Practice 3

Vectorization methods

As we know that many Machine Learning algorithms and almost all Deep Learning Architectures are not capable of processing strings or plain text in their raw form. In a broad sense, they require numerical values as input to perform any kind of task, such as classification, regression, clustering, etc (Figure 3.1).

Task 1. Suggest one example of NLP for all types of tasks shown in Figure 3.1.

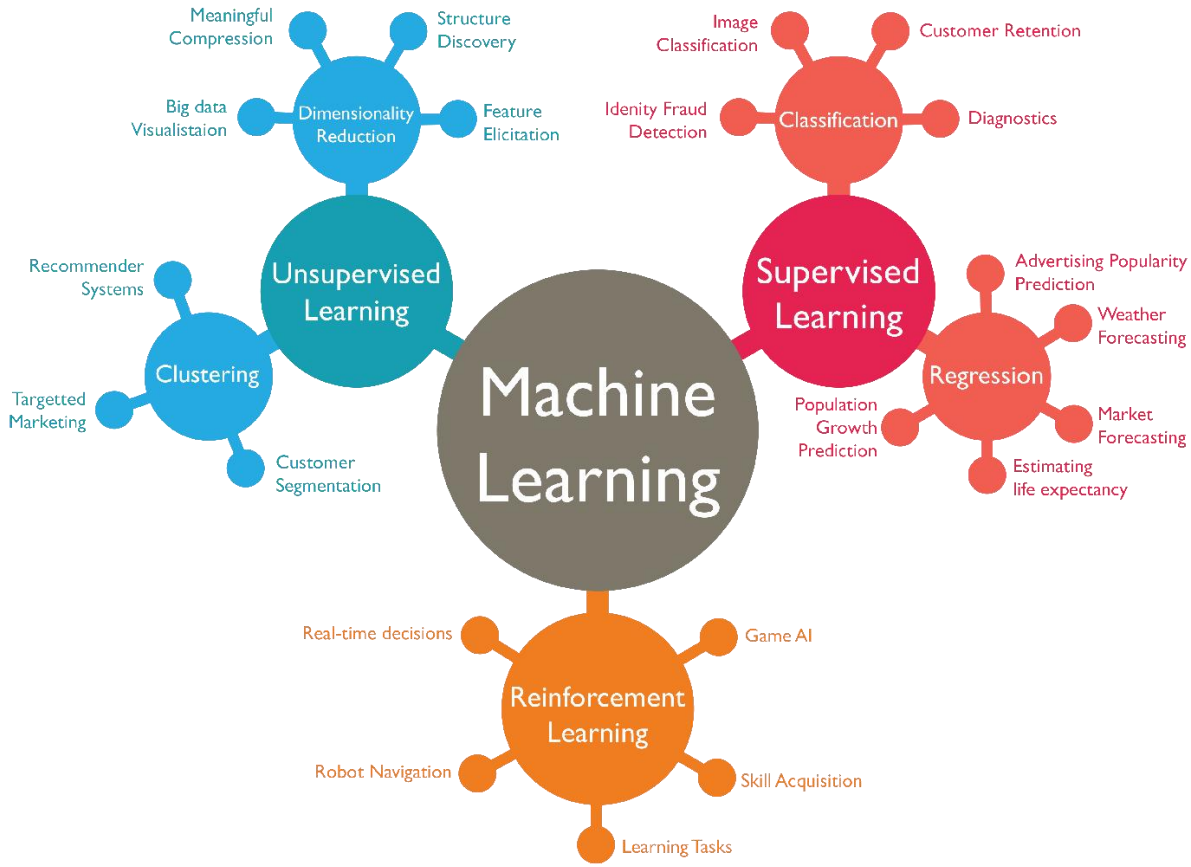

Figure 3.1. Machine learning task classification

In addition, from the huge amount of data presented in textual format, it is necessary to extract some knowledge and create any useful applications. Shortly, we can say that in order to build any model in machine or deep learning, the end-level data must be in numerical form, because models do not understand textual or graphical data directly, as people do.

In order to convert text data to numeric type, we need some clever ways known as **vectorization**, or in the NLP world, this is known as **word embedding**. Therefore, **vectorization** or **word embedding** is the process of converting text data into numeric vectors.

There are many methods of text data vectorization: One-Hot Encoding, Bag-of-Words, N-grams Vectorization, TF-IDF Vectorization, CountVectorizer, Word2Vec, GloVe, FastText, Transformer Encoders.

Task 2. Give the definition to text vectorization (word embedding).

1. **CountVectorizer (Bag-of-words)**

This is a great tool provided by the scikit-learn library in Python. It is used to transform a given text into word frequency vector that shows how often a word occurs throughout the text. The **CountVecorizer** creates a matrix where each unique word is represented by a matrix column, and each text instance from the document is represented by a row in the matrix. The value of each cell is nothing but the number of the word in that particular text sample.
CountVecorizer is the simplest technique involves three operations:

- **Tokenization**

First, the input text is tokenized. A sentence is presented as a list of its constituent words, and it's done for all the input sentences.

- **Vocabulary creation**

From all obtained tokenized words, only unique words are selected to create the vocabulary and then sorted by alphabetical order.

- **Vector representation**

Finally, the sparse matrix is filled with vocabulary word frequencies. In this sparse matrix, each row is a sentence vector whose length (the columns of the matrix) is equal to the size of the dictionary. Let's work with an example and see what it looks like in practice.

```
from sklearn.feature_extraction.text import CountVectorizer
# max_features: The CountVectorizer will select the
# words/features/terms which occur the most frequently.
# It takes absolute values so if you set the 'max_features = 3',
# it will select the 3 most common words in the data.
countv = CountVectorizer(max_features = 1000)

X = countv.fit_transform(df.clean_text).toarray()
y = df.iloc[:, 1].values
```

Split data for further usage in the model

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.20, random_state = 123)
```

Task 3. Print out the dictionary that you get with CountVectorizer, describe the output.
Task 4. Print out and describe the structure of "X_train" and "Y_train" objects.

Task 5. What does the "random_state" parameter influence on? can it be initialized with another value?

2. Term frequency-inverse document frequency (TF-IDF)

It gives a measure that takes into account the importance of a word depending on how often it occurs in a document and corpus.

To understand **TF-IDF**, we first understand the two terms separately:

- **Term frequency (*TF*)** denotes the frequency of a word in a document. For a specified word, it is defined as the ratio of the number of times a word appears in a document to the total number of words in the document.

$$TF = \frac{Frequency\ of\ word\ in\ a\ document}{Total\ number\ of\ words\ in\ that\ document}$$

- **Inverse document frequency (*IDF*)** measures the importance of the word in the corpus. It measures how common a particular word is across all the documents in the corpus.

$$IDF(W) = \log\left(\frac{Total\ number\ of\ documents}{Documents\ containing\ word\ W}\right)$$

The final **TF-IDF** score comes out to be:

$$TF\_IDF = TF \times IDF$$

Important points about **TF-IDF Vectorizer:**

1. Similar to the **CountVecorizer** method, the **TF-IDF** method creates a matrix of document terms, and each column represents a single unique word.

2. The difference of the **TF-IDF** method is that each cell doesn't indicate the term frequency, but contains a weight value that signifies how important a word is for an individual text message or document.

3. This method is based on the frequency method but it is different from the **CountVecorizer** in the sense that it takes into the occurrence of a word not only in one document, but in the entire corpus.

4. **TF-IDF** gives more weight to less frequent events and less weight to expected events. Thus, it penalizes frequently occurring words that appear frequently in a document, such as "the", "is", but assigns more weight to less frequent or rare words.

5. The product of **TF × IDF** of a word indicates how often the token is found in the document and how unique the token is to the whole entire corpus of documents.

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
X = df.clean_text
y = df.iloc[:, 1].values
```

Task 6. Describe the code above and the output results.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.20, random_state = 123)
```

```
my_tfidf = TfidfVectorizer(stop_words='english', max_df=0.7)

# fit the vectorizer and transform X_train into a tf-idf matrix,
# then use the same vectorizer to transform X_test
tfidf_train = my_tfidf.fit_transform(X_train)
tfidf_test = my_tfidf.transform(X_test)

tfidf_train
```

Task 7. Get TF-IDF score for each line of dataframe and put the result into column "TF-IDF score".

Task 8. Describe in your words considered vectorization methods CountVectorizer and TF-IDF.