รายงาน

Credit Approval Data

จัดทำโดย

นายอิทธิพร แก้วอำไพ

รหัสนักศึกษา 590610681

รายงานนี้เป็นส่วนหนึ่งของวิชา

Data Mining for CPE

261448

Title: Credit Approval

Data types: Multivariate

Instances: 690

Attribute Types: Categorical, Integer, Real

Attributes: 15 + 1(class attribute)

Attribute Information:

A1 – b, a.

A2 – continuous.

A3 – continuous.

A4 – u, y, l, t.

A5 – g, p, gg.

A6 – c, d, cc, i, j, k, m, r, q, w, x, e, aa, ff.

A7 – v, h, bb, j, n, z, dd, ff,o

A8 – continuous.

A9 – t, f.

A10 – t, f.

A11 – continuous.

A12 – t, f.

A13 – g, p, s.

A14 – continuous.

A15 – continuous.

A16 – (+, -) (class attribute)

Missing Attribute Values:

37 cases (5%) have one or more missing values. The missing values from particular attributes are:

A1: 12

A2: 12

A4: 6

A5: 6

A6: 9

A7: 9

A14: 13

Class Distribution:

+: 307 (44.5%)

-: 383 (55.5%)

Source: Submitted by quinlan@cs.su.oz.au

ขั้นตอนการทำ pre processing

- จัดการ row ที่มี missing values ตั้งแต่ 20% ขึ้นไปโดยการตัดทั้ง row เนื่องจากมี missing values มากเกินไป โดยจะมีดังนี้ ข้อมูลที่ 207(5), 271(5), 331(5), 457(5), 480(3), 593(5), 602(3), 623(5)
- จัดการเติม missing values ของ A1 โดยการเติม highest frequency value(b) เข้าไปในทุกที่ที่เป็น missing values(A1) เพื่อที่จะได้ไม่ทำให้ข้อมูลเพี้ยนจากเดิม
  โดยจะมีดังนี้ ข้อมูลที่ 249, 328, 347, 375, 454, 490, 521, 599, 642, 674
- เติม missing values ของ A2 โดยเติมตาม mean โดย mean = 31.5247 เพื่อไม่ให้ค่า mean เปลี่ยนไปจากเดิม โดยเติมข้อมูลที่ 84, 87, 93, 98, 254, 285, 328, 443, 448, 496, 511, 602
- เติม missing values ของ A6 และ A7 ของข้อมูลที่ 535 โดยค่าที่เติมไปคือ c และ v ตามลำดับโดยนำค่ามาจาก mod ของแต่ละ column
- เติม missing values ของ A14 โดนการนำ mean ไปแทนทุกค่าที่หายไป

การแบ่งชุดข้อมูลในการ Train และ Test

โดยในที่นี้ได้แบ่งข้อมูลในการ Train และ Test ดังนี้

Train 90 percent

Test 10 percent

โดยการที่แบ่ง Train 90% และ Test 10% ก็เพราะว่า ใน Credit Approval Data นั้นมี Attribute Value ที่มาก จึงจำเป็นต้องนำไป Train มากแต่ไม่มากจนเกินไปไม่งั้นจะเกิดการ overfitting ดังนั้นจึงต้องลองแบ่งข้อมูลหลายๆแบบไปทดลองจริงจนได้ค่า accuracy ที่ดีที่สุดหรือเรียกได้ว่า good fit นั่นเอง

สมมติฐานการทดลองที่ 1

การทดลองเพื่อทดสอบผลความถูกต้องจาก การจำแนก โดยการ preprocess ที่ต่างกัน

โดยข้อมูลชุดแรกเป็นข้อมูลที่ไม่ได้ผ่านการ preprocessing by human แต่เป็นการ preprocessing ด้วย code ดังนี้

```
# df.replace('?', np.nan, inplace=True)
# df.dropna(inplace=True)
```

