DEEP LEARNING FINAL PROJECT REPORT

(DS-6670-01-F19)

On

CREDIT CARD ELIGIBILITY

By

RAHUL AKKINENI (ID: 00689658)

Under the guidance of

KEITH DILLON, Ph.D.

Tagliatela College of Engineering

Department of Data Science

300 Boston Post Rd, West Haven, CT 06516

ACKNOWLEDGEMENT

My most sincere and grateful acknowledgement is due to this sanctum, **University Of Newhaven,** for giving us the opportunity to fulfill my aspirations and for good career.

I express my heartfelt gratitude to my guide **Mr. Keith Dillon, Ph.D,** for his esteemed guidance. I am indebted for his guidelines that proved to be very much helpful in completing my project successfully in time.

My special thanks to my classmates and seniors for providing their special guidance and support for the completion of my project successfully.

Submitted By

Rahul Akkineni

(00689658)

1. THE DATA AND THE METHODS I AM TRYING TO SOLVE

1.1 THE DATA:

The columns in dataset are,

- Customer_id: The customer’s identification number.
- Demographic_slice: The area under which he comes.
- Country_reg: The region of the country he comes from.
- Ad_exp: The add on experience of the customer (i.e., if there is any previous impression on customer)
- Est_income: The estimated income of the customer.
- Hold_bal: Is there any amount on hold on the customer’s present account.
- Pref_cust_prob: How much is the customer preferred to be given the card.
- Imp_cscore: The credit score of the customer.
- Risk_score: How much risk is there in providing the customer with the credit card.
- Imp_crediteval: The credit evaluation of the customer.
- Axio_score: The probability score of getting the card.
- Card_offer: Whether the card is offered to the customer.

The number of samples in test data are 10000 and number of samples in train data are 10000. The number of features are 12.

1.2 METHODS I AM TRYING TO SOLVE:

- Tabular data
- Preprocessing data
- Multilayer Layer Neural Network (in Keras)
- Generators

2. THE BACKGROUND

References : https://www.kaggle.com/amarvw/customercreditcard

There is no Keras code for this dataset.

Data Generator - https://github.com/anujshah1003/custom_data_generator/blob/master/flowers_recognition/data_generator_demo.ipynb

https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly

Understanding and Exploring - https://towardsdatascience.com/tabular-data-analysis-with-deep-neural-nets-d39e10efb6e0

Model Building - https://machinelearningmastery.com/how-to-improve-deep-learning-model-robustness-by-adding-noise/

3. APPROACHES

USED AND MULTIPLE APPROACHES

3.1. TABULAR DATA:

First of all I have combined the dataset of train and test, which I later used train_test_split to separate them for training and testing according to required proportions.

In [15]:

```
credit_data = pd.read_csv('F:/Rahul Subjects/Deep Learning/Total/DataSet.csv')
```

In [18]:

```
credit_data.head()
```

Out[18]:

	customer_id	demographic_slice	country_reg	ad_exp	est_income	hold_bal	pref_cust_pro
0	713782	AX03efs	W	N	33407.90175	3.000000	0.53111
1	515901	AX03efs	E	N	19927.53353	20.257927	0.29743
2	95166	AX03efs	W	Y	51222.47100	4.000000	0.01846
3	425557	AX03efs	E	Y	67211.58747	18.653631	0.08934
4	624581	AX03efs	W	N	20093.34216	4.000000	0.09494

3.2. PREPROCESSING DATA:

Since there are categorical variables in different columns I have converted them into integer values.

Preprocessing Data:

Creating new columns in the dataset and loading the columns with categorical values into their respective label columns and converting the values in the columns to integers.

In [20]:

```
data['country_reg_label'] = credit_data['country_reg']
cleanup_nums = {"country_reg_label": {"W" : 0, "E" : 1}}
data.replace(cleanup_nums, inplace=True)
data.head()
```

In [21]:

```
data['ad_exp_label'] = credit_data['ad_exp']
cleanup_nums = {"ad_exp_label": {"N" : 0, "Y" : 1}}
data.replace(cleanup_nums, inplace=True)
data.head()
```

In [23]:

```
data['card_offer_label'] = credit_data['card_offer']
```

In [24]:

```
data['card_offer_label'] = (data['card_offer_label'] == True).astype(int)
```

In [26]:

```
data['demographic_slice'].unique()
```

Out[26]:

```
array(['AX03efs', 'BWEsk45', 'CARDIF2', 'DERS3w5'], dtype=object)
```

In [27]:

```
data['demographic_slice_label'] = credit_data['demographic_slice']
cleanup_nums = {"demographic_slice_label": {"AX03efs" : 1, "BWEsk45" : 2, "CARDIF2" :
data.replace(cleanup_nums, inplace=True)
data.head()
```

Now dropping the categorical columns and retaining the corresponding label columns with integers which can be later used for building models and applying functions.

