Assignment 2 - COMP 6721

Applied Artificial Intelligence

Group Name: AI_Bots
Group Members:

1. Huzaifa Mohammed (40242080),
2. Mohammed Shurrab (40323793),
3. Oleksandr Yasinovskyy (40241188)

All members have contributed equally to the solution of the Assignment

Professor
Prof. Arash Azafar

Date
22/06/2025

Question 1:

The code is attached

a) K-means clustering

The code to create the K-means model with $n\_clusters = 3, random\_state = 6721, init = k - means + +$, where the default distance metric is Euclidean distance. X_train holds the training data which are the extracted features from the images training dataset (1860 features using color histogram and HOG).

```
kmeans = KMeans(n_clusters=3, random_state=6721, init='k-means++', n_init='auto')
kmeans.fit(X_train)
```

```
Extracted features using Color Histogram + HOG

Cluster 0:
['library-indoor'] :     1652.0
['museum-indoor'] :      1342.0
['shopping_mall-indoor'] :       2046.0

Cluster 1:
['library-indoor'] :     2044.0
['museum-indoor'] :      1598.0
['shopping_mall-indoor'] :       2628.0

Cluster 2:
['library-indoor'] :     1304.0
['museum-indoor'] :      2060.0
['shopping_mall-indoor'] :       326.0
```

- **Results Evaluation**

  A good metric to evaluate the results obtained by K-means clustering is Adjusted Rand Index (ARI), which measure the similarity between predicted clusters and the true labels. It evaluates how well pairs of samples are grouped together. A high ARI indicates better performance (1 means perfect match with ground truth, while 0 indicates random labelling).

  Another metric is Normalized Mutual Information (NMI), which is based on information theory and measures how much information the predicted clusters share with the true labels. Unlike ARI, NMI measures how similar the overall distribution of clusters is to the label distribution, and not just pairwise matches. A high NMI indicates better performance where 1 means the clusters are perfectly aligned with labels, whereas 0 indications completely independent data (no info shared).

```
from sklearn.metrics import adjusted_rand_score, normalized_mutual_info_score

ari = adjusted_rand_score(y_train_enc, y_kmeans)
nmi = normalized_mutual_info_score(y_train_enc, y_kmeans)

print(f"Adjusted Rand Index (ARI): {ari:.4f}")
print(f"Normalized Mutual Information (NMI): {nmi:.4f}")

```

✓ 0.0s

```
Adjusted Rand Index (ARI): 0.0453
Normalized Mutual Information (NMI): 0.0556
```

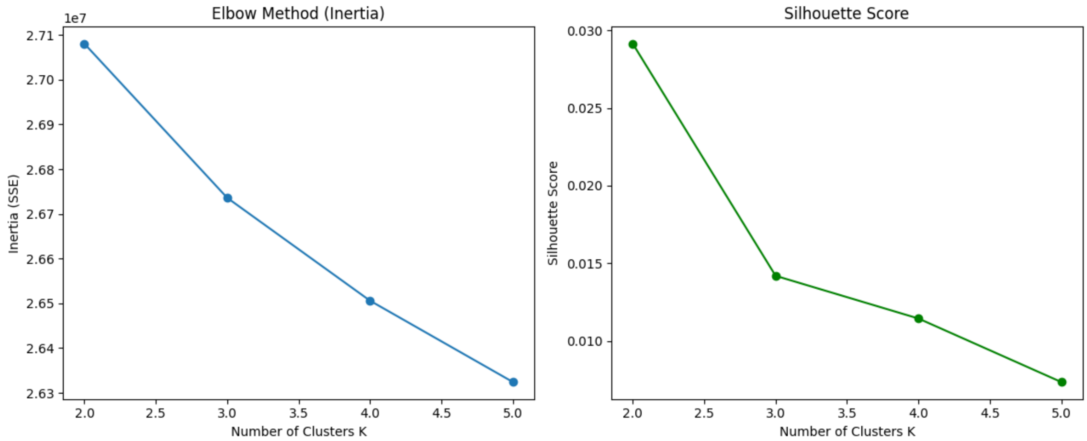
Another way to evaluate the clusters is by assigning each cluster a label based on the most frequent class within the cluster and thus we can calculate accuracy and the confusion matrix. However as seen in the first figure, shopping mall is the dominant class in two clusters (0 and 1), thus this naïve approach will not add any insight in this case. Moreover, this approach is not reproducible since the clusters numbering is not always consistent.