โดย code ชุดนี้หมายถึง เมื่อ ? หมายถึง missing data ให้ drop ข้อมูลทั้งแถวทิ้งไปและข้อมูลอีกชุดคือข้อมูลที่ผ่านการ preprocessing by human โดยจะทำตามขั้นตอนการ preprocessing ที่กล่าวไว้ก่อนหน้านี้

ทำการทดลองโดยนำข้อมูลทั้งสองชุดไปผ่านการ classify และตรวจดู accuracy รวมทั้ง loss ของข้อมูลและเปรียบเทียบกันโดยที่มีเงื่อนไขที่เหมือนกันดังนี้

1. มี 2 hidden layers
2. แต่ละ layer มี node เท่ากันคือ 20
3. ทั้งสองชุดข้อมูลมี step คือ 1000

ข้อมูลชุดที่ผ่านการตัดออกด้วย code

```
accuracy : 0.8181818127632141
loss : 0.9992094039916992
step : 1000
```

ข้อมูลที่ผ่านการ preprocessing ด้วยมือ

```
accuracy : 0.8695651888847351
loss : 0.3574884533882141
step : 1000
```

จากการทดลองแสดงให้เห็นว่าข้อมูลที่ผ่านการ preprocessing ด้วยคำสั่ง มี loss ที่สูงมากเนื่องจากมีการตัดออกของข้อมูลที่มากเกินไปทำให้มีข้อมูลในการ Train และ Test ไม่พอ รวมทั้งมี accuracy ที่น้อยกว่าข้อมูลที่ผ่านการปรับปรุงด้วยมือ แสดงให้เห็นว่าการ preprocessing ส่งผลต่อการ classify data ที่เรามี

สมมติฐานการทดลองที่ 2

การทดลองเพื่อทดสอบโครงสร้าง NN ที่เหมาะสม เช่น มี output layer, มี 1 hidden layer มี 2 hidden layer

สมมติฐาน ถ้ายิ่งมี hidden layer มาก จะยิ่งมี accuracy ที่มากขึ้น

ทดลองโดยกำหนดให้ จำนวน node ของแต่ละ hidden layer คือ 20 และ step คือ 2000

1. Output ของ hidden layer = 1 (20 node per layer , step 2000)

ครั้งที่ 1

```
'loss': 0.32399812, 'accuracy': 0.8405797
```

ครั้งที่ 2

```
'loss': 0.2881121, 'accuracy': 0.8695652
```

ครั้งที่ 3

```
'loss': 0.33559856, 'accuracy': 0.8405797,
```

2. Output ของ hidden layer = 2 (20 node per layer , step 2000)

ครั้งที่ 1

```
'loss': 0.34216517, 'accuracy': 0.8405797
```

ครั้งที่ 2

```
'loss': 0.29935575, 'accuracy': 0.884058
```

ครั้งที่ 3

```
'loss': 0.30632317, 'accuracy': 0.89855075
```

3. Output ของ hidden layer = 3 (20 node per layer , step 2000)

ครั้งที่ 1

```
'loss': 0.50454694, 'accuracy': 0.79710144
```

ครั้งที่ 2

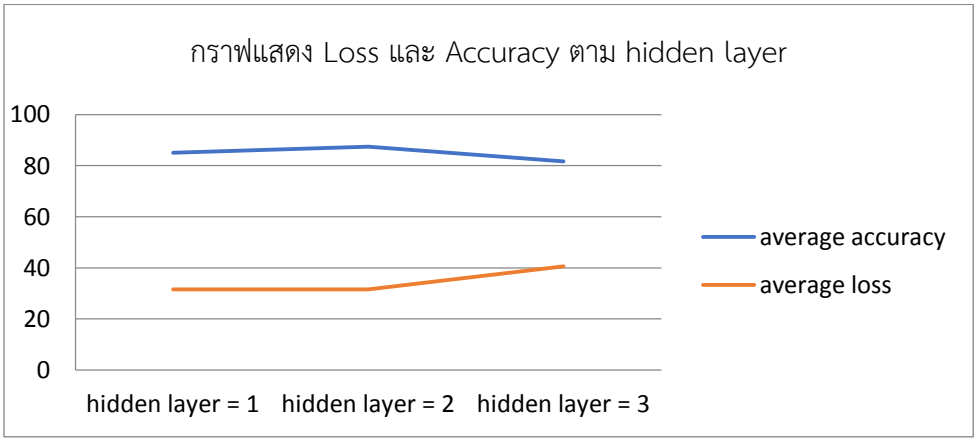
```
'loss': 0.3652779, 'accuracy': 0.8115942
```

