Anonymous Data Classification Machine Learning Model

Daniel Byrne

12/8/2019

# Anonymous Data Classification Machine Learning Model

## Preface

Engles Family Enterprises inc., client, and Byrne’s Brains, the developer, have partnered on the Pathways cloud machine learning development project, pursuant to Statement of Work number 29155, to develop a new product setup application which will feed the client’s customer metrics into a cloud hosted machine learning model to predict a customer viability metric.

## Introduction

Byrne’s Brains is designing a cloud hosted machine learning model with a customized HTTP REST which will accept and classify anonymized data in a highly parallelized environment, from Engles family agents in the field. The Cloud Interface Unit will be a compliant REST server designed to classify a collection of customer data points to give instant feedback to Engles Family agents to help in determining the viability of a sales decision.

This cloud hosted Machine Learning model will enable quick decision making abilities by instantly quantifying a collection of diverse metrics gathered by the client and then classifying this quantity into a prediction which will then be fed back over the API to the client’s calling computer.

This inaugural “Request For Comment”, RFC, document is being issued as a first step attempt to stimulate a dialog on the emerging design of this cloud hosted machine learning model.

This paper will go over the initial design of the machine learning model, challenges that emerged in the development process, and proposals for hosting.

## The Data

The client has generously provided the developer a collection of 160,000 anonymized records which has been used in developing an initial classification model. The data consisted variously of 46 7 columns of float data, and 3 columns of categorical data. After eliminating rows with missing data we were left with 159,913 rows of data.

Each row was pre-labeled with the proper classification when we received the data. In order to improve convergence of our models we also normalized the data for each feature. This is of note because during prediction, incoming data will need to be normalized to fit the model’s expectations.

We encoded the categorical columns with the one hot encoding technique. One hot encoding eliminates the possibility of introducing level bias effects by limiting the identifying factor to a maximum value of 1 when present, and zero otherwise. However, adding one-hot- encoding introduces its own set of problems considering that the dataset becomes very sparse with increasing numbers of encoded features. Sparse datasets struggle to converge. As such we tested our models with and without these encoded features to gauge the relative performance of leaving these features in the compiled model or removing them. The categorical features in particular seemed to be related to time and location (month,weekday, and continent). The month and weekday columns might not only appropriate in a time series modeling approach. This insight led us to attempt to train our models, which are not time series models, without these features.

## The Model

Because of the large number of features, we settled on a deep neural network, DNN, architecture as our model of choice. Deep neural networks provide the ability to encode and reduce a large numbers of feature combinations into a finite set of classification labels with relative ease.

The model consisted of 4 fully connected layers of 300, 250,200, and 150 nodes. The output layer is a single logistic regression node to squash the prediction to a 1 or a 0 class label. We used rely activations on every layer except the output where we use the logistic. Dropout was used only on the last layer. We trained the model using gradient decent with ADAM optimizations.

## Training

Training on average took about 10 minutes on a MacBook Pro equipped with a 2.3 GHz 8-Core Intel Core i9, 32 GB 2400 MHz DDR4, and training took place on the built int Radeon Pro Vega 16 4 GB using the PlaidML Keras API.

## Results

We trained our model on the 3 filtered datasets below. Results from validation are listed with each model.

nofactors.csv  
Test loss: 0.25880000801213704  
Test accuracy: 0.9369949494949495  
  
cleanedonehot.csv  
Test loss: 0.5000135238940894  
Test accuracy: 0.884280303030303  
  
cleandeddata.csv  
Test loss: 0.33758714681872254  
Test accuracy: 0.9227272727272727

## Hosting

We recommend hosting the model and API on Azure Functions. Azure functions have the benefit of configuring the service so that the client only pays when the service is actively being used. Otherwise the service goes dormant. Furthermore , since Azure functions can store their models in long term, low cost storage buckets hosting for the model and API will be relatively inexpensive.

## Source

[Github Repository https://github.com/realdanielbyrne/AnonymousDataAnalysis](https://github.com/realdanielbyrne/AnonymousDataAnalysis)

import pandas as pd  
import numpy as np  
import argparse  
import sklearn  
import sys  
import os  
os.environ["KERAS\_BACKEND"] = "plaidml.keras.backend"  
import plaidml.keras  
import keras  
from keras.models import load\_model  
from keras.models import Sequential, Model  
from keras.layers import Input,Dense, Dropout, Activation, SpatialDropout1D  
from keras.optimizers import SGD  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import MinMaxScaler  
  
def GetAllData(filepath):  
 """  
 Reads data from csv, extract features between start and end, and then spilt into train and test sets  
 # Arguments  
 start: Column of first feature  
 end: Column of last feature  
 filepath: path to csv  
 """  
 filepath = "nofactors.csv"  
 data = np.genfromtxt(filepath, delimiter=',',skip\_header=1)  
 data\_no\_nan = data[~np.isnan(data).any(axis=1)]  
  
 X = data\_no\_nan[:,0:-1].astype(float)  
 y = data\_no\_nan[:,-1].astype(int)  
  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = .1)  
 X\_train = MinMaxScaler().fit\_transform(X\_train)  
 X\_test = MinMaxScaler().fit\_transform(X\_test)  
  
 return X\_train, X\_test, y\_train, y\_test  
  
def get\_data(filepath="cleaneddata.csv"):  
 """  
 Reads data from csv, extract features between start and end, and then spilt into train and test sets  
 # Arguments  
 filepath: path to csv  
 'y' column is the label  
 """  
 # get data  
 df = pd.read\_csv(filepath)  
  
 # one hot encode  
 pdc = pd.get\_dummies(df['continent'])  
 pdm = pd.get\_dummies(df['month'])  
 pdw = pd.get\_dummies(df['weekday'])  
  
 # drop factors  
 df.drop(['continent','month','weekday'],inplace=True,axis=1)  
 df.dropna()  
  
 # split data and labels  
 y = df['y'].astype(int).to\_numpy()  
 X = df.drop('y', axis = 1).to\_numpy()  
  
 # split  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = .1)  
  
 # normalize  
 X\_train = MinMaxScaler().fit\_transform(X\_train)  
 X\_test = MinMaxScaler().fit\_transform(X\_test)  
  
 return X\_train, X\_test, y\_train, y\_test  
  
def train\_test\_split\_pd(df, train\_percent=.9, seed = [3,14]):  
 np.random.seed(seed)  
 perm = np.random.permutation(df.index)  
 m = len(df.index)  
  
 train\_end = int(train\_percent \* m)  
 train = df.iloc[perm[:train\_end]].to\_numpy()  
 test = df.iloc[perm[train\_end:]].to\_numpy()  
 return train, test  
  
def dnn\_dropout\_model(shape, dropoutprob):  
 model = Sequential()  
 model.add(Dense(units = 300, activation = "relu", input\_dim=shape))  
 model.add(Dense(units = 250, activation = "relu"))  
 model.add(Dense(units = 200, activation = "relu"))  
 model.add(Dense(units = 150, activation = "relu"))  
 model.add(Dropout(rate = dropoutprob))  
 model.add(Dense(1, activation='sigmoid'))  
 return model  
  
def compile\_model(model):  
 """  
 Compiles the model.  
  
 # Arguments :  
 model - The untrained model  
 lr - learning rate  
  
 decay - the learning rate decay rate  
 momentum - the momentum parameter  
 """  
 model.compile(loss = 'binary\_crossentropy',  
 optimizer = "adam",  
 metrics = ['accuracy'])  
 return model  
  
def score(model, X\_test, y\_test):  
 """  
 Scores the model and prints out the results.  
  
 # Arguments :  
 model - the trained model  
 X\_test - test set  
 y\_test - test labels  
 """  
 scores = model.evaluate(X\_test, y\_test, verbose=1)  
 print('Test loss:', scores[0])  
 print('Test accuracy:', scores[1])  
  
def predict(query,model\_path):  
  
 model = load\_model(model\_path)  
 data = np.genfromtxt(query, delimiter=',')  
 model.predict(data)  
  
def main():  
 parser = argparse.ArgumentParser(description='Build DNN Classifier for anonymized dataset.')  
 parser.add\_argument('-f','--filepath',  
 action='store',  
 type=str,  
 dest='filepath',  
 default="nofactors.csv",  
 help="Filename of training and test data.\n Last column should be the label. Assumes column headers are present.")  
  
 parser.add\_argument('-b','--batch\_size',  
 action='store',  
 type=int,  
 dest='batch\_size',  
 default= 50,  
 help="Sets the batch size. Default = 1000.")  
  
 parser.add\_argument('-e','--epochs',  
 action='store',  
 type=int,  
 dest='epochs',  
 default= 90,  
 help="Sets the # of epochs. Default = 70.")  
  
 parser.add\_argument('-v','--validation\_split',  
 action='store',  
 type=float,  
 dest='validation\_split',  
 default=.1,  
 help="Float between 0 and 1. Fraction of the training data to be used as validation data. Default = .1")  
  
 parser.add\_argument('-dp','--dropout\_prob',  
 action='store',  
 type=float,  
 dest='dropout\_prob',  
 default=1e-6, # original model default  
 help="Sets the learning rate decay rate. Default = 1e-6.")  
  
 parser.add\_argument('-p','--predict',  
 action='store',  
 type=str,  
 dest='query',  
 help="Path to a file containing the data to use to make a prediction.")  
  
 args = parser.parse\_args()  
  
 # parameters  
 save\_dir = os.path.join(os.getcwd(), 'saved\_models')  
 model\_name = 'dnndropout.h5'  
 model\_path = os.path.join(save\_dir, model\_name)  
  
 if (args.query):  
 model = load\_model(model\_path)  
 x = np.genfromtxt(args.query, delimiter=',')  
 p = model.predict(x)  
 return p  
  
 # Load Data and Split  
 X\_train, X\_test, y\_train, y\_test = GetAllData(args.filepath)  
  
 # Build Model  
 model = dnn\_dropout\_model(X\_train.shape[1], args.dropout\_prob)  
  
 # Compile Model  
 model = compile\_model(model)  
  
 print("Training model")  
 model.fit(X\_train, y\_train, epochs = args.epochs, batch\_size = args.batch\_size, validation\_split = args.validation\_split)  
  
 # Save model and weights  
 if not os.path.isdir(save\_dir):  
 os.makedirs(save\_dir)  
 model.save(model\_path)  
 print('Saved trained model at %s ' % model\_path)  
  
 score(model, X\_test, y\_test)  
  
main()