predictionfinal

June 8, 2015

In [1]:
```
# Equivalent R models for Google API predictions can be found on my git, in many
#cases they were finer tuned, albeit at the expense of a programmer and looking at the data
# (time and money) compared to Google's Prediction Api. Would typically never put keys in a pub
# directory. But this is a sample and limited account, and I want to show the handshake/authori

    # OAuth handshake and creating Google Prediction API method caller

import httplib2
    #settings is an imported py doc that contains client id = '' and client_secret=''fields
    #for OAuth
import settings

client_id = settings.client_id
client_secret = settings.client_secret

from apiclient.discovery import build
from oauth2client.file import Storage
from oauth2client.client import OAuth2WebServerFlow
from oauth2client.tools import run


scope = {'https://www.googleapis.com/auth/devstorage.full_control',
            'https://www.googleapis.com/auth/devstorage.read_only',
            'https://www.googleapis.com/auth/devstorage.read_write',
            'https://www.googleapis.com/auth/prediction'}

flow = OAuth2WebServerFlow(client_id, client_secret, scope)

storage = Storage("credentials.dat")
credentials = storage.get()

if credentials is None or credentials.invalid:
    credentials = run(flow, storage)

http = httplib2.Http()
http = credentials.authorize(http)


service = build('prediction', 'v1.6', http=http)

class TrainedModel(object):

    def __init__(self, project_id, model_name):
```

```
        self.p = project_id
        self.m = model_name

    #Train a Prediction API model
    def insert(self, storage_data_location=None, output_value=None, features=None):
        body= {
                "storageDataLocation": storage_data_location,
                "id": self.m,
                "trainingInstances": [
                            {"output": output_value,
                             "csvInstance": features
                             }
                           ]
             }
        return service.trainedmodels().insert(project=self.p, body=body).execute()

    #Train a Prediction API model using a dataset
    def insert_dataset(self, training_data):
            body= {
                   "id": self.m,
                   "trainingInstances": training_data
                   }
            return service.trainedmodels().insert(project=self.p, body=body).execute()

    #Check training status of your model
    def get(self):
        return service.trainedmodels().get(project=self.p, id=self.m).execute()

    #Submit model id and request a prediction
    def predict(self, features):
        body={
              "input": {
                "csvInstance": features
                }
            }
        return service.trainedmodels().predict(project=self.p, id=self.m, body=body).execute()

    #List available models
    def list(self):
        return service.trainedmodels().list(project=self.p).execute()

    #Delete a trained model
    def delete(self):
        return service.trainedmodels().delete(project=self.p, id=self.m).execute()

    #Get analysis of the model and the data the model was trained on
    def analyze(self):
        return service.trainedmodels().analyze(project=self.p, id=self.m).execute()

    #Add new data to a trained model
    def update(self, output, features):
        body= {
               "output": output,
                "csvInstance":[
```

```
                features
                ]
              }
        return service.trainedmodels().update(project=self.p, id=self.m, body=body).execute()

class HostedModel(object):

    Hosted_model_id = 414649711441

    #Submit input and request an output against a hosted model
    def predict(self, model_name, csv_instances):
        body={
            "input":{
            "csvInstance": csv_instances
            }
        }
        return service.hostedmodels().predict()


##Callling all models  in projectID and predictions
all_models = TrainedModel("615292928655", "credit.csv").list()

####Print Model Names
    #Telling us which models are being trained or completed each time we refresh the script

def parseDictListDict(all_models):

    id_kind_list=[];

    # picking all the items list from the first dictionary
    itemsList = all_models['items'];

    # running a loop pver all the items, one at a time
    for item in itemsList: #item here is index number

        # pick out id from dictionary
        idVar = item['id'];

        # pick out kind from dictionary
        kindVar = item['kind'];

        # filling id and kind into a 2d array
        id_kind_list.append([idVar, kindVar]);
    return id_kind_list; #need return statement because all of these operations are
    #happening in memory

def parseUserCreatedList(id_kind_list):

    # Getting out 1d array from 2d array
    for item in id_kind_list:

        #printing each index in 1d array
        print(item[0]+" "+item[1]+"\n");
```

```
#printing all_models and their statuses
id_kind_list = parseDictListDict(all_models);
parseUserCreatedList(id_kind_list);
print("Models and training status")