ครั้งที่ 3

```
'loss': 0.34766418, 'accuracy': 0.8405797
```

จากการทดลองของ hidden layer สรุปได้ดังตารางนี้

จำนวน hidden layer	average accuracy	average loss
1	85.02%	31.59%
2	87.44%	31.58%
3	81.64%	40.58%



จากกราฟ จะสรุปได้ว่า hidden layer มากไม่ได้ส่งผลดีเสมอไป ดังนั้นแต่ละข้อมูลจะมี hidden layer ที่เหมาะสมต่างกันไป ดังนั้นในข้อมูลชุดนี้ hidden layer ที่เหมาะสมคือ 2 นั่นเอง แต่อาจจะแตกต่างกันไปหากใช้ node ในแต่ละ layer ต่างกันไป

สมมติฐานการทดลองที่ 3

การทดลองเพื่อทดสอบ จำนวน node ที่เหมาะสม

สมมติฐานการทดลอง เมื่อ node ยิ่งมากเท่าไหร่ accuracy ยิ่งมากขึ้นเท่านั้น

ทำการทดลองโดยกำหนดให้

จำนวน hidden layer คือ 1 และ step คือ 2000

ผลการทดลอง

1. จำนวน node 5 ตัว

2. จำนวน node 40 ตัว

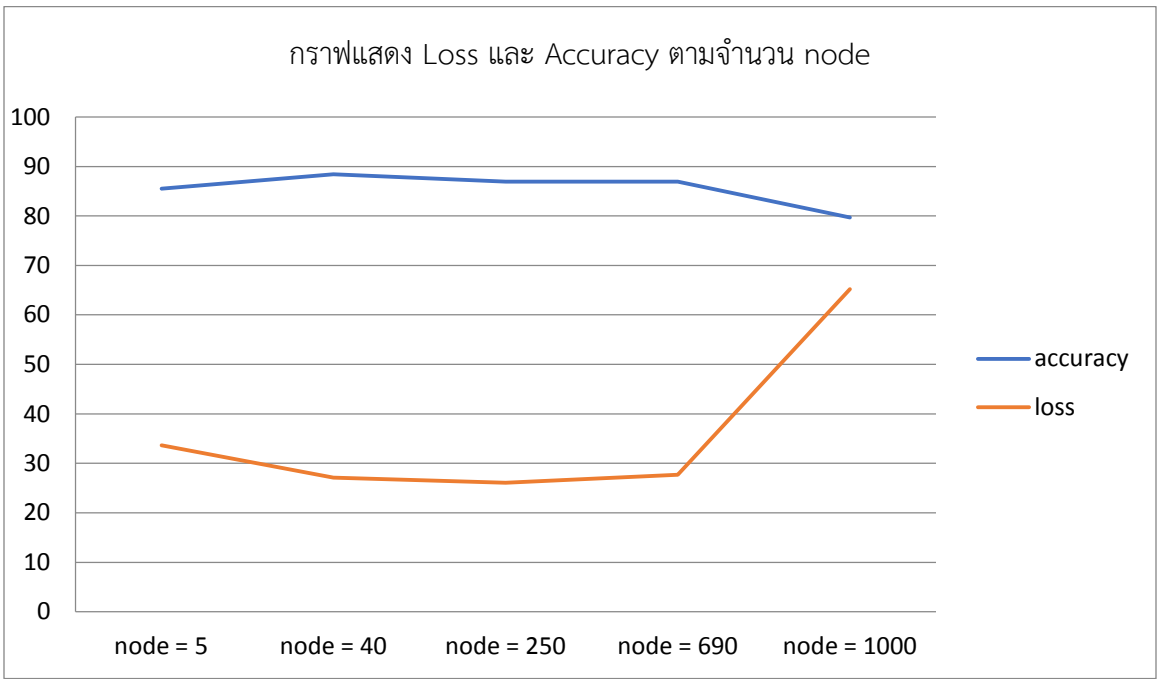
3. จำนวน node 250 ตัว

4 จำนวน node 690 ตัว

5 จำนวน node 1000 ตัว

จากการทดลองของ node สรุปได้ดังตารางดังนี้

จำนวน node	accuracy	loss
5	85.50%	33.66%
40	88.40%	27.11%
250	86.96%	26.06%
690	86.96%	27.68%
1000	79.71%	65.18%



จากกราฟแสดงให้เห็นว่าการที่มี node มากไปจะทำให้ output ที่ออกมามีค่า loss ที่สูงมากรวมทั้งค่า accuracy ลดลงดังนั้นจึงสรุปได้ว่า ในแต่ละ data จะมีค่า node ที่เหมาะสมกับข้อมูลอยู่แล้วไม่จำเป็นต้องใช้ node ที่มากเกินไป

