Driving sales through Machine Learning

June 17, 2020

1 Driving sales through targeted advertisement

Businesses are profit driven, they aim to increase their revenue while keeping the cost low. One of the tools businesses employed to drive their revenue is marketing. However, marketing is expensive. In fact, Singapore's marketing expenditure on digital marketing alone is projected to reach US$760m in 2020 (statista, 2020).
Marketing spending does not always translate to profit, if a company wrongly designed a marketing campaign, perhaps by targeting the wrong target audience, the company might suffer loss instead. Therefore, there is a need to investigate the right customer for marketing.

In this project, we aim to increase the sucess rate for an audiobook company's marketing campaign by predicting the customers who are most likely to purchase an audiobook again.
We will be working with a dataset containing past user usage records, and we will be predicting the customer who will purchase another audiobook in 6 months time.

We are interested in maximising the potential user growth while minimising cost. Since F1-score measures the trade off between precision and recall, we will use F1-score as our business metrics. We will also set recall 3 times more important as precision to maximise the potential user growth while keeping cost at a reasonable level.
We will be solving this problem using a neural network model.

Dataset taken from: https://www.kaggle.com/faressayah/audiobook-app-data
Singapore's projected digital spending in 2020: https://www.statista.com/outlook/216/124/digital-advertising/singapore#market-revenue

2 Preprocessing

2.0.1 Inspecting data

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.utils import resample

from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report
```

```
import tensorflow as tf
import tensorflow.keras as keras
import tensorflow.keras.layers as layers

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import Recall
from tensorflow_addons.metrics import F1Score
from tensorflow.keras.callbacks import EarlyStopping
import kerastuner as kt
import keras.backend as K
```

```
Using TensorFlow backend.
```

[2]:
```
def report(true_y, predict):
    print('confusion matrix:\n')
    print(confusion_matrix(true_y, predict, labels=[0,1]))
    print('\nclassification report:\n')
    print(classification_report(true_y, predict))
```

[3]:
```
data = pd.read_csv('data/audiobook_data_2.csv',index_col=0)
```

[4]:
```
data.head()
```

[4]:
```
      Book_length(mins)_overall  Book_length(mins)_avg  Price_overall  \
994                      1620.0                   1620          19.73
1143                     2160.0                   2160           5.33
2059                     2160.0                   2160           5.33
2882                     1620.0                   1620           5.96
3342                     2160.0                   2160           5.33

      Price_avg  Review  Review10/10  Completion  Minutes_listened  \
994       19.73       1        10.00        0.99            1603.8
1143       5.33       0         8.91        0.00               0.0
2059       5.33       0         8.91        0.00               0.0
2882       5.96       0         8.91        0.42             680.4
3342       5.33       0         8.91        0.22             475.2

      Support_Request  Last_Visited_mins_Purchase_date  Target
994                 5                               92       0
1143                0                                0       0
2059                0                              388       0
2882                1                              129       0
3342                0                              361       0
```

[5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 14084 entries, 994 to 251
Data columns (total 11 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   Book_length(mins)_overall        14084 non-null  float64
 1   Book_length(mins)_avg            14084 non-null  int64  
 2   Price_overall                    14084 non-null  float64
 3   Price_avg                        14084 non-null  float64
 4   Review                           14084 non-null  int64  
 5   Review10/10                      14084 non-null  float64
 6   Completion                       14084 non-null  float64
 7   Minutes_listened                 14084 non-null  float64
 8   Support_Request                  14084 non-null  int64  
 9   Last_Visited_mins_Purchase_date  14084 non-null  int64  
 10  Target                           14084 non-null  int64  
dtypes: float64(6), int64(5)
memory usage: 1.3 MB
```

Good news for us, there is no missing data and no categorical data

[6]:
```
data.describe()
```

[6]:

	Book_length(mins)_overall	Book_length(mins)_avg	Price_overall	\
count	14084.000000	14084.000000	14084.000000	
mean	1591.281685	1678.608634	7.103791	
std	504.340663	654.838599	4.931673	
min	216.000000	216.000000	3.860000	
25%	1188.000000	1188.000000	5.330000	
50%	1620.000000	1620.000000	5.950000	
75%	2160.000000	2160.000000	8.000000	
max	2160.000000	7020.000000	130.940000	

	Price_avg	Review	Review10/10	Completion	\
count	14084.000000	14084.000000	14084.000000	14084.000000	
mean	7.543805	0.160750	8.909795	0.125659	
std	5.560129	0.367313	0.643406	0.241206	
min	3.860000	0.000000	1.000000	0.000000	
25%	5.330000	0.000000	8.910000	0.000000	
50%	6.070000	0.000000	8.910000	0.000000	
75%	8.000000	0.000000	8.910000	0.130000	
max	130.940000	1.000000	10.000000	1.000000	

