Importing the libraries needed

In [1]:

```
import numpy as np
import pandas as pd
import time


import matplotlib.pyplot as plt
import seaborn as sns

import re
import string

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

import gensim
from gensim.models import KeyedVectors

from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

import tensorflow as tf
from keras.models import Sequential
from tensorflow.keras.layers import SpatialDropout1D, Conv1D, Bidirectional, LSTM, Dense, Input, Dropout, GlobalMaxPooling1D
from keras.layers.embeddings import Embedding
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStopping
from tensorflow.keras.optimizers import Adam

import itertools
from numpy import loadtxt
from keras.models import load_model

import warnings
warnings.filterwarnings("ignore")
```

Connecting to google drive

In [2]:

```
from google.colab import drive
drive.mount("/content/gdrive")
```

```
Mounted at /content/gdrive
```

Uploading the dataset

In [3]:

```
path_data = "/content/gdrive/MyDrive/thesis/HARD.xlsx"

HARD = pd.read_excel(path_data)
```

In [4]:

```
data = HARD
```

printing the first 3 rows of the data

In [5]:

```
data.head(3)
```

Out[5]:

	no	Hotel name	rating	user type	room type	nights	review
0	2	فندق 72	2	مسافر منفرد	غرفة ديلوكس مزدوجة أو توأم	أقمت ليلة واحدة	ممتاز”. النظافة والطاقم متعاون“.
1	3	فندق 72	5	زوج	غرفة ديلوكس مزدوجة أو توأم	أقمت ليلة واحدة	استثنائي. سهولة إنهاء المعاملة في الاستقبال. ل...
2	16	فندق 72	5	زوج	-	أقمت ليلتين	استثنائي. انصح بأختيار الاسويت و بالاخص غرفه ر...

printing the shape of the dataset nbr of row and columns

In [6]:

```
print("Data contient {} lignes et {} colonnes.".format(data.shape[0], data.shape
[1]))
```

```
Data contient 105698 lignes et 7 colonnes.
```

printing the fiels with missed values

In [7]:

```
data.isnull().sum()
```

Out[7]:

```
no            0
Hotel name    0
rating        0
user type     0
room type     0
nights        0
review        0
dtype: int64
```

printing the number of the duplicated rows

In [8]:

```
print("On a  {} doublons dans Data.".format(data.duplicated().sum()))
```

```
On a  0 doublons dans Data.
```

checking the types of the fiels in the data

In [9]:

```
data.dtypes
```

Out[9]:

```
no               int64
Hotel name      object
rating           int64
user type       object
room type       object
nights          object
review          object
dtype: object
```

function for printing the pie

In [10]:

```
def pie(data,col):
    labels = data[col].value_counts().keys().tolist()
    n = len(labels)
    if n==2:
        colors = ['#66b3ff', '#fb3999']
    elif n==3:
        colors = ['#66b3ff', '#fb3999', '#ffcc99']
    elif n==4:
        colors = ['#66b3ff', '#fb3999', '#ffcc99',"#66f3ff"]
    elif n==5:
        colors = ['#66b3ff', '#fb3999', '#ffcc99',"#66f3ff",'#adcc99']
    elif n==6:
        colors = ['#66b3ff', '#fb3999', '#ffcc99',"#66f3ff",'#adcc99',"#db7f23"]

    fig1, f1 = plt.subplots()
    f1.pie(data[col].value_counts(), labels=labels, colors = colors, autopct='%1.1f%%',shadow=False, startangle=60)
    f1.axis('equal')
    plt.tight_layout()
    plt.show()

def histo(data,col):
    plt.figure(figsize = (10, 8))
    sns.histplot(data=data, x=col, hue = data[col], fill=True)
```