สมมติฐานการทดลองที่ 4

ความสัมพันธ์ระหว่างการกำหนดค่า learning rate และ จำนวน epoch

สมมติฐาน ยิ่ง step มากขึ้นเท่าไหร่ accuracy จะมากขึ้นและ loss จะน้อยลง

ทดลองโดยมี 2 hidden layer (20 nodes per layer)

ผลการทดลอง

1. 100 steps

```
accuracy : 0.6666666865348816
loss : 1.8329358100891113
step : 100
```

2. 500 steps

```
accuracy : 0.8260869383811951
loss : 0.5072159767150879
step : 500
```

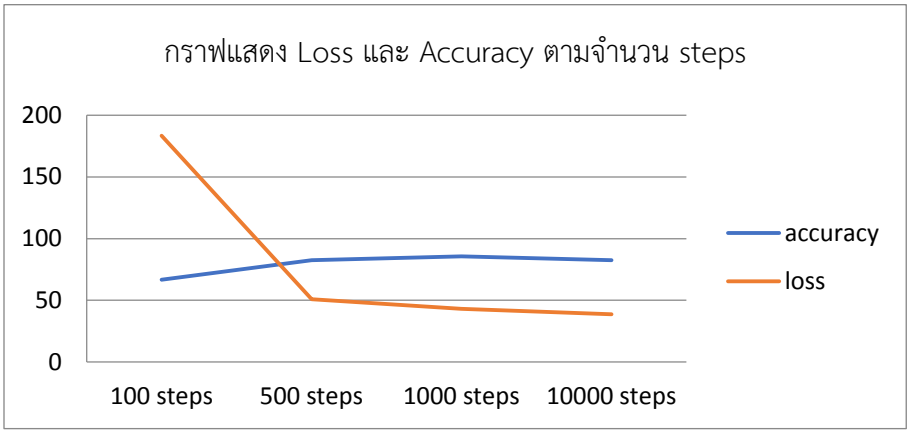
3. 1000 steps

```
accuracy : 0.8550724387168884
loss : 0.4304647743701935
step : 1000
```

4. 5000 steps

```
accuracy : 0.8260869383811951
loss : 0.38751572370529175
step : 5000
```

จะได้กราฟดังภาพ



จากกราฟสามารถสรุปได้ว่าจำนวน step หรือที่เรียกว่า epoch นั่นจะส่งผลต่อ accuracy แต่ที่เห็นได้ชัดคือ loss ที่ลดลงอย่างเห็นได้ชัดเจน ยิ่ง step ยิ่งดี แต่ก็จะส่งผลต่อความรวดเร็วในการคำนวณ ดังนั้นการเลือก step ที่เหมาะสมจะช่วยให้การคำนวณเป็นไปอย่างมีประสิทธิภาพโดยการคำนวณไม่ได้ขึ้นอยู่กับค่า step อย่างดียังขึ้นอยู่กับค่าอื่นๆอีกด้วย

สรุปผลการทดลอง

1. ข้อมูลนี้มี hidden layer ที่เหมาะสมคือ 2
2. มีจำนวน node ต่อ hidden layer คือ 20
3. step ที่เหมาะสมกับข้อมูลชุดนี้คือ 1000

Code โดยรวมทั้งหมด

```python
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import tensorflow as tf
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

import os

def input_fn(df,labels):
    feature_cols = {k:tf.constant(df[k].values,shape = [df[k].size,1]) for k in columns}
    label = tf.constant(labels.values, shape = [labels.size,1])
    return feature_cols,label

header = ['cat1','num1','num2','cat2','cat3','cat4','cat5','num3','cat6','cat7','num4','cat8','cat9','num5','num6','class']
df = pd.read_csv("../DATAMINING/crx.dataFixed.csv", names=header)

# df.replace('?', np.nan, inplace=True)
# df.dropna(inplace=True)

df['class'].replace('-',0,inplace=True)
df['class'].replace('+',1,inplace=True)

df = pd.get_dummies(df, columns=header[0:1])
df = pd.get_dummies(df, columns=header[3:7])
df = pd.get_dummies(df, columns=header[8:10])
df = pd.get_dummies(df, columns=header[11:13])

df = df.reindex(columns=(['class'] + list([a for a in df.columns if a != 'class']) ))

x_train, x_test, y_train, y_test = train_test_split(df.iloc[:,1:47], df["class"], test_size=0.1, random_state=30)

columns = df.columns[1:47]

feature_columns = [tf.contrib.layers.real_valued_column(k) for k in columns]
classifier = tf.contrib.learn.DNNClassifier(feature_columns=feature_columns,hidden_units=[20,20],n_classes = 2
    # ,optimizer=tf.train.ProximalAdagradOptimizer(
    #         learning_rate=0.5,
    #         # l1_regularization_strength=0.001
    #     )
        )

classifier.fit(input_fn=lambda: input_fn(x_train,y_train),steps = 1000)

acc = classifier.evaluate(input_fn=lambda: input_fn(x_test,y_test),steps=1)["accuracy"]
loss= classifier.evaluate(input_fn=lambda: input_fn(x_test,y_test),steps=1)["loss"]
step= classifier.evaluate(input_fn=lambda: input_fn(x_test,y_test),steps=1)["global_step"]
print("accuracy : "+ format(acc))
print("loss : "+ format(loss))
print("step : "+ format(step))

```

อธิบาย code ที่ใช้งาน (เขียนในภาษา python)

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import tensorflow as tf
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import os
```

import สิ่งที่จำเป็นในการทำการสิ่งที่จำเป็นในการทำงานครั้งนี้

tensorflow – เป็นไลบรารีที่ใช้ในการพัฒนา Machine Learning

numpy – เป็นโมดูลส่วนเสริมของ Python ที่มีฟังก์ชันเกี่ยวกับคณิตศาสตร์และการคำนวณต่างๆ มาให้ใช้งาน

pandas – Library ที่มี Data Structure และ Function สำหรับการจัดการและเตรียมข้อมูลใน Python

ขั้นตอนการนำเข้าข้อมูล

```
header = ['cat1','num1','num2','cat2','cat3','cat4','cat5','num3','cat6','cat7','num4','cat8','cat9','num5','num6','class']
df = pd.read_csv("../DATAMINING/crx.dataFixed.csv", names=header)
```

กำหนด header ให้ในแต่ละ column เพื่อให้ง่ายต่อการเรียกใช้ในภายหลัง

ขั้นตอนการจัดการข้อมูล

```
df['class'].replace('-',0,inplace=True)
df['class'].replace('+',1,inplace=True)

df = pd.get_dummies(df, columns=header[0:1])
df = pd.get_dummies(df, columns=header[3:7])
df = pd.get_dummies(df, columns=header[8:10])
df = pd.get_dummies(df, columns=header[11:13])
```

เปลี่ยนค่า class + - ให้เป็นตัวเลขเพื่อนำไปใช้งานต่อรวมทั้งการ get_dummies ของ pandas จะช่วยให้จัดการข้อมูลที่เป็น categories ได้ง่ายยิ่งขึ้น

แบ่งข้อมูลก่อนนำไป train และ test

```
x_train, x_test, y_train, y_test = train_test_split(df.iloc[:,1:47], df["class"], test_size=0.1, random_state=30)
```

จากภาพจะเห็นว่าพารามิเตอร์ของ test_size คือ 0.1 หมายถึง train 90% test 10% random_state คือการสุ่มสลับข้อมูล

ขั้นตอนการ train test , create model และวิเคราะห์ผล

```
columns = df.columns[1:47]
# Specify that all features have real-value data
feature_columns = [tf.contrib.layers.real_valued_column(k) for k in columns]
# Build 2 layer DNN with 20, 20 units respectively.
classifier = tf.contrib.learn.DNNClassifier(feature_columns=feature_columns,hidden_units=[20,20],n_classes = 2
    # ,optimizer=tf.train.ProximalAdagradOptimizer(
    #         learning_rate=0.5,
    #         # l1_regularization_strength=0.001
    #     )
        )
```

จากภาพเป็นการกำหนด feature columns เพื่อนำไป train โดยมี hidden layers 2 ชั้นดูจาก พารามิเตอร์ hidden_units และมีการกำหนด class ไว้ว่า 2 ในที่นี้ data มีเพียง 2 class

```
def input_fn(df,labels):
    feature_cols = {k:tf.constant(df[k].values,shape = [df[k].size,1]) for k in columns}
    label = tf.constant(labels.values, shape = [labels.size,1])
    return feature_cols,label

classifier.fit(input_fn=lambda: input_fn(x_train,y_train),steps = 1000)

acc = classifier.evaluate(input_fn=lambda: input_fn(x_test,y_test),steps=1)["accuracy"]
loss= classifier.evaluate(input_fn=lambda: input_fn(x_test,y_test),steps=1)["loss"]
step= classifier.evaluate(input_fn=lambda: input_fn(x_test,y_test),steps=1)["global_step"]
print("accuracy : "+ format(acc))
print("loss : "+ format(loss))
print("step : "+ format(step))
```

จากนั้นจะมีการนำไป train ในคำสั่ง classifier.fit และวิเคราะห์ผลออกมาตามที่ต้องการในคำสั่ง classifier.evaluate โดย lambda นั่นหมายถึงการประมวลผลแบบ ไร้เซิร์ฟเวอร์ จะทำให้สามารถประมวลผลได้ในเครื่องของตนเอง

ตัวอย่าง Output

```
WIN7PRO@WIN7PRO-PC MINGW64 /e/Github/Datamining (master)
$ py with_style.py

WARNING: The TensorFlow contrib module will not be included in TensorFlow 2.0.
For more information, please see:
  * https://github.com/tensorflow/community/blob/master/rfcs/20180907-contrib-sunset.md
  * https://github.com/tensorflow/addons
If you depend on functionality not listed there, please file an issue.

