         categorical_variables = ["trafficSource_source", "device_operatingSystem", "device_browser", "
                                  "pagetype", "webshop", "device_deviceCategory", "trafficSource_medium
                                  "PRODUCTTYPEID", "BRANDID"]
         categorical_dict = {}
         other_product_var = {}

         for var in categorical_variables:
             if var == 'productid':
                 all_train = sorted([(x[1], x[0].lower().strip()) for x in train.groupBy(var).count().c
                 all_test = sorted([(x[1], x[0].lower().strip()) for x in test.groupBy(var).count().col
                 common = set([x[1] for x in all_train]).intersection(set([x[1] for x in all_test]))
                 series = Series(list(common))
                 other_product_var = {v: k for k, v in series.to_dict().items()}
             all_train = sorted([(x[1], x[0].lower().strip()) for x in train.groupBy(var).count().colle
             all_test = sorted([(x[1], x[0].lower().strip()) for x in test.groupBy(var).count().collect
             common = set([x[1] for x in all_train]).intersection(set([x[1] for x in all_test]))
             series = Series(list(common))
             categorical_dict[var] = {v: k for k, v in enumerate(common)}

         print len(other_product_var)

47187

In [11]: for k, v in categorical_dict.items():
             print '%s: %s' %(k, len(v))

webshop: 331
trafficSource_medium: 56
geoNetwork_country: 114
BRANDID: 938
trafficSource_source: 670
device_operatingSystem: 11
pagetype: 20
device_browser: 22
PRODUCTTYPEID: 910
device_deviceCategory: 3
productid: 695

In [12]: print categorical_dict["device_deviceCategory"]

{u'mobile': 0, u'tablet': 1, u'desktop': 2}

In [13]: print categorical_dict["geoNetwork_country"]["netherlands"]
```

21

```
In [14]: data = test.rdd + train.rdd

In [15]: def type_casting(x):
             inner = ()

             gender = x.GENDER
             timestamp = datetime.strptime('%s %s:%s:00.0' %(x.date, x.hour, x.minute) , '%Y%m%d %H:%M:
             inner += (timestamp,)
             inner += (timestamp.hour, timestamp.minute, timestamp.day, timestamp.weekday(), timestamp.

             inner += (x.totals_pageviews,)
             inner += (x.totals_timeOnSite,)

             inner += (x.trafficSource_source.lower().strip(),)
             inner += (x.device_operatingSystem.lower().strip(),)
             inner += (x.device_browser.lower().strip(),)
             inner += (x.geoNetwork_country.lower().strip(),)
             inner += (x.pagetype.lower().strip(),)
             inner += (x.webshop.lower().strip(),)
             inner += (x.device_deviceCategory.lower().strip(),)
             inner += (x.trafficSource_medium.lower().strip(),)

             inner += (x.productid,)
             inner += (x.PRODUCTTYPEID,)
             inner += (x.BRANDID,)

             inner += (gender,)

             return ('%s-%s ' %(x.FVID, x.VID), inner)

In [16]: # Map again by grouping by key email
         data = data.map(type_casting)
         print '\nMapReduce Job 1 - Sample Output:\n'
         print data.take(1)

MapReduce Job 1 - Sample Output:

[(u'65514942951814451-1447235808 ', (datetime.datetime(2015, 11, 11, 11, 23), 11, 23, 11, 2, 46, u'25',

In [17]: # Reduce by adding all the lists for each session together
         data = data.reduceByKey(lambda x, y: x + y)

In [18]: def mapper_timestamp_sort(x):
             """
             Map the resulting data by session again and sort the records by timestamp
             """
             y = x[1]
             inner = []
             for indx in xrange(0, len(y), 20):
                 tmp = [item for sublist in [[y[indx]], y[indx+1:indx+20]] for item in sublist]
                 inner.append(tuple(tmp))
             inner = sorted(inner, key=lambda x: x[0])
             return (x[0], inner)
```

```
In [19]: data = data.map(mapper_timestamp_sort)

In [20]: def median(lst):
             return numpy.median(numpy.array(lst))

