Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE".

Also, please write how much time it took you to finish the homework. This will not affect your grade in any way and is used for statistical purposes.

In [1]:
```
TIME_SPENT = "00h00m"
```

Homework 2

Text Representations & Classification

Welcome to Homework 2!

The homework contains several tasks. You can find the amount of points that you get for the correct solution in the task header. Maximum amount of points for each homework is *four*.

The **grading** for each task is the following:

- correct answer - **full points**
- insufficient solution or solution resulting in the incorrect output - **half points**
- no answer or completely wrong solution - **no points**

Even if you don't know how to solve the task, we encourage you to write down your thoughts and progress and try to address the issues that stop you from completing the task.

When working on the written tasks, try to make your answers short and accurate. Most of the times, it is possible to answer the question in 1-3 sentences.

When writing code, make it readable. Choose appropriate names for your variables (`a = 'cat'` - not good, `word = 'cat'` - good). Avoid constructing lines of code longer than 100 characters (79 characters is ideal). If needed, provide the commentaries for your code, however, a good code should be easily readable without them :)

Finally, all your answers should be written only by yourself. If you copy them from other sources it will be considered as an academic fraud. You can discuss the tasks with your classmates but each solution must be individual.

Important: **before sending your solution, do the `Kernel -> Restart & Run All` to ensure that all your code works.**

In [2]:
```
from string import punctuation
from pathlib import Path

import pandas as pd
from nltk.tokenize import word_tokenize
from tqdm.notebook import tqdm

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelBinarizer
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.metrics import classification_report
```

Task 1. Read and preprocess the data (1 point)

For this Homework, you are going to use IMDB movie reviews dataset. It contains 50,000 reviews, from which 25,000 are labeled as "positive" and the other 25,000 as "negative". This dataset is very frequently used for benchmarking the binary classification models.

For your convenience, the dataset is transformed into the .csv format and split into three files, each containing the train, validation, and test data accordingly. The labels are transformed into the binary format with `1` for "positive" and `0` for "negative".

In [3]:
```
data_train = pd.read_csv(Path("imdb_dataset/Train.csv"))
data_val = pd.read_csv(Path("imdb_dataset/Valid.csv"))
data_test = pd.read_csv(Path("imdb_dataset/Test.csv"))
```

In [4]:
```
data_train.head()
```

Out[4]:

	text	label
0	I grew up (b. 1965) watching and loving the Th...	0
1	When I put this movie in my DVD player, and sa...	0
2	Why do people who do not know what a particula...	0
3	Even though I have great interest in Biblical ...	0
4	Im a die hard Dads Army fan and nothing will e...	1

Your preprocessing function should minimally:

- tokenize each text
- lowercase
- remove punctuation

Optionally:

- remove stopwords
- lemmatize or stem each text

In [5]:
```
def preprocess(texts):
    # YOUR CODE STARTS HERE
    import nltk
    from nltk.corpus import stopwords
    nltk.download('stopwords')
    nltk_stopwords = set(stopwords.words('english'))

    processed = []

    for text in tqdm(texts):
        tokens = word_tokenize(text.lower())
        tokens = [word for word in tokens if word not in punctuation]
        tokens = [word for word in tokens if word not in nltk_stopwords]

        processed.append(tokens)

    return processed
    # YOUR CODE ENDS HERE
```

In [6]:
```
train_reviews = preprocess(data_train.text)
val_reviews = preprocess(data_val.text)
test_reviews = preprocess(data_test.text)
```

```
[nltk_data] Downloading package stopwords to /home/utlab/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
[nltk_data] Downloading package stopwords to /home/utlab/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
[nltk_data] Downloading package stopwords to /home/utlab/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Task 2. Transform the inputs for the model (1 point)

Use sklearn's `TfidfVectorizer` to transform the input texts.

Play with different hyperparameters.

Since our texts are already preprocessed, you can use a dummy function `lambda x: x` for the `tokenizer` and `preprocessor` arguments.

You can call the final variables `train_X`, `val_X`, `test_X`.

In [7]:
```
tfidf = TfidfVectorizer(tokenizer=lambda x: x, preprocessor=lambda x: x)
# YOUR CODE STARTS HERE
train_X = tfidf.fit_transform(train_reviews)
val_X = tfidf.transform(val_reviews)
test_X = tfidf.transform(test_reviews)
# YOUR CODE ENDS HERE
```

Use sklearn's `LabelBinarizer` to prepare the labels.

You can call the final variables `train_y`, `val_y`, `test_y`.

In [8]:
```
lbr = LabelBinarizer()
# YOUR CODE STARTS HERE
train_y = lbr.fit_transform(data_train.label)
val_y = lbr.transform(data_val.label)
test_y = lbr.transform(data_test.label)
# YOUR CODE ENDS HERE
```

Task 3. Initialize and train the classifier (1 point)

Initialize and train a logistic regression classifier. Refer to the sklearn documentation for more details on different hyperparameters.

Try to train several models with different hyperparameters and compare them with each other on the validation dataset. Use sklearn's `classification report` to get the scores with the precision of 4 digits, i.e. your score should have 4 digits after the decimal point (e.g. 0.8896).

In [9]:
```
clf = LogisticRegression()  # use appropriate arguments
# YOUR CODE STARTS HERE
model_l2 = LogisticRegression(penalty='none')
model_l2.fit(train_X, train_y.ravel())
y_pred = model_l2.predict(val_X)
print('Model: with penalty none')
print(classification_report(val_y, y_pred, digits=4))
print('----'*20)

model_none = LogisticRegression(penalty='l2')
model_none.fit(train_X, train_y.ravel())
y_pred = model_none.predict(val_X)
print('Model: with penalty l2')
print(classification_report(val_y, y_pred, digits=4))
print('----'*20)
# YOUR CODE ENDS HERE
```

```
Model: with penalty none
              precision    recall  f1-score   support

           0     0.8931    0.8705    0.8816      2486
           1     0.8750    0.8970    0.8859      2514

    accuracy                         0.8838      5000
   macro avg     0.8841    0.8837    0.8838      5000
weighted avg     0.8840    0.8838    0.8838      5000

--------------------------------------------------------------------------------
Model: with penalty l2
              precision    recall  f1-score   support

           0     0.9053    0.8729    0.8888      2486
           1     0.8786    0.9097    0.8939      2514

    accuracy                         0.8914      5000
   macro avg     0.8919    0.8913    0.8913      5000
weighted avg     0.8919    0.8914    0.8914      5000

--------------------------------------------------------------------------------
```

Task 4. Test the model and prepare for inference (1 point)

Test your model on the test set.

In [10]:
```
# YOUR CODE STARTS HERE
model_l2 = LogisticRegression(penalty='none')
model_l2.fit(train_X, train_y.ravel())
y_pred = model_l2.predict(test_X)
print('Model: with penalty none')
print(classification_report(test_y, y_pred, digits=4))
print('----'*20)

model_none = LogisticRegression(penalty='l2')
model_none.fit(train_X, train_y.ravel())
y_pred = model_none.predict(test_X)
print('Model: with penalty l2')
print(classification_report(test_y, y_pred, digits=4))
print('----'*20)
# YOUR CODE ENDS HERE
```

```
Model: with penalty none
              precision    recall  f1-score   support

           0     0.8932    0.8786    0.8858      2495
           1     0.8810    0.8954    0.8881      2505

    accuracy                         0.8870      5000
   macro avg     0.8871    0.8870    0.8870      5000
weighted avg     0.8871    0.8870    0.8870      5000

--------------------------------------------------------------------------------
Model: with penalty l2
              precision    recall  f1-score   support

           0     0.9074    0.8778    0.8925      2495
           1     0.8906    0.9098    0.9001      2505

    accuracy                         0.8988      5000
   macro avg     0.8990    0.8988    0.8988      5000
weighted avg     0.8990    0.8988    0.8988      5000

--------------------------------------------------------------------------------
```

Write a code that would allow you to input any text into the model and get the prediction. To do that, use the same preprocessing and TfidfvectorizerX as for the training data to transform the input text for the model.

Predict a label for the example text below.

In [11]:
```
example_text = """Don't Look Up" tells a chilling story of lies, oppression, explosion, and deceit in modern c
```

In [12]:
```
# YOUR CODE STARTS HERE
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity


test_tokenized = preprocess(example_text)
test_X = tfidf.transform(test_tokenized)


def predict_label(test_X, X, k=5):
    return np.argsort(cosine_similarity(test_X,X))[0][::-1][:k]

top_pred=predict_label(test_X,train_X)
print(top_pred)
# YOUR CODE ENDS HERE
```

```
[nltk_data] Downloading package stopwords to /home/utlab/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
[30967 21037 22173 24554  7895]
```

Using the knowledge of how a bag-of-words approach works, try to come up with four short movie reviews that would be predicted as true positive, true negative, false positive, false negative.

Usually, just one or two short sentences are enough. Also, your writing skills are not assessed here, so you can write anything as long as it works! If you cannot come up with anything that meets the criteria, you can write down below why do you think it didn't work and what was your strategy.

In [13]:
```
# YOUR CODE STARTS HERE

# YOUR CODE ENDS HERE
```

In [ ]:
```

```