Counting the % of each classe

In [11]:

```
data.rating.value_counts(normalize = True)
```

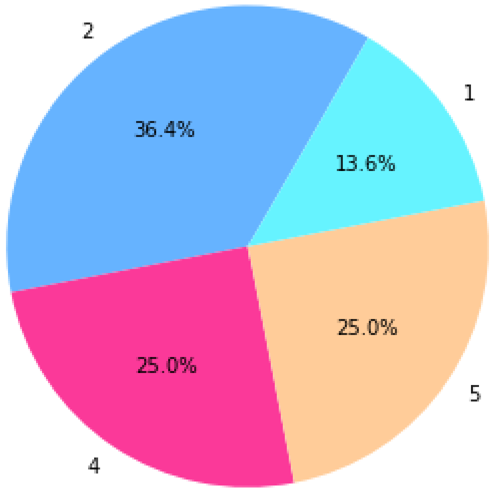
Out[11]:

```
2    0.363933
4    0.250241
5    0.249759
1    0.136067
Name: rating, dtype: float64
```

Printing the distribution of the classes

In [12]:

```
pie(data, "rating")
```



Repartitionning the data to 2 classes

In [13]:

```
positive_reviews = data[data["rating"] > 3]
positive_reviews["sentiment"] = 1

negative_reviews = data[data["rating"] < 3]
negative_reviews["sentiment"] = 0

data = pd.concat([positive_reviews, negative_reviews], ignore_index = True)
```

printing the number of rows in both classes

In [14]:

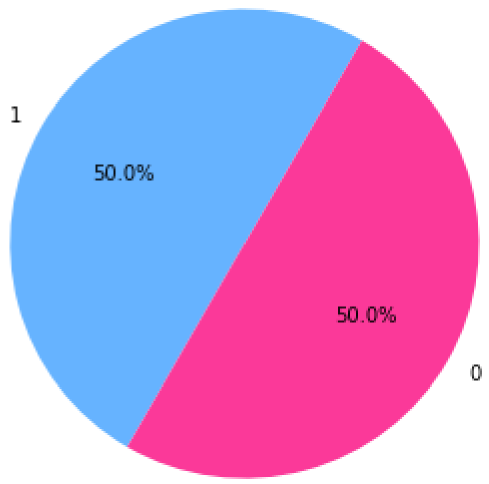
```
print("data contient {} lignes.".format(data.shape[0]))

print("Positive_reviews contient {} lignes.".format(positive_reviews.shape[0]))

print("Negative_reviews contient {} lignes.".format(negative_reviews.shape[0]))
```

```
data contient 105698 lignes.
Positive_reviews contient 52849 lignes.
Negative_reviews contient 52849 lignes.
```

printing the new distribution of the data

In [15]:

