Precision Medicine Modeling using Deep Learning (TensorFlow)

Yupeng Wang, Ph.D, Data Scientist

Overview

Deep learning is a powerful machine learning approach which has been widely used in automatic speech recognition, image recognition and natural language processing. Here I show an example of building Deep Neural Network (DNN) models for a precision medicine problem of classifying disease subtypes, using the programs I developed on top of TensorFlow. My DNN program takes in a training and a testing data file of Pandas data fame format using command line arguments, so can be easily applied to other predictive modeling problems. The source code can be downloaded from https://github.com/wyp1125/Python-Deep-Learning-Toolset

Key techniques: Python, Deep Learning, TensorFlow, NumPy, Pandas, Deep Neural Network

Detailed procedure

1. Simulating a precision medicine scenario

Here I simulate a complex disease which is determined by 100 SNPs, and five lifestyle factors including smoking, alcohol drinking, physical exercise, substance abuse and depression. The disease has three subtypes. Thus, the dependent variable (i.e. label) has four outcomes (classes): normal (0), disease subtype I (1), disease subtype II (2) and disease subtype III (3).

For each SNP, a guiding alternate allele frequency is generated according to an exponential distribution with scale=2. Its genotype is coded by 0 (homozygous for the reference allele), 1 (heterozygous) or 2 (homozygous for the alternate allele). In normal individuals, genotypes are generated according to the guiding alternate allele frequencies. In disease individuals, the alternate allele frequencies have a 2~4 fold increase and genotypes are generated accordingly. However, not all SNPs are effective in a disease subtype. In subtypes I and II, 50 effective SNPs are randomly selected. In subtype III, 70 effective SNPs are randomly selected.

Each lifestyle factor has two levels: "Y" or "N". "Y" is generated according to a guiding frequency. In subtype I, there is a 2-fold frequency increase for smoking and alcohol drinking. In subtype II, there is a 3-fold frequency increase for substance abuse and depression, and 0.7 fold decrease for physical exercise. In subtype III, there is no frequency change in lifestyle factors.

Program name: simulate_pm.py

Linux command: python simulate_pm.py

```
from __future__ import print_function
import numpy as np
import pandas as pd
from collections import Counter
maf=[x for x in np.random.exponential(2,300)/10 if x<=0.5]
g_maf=maf[:100]
#examine the generated mafs
g_fl=[int(x*20)/20.0 for x in g_maf]
stat=Counter(g_fl)
for k, v in sorted(stat.items()): print(str(k)+'~'+str(k+0.05), v)
#value generators
def gt(guide):
    cd=0
    for i in range(2):
        r=np.random.uniform()
        if r<guide:
            cd=cd+1
    return cd
def yn(guide):
    ch='N'
    r=np.random.uniform()
    if r<guide:
        ch='Y'
    return ch
cl_nm=['snp'+str(x) for x in range(1,101)]
cl_nm.extend(['smoking','alcohol','excercise','substance','depression','label'])
g_rt=[0.1,0.2,0.8,0.05,0.05]
df=pd.DataFrame(columns=cl_nm)
#simulate 400 normal individuals
grp1=[]
for i in range(400):
    line=[]
    for i in range(100):
        line.append(gt(g_maf[i]))
    for i in range(5):
        line.append(yn(g_rt[i]))
    line.append(0)
    grp1.append(line)
df = df.append(pd.DataFrame(grp1, columns=cl_nm),ignore_index=True)
#simulate 200 disease subtype 1
grp2=[]
for i in range(200):
    line=[]
    #randomly select 20 risky SNPs
    sel=np.random.choice(100, 50, replace=False)
    for i in range(100):
        if i not in sel:
            line.append(gt(g_maf[i]))
        else:
            factor=np.random.uniform(3,5)
            line.append(gt(factor*g_maf[i]))
    line.append(yn(3*g_rt[0]))
    line.append(yn(3*g_rt[1]))
    line.append(yn(g_rt[2]))
    line.append(yn(g_rt[3]))
    line.append(yn(g_rt[4]))
    line.append(1)
```

```
    grp2.append(line)
df = df.append(pd.DataFrame(grp2, columns=cl_nm),ignore_index=True)
#simulate 200 disease subtype 2
grp3=[]
for i in range(200):
    line=[]
    #randomly select 20 risky SNPs
    sel=np.random.choice(100, 50, replace=False)
    for i in range(100):
        if i not in sel:
            line.append(gt(g_maf[i]))
        else:
            factor=np.random.uniform(3,5)
            line.append(gt(factor*g_maf[i]))
    line.append(yn(g_rt[0]))
    line.append(yn(g_rt[1]))
    line.append(yn(0.3*g_rt[2]))
    line.append(yn(4*g_rt[3]))
    line.append(yn(4*g_rt[4]))
    line.append(2)
    grp3.append(line)
df = df.append(pd.DataFrame(grp3, columns=cl_nm),ignore_index=True)
#simulate 200 disease subtype 3
grp4=[]
for i in range(200):
    line=[]
    #randomly select 20 risky SNPs
    sel=np.random.choice(100, 70, replace=False)
    for i in range(100):
        if i not in sel:
            line.append(gt(g_maf[i]))
        else:
            factor=np.random.uniform(3,5)
            line.append(gt(factor*g_maf[i]))
    line.append(yn(g_rt[0]))
    line.append(yn(g_rt[1]))
    line.append(yn(g_rt[2]))
    line.append(yn(g_rt[3]))
    line.append(yn(g_rt[4]))
    line.append(3)
    grp4.append(line)
df = df.append(pd.DataFrame(grp4, columns=cl_nm),ignore_index=True)
#output simulated data
df.to_csv('precision.csv', index=False)
```

Here we get the output file precision.csv, which contains 400 normal, 200 subtype I, 200 subtype II and 200 subtype III individuals.

2. Generating training and testing data

Here we divide the simulated data into a training and a testing dataset according to a partition fold.

Program name: divide_train_test.py

Linux command: python divide_train_test.py precision.csv pm 3

```
from __future__ import print_function
import sys
import random
if len(sys.argv)<4:
    print("Usage:python divide_train_test.py input output_prefix fold")
    quit()
with open(sys.argv[1]) as f:
    content=[x.strip('\n') for x in f.readlines()]
header=content[0]
m=len(content)-1
k=int(sys.argv[3])+1
n=int(round(m/k))
id=list(range(m))
sel=random.sample(id,n)
with open(sys.argv[2]+'.train.csv','w') as out1:
    out1.write(header+'\n')
    with open(sys.argv[2]+'.test.csv','w') as out2:
        out2.write(header+'\n')
        for i in range(m):
            if i in sel:
                out2.write(content[i+1]+'\n')
            else:
                out1.write(content[i+1]+'\n')
```

3. Building Deep Neural Network models

With input data files being ready, we can quickly build DNN models using the DNN program I developed. We need to feed in several arguments such as the categorical variable names, number of classes, number of units in each layer and number of steps.

Program name: dnn_model_pandas_cat.py

Linux commands: source /tensorflow/bin/activate

```
python3 dnn_model_pandas_cat.py pm.train.csv pm.test.csv \
smoking,alcohol,excercise,substance,depression 4 600,600,600 800
```

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
import os
import sys
import numpy as np
import tensorflow as tf
import pandas as pd
if len(sys.argv)<7:
    print("Usage:python3 dnn_model_pandas_cat.py training_file testing_file categorical_columns(comma delimited) \
#classes layer_units(comma delimited) #steps")
    quit()
#process header
with open(sys.argv[1],'r') as fl:
    line=fl.readline().strip('\n')
header=line.split(',')
n_col=len(header)
features=[]
for i in range(n_col-1):
    features.append(header[i])
LABEL_COLUMN=header[n_col-1]
#process arguments
nsteps=int(sys.argv[6])
n_cls=int(sys.argv[4])
h_units=[]
lt=sys.argv[5].split(',')
for w in lt:
    h_units.append(int(w))
print("DNN units:")
print(h_units)
#read in training and testing data
df_train = pd.read_csv(sys.argv[1], names=header, skiprows=1, skipinitialspace=True)
df_test = pd.read_csv(sys.argv[2], names=header, skiprows=1, skipinitialspace=True)
#assign feature types for the DNN model
CATEGORICAL_COLUMNS=[]
categorical_ind=[]
CONTINUOUS_COLUMNS=[]
continuous_ind=[]
tt=sys.argv[3].split(',')
for w in tt:
    CATEGORICAL_COLUMNS.append(w)
    categorical_ind.append(tf.contrib.layers.embedding_column(tf.contrib.layers.sparse_column_with_hash_bucket(
        w, hash_bucket_size=1000),dimension=8))
for i in range(n_col-1):
    if header[i] not in CATEGORICAL_COLUMNS:
        CONTINUOUS_COLUMNS.append(header[i])
        continuous_ind.append(tf.contrib.layers.real_valued_column(header[i]))
deep_columns=[]
for s in categorical_ind:
    deep_columns.append(s)
for s in continuous_ind:
    deep_columns.append(s)
#convert input data into tensors
def input_fn(df):
    continuous_cols = {k: tf.constant(df[k].values) for k in CONTINUOUS_COLUMNS}
```

```
    categorical_cols = {
        k: tf.SparseTensor(
            indices=[[i, 0] for i in range(df[k].size)],
            values=df[k].values,
            dense_shape=[df[k].size, 1])
        for k in CATEGORICAL_COLUMNS}
    feature_cols = dict(continuous_cols)
    feature_cols.update(categorical_cols)
    label = tf.constant(df[LABEL_COLUMN].values)
    return feature_cols, label
#train the DNN model and evaluate its performance
def main(unused_argv):
    m = tf.contrib.learn.DNNClassifier(
        feature_columns=deep_columns,
        hidden_units=h_units,
        n_classes=n_cls)
    m.fit(input_fn=lambda: input_fn(df_train), steps=nsteps)
    results = m.evaluate(input_fn=lambda: input_fn(df_test), steps=1)
    print(results)
if __name__ == "__main__":
    tf.app.run()
```

A model accuracy of 0.764 is printed on the screen. The accuracy may differ a little in different executions. It is also desired to fine-tune the model by adjusting the parameters including numbers of layers and units, and number of steps.