- **Binary classification example: Imdb reviews**

IMDB dataset

Description

IMDB dataset having 50K movie reviews for natural language processing or Text analytics.
This is a dataset for binary sentiment classification containing substantially more data than previous benchmark datasets. We provide a set of 25,000 highly polar movie reviews for training and 25,000 for testing. So, predict the number of positive and negative reviews using either classification or deep learning algorithms.
For more dataset information, please go through the following link,
http://ai.stanford.edu/~amaas/data/sentiment/

Comments used for dataset

```
In [28]: from keras.datasets import imdb

In [29]: (train_data, train_labels), (test_data, test_labels)=imdb.load_data(num_words=10000)

In [30]: len(train_data)
Out[30]: 25000

In [31]: len(test_data)
Out[31]: 25000

In [32]: print(train_data[0])
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100,
43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172,
4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469,
4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2,
5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38,
619, 5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16, 82,
2, 8, 4, 107, 117, 5952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317,
46, 7, 4, 2, 1029, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18,
4, 226, 22, 21, 134, 476, 26, 480, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226,
65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 19, 178, 32]

In [33]: print(train_labels[0])
1

In [34]: print(train_labels[100])
0
```

Representation of comments by word indexes

https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

Comments

stunning film to watch. Mr. Mattei offers ...	
Probably my all-time favorite movie, a story of selflessness, sacrifice and dedication to a noble ca...	positive
I sure would like to see a resurrection of a up dated Seahunt series with the tech they have today i...	positive
This show was an amazing, fresh & innovative idea in the 70's when it first aired. The first 7 or 8 ...	negative
Encouraged by the positive comments about this film on here I was looking forward to watching this f...	negative
If you like original gut wrenching laughter you will like this movie. If you are young or old then y...	positive
Phil the Alien is one of those quirky films where the humour is based around the oddness of everythi...	negative
I saw this movie when I was about 12	negative

IMDB dataset

imdb function in keras allows you to reconstruct comments from work indices.

```
In [120]: word_index = imdb.get_word_index()

In [121]: word_index = {k:(v+3) for k,v in word_index.items()}

In [122]: reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])

In [123]: str=[reverse_word_index.get(i, '?') for i in train_data[0]]

In [124]: print(str)
['?', 'this', 'film', 'was', 'just', 'brilliant', 'casting', 'location', 'scenery',
'story', 'direction', "everyone's", 'really', 'suited', 'the', 'part', 'they', 'played',
'and', 'you', 'could', 'just', 'imagine', 'being', 'there', 'robert', '?', 'is', 'an',
'amazing', 'actor', 'and', 'now', 'the', 'same', 'being', 'director', '?', 'father',
'came', 'from', 'the', 'same', 'scottish', 'island', 'as', 'myself', 'so', 'i', 'loved',
'the', 'fact', 'there', 'was', 'a', 'real', 'connection', 'with', 'this', 'film', 'the',
'witty', 'remarks', 'throughout', 'the', 'film', 'were', 'great', 'it', 'was', 'just',
'brilliant', 'so', 'much', 'that', 'i', 'bought', 'the', 'film', 'as', 'soon', 'as', 'it',
'was', 'released', 'for', '?', 'and', 'would', 'recommend', 'it', 'to', 'everyone', 'to',
'watch', 'and', 'the', 'fly', 'fishing', 'was', 'amazing', 'really', 'cried', 'at', 'the',
'end', 'it', 'was', 'so', 'sad', 'and', 'you', 'know', 'what', 'they', 'say', 'if', 'you',
'cry', 'at', 'a', 'film', 'it', 'must', 'have', 'been', 'good', 'and', 'this',
'definitely', 'was', 'also', '?', 'to', 'the', 'two', 'little', "boy's", 'that', 'played',
'the', '?', 'of', 'norman', 'and', 'paul', 'they', 'were', 'just', 'brilliant', 'children',
'are', 'often', 'left', 'out', 'of', 'the', '?', 'list', 'i', 'think', 'because', 'the',
'stars', 'that', 'play', 'them', 'all', 'grown', 'up', 'are', 'such', 'a', 'big',
'profile', 'for', 'the', 'whole', 'film', 'but', 'these', 'children', 'are', 'amazing',
'and', 'should', 'be', 'praised', 'for', 'what', 'they', 'have', 'done', "don't", 'you',
'think', 'the', 'whole', 'story', 'was', 'so', 'lovely', 'because', 'it', 'was', 'true',
'and', 'was', "someone's", 'life', 'after', 'all', 'that', 'was', 'shared', 'with', 'us',
'all']
```

Preparetion of dataset

```python
def vectorize_sequences(sequences, dimension=10000):
    # Sıfırlardan oluşan, (len(sequences), dimension
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results

(train_data, train_labels),(test_data, test_labels)\
=imdb.load_data(num_words=10000)

# Eğitim ve test verilerini vektöre dönüştür
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)

# Etiketleri vektöre dönüştür
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

The words used in the comment are set to one.

Use first 10000 words that are frequently used in comments

Vectorize sequences

```
In [214]: train_data[0]
Out[214]:
[1,
 14,
 22,
 16,
 43,
 530,
 973,
 1622,
 1385,
 65,
 458,
 4468,
 66,
 3941,
 4,
 173,
 36,
 256,
 5,
 25,
 100,
 43,
 838,
 112,
 50,
 670,
 2,
 9,
 35,
```

```
In [262]: y=np.sort(train_data[0]).reshape(len(train_data[0]),1)