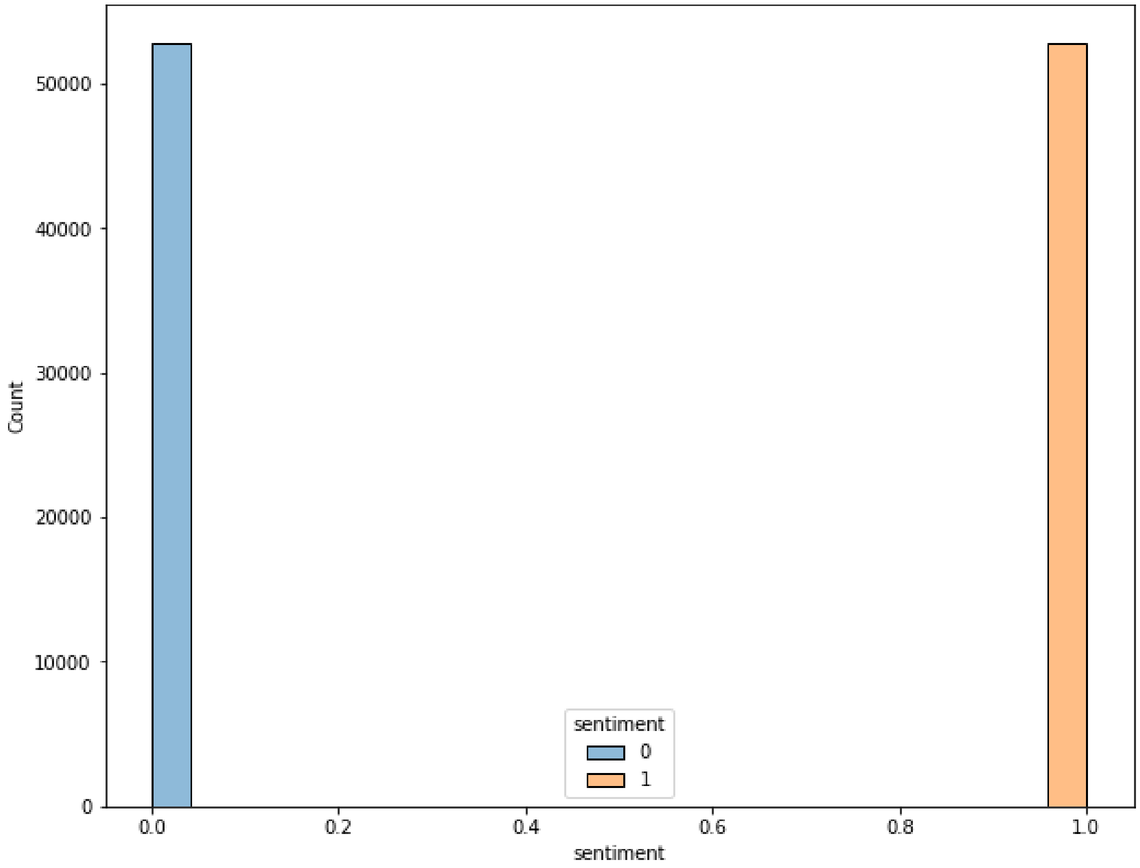
```
pie(data,"sentiment")
```



printing the new distribution in histogramme

In [16]:

```
histo(data,"sentiment")
```



function to count the length of reviews

In [17]:

```
def compte_mots(phrase):
    return len(phrase.split())

data["len_review"] = data["review"].apply(compte_mots)
positive_reviews['len_review'] = positive_reviews["review"].apply(compte_mots)
negative_reviews['len_review'] = negative_reviews["review"].apply(compte_mots)
```

printing the max length of the positive and negative reviews

In [18]:

```
print("Le maximum de mots utilisé dans les reviews positives est :", max(positiv
e_reviews.len_review))
print("Le moyen de mots utilisé dans les reviews positives est :", np.mean(posit
ive_reviews.len_review))
print("-------------------------------------------------------------------------
------------------------------")
print("Le maximum de mots utilisé dans les reviews négatives est :", max(negativ
e_reviews.len_review))
print("Le moyen de mots utilisé dans les reviews négatives est :", np.mean(negat
ive_reviews.len_review))
```

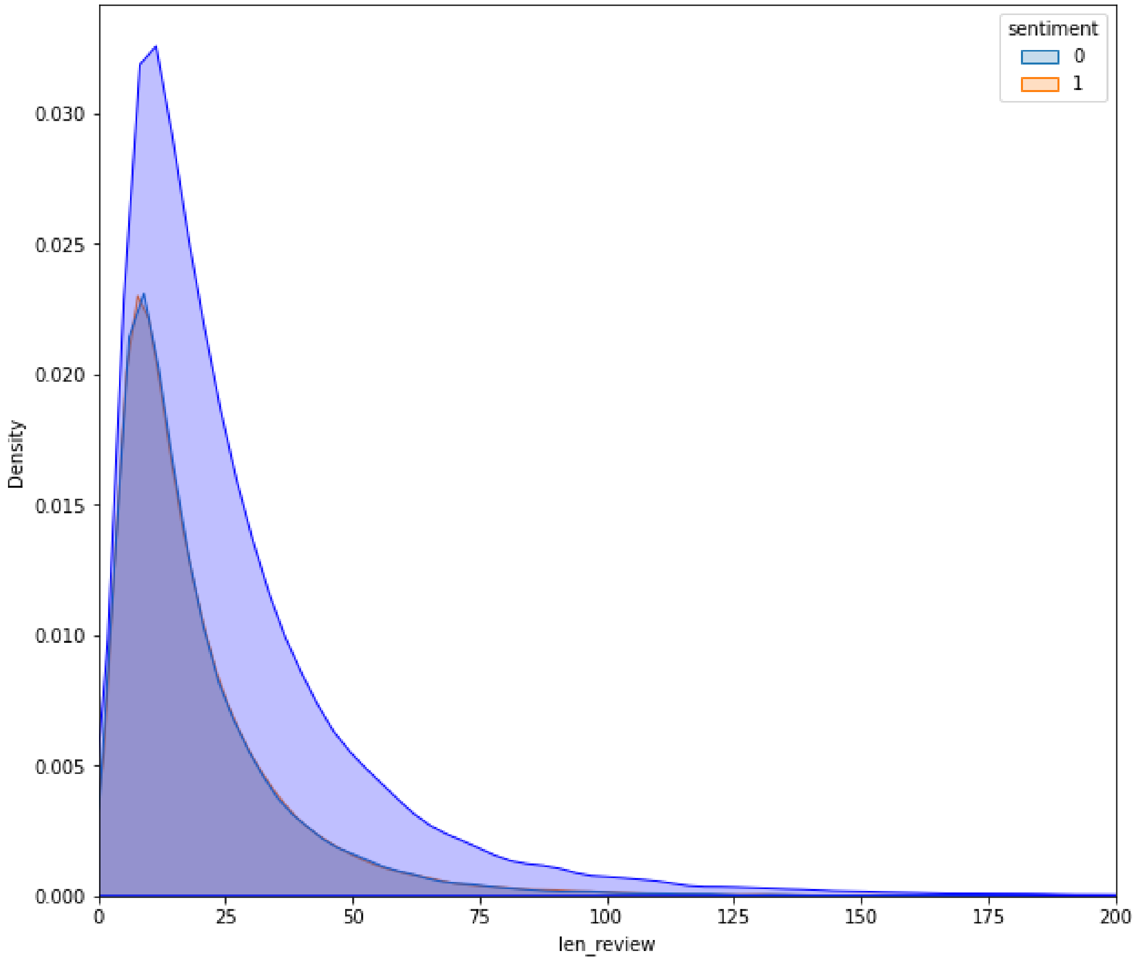
```
Le maximum de mots utilisé dans les reviews positives est : 570
Le moyen de mots utilisé dans les reviews positives est : 19.8297034
94862724
--------------------------------------------------------------------
-----------------------------------
Le maximum de mots utilisé dans les reviews négatives est : 614
Le moyen de mots utilisé dans les reviews négatives est : 28.0947227
00524134
```

In [19]:

```
plt.figure(figsize=(10,9))

p1=sns.kdeplot(positive_reviews['len_review'], hue = data['sentiment'],  shade=True, color="r")
p1=sns.kdeplot(negative_reviews['len_review'], shade=True, color="b")

plt.xlim(0, 200)
```

Out[19]:

```
(0.0, 200.0)
```



Deleting unused fields

In [20]:

```
data.drop(['no','Hotel name','rating','user type','room type','nights'], axis =
1, inplace = True)
data.head(3)
```

Out[20]:

	review	sentiment	len_review
0	استثنائي. سهولة إنهاء المعاملة في الاستقبال. ل...	1	7
1	استثنائي. انصح بأختيار الاسويت و بالاخص غرفه ر...	1	11
2	جيد. المكان جميل وهاديء. كل شي جيد ونظيف بس كا...	1	23

In [21]:

```
df = data
```

the function of the preprocessing

In [22]:

```
def preprocessing(x):
    x = re.sub('@[^\s]+', ' ', x)
    x = re.sub('((www\.[^\s]+)|(https?://[^\s]+))',' ',x)

    emoji_pattern = re.compile("["
                           u"\U0001F600-\U0001F64F"  # emoticons
                           u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                           u"\U0001F680-\U0001F6FF"  # transport & map symbo
ls
                           u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           u"\U00002500-\U00002BEF"  # chinese char
                           u"\U00002702-\U000027B0"
                           u"\U00002702-\U000027B0"
                           u"\U000024C2-\U0001F251"
                           u"\U0001f926-\U0001f937"
                           u"\U00010000-\U0010ffff"
                           u"\u2640-\u2642"
                           u"\u2600-\u2B55"
                           u"\u200d"
                           u"\u23cf"
                           u"\u23e9"
                           u"\u231a"
                           u"\ufe0f"  # dingbats
                           u"\u3030""]+", flags=re.UNICODE)
    emoji_pattern.sub(r'', x)

    ar_punctuations = '''`÷×؛<>_()*&^%][ـ،/:"؟.,'{}~¦+|!”…“–ـ'''
    en_punctuations = string.punctuation
    punctuations = ar_punctuations + en_punctuations
    x = x.translate(str.maketrans('', '', punctuations))

    arabic_diacritics = re.compile(""" ّ    | # Tashdid
                             َ    | # Fatha
                             ً    | # Tanwin Fath
                             ُ    | # Damma
                             ٌ    | # Tanwin Damm
                             ِ    | # Kasra
                             ٍ    | # Tanwin Kasr
                             ْ    | # Sukun
                             ـ     # Tatwil/Kashida
                         """, re.VERBOSE)
    x = re.sub(arabic_diacritics, '', str(x))

#     x = re.sub("[إأآا]", "ا", x)
#     x = re.sub("ى", "ي", x)
#     x = re.sub("ة", "ه", x)
#     x = re.sub("گ", "ك", x)
#     x = re.sub(r'(.)\1+', r'\1', x)

    return x
```

preprocessing the reviews and printing the time spent

In [23]:

```
%%time
data["Clean_reviews"] = data.review.apply(lambda x: preprocessing(x))
```

```
CPU times: user 3.15 s, sys: 16.3 ms, total: 3.17 s
Wall time: 3.17 s
```

printing a review before and after preprocessing

In [24]:

```
print('- Avant le prétraitement \n\n',data["review"][4])
print("\n-----------------------------------------------\n")
print('- Après le prétraitement \n\n',data["Clean_reviews"][4])
```

```
- Avant le prétraitement

 جيدجداً”. الافطار جيد والسرير ممتاز ومريح واطلالة الغرفة رائعه. فرش ا“
رضية الغرفه

-----------------------------------------------

- Après le prétraitement

 جيدجدا الافطار جيد والسرير ممتاز ومريح واطلالة الغرفة رائعه فرش ارضية
الغرفه
```

Saving the cleaned data in a csv file

In [25]:

```
data.to_csv("cleaned_hard.csv")
```

asigning the reviews and classes to a new variables

In [26]:

```
X = data.Clean_reviews
y = data.sentiment
```

spliting the data to train and test set

In [27]:

```
X_train, X_test, y_train, y_test = train_test_split(X,
                                                     y,
                                                     test_size = 0.20,
                                                     random_state = 42)
```

printing the number of the train set and the test set

In [28]:

```
print('Train set', X_train.shape)
print('Test set', X_test.shape)
```

```
Train set (84558,)
Test set (21140,)
```

In [29]:

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly rem
ount, call drive.mount("/content/gdrive", force_remount=True).
```

Uploading the fsttext pretrained word embedding with 150 dimension

In [30]:

```
%%time
target_word_vec = KeyedVectors.load_word2vec_format("/content/gdrive/MyDrive/the
sis/cc.ar.150.vec", binary = False)
```

```
CPU times: user 2min 29s, sys: 4.31 s, total: 2min 34s
Wall time: 2min 41s
```

tokenization of the reviews

In [31]:

```
%%time
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train)
```

```
CPU times: user 3.36 s, sys: 41.2 ms, total: 3.4 s
Wall time: 3.4 s
```

In [32]:

```
word_index = tokenizer.word_index
vocab_size = len(tokenizer.word_index) + 1
```

making all reviews of the same length 615

In [33]:

```
%%time
MAX_SEQUENCE_LENGTH = 615

X_train = pad_sequences(tokenizer.texts_to_sequences(X_train),
                        maxlen = MAX_SEQUENCE_LENGTH)
X_test = pad_sequences(tokenizer.texts_to_sequences(X_test),
                       maxlen = MAX_SEQUENCE_LENGTH)

print("Training X Shape:", X_train.shape)
print("Testing X Shape:", X_test.shape)
```

```
Training X Shape: (84558, 615)
Testing X Shape: (21140, 615)
CPU times: user 3.24 s, sys: 128 ms, total: 3.37 s
Wall time: 3.37 s
```

Construction of the embedding matrix

In [34]:

```
%%time
embedding_matrix = np.zeros((vocab_size, 150))

for word, i in word_index.items():
    if word in target_word_vec :
        embedding_vector = target_word_vec[word]
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
```

```
CPU times: user 331 ms, sys: 72.1 ms, total: 403 ms
Wall time: 401 ms
```

In [35]:

```
embedding_matrix.shape[0] == vocab_size
```

Out[35]:

```
True
```

Creating the model

In [36]:

```
model = Sequential()
embedding_layer = Embedding(vocab_size,
                            150,
                            weights = [embedding_matrix],
                            input_length = MAX_SEQUENCE_LENGTH,
                            trainable=False)
model.add(embedding_layer)
model.add(LSTM(100))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer = Adam(learning_rate=0.001),
              loss = 'binary_crossentropy',
              metrics = ['accuracy'])

# es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding (Embedding)       (None, 615, 150)          19810200  
                                                                 
 lstm (LSTM)                 (None, 100)               100400    
                                                                 
 dropout (Dropout)           (None, 100)               0         
                                                                 
 dense (Dense)               (None, 1)                 101       
                                                                 
=================================================================
Total params: 19,910,701
Trainable params: 100,501
Non-trainable params: 19,810,200
_________________________________________________________________
None
```

fitting the model to the dataset

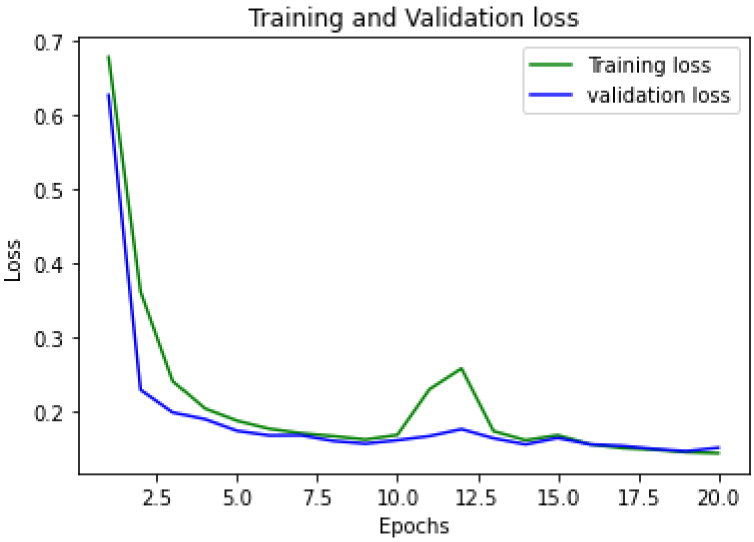
In [37]:

```
# history = model.fit(X_train, y_train, validation_split=0.15, batch_size = 128, epochs=20, verbose=1, callbacks=[es])

history = model.fit(X_train, y_train, validation_split=0.15, batch_size = 128, epochs=20, verbose=1)
```

```
Epoch 1/20
562/562 [==============================] - 32s 44ms/step - loss: 0.6
786 - accuracy: 0.6287 - val_loss: 0.6272 - val_accuracy: 0.7138
Epoch 2/20
562/562 [==============================] - 24s 42ms/step - loss: 0.3
618 - accuracy: 0.8645 - val_loss: 0.2293 - val_accuracy: 0.9237
Epoch 3/20
562/562 [==============================] - 24s 43ms/step - loss: 0.2
409 - accuracy: 0.9154 - val_loss: 0.1987 - val_accuracy: 0.9306
Epoch 4/20
562/562 [==============================] - 25s 44ms/step - loss: 0.2
040 - accuracy: 0.9282 - val_loss: 0.1896 - val_accuracy: 0.9339
Epoch 5/20
562/562 [==============================] - 25s 45ms/step - loss: 0.1
875 - accuracy: 0.9333 - val_loss: 0.1738 - val_accuracy: 0.9386
Epoch 6/20
562/562 [==============================] - 24s 43ms/step - loss: 0.1
767 - accuracy: 0.9372 - val_loss: 0.1677 - val_accuracy: 0.9405
Epoch 7/20
562/562 [==============================] - 24s 43ms/step - loss: 0.1
706 - accuracy: 0.9392 - val_loss: 0.1678 - val_accuracy: 0.9394
Epoch 8/20
562/562 [==============================] - 24s 43ms/step - loss: 0.1
667 - accuracy: 0.9412 - val_loss: 0.1602 - val_accuracy: 0.9422
Epoch 9/20
562/562 [==============================] - 25s 45ms/step - loss: 0.1
621 - accuracy: 0.9430 - val_loss: 0.1566 - val_accuracy: 0.9436
Epoch 10/20
562/562 [==============================] - 24s 43ms/step - loss: 0.1
684 - accuracy: 0.9423 - val_loss: 0.1612 - val_accuracy: 0.9427
Epoch 11/20
562/562 [==============================] - 24s 43ms/step - loss: 0.2
300 - accuracy: 0.9049 - val_loss: 0.1668 - val_accuracy: 0.9403
Epoch 12/20
562/562 [==============================] - 24s 43ms/step - loss: 0.2
582 - accuracy: 0.9005 - val_loss: 0.1762 - val_accuracy: 0.9395
Epoch 13/20
562/562 [==============================] - 24s 43ms/step - loss: 0.1
735 - accuracy: 0.9392 - val_loss: 0.1639 - val_accuracy: 0.9424
Epoch 14/20
562/562 [==============================] - 24s 43ms/step - loss: 0.1
612 - accuracy: 0.9438 - val_loss: 0.1556 - val_accuracy: 0.9439
Epoch 15/20
562/562 [==============================] - 24s 43ms/step - loss: 0.1
680 - accuracy: 0.9410 - val_loss: 0.1643 - val_accuracy: 0.9419
Epoch 16/20
562/562 [==============================] - 24s 43ms/step - loss: 0.1
557 - accuracy: 0.9458 - val_loss: 0.1556 - val_accuracy: 0.9437
Epoch 17/20
562/562 [==============================] - 24s 43ms/step - loss: 0.1
506 - accuracy: 0.9473 - val_loss: 0.1536 - val_accuracy: 0.9422
Epoch 18/20
562/562 [==============================] - 24s 43ms/step - loss: 0.1
485 - accuracy: 0.9483 - val_loss: 0.1494 - val_accuracy: 0.9476
Epoch 19/20
562/562 [==============================] - 24s 43ms/step - loss: 0.1
450 - accuracy: 0.9495 - val_loss: 0.1464 - val_accuracy: 0.9477
Epoch 20/20
562/562 [==============================] - 25s 45ms/step - loss: 0.1
437 - accuracy: 0.9501 - val_loss: 0.1511 - val_accuracy: 0.9469
```

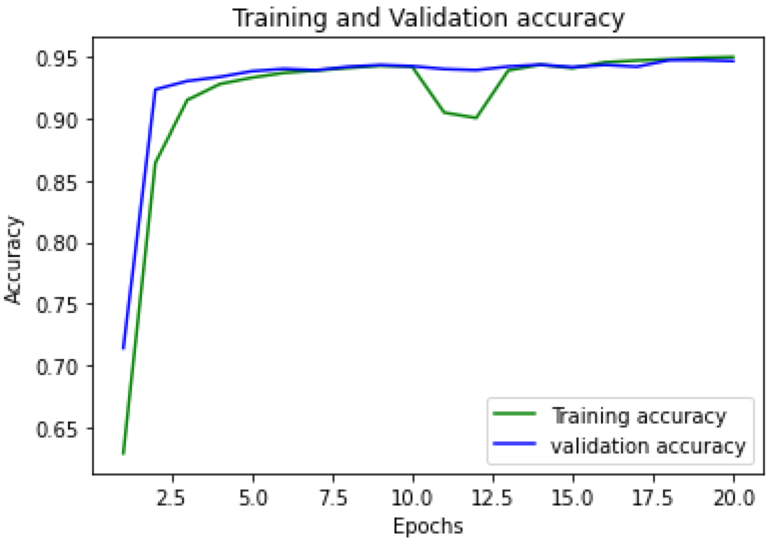
In [38]:

```
loss_train = history.history['loss']
loss_val = history.history['val_loss']
epochs = range(1,21)
plt.plot(epochs, loss_train, 'g', label='Training loss')
plt.plot(epochs, loss_val, 'b', label='validation loss')
plt.title('Training and Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [39]:

```
loss_train = history.history['accuracy']
loss_val = history.history['val_accuracy']
epochs = range(1,21)
plt.plot(epochs, loss_train, 'g', label='Training accuracy')
plt.plot(epochs, loss_val, 'b', label='validation accuracy')
plt.title('Training and Validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



In [40]:

```
score = model.evaluate(X_test, y_test, verbose=1)
print("%s: %.2f%%" % (model.metrics_names[1], score[1]*100))
```

```
661/661 [==============================] - 8s 12ms/step - loss: 0.15
46 - accuracy: 0.9445
accuracy: 94.45%
```

In [41]:

```
def decode_sentiment(score):
    return 1 if score>0.5 else 0
```

In [42]:

```
scores = model.predict(X_test, verbose=1)

y_pred = [decode_sentiment(x) for x in scores]
```

```
661/661 [==============================] - 7s 10ms/step
```

In [43]:

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.94      0.94     10600
           1       0.94      0.95      0.94     10540

    accuracy                           0.94     21140
   macro avg       0.94      0.94      0.94     21140
weighted avg       0.94      0.94      0.94     21140
```

function for creating confusion matrix

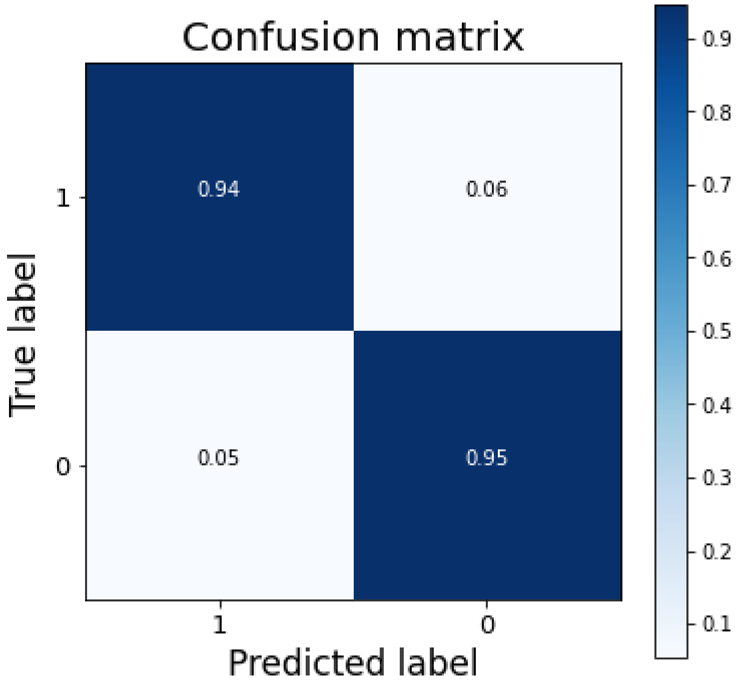
In [44]:

```
def plot_confusion_matrix(cm, classes,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """

    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title, fontsize=20)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, fontsize=13)
    plt.yticks(tick_marks, classes, fontsize=13)

    fmt = '.2f'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label', fontsize=17)
    plt.xlabel('Predicted label', fontsize=17)
```

printing the confusion matrix

In [45]:

```
cnf_matrix = confusion_matrix(y_test.to_list(), y_pred)
plt.figure(figsize=(6,6))
plot_confusion_matrix(cnf_matrix, classes=y_test.unique(), title="Confusion matrix")
plt.show()
```


Confusion matrix
True label
Predicted label