WARNING:tensorflow:From C:\Users\WIN7PRO\AppData\Local\Programs\Python\Python36\lib\site-packages\tensorflow\contrib\learn\python\learn\estimators\dnn.py:378: multi_class_head (f
rom tensorflow.contrib.learn.python.learn.estimators.head) is deprecated and will be removed in a future version.
Instructions for updating:
Please switch to tf.contrib.estimator.*_head.
WARNING:tensorflow:From C:\Users\WIN7PRO\AppData\Local\Programs\Python\Python36\lib\site-packages\tensorflow\contrib\learn\python\learn\estimators\estimator.py:1179: BaseEstimato
r.__init__ (from tensorflow.contrib.learn.python.learn.estimators.estimator) is deprecated and will be removed in a future version.
Instructions for updating:
Please replace uses of any Estimator from tf.contrib.learn with an Estimator from tf.estimator.*
WARNING:tensorflow:From C:\Users\WIN7PRO\AppData\Local\Programs\Python\Python36\lib\site-packages\tensorflow\contrib\learn\python\learn\estimators\estimator.py:427: RunConfig.__i
nit__ (from tensorflow.contrib.learn.python.learn.estimators.run_config) is deprecated and will be removed in a future version.
Instructions for updating:
When switching to tf.estimator.Estimator, use tf.estimator.RunConfig instead.
WARNING:tensorflow:Using temporary folder as model directory: C:\Users\WIN7PRO\AppData\Local\Temp\tmpi0_e0ly4
WARNING:tensorflow:From C:\Users\WIN7PRO\AppData\Local\Programs\Python\Python36\lib\site-packages\tensorflow\python\framework\op_def_library.py:263: colocate_with (from tensorflo
w.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From C:\Users\WIN7PRO\AppData\Local\Programs\Python\Python36\lib\site-packages\tensorflow\contrib\layers\python\layers\feature_column.py:1874: to_float (from t
ensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
WARNING:tensorflow:Casting <dtype: 'int64'> labels to bool.
WARNING:tensorflow:From C:\Users\WIN7PRO\AppData\Local\Programs\Python\Python36\lib\site-packages\tensorflow\python\ops\metrics_impl.py:788: div (from tensorflow.python.ops.math_
ops) is deprecated and will be removed in a future version.
Instructions for updating:
Deprecated in favor of operator or tf.math.divide.
WARNING:tensorflow:Casting <dtype: 'int64'> labels to bool.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect PR-AUCs; please switch to "careful_interpolation" instead.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect PR-AUCs; please switch to "careful_interpolation" instead.
WARNING:tensorflow:From C:\Users\WIN7PRO\AppData\Local\Programs\Python\Python36\lib\site-packages\tensorflow\contrib\learn\python\learn\estimators\head.py:677: ModelFnOps.__new__
 (from tensorflow.contrib.learn.python.learn.estimators.model_fn) is deprecated and will be removed in a future version.
Instructions for updating:
When switching to tf.estimator.Estimator, use tf.estimator.EstimatorSpec. You can use the `estimator_spec` method to create an equivalent one.
2019-04-17 17:14:01.333793: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
WARNING:tensorflow:Casting <dtype: 'int64'> labels to bool.
WARNING:tensorflow:Casting <dtype: 'int64'> labels to bool.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect PR-AUCs; please switch to "careful_interpolation" instead.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect PR-AUCs; please switch to "careful_interpolation" instead.
WARNING:tensorflow:From C:\Users\WIN7PRO\AppData\Local\Programs\Python\Python36\lib\site-packages\tensorflow\python\training\saver.py:1266: checkpoint_exists (from tensorflow.pyt
hon.training.checkpoint_management) is deprecated and will be removed in a future version.
Instructions for updating:
Use standard file APIs to check for files with this prefix.
WARNING:tensorflow:Casting <dtype: 'int64'> labels to bool.
WARNING:tensorflow:Casting <dtype: 'int64'> labels to bool.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect PR-AUCs; please switch to "careful_interpolation" instead.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect PR-AUCs; please switch to "careful_interpolation" instead.
WARNING:tensorflow:Casting <dtype: 'int64'> labels to bool.
WARNING:tensorflow:Casting <dtype: 'int64'> labels to bool.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect PR-AUCs; please switch to "careful_interpolation" instead.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect PR-AUCs; please switch to "careful_interpolation" instead.
accuracy : 0.8840579986572266
loss : 0.32669389247894287
step : 1000
```

For more information about this project see at : https://github.com/RAIDSouL/Datamining