- **Results Interpretation**

K-Means groups images based solely on similarity in their color and edge-based feature vectors. If the clusters roughly correspond to the actual categories (e.g., shopping malls cluster together), it indicates that the feature extraction method was meaningful, where the model is capturing underlying visual patterns without labels. However, misclusterings can occur because: 1) K-Means assumes spherical clusters with equal variance, 2) there is a big visual overlap between classes (e.g., museums and libraries may share textures), and 3) the high dimensionality of the data can distort Euclidean distances.

Thus, while K-Means isn't perfect for complex image datasets, it's useful for exploring feature space structure and verifying that the preprocessing pipeline captures meaningful information.

b) Determining number of clusters (k)

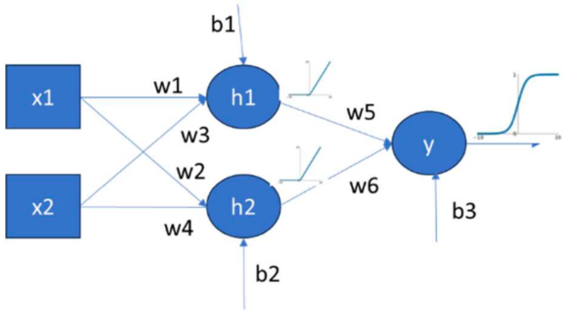


The best number of clusters can be determined when the graph converges to a minimum. We can see that the sillhouette analysis shows that 3 clusters is the best number of clusters needed, however the elbow method is not converging.

c) K-means vs K-medoids

K-Medoids is more robust to outliers and noise than K-Means because it uses actual data points (medoids) as cluster centers rather than means, which can be skewed by extreme values. Unlike K-Means, it minimizes a sum of pairwise dissimilarities, not squared distances.

Question 2:



Q2| a)

output y?

$h_1' = w_1 x_1 + w_3 x_2 + b_1$
$= 0.1(1) + 0.3(1) + 0.1$
$= 0.5$
$\Rightarrow$ ReLU (0.5) $\boxed{h_1 = 0.5}$

$h_2' = w_2 x_1 + w_4 x_2 + b_2$
$= 0.2(1) + 0.4(1) + 0.2$
$= 0.8$
$\Rightarrow$ ReLU(0.8) $\Rightarrow$ $\boxed{h_2 = 0.8}$

$y' = w_5 h_1 + w_6 h_2 + b_3$
$= 0.5(0.5) + 0.6(0.8) + 0.3$
$= 1.03$

$\Rightarrow$ Sigmoid (1.03) $\Rightarrow$ $\sigma(1.03)$

$y = \frac{1}{1+e^{-1.03}} \approx \boxed{0.737}$

b) Loss?

$L = \frac{1}{2}(O-T)^2 + \frac{1}{2}\sum w_i^2$

$= \frac{1}{2}(0.737-1)^2 + \frac{1}{2}(0.1^2 + 0.2^2 + 0.3^2 + 0.4^2 + 0.5^2 + 0.6^2)$

$= \frac{1}{2}(0.0692) + \frac{1}{2}(0.91)$

$\boxed{L \cong 0.4896}$

c) weights update ... ⊛ z terms are before activation

h terms are after activation

at the output layer:

$$\frac{\partial L}{\partial z_3} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z_3} = (y-T)\, y\,(1-y)$$

$$= (0.737-1)(0.737)(1-0.737)$$

$$= -0.051$$

$$\frac{\partial L}{\partial w_5} = \frac{\partial L_P}{\partial w_5} + \frac{\partial L_R}{\partial w_5}$$

L_P is prediction loss

L_R is regularization term

$$= \frac{\partial L_P}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_5} + \frac{\partial\left(\frac{1}{2}w_5^2\right)}{\partial w_5}$$

$$= (-0.051)(h_1) + w_5 = -0.051(0.5) + 0.5$$

$$= 0.4745$$

$$\frac{\partial L}{\partial w_6} = \frac{\partial L_P}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_6} + w_6 = -0.051\,(0.8) + 0.6$$

$$= 0.5592$$

$$\frac{\partial L}{\partial b_3} = \frac{\partial L}{\partial z_3} = -0.051$$

⟹ for the hidden layers

$$\frac{\partial L}{\partial h_1} = \frac{\partial L}{\partial z_3} \cdot \frac{\partial z_3}{\partial h_1} = -0.051 \times \overset{0.5}{w_5} = -0.0255$$

ReLu derevative = 1 for $x > 0$ otherwise 0

$$\frac{\partial L}{\partial z_1} = \frac{\partial L}{\partial h_1} \cdot \frac{\partial h_1}{\partial z_1} = -0.0255 \cdot (1) = -0.0255$$

$$\frac{\partial L}{h_2} = \frac{\partial L}{\partial z_3} \cdot \frac{\partial z_3}{\partial h_2} = -0.051 \times w_6 = -0.0306$$

$$\frac{\partial L}{\partial z_2} = \frac{\partial L}{\partial h_2} \cdot \frac{\partial h_2}{\partial z_2} = -0.0306$$

$$\Rightarrow \frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1} + w_1 = -0.0255 \times x_1 + w_1$$

$$= -0.0255\,(1) + 0.1 = 0.0745$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_3} + w_3 = -0.0255\,(1) + 0.3$$

$$= 0.2745$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial z_1} = -0.0255$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2} + w_2 = -0.0306\,(x_2) + 0.2$$

$$= 0.1694$$

$$\frac{\partial L}{\partial w_4} = \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_4} + w_4 = -0.0306\,(1) + 0.4$$

$$= 0.3694$$

$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial z_2} = -0.0306$$

new weights: $w = w + \Delta w \Rightarrow \Delta w = -\eta \frac{\partial L}{\partial w}$

$$w_1 = w_1 - 0.1(0.0745) = 0.0926$$
$$w_2 = w_2 - 0.1(0.1694) = 0.1831$$
$$w_3 = w_3 - 0.1(0.2745) = 0.2726$$
$$w_4 = w_4 - 0.1(0.3694) = 0.3631$$
$$w_5 = w_5 - 0.1(0.4745) = 0.4526$$
$$w_6 = w_6 - 0.1(0.5592) = 0.5441$$
$$b_1 = b_1 - 0.1(-0.0255) = 0.1026$$
$$b_2 = b_2 - 0.1(-0.0306) = 0.2031$$
$$b_3 = b_3 - 0.1(-0.051) = 0.3051$$

d)

d) $\alpha = 0.9$ & velocity $= 0.5$

$$v_{new} = \alpha \cdot v_{prev} + \eta \nabla L = 0.9(0.5) + 0.1(0.4745)$$

$$= 0.49745$$

$$w_5 = w_5 - v_{new} = 0.5 - 0.49745$$

$$\boxed{w_5 = 0.00255}$$

Question 3:

Q3

a)

Conv 1 : 16 filters with size 3x3

stride = 1 , padding = 1

$$\text{output size} = \frac{\overset{\text{input size}}{32} + 2 - 3}{1} + 1 = 32$$

output feature map has size 32x32 as we have 16 feature map

$\Rightarrow$ Total # of neurons $= 32 \times 32 \times 16 = \boxed{16\,384}$

b) # of weights

This is equivalent to # of trainable parameters.

$$\#\text{ weights} = \underbrace{3 \times 3}_{\text{filter size}} \times \underbrace{3}_{\text{input depth}} \times \underbrace{16}_{\text{\# of filters.}} = \boxed{432}$$

c) Similar to a.

Conv 2 : 16 filters , 3x3

stride =1 , padding =0

input to Conv2 is $32 \times 32 \times 16$

$$\text{output size} = \frac{32 - 3}{1} + 1 = 30$$

hence output feature map is 30×30 , with 16 depth

# of neurons $= 30 \times 30 \times 16 = \boxed{14\,400}$

d) Similar to b.

$$\text{weights} = \underbrace{3 \times 3}_{\text{filter size}} \times \underbrace{16}_{\text{conv1 depth}} \times \underbrace{16}_{\text{Conv2 filters number}} = \boxed{2304}$$

e) output size of Conv2 = $30 \times 30 \times 16$

After 2×2 maxpooling with no overlapping, we get:

$$\frac{30}{2} = 15 \Rightarrow \underbrace{15 \times 15}_{\text{new size}} \times \underbrace{16}_{\text{\# of filters.}}$$

# of features = $15 \times 15 \times 16 = \boxed{3600}$

f)

* Using multiple small filters is better since it can capture more localized details of the input, enabling deeper networks with more non-linearities, as allowing it to learn more complex features & patterns.
* It provides better feature hierarchy which improves the model generalization capability.
* It also results in fewer trainable parameters. i.e., 3×3 produces 9 weights whereas 7×7 produces 49 weights.

Question 4:

a) CNN model structure

The CNN model consists of two 2D convolutional layers with a filter size of 3x3 and depth of 16 (# of filters) and padding of 1 (to retain the original image size). After each Conv layer there is a 2D batch normalization layer and a LeakyReLU activation function. These layers are followed by a 2D MaxPool layer of size 2x2. Afterwards the output is connected to a fully connected layer with size 14x14x16 = (3136,32), followed by another fully connected layer of size (32,16) and finally the output layer cosists of (16,10), where 10 is the number of classes. The activation function for the conv layers is LeakyReLU, whereas the fully connected layers have normal ReLU as an activation function. The model uses CrossEntropyLoss, which applies softmax that is suitable for the task at hand. A snapshot of the model summary is provided below:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 16, 28, 28]             160
       BatchNorm2d-2           [-1, 16, 28, 28]              32
         LeakyReLU-3           [-1, 16, 28, 28]               0
            Conv2d-4           [-1, 16, 28, 28]           2,320
       BatchNorm2d-5           [-1, 16, 28, 28]              32
         LeakyReLU-6           [-1, 16, 28, 28]               0
         MaxPool2d-7           [-1, 16, 14, 14]               0
            Linear-8                   [-1, 32]         100,384
              ReLU-9                   [-1, 32]               0
           Linear-10                   [-1, 16]             528
             ReLU-11                   [-1, 16]               0
           Linear-12                   [-1, 10]             170
================================================================
Total params: 103,626
Trainable params: 103,626
Non-trainable params: 0
----------------------------------------------------------------
```

- The small 3x3 Conv layers are essential to capture local edge/texture patterns
- The batch-norm layers stabilise learning
- MaxPool layer reduces the spatial size which reduces the number of neurons required to implement the fully connected layer
- The above layers are treated as the encoder
- The fully connected layers maps features to class scores (classifier)

b)

The activation functions used are LeakyRelu, standard ReLU, and softmax is used internally via the CrossEntropyLoss

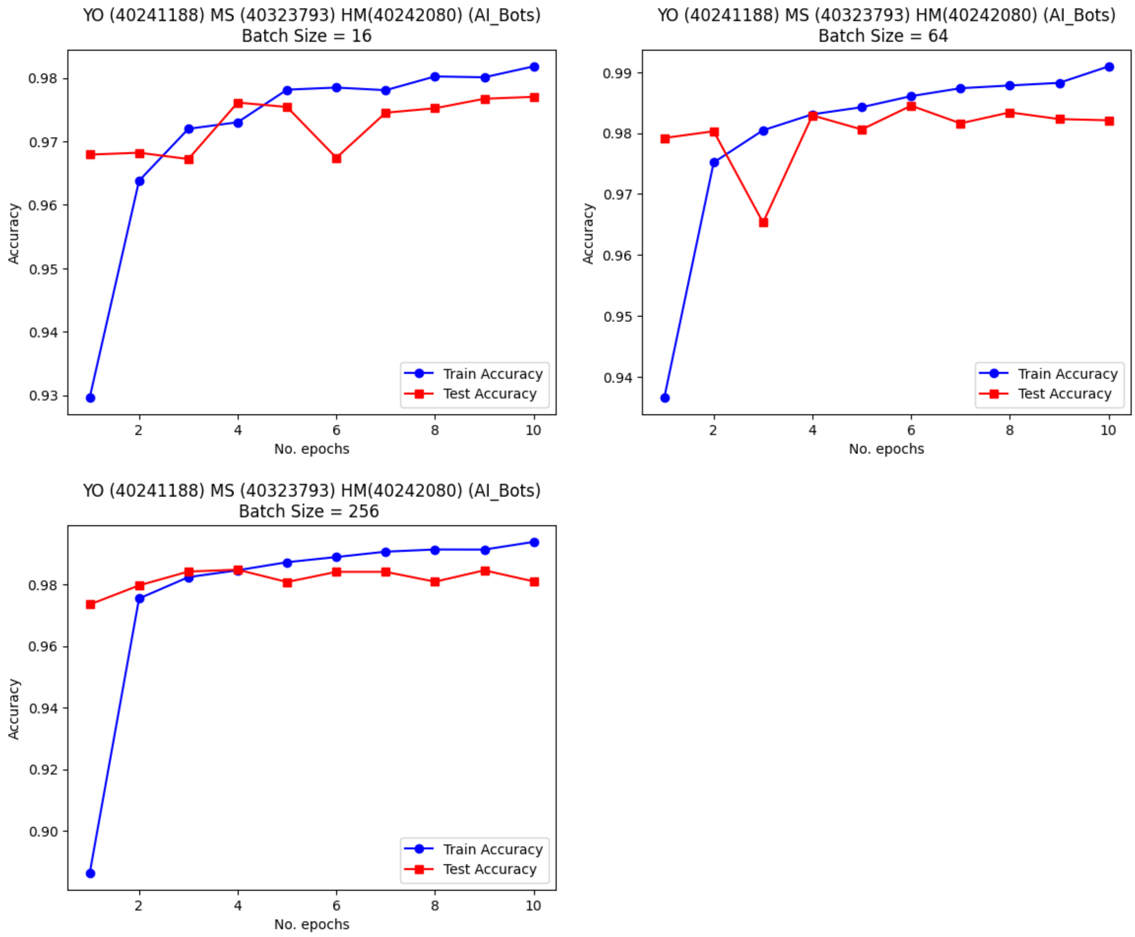
The LeakyRelu avoids dead-ReLU problem, whereas ReLU is the standard choice for fully connected layers. Softmax is suitable for multi-class problems c)

- *What are these bias and weight parameters printed here? What is their role? Justify the number of parameters that we see.*

These parameters belong to the batch normalization (BN) layer where the first two tensors consist of the biases and scaling learnable parameters that belong to the first BN layer. And the last two tensors belong to the second BN layer. The role of these two parameters is to make the distribution close to standard normal that helps in optimizing the network, while still getting the benefits of stable mean/variance during training.

Since the BN size is 16, we need to learn 16 biases and 16 scaling parameter per layer. Hence, each BN layer produces 32 trainable parameters. The model consists of 2 BN layers, making the total number of trainable parameters for these layers 64, which is consistent with the output in the image.

e)



We can see that as we increase the batch size, the training curve converges faster, where it reachs ~98% training accuracy just after 1 epoch for the case of 256.

It is also seen that with batch size 16, training was slower initially, due to noisy gradients, but it achieved better generalization since test accuracy closely tracks the training accuracy. As for batch size 64, we can see that it provided a balance between stability and generalization, resulting I smooth and rapid convergence with the highest overall performance. In contrast, batch size 256 led to very stable (less noisy gradients) and fast training, but showed a noticable gap between training and test accuracy after a few epochs. This indicates signs of overfitting.

f)

It is observed that using batch size 1 results in poor performance (accuracy of ~10%), where the model shows very slow and unstable training progress, with both training and test accuracy fluctuating and remaining low. This is due to the extremely noisy gradient estimates from processing only one image at a time. Also, high variance forces the optimiser to take very small effective steps which significantly increases the training time.

Batch size 256 leads to rapid and stable improvements in training accuracy, as the gradients are computed from a large number of samples and thus more reliable (less noisy gradient). However, while training accuracy improves steadily, test accuracy lags slightly behind, indicating the model may be overfitting to the training data.

g)

This gap between training and testing accuracy indicates overfitting of the model. This can be mitigated by using 1) dropout, 2) data augmentation (Transform), 3) weight decay in the optimizer, and 4) increasing the momentum in the BN layer.

Question 5:

a)

Code:

```
from sklearn.feature_extraction.text import CountVectorizer
# Creating Bag of Words
headlines = df['short_description']

# with stop-words
cv_with = CountVectorizer()
bow_with  = cv_with.fit_transform(headlines)
vocab_size_with = len(cv_with.get_feature_names_out())

# without stop-words
cv_no = CountVectorizer(stop_words='english')
bow_no = cv_no.fit_transform(headlines)
vocab_size_no = len(cv_no.get_feature_names_out())

# three most-frequent tokens (stop-words removed version)
word_counts = bow_no.sum(axis=0).A1          # flatten matrix to 1-D
vocab       = cv_no.get_feature_names_out()
top_idx     = word_counts.argsort()[-3:][::-1]
top_words   = [(vocab[i], int(word_counts[i])) for i in top_idx]

print("BoW vocab size (WITH stop-words) :", vocab_size_with)
print("BoW vocab size (NO  stop-words) :", vocab_size_no)
print("Top 3 frequent words (stop-words removed) :", top_words)
```

Output:

```
BoW vocab size (WITH stop-words) : 75726
BoW vocab size (NO  stop-words) : 75420
Top-3 frequent words (stop-words removed) : [('new', 10730), ('people', 9801), ('time', 9791)]
```

b)

```
## Q5) b
import spacy
nlp = spacy.load("en_core_web_sm")
sample_idx   = 6721
sample_text  = df.iloc[sample_idx]['short_description']
doc          = nlp(sample_text)

print(f"\nspaCy NER for row {sample_idx}:")
for ent in doc.ents:
    print(f"{ent.text:30}  →  {ent.label_}")
```

✓ 1.8s

```
spaCy NER for row 6721:
Joe Perricone                   →  PERSON
95                              →  DATE
Bill William Arnold Craddock    →  PERSON
85                              →  DATE
more than 65 years              →  DATE
```

c)

Code:

```
## Q5) c
from transformers import pipeline

hf_ner = pipeline("ner", grouped_entities=True)
hf_entities = hf_ner(sample_text)
(hf_entities)
```

Output:

```
[{'entity_group': 'PER',
  'score': np.float32(0.9983566),
  'word': 'Joe Perricone',
  'start': 0,
  'end': 13},
 {'entity_group': 'PER',
  'score': np.float32(0.96669865),
  'word': 'Bill William Arnold Craddock',
  'start': 23,
  'end': 51}]
```

The hugging face model that we are using is not trained to detect time/date, it only contains the four tags PER, ORG, LOC, MISC; hence the model output only detects persons when compared to Spacy in part b

d)

Code:

```
## Q5) d
from transformers import pipeline
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

extractor = pipeline("feature-extraction",
                     model="bert-base-uncased",
                     tokenizer="bert-base-uncased")

def sentence_embedding(text):
    """average the token vectors to get one vector per sentence"""
    token_vectors = np.array(extractor(text)[0])   # (tokens, 768)
    return token_vectors.mean(axis=0)              # (768,)

vec1 = sentence_embedding("I do not understand this assignment.")
vec2 = sentence_embedding("This assignment is pretty clear.")

print("\nEmbedding dimension:", vec1.shape[0])

emb_montreal = extractor("Montreal")
aa = np.array(emb_montreal)
print ('Embedding shape for Montreal: ',aa.shape)

sim = cosine_similarity(vec1.reshape(1,-1), vec2.reshape(1,-1))[0][0]
print("Cosine similarity between the two example sentences:", f"{sim:.3f}")
```

Output:

```
Embedding dimension: 768
embedding shape for Montreal:  (1, 3, 768)
Cosine similarity between the two example sentences: 0.725
```

- Dataset References:

1. Misra, Rishabh. "News Category Dataset." arXiv preprint arXiv:2209.11429 (2022).
2. Misra, Rishabh and Jigyasa Grover. "Sculpting Data for ML: The first act of Machine Learning." ISBN 9798585463570 (2021).