The data after pre processing looks like following:

In [28]:

```
data = data.drop(['demographic_slice', 'country_reg', 'ad_exp', 'card_offer'], axis=1)
```

In [29]:

```
data.head()
```

Out[29]:

	customer_id	est_income	hold_bal	pref_cust_prob	imp_cscore	RiskScore	imp_crediteva
0	713782	33407.90175	3.000000	0.531112	619	503.249027	23.977827
1	515901	19927.53353	20.257927	0.297439	527	820.108146	22.986398
2	95166	51222.47100	4.000000	0.018463	606	586.605795	24.939219
3	425557	67211.58747	18.653631	0.089344	585	634.701982	24.841147
4	624581	20093.34216	4.000000	0.094948	567	631.949979	24.67936:

Now extracting the features and targets for model building:

In [30]:

```
X = data.drop('card_offer_label', axis=1)
y = data['card_offer_label']
y1 = to_categorical(y)
```

Normalizing Data:

In [31]:

```
np.linalg.norm(keras.utils.normalize([[1,2,3],[2,3,4]],axis=0)[:,0])
```

Out[31]:

```
0.9999999999999999
```

Splitting data for training and testing:

In [32]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y1)
```

3.2. MULTILAYER NUERAL NETWORK(KERAS):

Finally stepping into creating models and evaluating the results. I have used a lot of combinations of dense layers and activation functions to see which works better. I am only listing a few here. The remaining can be found in the appendix and the code.

Loss - Categorical Crossentropy and Optimizer – Adam

In [33]:

```
model = models.Sequential()
model.add(layers.Dense(output_dim=128, init='uniform', activation='relu', input_dim=11
model.add(layers.Dense(output_dim=32, init='uniform', activation='relu'))
model.add(layers.Dense(output_dim=2, activation='softmax', input_dim=30))
model.add(layers.Dense(2, input_dim = 11, activation = 'sigmoid'))
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               1536
_________________________________________________________________
dense_2 (Dense)              (None, 32)                4128
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 66
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 6
=================================================================
Total params: 5,736
Trainable params: 5,736
Non-trainable params: 0
_________________________________________________________________
```

In [34]:

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

In [36]:

```
test = model.evaluate(X_test, y_test)
print(test[1]*100)
```

```
5000/5000 [==============================] - 0s 22us/step
92.10000038146973
```

Adding Gaussian Noise

In [37]:

```
model = models.Sequential()
model.add(layers.Dense(output_dim=128, init='uniform', activation='relu', input_dim=11
model.add(layers.Dense(output_dim=32, init='uniform', activation='relu'))
model.add(GaussianNoise(0.1))
model.add(layers.Dense(output_dim=2, activation='softmax'))
model.add(layers.Dense(output_dim=2, activation = 'sigmoid'))
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 128)               1536
_________________________________________________________________
dense_6 (Dense)              (None, 32)                4128
_________________________________________________________________
gaussian_noise_1 (GaussianNo (None, 32)                0
_________________________________________________________________
dense_7 (Dense)              (None, 2)                 66
_________________________________________________________________
dense_8 (Dense)              (None, 2)                 6
=================================================================
Total params: 5,736
Trainable params: 5,736
Non-trainable params: 0
```

In [79]:

```
test = model.evaluate(X_test, y_test)
print(test[1]*100)
```

```
5000/5000 [==============================] - 0s 45us/step
91.86000227928162
```

Loss - Binary Crossentropy and Optimizer – Adam

In [41]:

```
model = models.Sequential()
model.add(layers.Dense(output_dim=128, init='uniform', activation='relu', input_dim=11
model.add(layers.Dense(output_dim=32, init='uniform', activation='relu'))
model.add(GaussianNoise(0.5))
model.add(layers.Dense(output_dim=2, activation='softmax'))
model.add(layers.Dense(output_dim=2, activation = 'sigmoid'))
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_9 (Dense)              (None, 128)               1536
_________________________________________________________________
dense_10 (Dense)             (None, 32)                4128
_________________________________________________________________
gaussian_noise_2 (GaussianNo (None, 32)                0
_________________________________________________________________
dense_11 (Dense)             (None, 2)                 66
_________________________________________________________________
dense_12 (Dense)             (None, 2)                 6
=================================================================
Total params: 5,736
Trainable params: 5,736
Non-trainable params: 0
```

In [42]:

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [44]:

```
test = model.evaluate(X_test, y_test)
print(test[1]*100)
```

```
5000/5000 [==============================] - 0s 30us/step
92.10000038146973
```

Adding Dropout to avoid overfitting

In [53]:

```
model = models.Sequential()
model.add(layers.Dropout(0.2, input_shape=(11,)))
model.add(layers.Dense(60, activation='relu', kernel_constraint=maxnorm(3)))
model.add(layers.Dense(30, activation='relu', kernel_constraint=maxnorm(3)))
model.add(layers.Dense(2, activation='sigmoid'))
sgd = SGD(lr=0.1, momentum=0.9)
model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dropout_1 (Dropout)          (None, 11)                0
_________________________________________________________________
dense_18 (Dense)             (None, 60)                720
_________________________________________________________________
dense_19 (Dense)             (None, 30)                1830
_________________________________________________________________
dense_20 (Dense)             (None, 2)                 62
=================================================================
Total params: 2,612
Trainable params: 2,612
Non-trainable params: 0
_________________________________________________________________
```

In [71]:

```
test = model.evaluate(X_test, y_test)
print(test[1]*100)
```

```
5000/5000 [==============================] - 0s 38us/step
71.72999978065491
```

Loss - Mean Squared Error and Optimizer – Adam

In [89]:

```
model = models.Sequential()
model.add(layers.Dense(60, input_dim=11, activation='relu'))
model.add(layers.Dense(30,  activation='relu'))
model.add(layers.Dense(2, activation='sigmoid'))
adam = keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, amsgrad=Fa
model.summary()
```

```
Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_47 (Dense)             (None, 60)                720
_________________________________________________________________
dense_48 (Dense)             (None, 30)                1830
_________________________________________________________________
dense_49 (Dense)             (None, 2)                 62
=================================================================
Total params: 2,612
Trainable params: 2,612
Non-trainable params: 0
_________________________________________________________________
```

In [90]:

```
model.compile(loss='mean_squared_error', optimizer=adam, metrics=['accuracy'])
```

In [92]:

```
test = model.evaluate(X_test, y_test)
print(test[1]*100)
```

```
5000/5000 [==============================] - 0s 41us/step
92.10000038146973
```

3.3. GENERATORS:

I have created generators using yield statement which returns a sample of the data when called. Due to this the accuracy may not be same overall, since, different samples give different accuracy.

In [101]:

```
def gen():
    while True:
        sample = data.sample()
        y = sample[['card_offer_label']]
        X = sample[['customer_id','est_income','hold_bal','pref_cust_prob', 'imp_cscor
        y = np.ravel(y.values)
        X = X.values
        X = np.asarray(X)
        y = np.ravel(y)
        X = np.ravel(X)
        X = X.reshape(1,11)
        yield X, y
```

In [90]:

```
model = models.Sequential()
model.add(layers.Dense(60, input_dim=11, activation='relu'))
model.add(layers.Dense(30,  activation='relu'))
#model.add(layers.Dense(2, activation='sigmoid'))
nadam = keras.optimizers.Nadam(learning_rate=0.002, beta_1=0.9, beta_2=0.999)
model.add(GaussianNoise(0.1))
model.add(layers.Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_56 (Dense)             (None, 60)                720
_________________________________________________________________
dense_57 (Dense)             (None, 30)                1830
_________________________________________________________________
gaussian_noise_6 (GaussianNo (None, 30)                0
_________________________________________________________________
dense_58 (Dense)             (None, 1)                 31
=================================================================
Total params: 2,581
Trainable params: 2,581
Non-trainable params: 0
_________________________________________________________________
```

In [104]:

```
model.compile(loss='mean_squared_error', optimizer=nadam, metrics=['accuracy'])
```

In [105]:

```
model.fit_generator(mygen,steps_per_epoch=1000,epochs=10)
```

```
Epoch 1/10
1000/1000 [==============================] - 5s 5ms/step - loss: 0.0730 - ac
curacy: 0.9270
Epoch 2/10
1000/1000 [==============================] - 4s 4ms/step - loss: 0.0850 - ac
curacy: 0.9150
Epoch 3/10
1000/1000 [==============================] - 4s 4ms/step - loss: 0.0790 - ac
curacy: 0.9210
Epoch 4/10
1000/1000 [==============================] - 4s 4ms/step - loss: 0.1000 - ac
curacy: 0.9000
Epoch 5/10
1000/1000 [==============================] - 4s 4ms/step - loss: 0.0890 - ac
curacy: 0.9110
Epoch 6/10
1000/1000 [==============================] - 4s 4ms/step - loss: 0.0710 - ac
curacy: 0.9290
Epoch 7/10
1000/1000 [==============================] - 4s 4ms/step - loss: 0.0860 - ac
curacy: 0.9140
Epoch 8/10
1000/1000 [==============================] - 4s 4ms/step - loss: 0.0730 - ac
curacy: 0.9270
Epoch 9/10
1000/1000 [==============================] - 4s 4ms/step - loss: 0.0680 - ac
curacy: 0.9320
Epoch 10/10
1000/1000 [==============================] - 4s 4ms/step - loss: 0.0770 - ac
curacy: 0.9230
```

In [93]:

```
test = model.evaluate_generator(mygen, steps=5)
print(test[1]*100)
```

```
80.0000011920929
```

4. CONCLUSION

1. I have extracted data and loaded it into a data-frame.
2. I have preprocessed the data and converted categorical values to numeric for model building and execution.
3. I have built the models and tested the accuracy for multiple activation layers, losses, metrics and optimizers.
4. I have used generators to get samples from big data and tested against the models that I built.

Lessons learnt:

1. Classification and Regression definition.
2. Perceptron and it's functioning.
3. Different optimizers, loss functions, hyper parameters and metrics.
4. How to create generators and use them (including yield statement).