In [21]: matrix = OrderedDict()
         for x in categorical_variables:
             matrix[x] = [0] * (len(categorical_dict[x]) + 1)

         def mapper_add_cumulative_features(x):
             """
             Transform, calculate, and add new features:
             TODO
             """
             def get_index(key, val):
                 try:
                     return categorical_dict[key][val]
                 except KeyError:
                     return -1

             inner_matrix = OrderedDict(deepcopy(matrix))

             y = x[1]
             timestamps = []
             hours, weekdays, woys, sources, os_s, browsers, shops, devices, mediums = [], [], [], [],
             page_types, products, producttypes, brands = [], [], [], []
             for timestamp, hour, minute, day, weekday, woy, page_views, time_on_site,\
                 source, os, browser, country, page_type, shop, device, medium, prod_id, prodtype_id, b

                 timestamps.append(timestamp)
                 hours.append(hour + (minute/60.0))
                 weekdays.append(weekday)
                 woys.append(woy)

                 try:
                     productid_all = other_product_var[prod_id]
                 except KeyError:
                     productid_all = -1
                 products.append(productid_all)

                 inner_matrix["trafficSource_source"][get_index("trafficSource_source", source)] += 1
                 inner_matrix["device_operatingSystem"][get_index("device_operatingSystem", os)] += 1
                 inner_matrix["device_browser"][get_index("device_browser", browser)] += 1
                 inner_matrix["geoNetwork_country"][get_index("geoNetwork_country", country)] += 1
                 inner_matrix["pagetype"][get_index("pagetype", page_type)] += 1
                 inner_matrix["webshop"][get_index("webshop", shop)] += 1
                 inner_matrix["device_deviceCategory"][get_index("device_deviceCategory", device)] += 1
                 inner_matrix["trafficSource_medium"][get_index("trafficSource_medium", medium)] += 1
                 inner_matrix["productid"][get_index("productid", prod_id)] += 1
                 inner_matrix["PRODUCTTYPEID"][get_index("PRODUCTTYPEID", prodtype_id)] += 1
                 inner_matrix["BRANDID"][get_index("BRANDID", brand)] += 1

             hour = median(hours)
             weekday = Counter(weekdays).most_common(1)[0][0]
             woy = Counter(woys).most_common(1)[0][0]
```

```
            # Exclude week of the year for now
            woy = 0
            product = Counter(products).most_common(1)[0][0]

            unique_products = len(set(products))
            unique_product_types = len(set(producttypes))
            unique_shops = len(set(shops))

            total_pages = len(hours)
            time_on_site = (timestamps[-1]-timestamps[0]).total_seconds()
            time_per_page = time_on_site / float(total_pages)

            data = [hour, weekday, woy, product, unique_products, unique_product_types, unique_shops,
                    time_per_page]

            data += [item for sublist in inner_matrix.values() for item in sublist]

            return LabeledPoint(gender, data)
```

In [22]:
```
data = data.map(mapper_add_cumulative_features)
```

In [23]:
```
train_test_data = data.filter(lambda x: x.label < 2.0)
test_data = data.filter(lambda x: x.label > 2.0)

all_1s = train_test_data.filter(lambda x: x.label==1.0)
all_0s = train_test_data.filter(lambda x: x.label==0.0)

# NOT using test_eval_data for now
# (training_data_1, test_eval_data_1) = all_1s.randomSplit([0.7, 0.3])
# (training_data_0, test_eval_data_0) = all_0s.randomSplit([0.7, 0.3])

# training_data = training_data_1 + training_data_0
training_data = all_1s + all_0s
# test_eval_data = test_eval_data_1 + test_eval_data_0
```

In [6]:
```
train_location = 'gender_train'
test_location = 'gender_test'
eval_location = 'gender_test_eval'
```

In [25]:
```
try:
    shutil.rmtree(train_location)
except:
    pass
try:
    shutil.rmtree(test_location)
except:
    pass
try:
    shutil.rmtree(eval_location)
except:
    pass
```

In [26]:
```
st = time.time()
MLUtils.saveAsLibSVMFile(training_data, train_location)
# MLUtils.saveAsLibSVMFile(test_eval_data, eval_location)
```

```
        MLUtils.saveAsLibSVMFile(test_data, test_location)
        print '\nData transformed in: %s' %(time.time()-st)
```

```
Data transformed in: 2091.66101503
```

```
In [ ]: all_1s.unpersist()
        all_0s.unpersist()

        training_data.unpersist()
        # test_eval_data.unpersist()
        train_test_data.unpersist()
        test_data.unpersist()
```

```
Out[ ]: PythonRDD[225] at RDD at PythonRDD.scala:43
```

```
In [12]: st = time.time()
         _ = os.system('cat %s/part-00* > %s.txt' %(train_location, train_location))
         _ = os.system('cat %s/part-00* > %s.txt' %(test_location, test_location))
         # _ = os.system('cat %s/part-00* > %s.txt' %(eval_location, eval_location))
         print '\nData saved in: %s' %(time.time()-st)
```