print("\n"+"\n"+"\n")


###Now let's parse the prediction results
def parseDictListDict(prediction_results):

    pred_id_outputMulti=[];

    # picking all the items list from the first dictionary
    itemsList = prediction_results['outputMulti'];

    # running a loop pver all the items, one at a time
    for item in itemsList: #item here is index number

        # pick out id from dictionary
        labelVar = item['label'];

        # pick out kind from dictionary
        scoreVar = item['score'];

        # filling id and kind into a 2d array
        pred_id_outputMulti.append([labelVar, scoreVar]);
    return pred_id_outputMulti; #need return statement because all of these operations are
    #happening in memory

def parseUserCreatedList(pred_id_outputMulti):
#        print(pred_id_outputMulti)
    # Getting out 1d array from 2d array
    for item in pred_id_outputMulti:

        #printing each index in 1d array
        print(item[0]+" "+item[1]+"\n");
    print("\n"+"\n"+"\n")
####Analysis
    #Note did not parse prediction dictionary output results in notebook like above, one can us
    #Spyder variable explorer to do cursory analysis before proceeding to tweek prediction
    # features for intended purposes, sorting, parsing results, boosting models, building ROC c
    # etc

model = "credit.csv"
prediction_credit = TrainedModel("615292928655", model).predict(['unknown',12,'good','furniture
print(model+" "+"prediction scores")
id_kind_list = parseDictListDict(prediction_credit);
parseUserCreatedList(id_kind_list);

#Does not do well with categorical variables and factor levels, using csvs with factors as stri
# ie 0-200DM, 200-400DM, >400DM, unknown Deutsche Marks in ones bank account
# For a credit.csv file I had, from which I tried to determine whether a customer
#would default on a credit card payment or not. ALso seems to be less accurate on larger
```

```
    # number of features and non-normally distributed IDV. Also seems overfit
    # for decision tree classifiers. I have to remove a good number of features, as opposed to some
    #the more efficient packages on R that are oprimized for this, to get the same level of accurac
    #thereby requiring more knowledge of the data you are sending to Google to blackbox.
    #In fact the API method initially couldn't even find a classifier to use for the data and
    #reported an accuracy of 0. It didn't work as well as other decision tree classifiers.

    model = "concrete.csv"
    prediction_concrete = TrainedModel("615292928655", model).predict([296,0,0,192,0,0,1085,765,7,1
    print(model+" "+"prediction scores")
    id_kind_list = parseDictListDict(prediction_concrete);
    parseUserCreatedList(id_kind_list);
    #Next I tried a concrete strength .csv that I had used an R ANN package on
    # Given it did not have the above factors that limited the previous model, it worked
    #much better, giving me an accuracy of .67 at determining the strength of cement
    # from its eight featues. Unlike the above model as well, there were many more relevant
    # and numerical features that helped contribute to the accuracy of the model.

    model = "letterdata.csv"
    prediction_letterdata = TrainedModel("615292928655", model).predict([2,8,3,5])
    print(model+" "+"prediction scores")
    id_kind_list = parseDictListDict(prediction_letterdata);
    parseUserCreatedList(id_kind_list);

    #Lastly I tried an SVM classifier for OCR of letter data from UCI's machine
    # learning repository. After vectorizing each variaton or 'glyph' for a letter
    # different features such as horizontal and vertical positions, proportion of
    # black and white, and average horizontal and vertical positions of pixels
    # are recorded for each letter of the alphabet to be OCR'd.

    #The model had an accuracy of .76 and correctly predicted the letter 'X' from 4 of 16 features
    # Slightly less than the equivalent model in R of .83
    # The reason for this, I can assume, given that Google's API predict is a blackbox method
    # is the different krenels used to build the classifier in higher dimension space.
    #I would assume much like I boosted the accuracy of my model in R, that the API
    # initially chose to select a linear kernel. Much like I did in R, one can
    # increase the accuracy of results by using a Gaussian kernel.



    ####Conclusion:
    #I learned Google's API is not well trained for certain machine learning tasks, but is robust,
    # some of the more processor intensive tasks quicker on a distributed system. However great mod
    #still require strong knowledge of the statistical algorithms used, many of which are better
    #optimized in R. Ultimately thoguh, you just have to know your data and how to train it.
    # It seems Google's Prediction API is a good start, much like AWS was for distributed computing
    #towards a #commercial and small business use of machine learning. It wants to democratize a
    #the future for machine learning in the cloud.
```

```
CTR prediction#training

CTRs prediction#training

concrete.csv prediction#training
```