In [263]: y
Out[263]:
array([[   1],
       [   2],
       [   2],
       [   2],
       [   2],
       [   2],
       [   2],
       [   4],
       [   4],
       [   4],
       [   4],
       [   4],
       [   4],
       [   4],
       [   4],
       [   4],
       [   4],
       [   4],
       [   4],
       [   4],
       [   4],
       [   4],
       [   5],
       [   5],
       [   5],
       [   5],
       [   5],
       [   5],
       [   5],
       [   5],
       [   5],
       [   6],
       [   6],
       [   6],
       [   7],
       [   7],
       [   8],
       [   8],
       [   8],
```

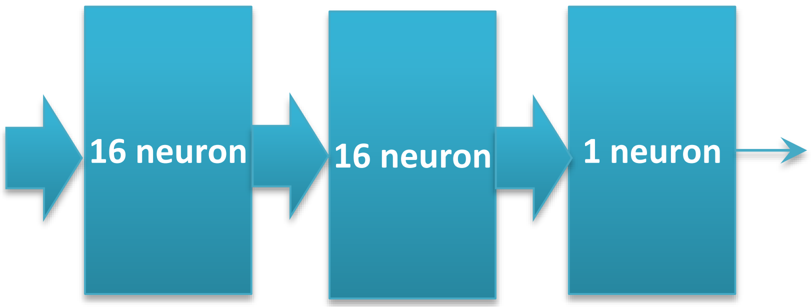
```
In [269]: y=x_train[0].reshape(len(x_train[0]),1)

In [270]: y[0:50]
Out[270]:
array([[0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
```

Each comment is represented with an array of 10000 elements. Only the words used in the comment are set to 1. Others are left zero.

Model architecture

Comment



Decision

0: negative

1: positive

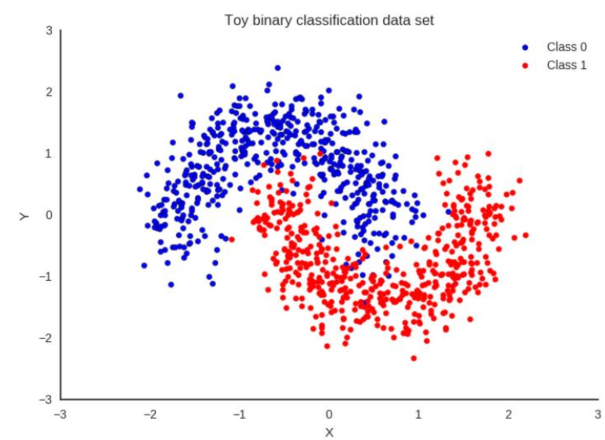
```
model=models.Sequential()

model.add(layers.Dense(16,
                       activation='relu',
                       input_shape=(10000,)))

model.add(layers.Dense(16,
                       activation='relu'))

model.add(layers.Dense(1,
                       activation='sigmoid'))
```

Binary classifier



https://www.kdnuggets.com/2017/04/must-know-evaluate-binary-classifier.html

Compile and fit with validation

```
# Optimisation parameters ---------------------------
model.compile(optimizer=optimizers.RMSprop(lr=0.01),
              loss=losses.binary_crossentropy,
              metrics=["acc"])
```

```
# Training ------------------------------
x_val = x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
partial_y_train = y_train[10000:]

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=10,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

Use some of the training data for validation

Training

- Validation loss and accuracy is computed at the end of the each step.
- Therefore we can monitor how the training is going for the data that is not used in training.
- Validation datasets can be used for regularization by early stopping
- To read more: https://en.wikipedia.org/wiki/Training,_validation,_and_test_sets

```
Epoch 1/10
30/30 [==============================] - 2s 48ms/step - loss: 0.6636 - acc: 0.6946 - val_loss: 0.3622 - val_acc: 0.8423
Epoch 2/10
30/30 [==============================] - 1s 29ms/step - loss: 0.2854 - acc: 0.8814 - val_loss: 0.3876 - val_acc: 0.8470
Epoch 3/10
30/30 [==============================] - 1s 28ms/step - loss: 0.2063 - acc: 0.9126 - val_loss: 0.3051 - val_acc: 0.8888
Epoch 4/10
30/30 [==============================] - 1s 28ms/step - loss: 0.1156 - acc: 0.9496 - val_loss: 0.4029 - val_acc: 0.8259
Epoch 5/10
30/30 [==============================] - 1s 28ms/step - loss: 0.0966 - acc: 0.9579 - val_loss: 0.4501 - val_acc: 0.8784
Epoch 6/10
30/30 [==============================] - 1s 28ms/step - loss: 0.1101 - acc: 0.9622 - val_loss: 0.5293 - val_acc: 0.8813
Epoch 7/10
30/30 [==============================] - 1s 28ms/step - loss: 0.0702 - acc: 0.9804 - val_loss: 0.5110 - val_acc: 0.8826
Epoch 8/10
30/30 [==============================] - 1s 28ms/step - loss: 0.0162 - acc: 0.9947 - val_loss: 1.2341 - val_acc: 0.7981
Epoch 9/10
30/30 [==============================] - 1s 28ms/step - loss: 0.1379 - acc: 0.9590 - val_loss: 0.7095 - val_acc: 0.8797
Epoch 10/10
30/30 [==============================] - 1s 28ms/step - loss: 0.0072 - acc: 0.9982 - val_loss: 0.9645 - val_acc: 0.8717
```

Plotting the accurcy and loss graphs

```python
# Plot accuracy and loss graphs-----------------------------------
history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']

epochs = range(1,
               len(loss_values ) + 1)

plt.figure(1)
plt.plot(epochs, loss_values, 'ro', label='Training loss')
plt.plot(epochs, val_loss_values, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')

plt.legend()
plt.show()

plt.figure(2)
acc = history.history['acc']
val_acc = history.history['val_acc']

plt.plot(epochs, acc, 'ro', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('acc')
plt.legend()

plt.show()
```