```
Data saved in: 1184.66550899
```

```
In [7]: st = time.time()

        pre = ''
        tr_xgb = xgb.DMatrix('%s%s.txt' %(pre, train_location))
        # ts_eval_xgb = xgb.DMatrix('%s%s.txt' %(pre, eval_location))
        print '\nData re-loaded in: %s' %(time.time()-st)
```

```
Data re-loaded in: 296.883789062
```

```
In [8]: model_loc = 'model/gender_4.model'
```

```
In [1]: #%%capture
        st = time.time()

        param = {
                "silent": 1,
                "objective": "binary:logistic",
                "eval_metric": "auc",           # evaluation metric
                "nthread": 40,                  # number of threads to be used
                "max_depth": 4,                 # maximum depth of tree
                # "eta": 0.10,                    # step size shrinkage
                "eta": 0.04,                    # step size shrinkage
                # "subsample": 0.6,               # part of data instances to grow tree
                "subsample": 0.4,               # part of data instances to grow tree
                # "colsample_bytree": 0.8,        # subsample ratio of columns when constructing each t
                "colsample_bytree": 0.4,        # subsample ratio of columns when constructing each tre
                # "min_child_weight": 3,          # minimum sum of instance weight needed in a child
                "min_child_weight": 1,          # minimum sum of instance weight needed in a child
        }
        plst = param.items()

        num_round = 4000
        try:
```

```
            bst = xgb.train(plst, tr_xgb, num_round, evals=((tr_xgb, 'eval'),), early_stopping_rounds=2
        except:
            bst = xgb.train(plst, tr_xgb, num_round, evals=((tr_xgb, 'eval'),), early_stopping_rounds=2
```

```
In [10]: print '\nTraining completed in: %s' %(time.time()-st)
```

```
Training completed in: 13583.680423
```

```
In [11]: bst.save_model(model_loc)
```

```
In [12]: print bst.best_iteration
         print bst.best_score
```

```
7999
0.749864
```

```
In [13]: # %matplotlib inline
         # _ = xgb.plot_importance(bst)
```

```
In [14]: del tr_xgb
         # del ts_eval_xgb
```

```
In [15]: os.system('sudo su -c "echo 1 > /proc/sys/vm/drop_caches"')
         os.system('sudo su -c "echo 2 > /proc/sys/vm/drop_caches"')
         os.system('sudo su -c "echo 3 > /proc/sys/vm/drop_caches"')
```

```
Out[15]: 0
```

```
In [16]: st = time.time()

         ts_xgb = xgb.DMatrix('%s%s.txt' %(pre, test_location))
         print '\nData re-loaded in: %s' %(time.time()-st)
```

```
Data re-loaded in: 291.265121222
```

```
In [17]: st = time.time()
         predicted_values = bst.predict(ts_xgb, ntree_limit=bst.best_ntree_limit)
         print '\nPredictions made in: %s' %(time.time()-st)
```

```
Predictions made in: 22.0992760658
```

```
In [18]: labels = ts_xgb.get_label()
         del ts_xgb
```

```
In [19]: os.system('sudo su -c "echo 1 > /proc/sys/vm/drop_caches"')
         os.system('sudo su -c "echo 2 > /proc/sys/vm/drop_caches"')
         os.system('sudo su -c "echo 3 > /proc/sys/vm/drop_caches"')
```

```
Out[19]: 0
```

```
In [20]: predictions = {}

         mapping = {}
         mdf = sql.read.format('com.databricks.spark.csv') \
              .option('header', 'true') \
              .option('inferschema', 'true') \
              .load('export_gender_mapping')
```

```
        for x in mdf.collect():
            (fullVisitorId, visitId, ID) = x.fullVisitorId, x.visitId, x.ID
            mapping[int(ID)] = (fullVisitorId, visitId)

        cnt = 0
        for indx, x in enumerate(predicted_values):
            key = int(labels[indx]/10)
            (fullVisitorId, visitId) = mapping[key]
            prediction = 1 if x > 0.49 else 0
            if prediction:
                cnt+=1
            # predictions[(fullVisitorId, visitId)] = prediction
            predictions[(fullVisitorId, visitId)] = x

        print cnt
```

```
5706
```

In [21]:
```
fl = open('gender_predictions.csv', 'w')

for k, v in predictions.items():
    prediction = v
    (fullVisitorId, visitId) = k[0], k[1]
    fl.write('%s,%s,%s\n' %(fullVisitorId, visitId, prediction))

fl.close()
```

In [ ]: