Sentiment analysis with BOW representation

Text classification is a machine learning technique that assigns a set of predefined categories to open-ended text. Text classifiers can be used to organize, structure, and categorize pretty much any kind of text – from documents, medical studies and files, and all over the web.

For example, new articles can be organized by topics; support tickets can be organized by urgency; chat conversations can be organized by language; brand mentions can be organized by sentiment; and so on.

Text classification is one of the fundamental tasks in natural language processing with broad applications such as **sentiment analysis**, topic labeling, spam detection, and intent detection.

Why is Text Classification Important?

It's estimated that around 80% of all information is unstructured, with text being one of the most common types of unstructured data. Because of the messy nature of text, analyzing, understanding, organizing, and sorting through text data is hard and time-consuming, so most companies fail to use it to its full potential.

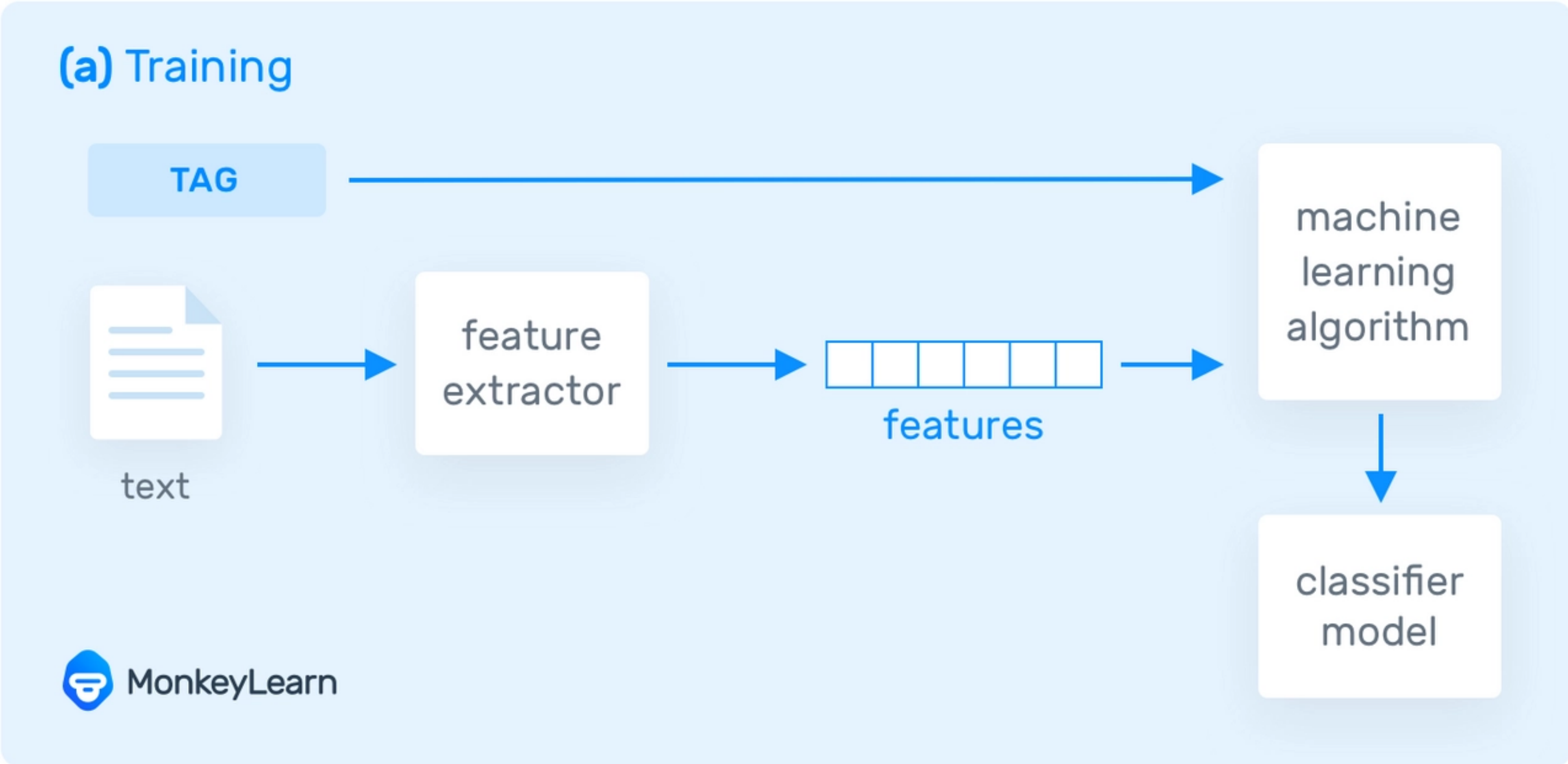
This is where text classification with machine learning comes in. Using text classifiers, companies can automatically structure all manner of relevant text, from emails, legal documents, social media, chatbots, surveys, and more in a fast and cost-effective way. This allows companies to save time analyzing text data, automate business processes, and make data-driven business decisions.

How Does Text Classification Work?

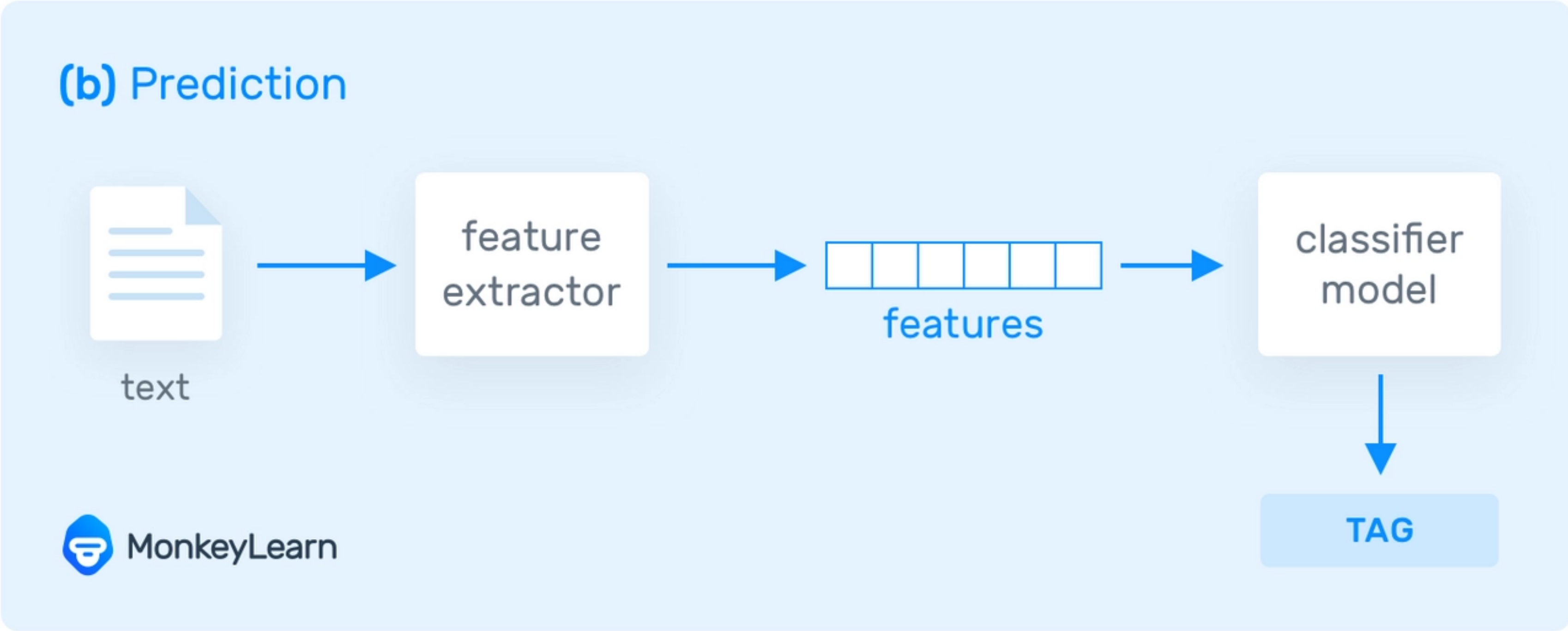
Instead of relying on manually crafted rules, machine learning text classification learns to make classifications based on past observations. By using pre-labeled examples as training data, machine learning algorithms can learn the different associations between pieces of text, and that a particular output (i.e., tags) is expected for a particular input (i.e., text). A "tag" is the pre-determined classification or category that any given text could fall into.

The first step towards training a machine learning NLP classifier is feature extraction: a method is used to transform each text into a numerical representation in the form of a vector. One of the most frequently used approaches is bag of words, where a vector represents the frequency of a word in a predefined dictionary of words.

Then, the machine learning algorithm is fed with training data that consists of pairs of feature sets (vectors for each text example) and tags (e.g. sports, politics) to produce a classification model:



Once it’s trained with enough training samples, the machine learning model can begin to make accurate predictions. The same feature extractor is used to transform unseen text to feature sets, which can be fed into the classification model to get predictions on tags (e.g., sports, politics):



Text classification with machine learning is usually much more accurate than human-crafted rule systems, especially on complex NLP classification tasks. Also, classifiers with machine learning are easier to maintain and you can always tag new examples to learn new tasks.

Today lab

In this lab we use part of the 'Amazon_Unlocked_Mobile.csv' dataset published by Kaggle. The dataset contain the following information:

- Product Name
- Brand Name
- Price
- Rating
- Reviews
- Review Votes

We are mainly interested by the 'Reviews' (X) and by the 'Rating' (y)

The goal is to try to predict the 'Rating' after reading the 'Reviews'. I've prepared for you TRAIN and TEST set.

Table of Contents

Load dataset

In [ ]:
```
import pandas as pd
import numpy as np
import nltk
nltk.download('popular')
```

```
[nltk_data] Downloading collection 'popular'
[nltk_data]    |
[nltk_data]    | Downloading package cmudict to
[nltk_data]    |     /home/joris/nltk_data...
[nltk_data]    |   Package cmudict is already up-to-date!
[nltk_data]    | Downloading package gazetteers to
[nltk_data]    |     /home/joris/nltk_data...
[nltk_data]    |   Package gazetteers is already up-to-date!
[nltk_data]    | Downloading package genesis to
[nltk_data]    |     /home/joris/nltk_data...
[nltk_data]    |   Package genesis is already up-to-date!
[nltk_data]    | Downloading package gutenberg to
[nltk_data]    |     /home/joris/nltk_data...
[nltk_data]    |   Package gutenberg is already up-to-date!
[nltk_data]    | Downloading package inaugural to
[nltk_data]    |     /home/joris/nltk_data...
[nltk_data]    |   Package inaugural is already up-to-date!
[nltk_data]    | Downloading package movie_reviews to
[nltk_data]    |     /home/joris/nltk_data...
[nltk_data]    |   Package movie_reviews is already up-to-date!
[nltk_data]    | Downloading package names to /home/joris/nltk_data...
[nltk_data]    |   Package names is already up-to-date!
[nltk_data]    | Downloading package shakespeare to
[nltk_data]    |     /home/joris/nltk_data...
[nltk_data]    |   Package shakespeare is already up-to-date!
[nltk_data]    | Downloading package stopwords to
[nltk_data]    |     /home/joris/nltk_data...
[nltk_data]    |   Package stopwords is already up-to-date!
[nltk_data]    | Downloading package treebank to
[nltk_data]    |     /home/joris/nltk_data...
[nltk_data]    |   Package treebank is already up-to-date!
[nltk_data]    | Downloading package twitter_samples to
[nltk_data]    |     /home/joris/nltk_data...
[nltk_data]    |   Package twitter_samples is already up-to-date!
[nltk_data]    | Downloading package omw to /home/joris/nltk_data...
[nltk_data]    |   Package omw is already up-to-date!
[nltk_data]    | Downloading package omw-1.4 to
[nltk_data]    |     /home/joris/nltk_data...
[nltk_data]    |   Package omw-1.4 is already up-to-date!
[nltk_data]    | Downloading package wordnet to
[nltk_data]    |     /home/joris/nltk_data...
[nltk_data]    |   Package wordnet is already up-to-date!
[nltk_data]    | Downloading package wordnet2021 to
```

```
[nltk_data]    |     /home/joris/nltk_data...
[nltk_data]    |   Package wordnet2021 is already up-to-date!
[nltk_data]    | Downloading package wordnet31 to
[nltk_data]    |     /home/joris/nltk_data...
[nltk_data]    |   Package wordnet31 is already up-to-date!
[nltk_data]    | Downloading package wordnet_ic to
[nltk_data]    |     /home/joris/nltk_data...
[nltk_data]    |   Package wordnet_ic is already up-to-date!
[nltk_data]    | Downloading package words to /home/joris/nltk_data...
[nltk_data]    |   Package words is already up-to-date!
[nltk_data]    | Downloading package maxent_ne_chunker to
[nltk_data]    |     /home/joris/nltk_data...
[nltk_data]    |   Package maxent_ne_chunker is already up-to-date!
[nltk_data]    | Downloading package punkt to /home/joris/nltk_data...
[nltk_data]    |   Package punkt is already up-to-date!
[nltk_data]    | Downloading package snowball_data to
[nltk_data]    |     /home/joris/nltk_data...
[nltk_data]    |   Package snowball_data is already up-to-date!
[nltk_data]    | Downloading package averaged_perceptron_tagger to
[nltk_data]    |     /home/joris/nltk_data...
[nltk_data]    |   Package averaged_perceptron_tagger is already up-
[nltk_data]    |       to-date!
[nltk_data]    |
[nltk_data]  Done downloading collection popular
```

Out[ ]:
```
True
```

In [ ]:
```python
TRAIN = pd.read_csv("http://www.i3s.unice.fr/~riveill/dataset/Amazon_Unlocked_Mobile/train.csv.gz")
TEST = pd.read_csv("http://www.i3s.unice.fr/~riveill/dataset/Amazon_Unlocked_Mobile/test.csv.gz")

TRAIN.head()
```

Out[ ]:

	Product Name	Brand Name	Price	Rating	Reviews	Review Votes
0	Samsung Galaxy Note 4 N910C Unlocked Cellphone...	Samsung	449.99	4	I love it!!! I absolutely love it!! 👌👍	0.0
1	BLU Energy X Plus Smartphone - With 4000 mAh S...	BLU	139.00	5	I love the BLU phones! This is my second one t...	4.0
2	Apple iPhone 6 128GB Silver AT&T	Apple	599.95	5	Great phone	1.0
3	BLU Advance 4.0L Unlocked Smartphone -US GSM -...	BLU	51.99	4	Very happy with the performance. The apps work...	2.0
4	Huawei P8 Lite US Version- 5 Unlocked Android ...	Huawei	198.99	5	Easy to use great price	0.0

Build X (features vectors) and y (labels)

In [ ]:

```
# Construct X_train and y_train
X_train = TRAIN['Reviews'].fillna("")
y_train = TRAIN['Rating']
X_train.shape, y_train.shape
```

Out[ ]: ((5000,), (5000,))

In [ ]:

```
# Construct X_test and y_test
X_test = TEST['Reviews'].fillna("")
y_test = TEST['Rating']
X_test.shape, y_test.shape
```

Out[ ]: ((1000,), (1000,))

Features extraction

A bag-of-words model is a way of extracting features from text so the text input can be used with machine learning algorithms or neural networks.

Each document, in this case a review, is converted into a vector representation. The number of items in the vector representing a document corresponds to the number of words in the vocabulary. The larger the vocabulary, the longer the vector representation, hence the preference for smaller vocabularies in the previous section.

Words in a document are scored and the scores are placed in the corresponding location in the representation.

In order to extract feature, you can use `CountVectorizer` or `TfidfVectorizer` and you can perform the desired text cleaning.

$$[TODO - Students]$$

> - Quickly remind what are `CountVectorizer` , `TfidfVectorizer` and how they work.\ `CountVectorizer` counts how many times a word occurs in the analysed text.\ `TfidfVectorizer` also counts how many times a word occurs in the analysed text, but then it compares it with its usual frequency within the whole corpus. This allows to get an idea of how a word is over-represented in the analyzed text and it disregards words that simply occur a lot in the whole corpus.

> - Build the BOW representation for train and test set

In [ ]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, classification_report
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

In [ ]:

```
preproc_pipe = make_pipeline(
    TfidfVectorizer(),
)
# Extract features
preproc_X_train = preproc_pipe.fit_transform(X_train)
preproc_X_test = preproc_pipe.transform(X_test)
```

Build a baseline with logistic regression.

Using the previous BOW representation, fit a logistic regression model and evaluate it.

$$[TODO - Students]$$

> - Quickly remind what are `LogisticRegression` and how they work.\ `LogisticRegression` computes a prediction probability for each of the classes

In [ ]:

```
y_train.value_counts().sort_index()
```

Out[ ]:

```
1     885
2     291
3     385
4     747
5    2692
Name: Rating, dtype: int64
```

> - what are the possible metrics. Choose one and justify your choice.\ We could use accuracy, recall, precision or f1-score. I choose to use the f1-score because our data set is unbalanced (class 1 -> 885, class 2 -> 291, class 3 -> 385, class 4 -> 747,

> class 5 -> 2692,)

In [ ]:

```
# Build your model
clf = LogisticRegression()

param_search = {
    # "penalty": ["l1", "l2"],
    "C": np.linspace(0.01,4),
    "max_iter": [1000],
}

search = RandomizedSearchCV(clf, param_search, n_jobs=-1, verbose=1, scoring="f1_weighted")
search.fit(preproc_X_train, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

Out[ ]:

```
RandomizedSearchCV(estimator=LogisticRegression(), n_jobs=-1,
                   param_distributions={'C': array([0.01      , 0.09142857, 0.17285714, 0.25428571, 0.33571429,
       0.41714286, 0.49857143, 0.58      , 0.66142857, 0.74285714,
       0.82428571, 0.90571429, 0.98714286, 1.06857143, 1.15      ,
       1.23142857, 1.31285714, 1.39428571, 1.47571429, 1.55714286,
       1.63857143, 1.72      , 1.80142857, 1.88285714, 1.96428571,
       2.04571429, 2.12714286, 2.20857143, 2.29      , 2.37142857,
       2.45285714, 2.53428571, 2.61571429, 2.69714286, 2.77857143,
       2.86      , 2.94142857, 3.02285714, 3.10428571, 3.18571429,
       3.26714286, 3.34857143, 3.43      , 3.51142857, 3.59285714,
       3.67428571, 3.75571429, 3.83714286, 3.91857143, 4.        ]),
                                         'max_iter': [1000]},
                   scoring='f1_weighted', verbose=1)
```

In [ ]:

```
log_reg_params = search.best_params_
log_reg_params
```

Out[ ]:

```
{'max_iter': 1000, 'C': 3.592857142857143}
```

In [ ]:

```
# Evaluate your model
y_pred_log_reg = search.predict(preproc_X_test)
log_reg_score = f1_score(y_test, y_pred_log_reg, average="weighted")
print(f"Logistic Regression best score: {round(log_reg_score, 4)}")
```

```
Logistic Regression best score: 0.6363
```

Build an MLP Classifier

In [ ]:

```
import tensorflow as tf
import tensorflow_addons as tfa
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.utils import plot_model
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

```
2022-01-20 18:22:23.084881: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic li
brary 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open shared object file: No such file or directory
2022-01-20 18:22:23.084909: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you d
o not have a GPU set up on your machine.
```

$$[TODO - Students]$$

> - Quickly remind what are `Multi Layer Perceptron` and how they work.\ `Multi Layer Perceptron` is a class of feed-forward neural network. They contain an input layer, an output layer and one or more hidden layer, each with an activation function.
> - If necessary, One hot encode the output vectors

In [ ]:

```
# Encode output vector if necessary.
from sklearn.preprocessing import OneHotEncoder
ohe_y = OneHotEncoder()
y_train_ohe = ohe_y.fit_transform(y_train.values.reshape(-1, 1))
y_test_ohe = ohe_y.transform(y_test.values.reshape(-1, 1))
```

$$[TODO - Students]$$

> - What is the size of the input vector and the output vector?

In [ ]:

```
# Define constant
input_dim = preproc_X_train.shape[1] # number of features of X_train
output_dim = y_train_ohe.shape[1] # number of classes (that we spread into 5 dimensions by OneHotEncod-ing)
```

$$[TODO - Students]$$

> - Build a simple network to predict the star rating of a review using the functional API. It should have the following characteristic : one hidden layer with 256 nodes and relu activation.
> - What is the activation function of the output layer?\ Softmax since we are tackling a classification problem.

In [ ]:

```
# Build your MLP
inputs = Input(shape=(input_dim,))
x = Dense(256, activation="relu")(inputs)
outputs = Dense(output_dim, activation="softmax")(x)
```

```
2022-01-20 18:22:24.861657: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:939] successful NUMA node read fro
m SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2022-01-20 18:22:24.862416: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic li
brary 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open shared object file: No such file or directory
2022-01-20 18:22:24.862491: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic li
brary 'libcublas.so.11'; dlerror: libcublas.so.11: cannot open shared object file: No such file or directory
2022-01-20 18:22:24.862558: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic li
brary 'libcublasLt.so.11'; dlerror: libcublasLt.so.11: cannot open shared object file: No such file or directory
2022-01-20 18:22:24.862625: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic li
brary 'libcufft.so.10'; dlerror: libcufft.so.10: cannot open shared object file: No such file or directory
2022-01-20 18:22:24.862693: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic li
brary 'libcurand.so.10'; dlerror: libcurand.so.10: cannot open shared object file: No such file or directory
2022-01-20 18:22:24.862762: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic li
brary 'libcusolver.so.11'; dlerror: libcusolver.so.11: cannot open shared object file: No such file or directory
2022-01-20 18:22:24.862827: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic li
brary 'libcusparse.so.11'; dlerror: libcusparse.so.11: cannot open shared object file: No such file or directory
2022-01-20 18:22:24.862893: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic li
brary 'libcudnn.so.8'; dlerror: libcudnn.so.8: cannot open shared object file: No such file or directory
2022-01-20 18:22:24.862903: W tensorflow/core/common_runtime/gpu/gpu_device.cc:1850] Cannot dlopen some GPU librarie
s. Please make sure the missing libraries mentioned above are installed properly if you would like to use GPU. Follow
the guide at https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your pla
tform.
Skipping registering GPU devices...
2022-01-20 18:22:24.863197: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is optimized
with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operat
ions:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

$$[TODO - Students]$$

We are now compiling and training the model.

> - Using the tensorflow documentation, explain the purpose the EarlyStopping callback and detail its arguments.\ As per the Tensorflow documentation:\ *Assuming the goal of a training is to minimize the loss. With this, the metric to be monitored would be 'loss', and mode would be 'min'. A model.fit() training loop will check at end of every epoch whether the loss is no longer decreasing, considering the min_delta and patience if applicable. Once it's found no longer decreasing, model.stop_training is marked True and the training terminates.*\ In other words, `EarlyStopping` stops the training process (*i.e.* the `fit` method) if no more progress is made in terms of the loss.
> - Compile the model
> - Fit the model

In [ ]:

```
# Compile the model and start training
# Stop training with early stopping with patience of 20
model = Model(inputs, outputs)
model.compile(optimizer="adam", loss="mean_squared_error", metrics=tfa.metrics.F1Score(5))

callback = EarlyStopping(monitor="loss", patience=20)
history = model.fit(x=preproc_X_train.toarray(), y=y_train_ohe.toarray(), epochs=100, callbacks=[callback], use_multi
```

```
Epoch 1/100
157/157 [==============================] - 2s 10ms/step - loss: 0.1142 - f1_score: 0.2362
Epoch 2/100
157/157 [==============================] - 2s 10ms/step - loss: 0.0773 - f1_score: 0.3880
Epoch 3/100
157/157 [==============================] - 2s 10ms/step - loss: 0.0561 - f1_score: 0.6689
Epoch 4/100
157/157 [==============================] - 2s 10ms/step - loss: 0.0374 - f1_score: 0.8482
Epoch 5/100
157/157 [==============================] - 2s 10ms/step - loss: 0.0252 - f1_score: 0.9078
Epoch 6/100
157/157 [==============================] - 2s 10ms/step - loss: 0.0191 - f1_score: 0.9372
Epoch 7/100
157/157 [==============================] - 2s 10ms/step - loss: 0.0156 - f1_score: 0.9520
Epoch 8/100
157/157 [==============================] - 2s 10ms/step - loss: 0.0135 - f1_score: 0.9591
Epoch 9/100
157/157 [==============================] - 2s 10ms/step - loss: 0.0120 - f1_score: 0.9638
Epoch 10/100
157/157 [==============================] - 2s 10ms/step - loss: 0.0112 - f1_score: 0.9647
Epoch 11/100
157/157 [==============================] - 2s 10ms/step - loss: 0.0105 - f1_score: 0.9673
Epoch 12/100
157/157 [==============================] - 2s 10ms/step - loss: 0.0100 - f1_score: 0.9691
Epoch 13/100
157/157 [==============================] - 2s 10ms/step - loss: 0.0097 - f1_score: 0.9711
Epoch 14/100
157/157 [==============================] - 2s 10ms/step - loss: 0.0094 - f1_score: 0.9705
Epoch 15/100
157/157 [==============================] - 2s 10ms/step - loss: 0.0091 - f1_score: 0.9701
Epoch 16/100
157/157 [==============================] - 2s 10ms/step - loss: 0.0089 - f1_score: 0.9720
Epoch 17/100
157/157 [==============================] - 2s 10ms/step - loss: 0.0088 - f1_score: 0.9701
Epoch 18/100
157/157 [==============================] - 2s 10ms/step - loss: 0.0086 - f1_score: 0.9724
Epoch 19/100
157/157 [==============================] - 2s 10ms/step - loss: 0.0083 - f1_score: 0.9734
Epoch 20/100
157/157 [==============================] - 2s 15ms/step - loss: 0.0081 - f1_score: 0.9738
Epoch 21/100
157/157 [==============================] - 2s 16ms/step - loss: 0.0080 - f1_score: 0.9745
Epoch 22/100
157/157 [==============================] - 2s 15ms/step - loss: 0.0078 - f1_score: 0.9750
```

```
Epoch 23/100
157/157 [==============================] - 2s 14ms/step - loss: 0.0077 - f1_score: 0.9739
Epoch 24/100
157/157 [==============================] - 2s 14ms/step - loss: 0.0076 - f1_score: 0.9754
Epoch 25/100
157/157 [==============================] - 2s 14ms/step - loss: 0.0075 - f1_score: 0.9757
Epoch 26/100
157/157 [==============================] - 2s 12ms/step - loss: 0.0075 - f1_score: 0.9760
Epoch 27/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0074 - f1_score: 0.9766
Epoch 28/100
157/157 [==============================] - 2s 12ms/step - loss: 0.0074 - f1_score: 0.9769
Epoch 29/100
157/157 [==============================] - 2s 12ms/step - loss: 0.0073 - f1_score: 0.9760
Epoch 30/100
157/157 [==============================] - 2s 12ms/step - loss: 0.0071 - f1_score: 0.9782
Epoch 31/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0072 - f1_score: 0.9772
Epoch 32/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0071 - f1_score: 0.9775
Epoch 33/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0068 - f1_score: 0.9782
Epoch 34/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0072 - f1_score: 0.9770
Epoch 35/100
157/157 [==============================] - 2s 12ms/step - loss: 0.0069 - f1_score: 0.9772
Epoch 36/100
157/157 [==============================] - 2s 14ms/step - loss: 0.0069 - f1_score: 0.9774
Epoch 37/100
157/157 [==============================] - 2s 12ms/step - loss: 0.0068 - f1_score: 0.9775
Epoch 38/100
157/157 [==============================] - 2s 12ms/step - loss: 0.0067 - f1_score: 0.9780
Epoch 39/100
157/157 [==============================] - 2s 12ms/step - loss: 0.0068 - f1_score: 0.9785
Epoch 40/100
157/157 [==============================] - 2s 12ms/step - loss: 0.0068 - f1_score: 0.9774
Epoch 41/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0069 - f1_score: 0.9778
Epoch 42/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0067 - f1_score: 0.9792
Epoch 43/100
157/157 [==============================] - 2s 12ms/step - loss: 0.0067 - f1_score: 0.9769
Epoch 44/100
157/157 [==============================] - 2s 12ms/step - loss: 0.0067 - f1_score: 0.9780
```

```
Epoch 45/100
157/157 [==============================] - 2s 12ms/step - loss: 0.0067 - f1_score: 0.9785
Epoch 46/100
157/157 [==============================] - 2s 12ms/step - loss: 0.0066 - f1_score: 0.9781
Epoch 47/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0064 - f1_score: 0.9783
Epoch 48/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0065 - f1_score: 0.9783
Epoch 49/100
157/157 [==============================] - 2s 12ms/step - loss: 0.0065 - f1_score: 0.9775
Epoch 50/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0065 - f1_score: 0.9781
Epoch 51/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0064 - f1_score: 0.9781
Epoch 52/100
157/157 [==============================] - 2s 14ms/step - loss: 0.0064 - f1_score: 0.9785
Epoch 53/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0063 - f1_score: 0.9789
Epoch 54/100
157/157 [==============================] - 2s 14ms/step - loss: 0.0064 - f1_score: 0.9792
Epoch 55/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0063 - f1_score: 0.9796
Epoch 56/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0064 - f1_score: 0.9775
Epoch 57/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0062 - f1_score: 0.9787
Epoch 58/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0062 - f1_score: 0.9795
Epoch 59/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0062 - f1_score: 0.9793
Epoch 60/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0061 - f1_score: 0.9796
Epoch 61/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0061 - f1_score: 0.9793
Epoch 62/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0062 - f1_score: 0.9798
Epoch 63/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0062 - f1_score: 0.9786
Epoch 64/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0062 - f1_score: 0.9785
Epoch 65/100
157/157 [==============================] - 2s 12ms/step - loss: 0.0060 - f1_score: 0.9799
Epoch 66/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0061 - f1_score: 0.9795
```

```
Epoch 67/100
157/157 [==============================] - 2s 12ms/step - loss: 0.0061 - f1_score: 0.9798
Epoch 68/100
157/157 [==============================] - 2s 12ms/step - loss: 0.0061 - f1_score: 0.9802
Epoch 69/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0060 - f1_score: 0.9792
Epoch 70/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0062 - f1_score: 0.9783
Epoch 71/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0060 - f1_score: 0.9787
Epoch 72/100
157/157 [==============================] - 2s 14ms/step - loss: 0.0060 - f1_score: 0.9788
Epoch 73/100
157/157 [==============================] - 2s 14ms/step - loss: 0.0060 - f1_score: 0.9797
Epoch 74/100
157/157 [==============================] - 2s 15ms/step - loss: 0.0059 - f1_score: 0.9799
Epoch 75/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0058 - f1_score: 0.9809
Epoch 76/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0059 - f1_score: 0.9797
Epoch 77/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0059 - f1_score: 0.9796
Epoch 78/100
157/157 [==============================] - 2s 14ms/step - loss: 0.0059 - f1_score: 0.9798
Epoch 79/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0059 - f1_score: 0.9792
Epoch 80/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0059 - f1_score: 0.9795
Epoch 81/100
157/157 [==============================] - 2s 14ms/step - loss: 0.0059 - f1_score: 0.9793
Epoch 82/100
157/157 [==============================] - 2s 15ms/step - loss: 0.0060 - f1_score: 0.9797
Epoch 83/100
157/157 [==============================] - 2s 15ms/step - loss: 0.0058 - f1_score: 0.9798
Epoch 84/100
157/157 [==============================] - 2s 14ms/step - loss: 0.0058 - f1_score: 0.9798
Epoch 85/100
157/157 [==============================] - 2s 15ms/step - loss: 0.0059 - f1_score: 0.9795
Epoch 86/100
157/157 [==============================] - 2s 15ms/step - loss: 0.0059 - f1_score: 0.9802
Epoch 87/100
157/157 [==============================] - 2s 14ms/step - loss: 0.0057 - f1_score: 0.9796
Epoch 88/100
157/157 [==============================] - 2s 14ms/step - loss: 0.0059 - f1_score: 0.9806
```

```
Epoch 89/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0059 - f1_score: 0.9796
Epoch 90/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0059 - f1_score: 0.9793
Epoch 91/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0059 - f1_score: 0.9793
Epoch 92/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0058 - f1_score: 0.9800
Epoch 93/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0058 - f1_score: 0.9797
Epoch 94/100
157/157 [==============================] - 2s 12ms/step - loss: 0.0057 - f1_score: 0.9811
Epoch 95/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0058 - f1_score: 0.9804
Epoch 96/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0058 - f1_score: 0.9799
Epoch 97/100
157/157 [==============================] - 2s 12ms/step - loss: 0.0057 - f1_score: 0.9811
Epoch 98/100
157/157 [==============================] - 2s 14ms/step - loss: 0.0057 - f1_score: 0.9807
Epoch 99/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0057 - f1_score: 0.9805
Epoch 100/100
157/157 [==============================] - 2s 13ms/step - loss: 0.0057 - f1_score: 0.9816
```

In [ ]:

```
model.summary()
plot_model(model, show_shapes=True)
```

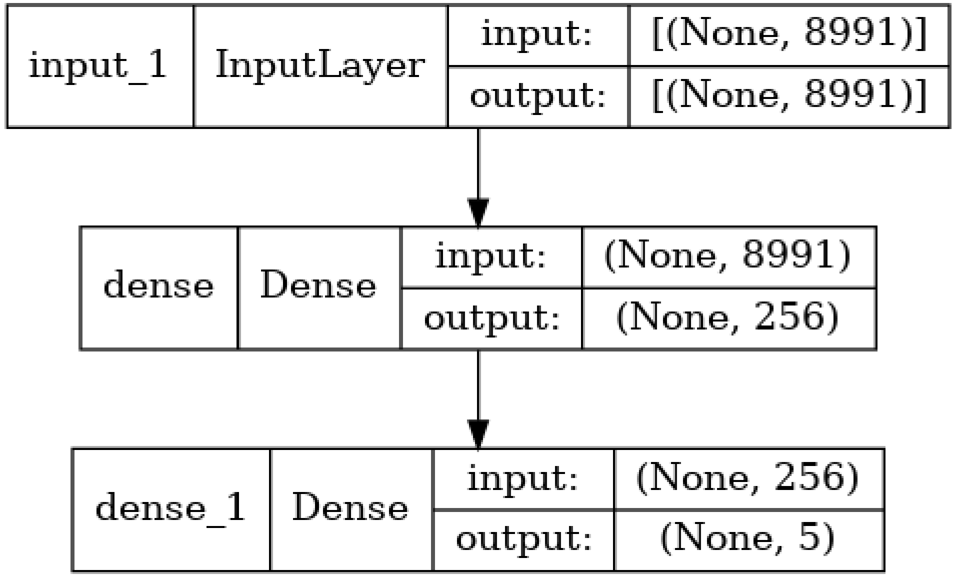
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 8991)]            0

 dense (Dense)               (None, 256)               2301952

 dense_1 (Dense)             (None, 5)                 1285

=================================================================
Total params: 2,303,237
Trainable params: 2,303,237
Non-trainable params: 0
_________________________________________________________________
```

Out[ ]:



$$[TODO - Students]$$

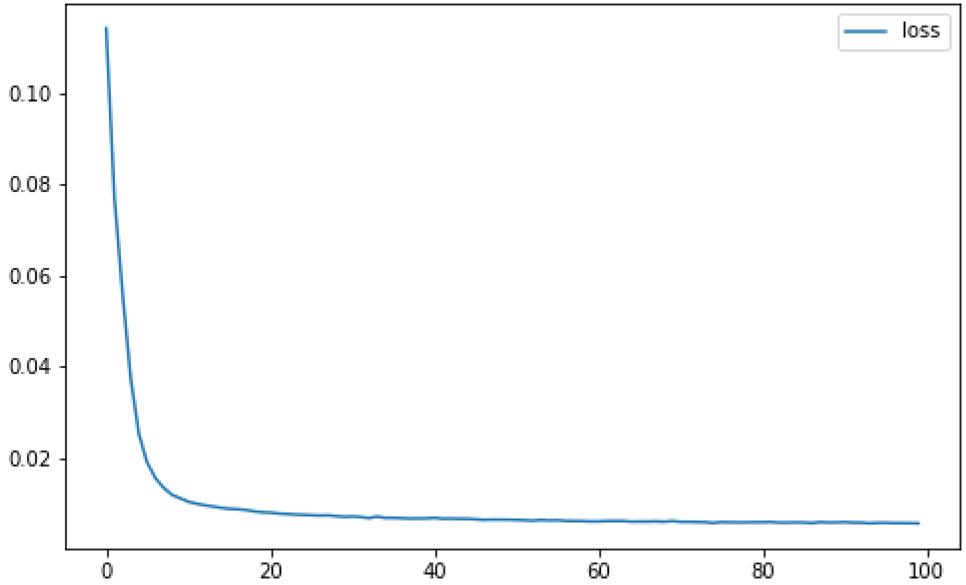
> - Babysit your model: plot learning curves

In [ ]:

```python
# Plot the learning curves and analyze them
# It's possible to plot them very easily using: pd.DataFrame(history.history).plot(figsize=(8,5))
pd.DataFrame(history.history).plot(figsize=(8,5))
```

Out[ ]:

```
<AxesSubplot:>
```



$$[TODO - Students]$$

> - How do you interpret those learning curves ?\ There is clearly something wrong since we see close to no improvement over the epochs. This could be due to overfitting.

The model appears to overfit the training data. Various strategies could reduce the overfitting but for this lab we will just change the number and size of layers. We will do that a little later.

> - Evaluate the model (on test part) and plot confusion matrix.
> - Are you doing better or worse than with our first attempt with Logistic regression.

In [ ]:

```
# Evaluate the model
pred_proba = model.predict(preproc_X_test.toarray())
y_pred = np.argmax(pred_proba, axis=1)+1
```

In [ ]:

```
mlp_initial_score = f1_score(y_test, y_pred, average="weighted")
print(f"Initial MLP best score: {round(mlp_initial_score, 4)}")
print(f"Logistic Regression best score: {round(log_reg_score, 4)}")
print("MLP performs worse than linear regression")
```

```
Initial MLP best score: 0.5925
Logistic Regression best score: 0.6363
MLP performs worse than linear regression
```

In [ ]:

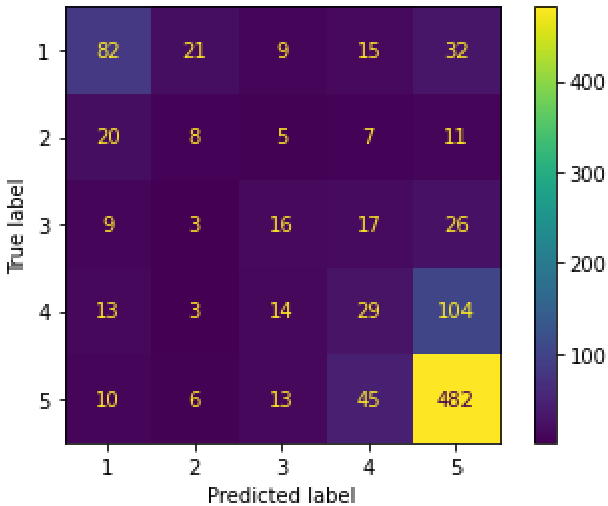
```
# Print/plot the confusion matrix
print("--> Using tensorflow:\n", tf.math.confusion_matrix(y_test, y_pred))
print("\n--> Using sklearn:")
cm = confusion_matrix(y_test, y_pred)
ConfusionMatrixDisplay(cm, display_labels=list(range(1,6))).plot()
cm
```

```
--> Using tensorflow:
 tf.Tensor(
[[  0   0   0   0   0   0]
 [  0  82  21   9  15  32]
 [  0  20   8   5   7  11]
 [  0   9   3  16  17  26]
 [  0  13   3  14  29 104]
 [  0  10   6  13  45 482]], shape=(6, 6), dtype=int32)

--> Using sklearn:
```

Out[ ]:

```
array([[ 82,  21,   9,  15,  32],
       [ 20,   8,   5,   7,  11],
       [  9,   3,  16,  17,  26],
       [ 13,   3,  14,  29, 104],
       [ 10,   6,  13,  45, 482]])
```



Hyper-parameters search

Using KerasTuner and modifying various hyper-parameters, improve your model. Change in particular the number of layers, the number of neurons per layer, the dropout, the regularization.

In [ ]:
```
import keras_tuner as kt


def build_model(hp):
    # Wrap a model in a function
    NUM_LAYERS = hp.Int("num_layers", 1, 3)
    # Define hyper-parameters
    NUM_DIMS = hp.Int("num_dims", min_value=32, max_value=128, step=32)
    ACTIVATION = hp.Choice("activation", ["relu", "tanh"])
    DROPOUT = hp.Boolean("dropout")
    DROP_RATE = hp.Choice("drop_rate", values=[0.2, 0.25, 0.5])
    # replace static value
    text_input = Input(shape=(input_dim,), name='input')
    h = text_input
    # with hyper-parameters
    for i in range(NUM_LAYERS):
        h = Dense(NUM_DIMS//(2*i+1), activation=ACTIVATION)(h)
        if DROPOUT:
            h = Dropout(rate=DROP_RATE)(h)
    ouputs = Dense(output_dim, activation='softmax', name='output')(h)
    model.compile(
        optimizer='adam',
        loss="categorical_crossentropy",
        metrics=tfa.metrics.F1Score(5),
    )
    return model


tuner = kt.BayesianOptimization(build_model,
                                objective="loss",
                                max_trials=10,
                                overwrite=True,
                                # directory='my_dir',
                                project_name='BOW_MLP')
ea = EarlyStopping(monitor='loss', mode='max',
                   patience=2, restore_best_weights=True)
```

```
tuner.search(preproc_X_train.toarray(), y_train_ohe.toarray(), epochs=10,
             validation_split=0.1,
             callbacks=[ea])
```

```
Trial 10 Complete [00h 00m 09s]
loss: 0.04891612380743027

Best loss So Far: 0.048436202108860016
Total elapsed time: 00h 01m 42s
INFO:tensorflow:Oracle triggered exit
```

In [ ]:

```
best_hp = tuner.get_best_hyperparameters()
model = tuner.hypermodel.build(best_hp[0])
H = model.fit(preproc_X_train.toarray(), y_train_ohe.toarray(),
              validation_split=0.1, callbacks=[ea])
```

```
141/141 [==============================] - 3s 16ms/step - loss: 0.0514 - f1_score: 0.9820 - val_loss: 0.1269 - val_f1
_score: 0.9596
```

In [ ]:

```
# evaluate the network
pred_proba = model.predict(preproc_X_test.toarray())
y_pred_tuned_MLP = pred_proba.argmax(axis=1)+1
y_test
```

Out[ ]:

```
0      4
1      4
2      1
3      1
4      5
      ..
995    5
996    4
997    5
998    5
999    1
Name: Rating, Length: 1000, dtype: int64
```

In [ ]:

```
mlp_tuned_score = f1_score(y_test, y_pred_tuned_MLP, average="weighted")

print(f"Logistic Regression best score: {round(log_reg_score, 4)}")
print(f"Initial MLP best score: {round(mlp_initial_score, 4)}")
print(f"Tuned MLP best score: {round(mlp_tuned_score, 4)}")
```

```
Logistic Regression best score: 0.6363
Initial MLP best score: 0.5925
Tuned MLP best score: 0.6025
```