	Minutes_listened	Support_Request	Last_Visited_mins_Purchase_date	\
count	14084.000000	14084.000000	14084.000000	
mean	189.888983	0.070222	61.935033	
std	371.084010	0.472157	88.207634	
min	0.000000	0.000000	0.000000	

```
25%          0.000000          0.000000                           0.000000
50%          0.000000          0.000000                          11.000000
75%        194.400000          0.000000                         105.000000
max       2160.000000         30.000000                         464.000000

             Target
count  14084.000000
mean       0.158833
std        0.365533
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        1.000000
```

However, the mean for target is 0.159. Implying that only ~16% of the customers purchased another audiobook after 6 months.
Since we have an imbalanced data, we will try to balance them later.
There is another interesting thing about Review10/10: the 25%, 50% and 75% are all 8.91.

2.0.2 Splitting data into train, validation, test datasets

[7]:
```python
X, y = data.drop('Target',axis=1), data['Target']
train_valid_x, test_x, train_valid_y, test_y =␣
↪train_test_split(X,y,random_state=0,test_size=0.1)
train_x, valid_x, train_y, valid_y =␣
↪train_test_split(train_valid_x,train_valid_y,random_state=0,test_size=0.1)
```

[8]:
```python
train_y.describe()
```

[8]:
```
count    11407.000000
mean         0.158324
std          0.365060
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          1.000000
Name: Target, dtype: float64
```

2.0.3 Solving unbalanced datasets

We'll attempt 3 different strategies here: 1. No changing of data but focus on optimising f1 score 2. Oversampling minority cases 3. Undersampling majority cases

Here's an article talking about some common techniques to handle imbalanced datasets: https://towardsdatascience.com/methods-for-dealing-with-imbalanced-data-5b761be45a18

```
[9]: NUM_MAJOR = np.sum(train_y == 0)
NUM_MINOR = np.sum(train_y == 1)
print('Majority case count: {}\nMinority case count: {}'.format(NUM_MAJOR,␣
 ↪NUM_MINOR))

major_class = pd.concat([train_x,train_y],axis=1)[train_y == 0]
minor_class = pd.concat([train_x,train_y],axis=1)[train_y == 1]
```

```
Majority case count: 9601
Minority case count: 1806
```

```
[10]: oversampling = resample(minor_class, replace=True, n_samples=NUM_MAJOR,␣
 ↪random_state=0)
undersampling = resample(major_class, replace=False, n_samples=NUM_MINOR,␣
 ↪random_state=0)
```

```
[11]: oversampled = pd.concat([major_class,oversampling],axis=0)
train_x_oversample, train_y_oversample = oversampled.drop('Target',axis=1),␣
 ↪oversampled['Target']
undersampled = pd.concat([undersampling, minor_class], axis=0)
train_x_undersample, train_y_undersample = undersampled.drop('Target',axis=1),␣
 ↪undersampled['Target']
```

```
[12]: train_x.shape, train_x_oversample.shape, train_x_undersample.shape
```

```
[12]: ((11407, 10), (19202, 10), (3612, 10))
```

Now we have matching cases where
Original training set = 11407
Oversampled = number of majority cases * 2 = 9601 * 2 = 19202
Undersampled = number of minority cases * 2 = 1806 * 2 = 3612

2.0.4 Scaling

Scaled data improves our accuracy and neural network requires the data to be scaled

```
[13]: pipe = make_pipeline(StandardScaler()).fit(train_x)
pipe_oversample = make_pipeline(StandardScaler()).fit(train_x_oversample)
pipe_undersample = make_pipeline(StandardScaler()).fit(train_x_undersample)
```

3 Neural Network

```
[14]: tf.random.set_seed(0)
```

```
[15]: #we weigh recall 3 times more important than precision
def f1_metric(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
```

```
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    recall = true_positives / (possible_positives + K.epsilon())
    f1_val = (1+9)*(precision*recall)/(9*precision+recall+K.epsilon())
    return f1_val
```

[16]:
```
class Audiobook:
    def __init__(self):
        self.model = keras.Sequential()
        self.callbacks = [EarlyStopping(monitor='f1_metric', mode='max',␣
 ↪patience=10,
                                       restore_best_weights=True, verbose=2)]

    def build(self):
        self.model.add(layers.Dense(10))
        self.model.add(layers.Dense(50, activation='relu'))
        self.model.add(layers.Dense(50, activation='relu'))
        self.model.add(layers.Dense(1, activation='sigmoid'))
        self.model.compile(optimizer=Adam(lr=0.001), loss='binary_crossentropy',
                           metrics=['accuracy',f1_metric])
        return self
```

We will run a maximum of 100 epochs for all models with a batch size 100

[17]:
```
EPOCHS = 100
BATCH_SIZE = 100
```

3.0.1 No resampling

[18]:
```
model = Audiobook().build()
model_report = model.model.fit(pipe.transform(train_x), train_y, epochs=EPOCHS,␣
 ↪batch_size=BATCH_SIZE,
         validation_data=(pipe.transform(valid_x), valid_y), verbose=0,␣
 ↪callbacks=model.callbacks)
```

```
Restoring model weights from the end of the best epoch.
Epoch 00059: early stopping
```

3.0.2 Oversampling

[19]:
```
oversample = Audiobook().build()
oversample_report = oversample.model.fit(pipe_oversample.
 ↪transform(train_x_oversample), train_y_oversample,
                                         epochs=EPOCHS, batch_size=BATCH_SIZE,
                                         validation_data=(pipe_oversample.
 ↪transform(valid_x), valid_y),
```

```
                                              verbose=0, callbacks=oversample.
↪callbacks)
```

```
Restoring model weights from the end of the best epoch.
Epoch 00020: early stopping
```

3.0.3 Undersampling

[20]:
```
undersample = Audiobook().build()
undersample_report = undersample.model.fit(pipe_undersample.
↪transform(train_x_undersample), train_y_undersample, epochs=EPOCHS,␣
↪batch_size=BATCH_SIZE,
         validation_data=(pipe_undersample.transform(valid_x), valid_y),␣
↪verbose=0, callbacks=undersample.callbacks)
```

```
Restoring model weights from the end of the best epoch.
Epoch 00035: early stopping
```

3.0.4 Confusion matrix

[21]:
```
MAX_0, MAX_1 = np.sum(valid_y == 0), np.sum(valid_y == 1)
print('Validation set:\nNon returning customers: {}\nReturning customers: {}'.
↪format(MAX_0,MAX_1))
```

```
Validation set:
Non returning customers: 1066
Returning customers: 202
```

[22]:
```
predict = np.where(model.model.predict(pipe.transform(valid_x)) > 0.5, 1, 0)
report(valid_y, predict)
```

```
confusion matrix:

[[1061    5]
 [ 108   94]]

classification report:

              precision    recall  f1-score   support

           0       0.91      1.00      0.95      1066
           1       0.95      0.47      0.62       202

    accuracy                           0.91      1268
   macro avg       0.93      0.73      0.79      1268
weighted avg       0.91      0.91      0.90      1268
```

[23]:
```python
predict_oversample = np.where(oversample.model.predict(pipe_oversample.
 ↪transform(valid_x))> 0.5, 1, 0)
report(valid_y, predict_oversample)
```

```
confusion matrix:

[[978  88]
 [ 58 144]]

classification report:

              precision    recall  f1-score   support

           0       0.94      0.92      0.93      1066
           1       0.62      0.71      0.66       202

    accuracy                           0.88      1268
   macro avg       0.78      0.82      0.80      1268
weighted avg       0.89      0.88      0.89      1268
```

[24]:
```python
predict_undersample = np.where(undersample.model.predict(pipe_undersample.
 ↪transform(valid_x))> 0.5, 1, 0)
report(valid_y, predict_undersample)
```

```
confusion matrix:

[[923 143]
 [ 36 166]]

classification report:

              precision    recall  f1-score   support

           0       0.96      0.87      0.91      1066
           1       0.54      0.82      0.65       202

    accuracy                           0.86      1268
   macro avg       0.75      0.84      0.78      1268
weighted avg       0.89      0.86      0.87      1268
```

Comparing the 3 techniques to handle imbalanced datasets, original datasets gives us the highest precision but the lowest recall. This means that all the potential customers identified by us are correct, but we miss out on a lot of potential customers.
Comparing oversampling and undersampling, both have very similiar recalls but different precision (0.62 vs 0.54). This means that oversampling is able to caputure a relatively good proportion of the actual potential customers while making smaller mistakes, and undersampling is able to

capture much higher proportion of the actual potential customers but makes a lot more mistakes. In business context, we will need to decide which is more important: more customers or more correct customers.

Since the business objective we set at first require us to balance between recall and precision, we will be using ***oversampling*** which has the highest f1-score to handle the imbalanced dataset.

3.0.5 Hyperparameter tunning

Since I'm using a Macbook without GPU, I ran the codes in Google Colab.
If you are interested in running the codes, you can uncomment the follow codes

[25]:
```
train_x, train_y = pipe_oversample.transform(train_x_oversample),␣
↪train_y_oversample
valid_x, valid_y = pipe_oversample.transform(valid_x), valid_y
test_x, test_y = pipe_oversample.transform(test_x), test_y
```

[26]:
```
EPOCHS = 100
BATCH_SIZE = 100
```

[27]:
```
def model_builder(hp):
    model = keras.Sequential()
    model.add(layers.Dense(10))

    #changing breaths
    hp_units = hp.Choice('unit', values=[10,50,100,500,1000])
    model.add(layers.Dense(hp_units, activation='relu'))
    model.add(layers.Dense(hp_units, activation='relu'))
    model.add(layers.Dense(1, activation='sigmoid'))
    #changing learning rates
    hp_lr = hp.Choice('learning rate', values=[0.01, 0.001, 0.0001])
    model.compile(optimizer=Adam(lr=hp_lr), loss='binary_crossentropy',
                           metrics=['accuracy',f1_metric])
    return model
```

[28]:
```
tuner = kt.Hyperband(model_builder, metrics=[f1_metric], objective=kt.
↪Objective("f1_metric", direction="max"),
                     max_epochs=10, seed=0, directory='report',␣
↪project_name='hyperparameter_tunning')
```

```
INFO:tensorflow:Reloading Oracle from existing project
report/hyperparameter_tunning/oracle.json
INFO:tensorflow:Reloading Tuner from report/hyperparameter_tunning/tuner0.json
```

[29]:
```
tuner.search(train_x, train_y, epochs=EPOCHS, batch_size=BATCH_SIZE,
            validation_data=(valid_x, valid_y),
             callbacks = [EarlyStopping(monitor='f1_metric', mode='max',␣
↪patience=3,
```

```
                                              restore_best_weights=True, verbose=2)],␣
↪verbose=0)
```

```
INFO:tensorflow:Oracle triggered exit
```

3.0.6 The best model

[30]:
```
tuner.results_summary(1)
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

[31]:
```python
best_model = tuner.get_best_models(num_models=1)[0]
```

[32]:
```python
best_model_report = best_model.fit(train_x, train_y,
                             epochs=EPOCHS, batch_size=BATCH_SIZE,
                             validation_data=(valid_x, valid_y),
                             verbose=0,␣
 ↪callbacks=[EarlyStopping(monitor='f1_metric', mode='max', patience=3,
                                                     restore_best_weights=True,␣
 ↪verbose=2)])
predict = best_model.predict(valid_x)
predict = np.where(predict> 0.5, 1, 0)
report(valid_y, predict)
```

```
Restoring model weights from the end of the best epoch.
Epoch 00007: early stopping
confusion matrix:

[[964 102]
 [ 51 151]]

classification report:

              precision    recall  f1-score   support

           0       0.95      0.90      0.93      1066
           1       0.60      0.75      0.66       202

    accuracy                           0.88      1268
   macro avg       0.77      0.83      0.80      1268
weighted avg       0.89      0.88      0.88      1268
```

Our final model is captures more potential customers than the pre-tunned oversampling (recall: 0.75 vs 0.71) at a cost of lowered accuracy (precision: 0.60 vs 0.62). Compared to the undersampling data, our final model is more accurate at capturing potential customers (precision: 0.60 vs 0.54) but still worse at capturing all the potential customers (recall: 0.75 vs 0.82)

3.0.7 Testing model

[33]:
```python
test_predict = best_model.predict(test_x)
test_predict = np.where(test_predict> 0.5, 1, 0)
report(test_y, test_predict)
```

```
confusion matrix:
```

```
[[1063  117]
 [  71  158]]

classification report:

              precision    recall  f1-score   support

           0       0.94      0.90      0.92      1180
           1       0.57      0.69      0.63       229

    accuracy                           0.87      1409
   macro avg       0.76      0.80      0.77      1409
weighted avg       0.88      0.87      0.87      1409
```

The test dataset contains data points our model has never seen before. Therefore, test dataset shows the expected performance when we deploy our model in the real world.
As seen from the classficaition report, our model is able to capture ~70% of all potential customers while being ~60% correct in average.

4 Business impact

Let us assume our marketing spending is 3 dollars/customer and each returned customer is able to generate 10 dollars in revenue. And we assume we target a total of 100 customers.

Before deploying our model
We targeted 100 customers randomly with 16% (assumed population from dataset) of the customers are potential returning customers.
Total marketing spending: 3x100=300 dollars
Total revenue: 10x16=160 dollars
Net profit: (140) dollars

After deploying our model
We targeted 100 customers with 57% (model precision) certain that these customers are returning customers.
Total marketing spending: 3x100=300 dollars
Total revenue: 10x0.57x100=570 dollars
Net profit: 270 dollars

As seen from the above simplified example, deploying machine learning algorithm indeed helps us to turn a lossing strategy into a profiting strategy!