credit.csv prediction#training

groceries.csv prediction#training

kibana prediction#training

letterdata.csv prediction#training

sms_spam.csv prediction#training

wisc_bc_data.csv prediction#training

Models and training status

credit.csv prediction scores
checking_balance 0.200000

< 0 DM 0.200000

1 - 200 DM 0.200000

unknown 0.200000

> 200 DM 0.200000

concrete.csv prediction scores
cement 0.007168

141.3 0.005596

168.9 0.002140

250 0.002218

266 0.005002

154.8 0.002166

255 0.003430

166.8 0.006472

251.4 0.001006

296 0.001934

155 0.002475

151.8 0.006266

173 0.003120

385 0.002372

237.5 0.006807

167 0.003765

213.8 0.003017

336 0.002218

190.7 0.000181

312.7 0.001986

229.7 0.001083

228 0.006962

236 0.002295

132 0.006601

331 0.002243

310 0.004229

304 0.005028

425 0.004332

166.1 0.000490

339 0.001547

475 0.004177

145.7 0.003584

313 0.004384

178 0.003997

165 0.001444

277.2 0.003223

325 0.000645

194.7 0.000232

246.8 0.000748

382 0.001057

149 0.004358

531.3 0.003249

387 0.002656

193.5 0.006446

326 0.006111

337.9 0.006008

200 0.001367

218.9 0.000567

234 0.004874

309.9 0.006318

350 0.001444

182 0.002424

480 0.002398

295.7 0.001109

233.8 0.001057

379.5 0.003017

332.5 0.005828

237 0.004590

238.1 0.000645

323.7 0.006988

342 0.006266

388.6 0.003610

147.8 0.003739

290.4 0.001083

500 0.002888

284 0.003919

218.2 0.002733

190.3 0.000438

116 0.003842

277 0.003326

376 0.001728

273 0.003945

212.5 0.000284

362.6 0.005957

275.1 0.001160

139.6 0.004899

427.5 0.004177

183.9 0.003352

318.8 0.005853

252 0.002553

149.5 0.005467

540 0.003765

380 0.004848

436 0.002475

281 0.000361

151.6 0.000335

326.5 0.003919

397 0.002501

238 0.000077

158.6 0.004899

302 0.001264

192 0.006988

155.6 0.006085

160 0.006137

222.4 0.000670

251.8 0.000593

213.5 0.001135

446 0.002811

133 0.005415

122.6 0.004461

290.2 0.005080

375 0.002063

181.4 0.000645

298.2 0.002682

162 0.006936

262 0.004641

213.7 0.002475

313.3 0.004513

322 0.004538

173.5 0.002553

299.8 0.002424

198.6 0.003326

286.3 0.005647

349 0.000490

520 0.003584

252.1 0.004435

255.5 0.003919

172.4 0.000980

212.1 0.000516

276 0.005647

393 0.002011

230 0.000903

389.9 0.006395

157 0.006601

359 0.002553

374 0.004899

102 0.002733

202 0.001934

252.3 0.000799

336.5 0.002321

315 0.005157

159 0.005338

231.8 0.000619

159.8 0.006111

164.6 0.001650

136.4 0.005209

190 0.006421

184 0.004796

424 0.002579

212 0.002269

156 0.006498

136 0.005467

203.5 0.002733

254 0.000825

220.8 0.002914

167.4 0.004255

144 0.003197

108.3 0.003662

214.9 0.002991

469 0.005415

522 0.002166

250.2 0.004848

439 0.006627

322.5 0.005931

153 0.003971

525 0.001625

259.9 0.005131

236.9 0.004564

366 0.007117

333 0.002475

145.9 0.005699

277.1 0.001160

166 0.006627

143 0.005699

181.9 0.005982

450.1 0.002269

528 0.003791

238.2 0.003301

186.2 0.002269

212.6 0.000309

491 0.003301

152.6 0.005647

252.5 0.000155

295 0.000645

150.7 0.001341

249.1 0.000645

505 0.003868

148.1 0.001031

143.7 0.006034

289 0.004332

298 0.005621

173.8 0.003584

135.7 0.003301

225 0.000026

305.3 0.006292

170.3 0.003430

330.5 0.006756

304.8 0.002475

150 0.005441

148 0.002269

297.8 0.006782

321.3 0.006395

134.7 0.001315

168 0.001521

300 0.001625

382.5 0.001341

321 0.005518

339.2 0.000903

288 0.005260

400 0.001831

155.2 0.006395

334 0.004126

261.9 0.004616

485 0.002218

356 0.004435

264.5 0.005492

317.9 0.003275

288.4 0.004435

275 0.004409

145.4 0.002527

297.2 0.001779

280 0.006240

451 0.003713

313.8 0.002733

164 0.004203

298.1 0.002604

381.4 0.000206

261 0.004435

145 0.001367

272.8 0.005028

260 0.005131

153.1 0.004461

355 0.002708

272.6 0.002347

133.1 0.005338

143.6 0.001418

210.7 0.006859

260.9 0.004461

295.8 0.000464

322.2 0.003146

405 0.001444

401.8 0.003765

255.3 0.004048

266.2 0.005002

307 0.001625

152 0.002785

160.2 0.006782

143.8 0.004126

151 0.001418

158.8 0.004538

139.9 0.005054

374.3 0.002347

287.3 0.005853

303.6 0.005157

140 0.005260

150.9 0.001521

135 0.001934

146.5 0.003842

146 0.003971

314 0.005673

516 0.003713

152.7 0.003094

305 0.002527

148.5 0.004358

154 0.004796

164.2 0.002682

139.7 0.006627

325.6 0.006550

279.8 0.006266

158.4 0.002321

265 0.005544

500.1 0.001753

141.9 0.006163

355.9 0.004048

318 0.003301

159.1 0.005054

285 0.005905

239.6 0.007065

297 0.001418

321.4 0.003120

316.1 0.005776

312.9 0.006137

153.6 0.004719

142 0.006189

287 0.005776

158 0.002295

147 0.003868

144.8 0.000980

276.4 0.005596

letterdata.csv prediction scores
letter 0.000000

T 0.004448

I 0.031414

D 0.000070

N 0.013106

G 0.011378

S 0.047718

B 0.000134

A 0.276733

J 0.042478

M 0.000374

X 0.453738

O 0.004572

R 0.001680

F 0.004557

C 0.005488

H 0.003472

W 0.000597

L 0.023187

P 0.003897

E 0.000595

V 0.015764

Y 0.018681

Q 0.009843

U 0.001007

K 0.024059

```
Z 0.001013
```

In [ ]: