DL: 10/5/20 10:00 am

ANN - Classification

Code :- on Tensorflow, Keras.

Tensorflow, CNTK, theano

Google | Micro soft | FaceBook

```
# PIP install tensorflow
# PIP install keras
```

```
import numpy as np
import pandas as pd
import tensorflow as tf
import keras
# from tensorflow import keras.
# df = pd.read_csv("churn_modelling.csv")
# df.head()
```

out:

Row no.	customer id	Surname	Credit Score	Geo graphy	Gender	Age	Tenure	Bal ance	num of Products	Has card	Is active numb	Estimated Salary
1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348
2	15647311	Hill	608	Spain	Female	41	1	83807	1	0	1	112542
3	15619304	Onio	502	France	Female	42	8	159660.	3	1	0	113931
4	15701354	Boni	699	Feance	Female	39	1	0.00	2	0	0	93826.
5	15737888	Mitchell	850	Spain	Femal	43	2	125510	1	1	1	79084.

```
# df.info()
```

out:

```
Range Index : 10000 entries , 0 to 9999
Data columns.                Nonnull columns      Dtype.
0.  Row number               10000 Non null       int64
1.  Customer Id              1000 Non null        int64
```

3	Credit score	10000 non null	int 64
2	Surname	10000 non null	object
categorical data 4	Geography	10000 non-null	object] convert to numeric
5	Gender	"	
6	Age	"	int64
7	Tenure	"	int 64
8	Balance	"	float 64
9	No. of Products	"	int 64
10	has Cr Card	"	int 64
11	Is Active member	"	int 64
12	Estimated Salary	"	float 64
13	Exited	"	int 64

we can use drop function also [x & y]

```
# x = df.iloc[:, 3:13].values
# y = df.iloc[:, 13].values
```

: → all; 3:13 → 3 to 12; [:,13] → only 13 column

original data: 13 input — 3 drop (row no., customer id, surname) → 10 inputs; 1 output → 1 column

```
# x.shape
```

Out: (10000, 10)

```
# y.shape
```

Out: (10000,)

ordinal data (sequence) — natural

↑ Label Encoder

```
# encoding [categorical data]
from sklearn.preprocessing import
# x[:, 2]
```

Out: array ["Female", "Female", ... "male", "Female"].

```
# labelencoder = LabelEncoder()
# x[:,2] = labelencoder.fit_transform(x[:,2])
# changed to numerical.
# x[:,2]
out: array [0,0,0 .... 0,1,0]
```

```
# one hot encoding (nominal data = Geography (column)) # We can use get_dummies.
from sklearn.preprocessing import OneHotEncoder
***
from sklearn.compose import ColumnTransformer
                              ↳ Transforms data.

# x[:,1]
out: array([["France", "spain", "France" ... "France", "Germany"].
           dtype= object.

# ct = ColumnTransformer([["encoder", OneHotEncoder(), [1]]
                          , remainder = "passthrough")
# x = np.array(ct.fit_transform(x))
# changed to numerical
# x[:,1]
out: array([[0.0, 0.0, 0.0, .... 0.0, 1.0, 0.0].
```

train_test

```
from sklearn.model_selection import train_test_split

# x_train, x_test, y_train, y_test = train_test_split(x, y,
                                     test_size = 0.2, random_state = 0)
```

```
# Shape.
# x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Out: (8000, 12), (2000, 12), (8000,), (2000,)

↳ after applying dummies it converts into 12 columns.

Scaling

```
from sklearn.preprocessing import StandardScaler

# sc = StandardScaler()
# x_train = sc.fit_transform(x_train)
# x_test = sc.fit_transform(x_test)
```

MODELLING

Initializing The ANN

```
from keras.models import Sequential
# ann = Sequential()
```

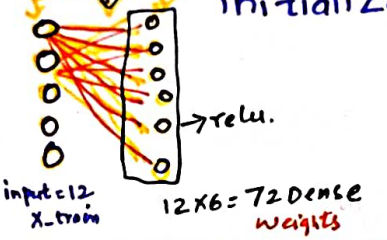
Adding The input layer and first hidden layer.

```
from keras.layers import Dense.

# ann.add(Dense(input_dim = 12, units = 6, kernel_
    initializer = "uniform", activation = "relu"
    ))
```

connections

Hidden Layer

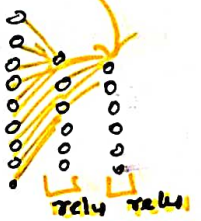


$\cup \left[\frac{-1}{\sqrt{in}}, \frac{+1}{\sqrt{in}}\right]$

$\cup \left[\frac{-1}{\sqrt{12}}, \frac{1}{\sqrt{12}}\right]$

Adding the second hidden Layer.

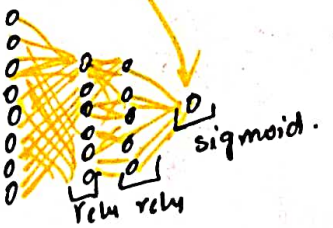
```
# ann.add(Dense(units = 6, kernel_initializer = "uniform",
    activation = "relu"))
```



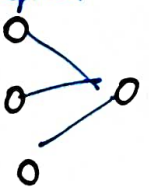
Adding the output layer

output

```
# ann.add(Dense(units = 1, kernel_initializer = "uniform",
    activation = "sigmoid"))
```



Ex:- Perceptron model.



```
(ann.add(Dense(input_dim = 3, units = 1, kernel_initialization
    = "uniform", activation = "relu")).
```

Part-3 Training the ANN

Compiling the ANN

Ex: current
past: Connection is made (wiring), (data)

$\sum(y-\hat{y})^2$
regression = "mean_squared_error"

```
# ann.compile(optimizer = "adam", loss = "binary_crossentropy",
                  Gradient descent          function
                  metrics = ["accuracy"])
```

model is created, fit the data

training the ANN on training Set

```
# ann.fit(x_train, y_train, epochs = 100)
```

epochs → no. of iteration. (backward, forward directions)

Out

```
Epoch 1/100
250/250 [=========] - 2s 1ms/step - loss: 0.6457, - accuracy - 0.7830)
Epoch 2/100
. . . . . .
. . . . . .
. . . . . .
Epoch 100/100
```

accu: 0.838

Part-4 Predictions & Evaluating model

making the predictions

Predict
Test accuracy.

```
# y_pred = ann.Predict(x_test)
# y_pred = (y_pred > 0.5)
```

Evaluating the model

```
from sklearn.metrics import confusion_matrix, accuracy_score

# print ("Test Accuracy:", accuracy_score (y_test, y_pred))

# confusion_matrix (y_test, y_pred)
```

Out: Test accuracy : 08455

```
array([[1536, 59
        [150, 155]].
```

cross validate the model

```
# user defined function.

def build_cross_classifier():
    classifier = Sequential()
    classifier.add(Dense(input_dim=12, units=6, kernel_
                         initializer="uniform", activation="relu
    classifier.add(Dense(units=6, kernel_initializer=
                         "uniform", activation="relu"))
    classifier.add(Dense(units=1, kernel_initializer
                         "uniform", activation="sigmo
    classifier.compile(optimizer="adam", loss="binary_
                       cross_entropy, metrics=["Acc

    return classifier
```

ann (or) anything we can use

input layer

hidden layer

output layer

connecting

Classifier

```
from keras.wrappers.scikit_learn import KerasClassifier

# classifier = KerasClassifier(build_fn = build_cross_classifier
                               , batch_size = 10 , epochs = 100)
```

build_fn = User defined function

batch_size = if stochastic descent

epochs = iterations

Cross_validation

```
from sklearn.model_selection import cross_val_score

# accuracies = cross_val_score(estimator = classifier ,
                               X = x_train , y = y_train, cv=5)
```

out :

```
Epoch 1/100
640/640 [======] - 1s  605us/step - loss: 0.4922, -accr: 0.7997
Epoch 2/100
640/640 [======] - 0s  617us/step - loss: 0.4296, -acc, 0.8003
. . . . . .
. . . . . .
. . . . . .
. . . . . .
Epoch 101/100
640/640 [======] 1s 805/step - loss: 0.4003. acc: 0.8342]
160/160 [======] 0s 666us/step - loss: 0.4110 . acc: 0.8375 ]
```

```
# print(accuracies)
# accuracies.mean()
[0.8237 . 0.8343 , 0.8362 , 0.8362, 0.8374)
```

out

0.83362

⇒ Hyperparameter Tunning

Part-5 - Improving and Tuning the ANN

This can done by 3 options

1. Hyperparamter tuning
2. Regularization (L1 & L2) to reduce overfitting if needed (for regression problems)
3. Dropout

Hyperparameter tuning Can be done for identifying no. of hidden Layers & no. of neurons in each hidden Layer.

- Layers = [[20],[10],[30],[40]] → 1 hidden Layer with different options of no. of neurons in hidden Layer
- Layers = [[20],[40,20],[45,30,15]] → Multiple hidden Layers with different options of no. of neurons.

Hyperparameter tuning Can be done for identifying best activation function for hidden Layers.

- activations = ["relu", "sigmoid"]

Hyperpameter tuning Can be done for identify best optimizers

- Optimizer = ["adam", "rmsprop"]
  - "adam" → gradient desent
  - "rmsprop" → Stochatic gradient desent.

Hyper parameter tunning for batchsize for building/training model.

- batch size = [10, 16, 32, 64, 128, 256]

Code: tunning :-

```
def build_classifier(optimizer):
    classifier = Sequential()
    classifier.add(Dense(input_dim=12, units=6, kernel_
                         initializer="uniform",
    classifier.add(Dense(units=6, kernel_initializer="uniform
                         ,activation="relu"))
    classifier.add(Dense(units=1, kernel_initializer=
                         "uniform", activation="sigmoid"))
    classifier.compile(optimizer=optimizer, loss="binary_
                       crossentropy", metrics=["accuracy"])
    return classifier

# Classifier = KerasClassifier(build_fn=build_classifier)
# parameters = {"batch_size": [10,32], "epochs": [50,100],
               "optimizer": ["adam", "rmsprop"]}
```

```
from sklearn.model_selection import GridSearchCV

# grid = GridSearchCV(estimator = classifier, param_grid =
                      parameters, scoring = "accuracy", cv= 5)

# grid_result = grid.fit(x_train, y_train)
```

Out: Epoch 1/100
200/200 [==========] - 0s 711us/step - loss: 0.5877, accu. 0.7117]
Epoch 2/100
200/200 [==========] - 0s 630us/step - loss: 0.4392 - acc -0.8003)

.....

Epoch 100/100
200/200 [==========] - 0s 83 0us/step - loss: 0.6392. acc-84]

```
# grid_result.best_params_
```

Out: {"batch_size": 32, "epochs": 50, "optimizer": "adam"}

best accuracy

```
# grid_result.best_score_
```

Out: 0.844125

```
# y_pred = grid_result.predict(x_test)
# y_pred = (y_pred > 0.5)
```

accuracy, confusion_matrix

```
from sklearn.metrics import confusion_matrix, accuracy_
                            score.

# print ("test accuracy:", accuracy_score (y_test, y_pred))
# confusion_matrix (y_test, y_pred)
```

Out: Test accuracy : 0.8405

```
array([[1530,  65]
       [ 254,  151]],
```

10/05/22

5:30 pm.