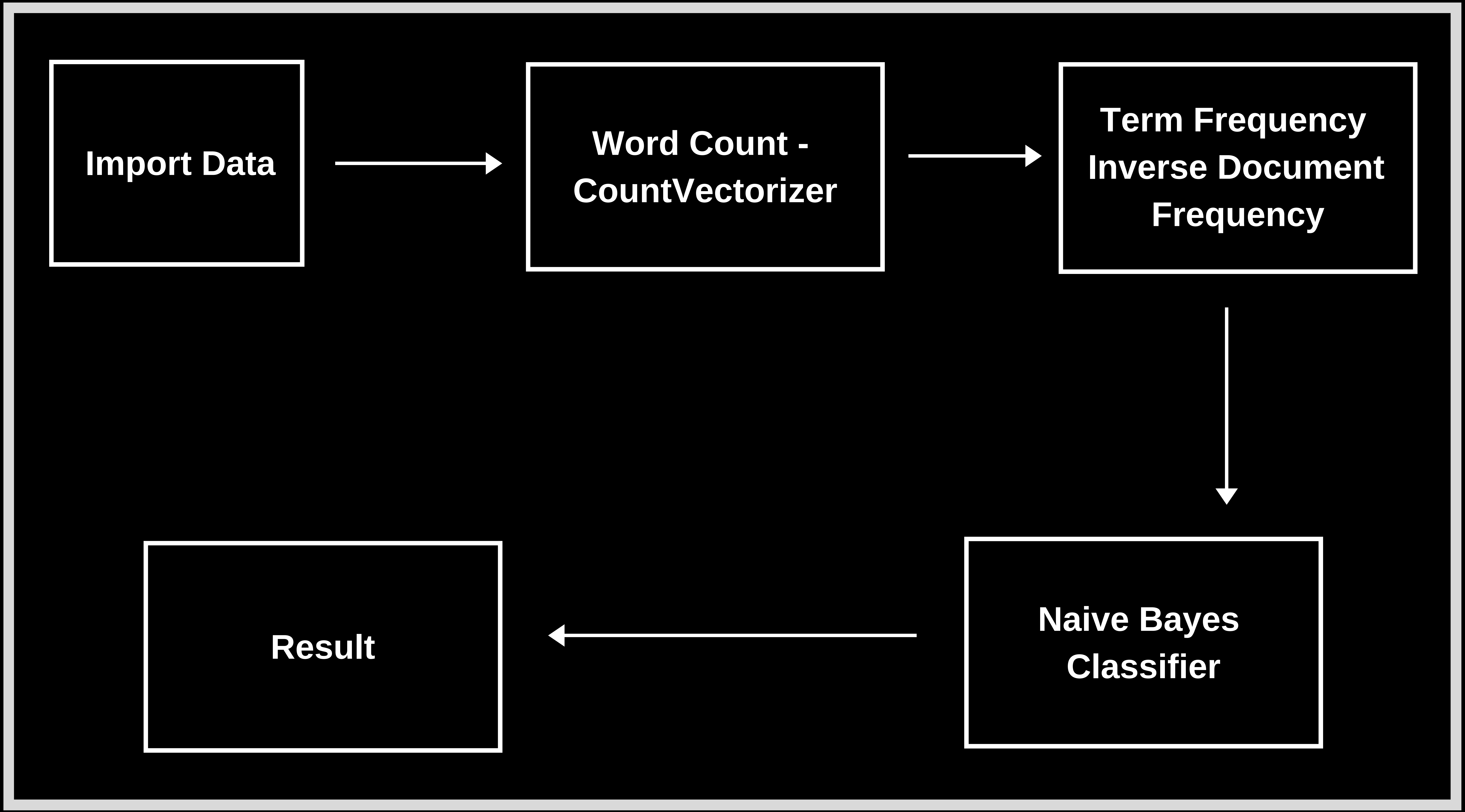
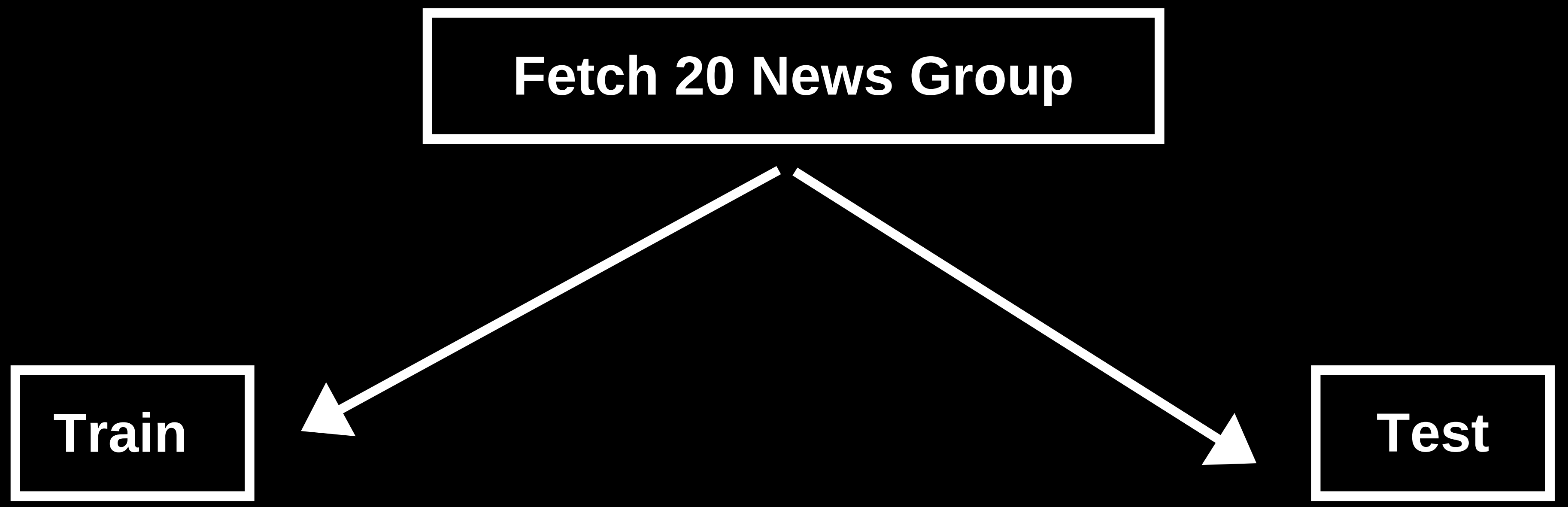
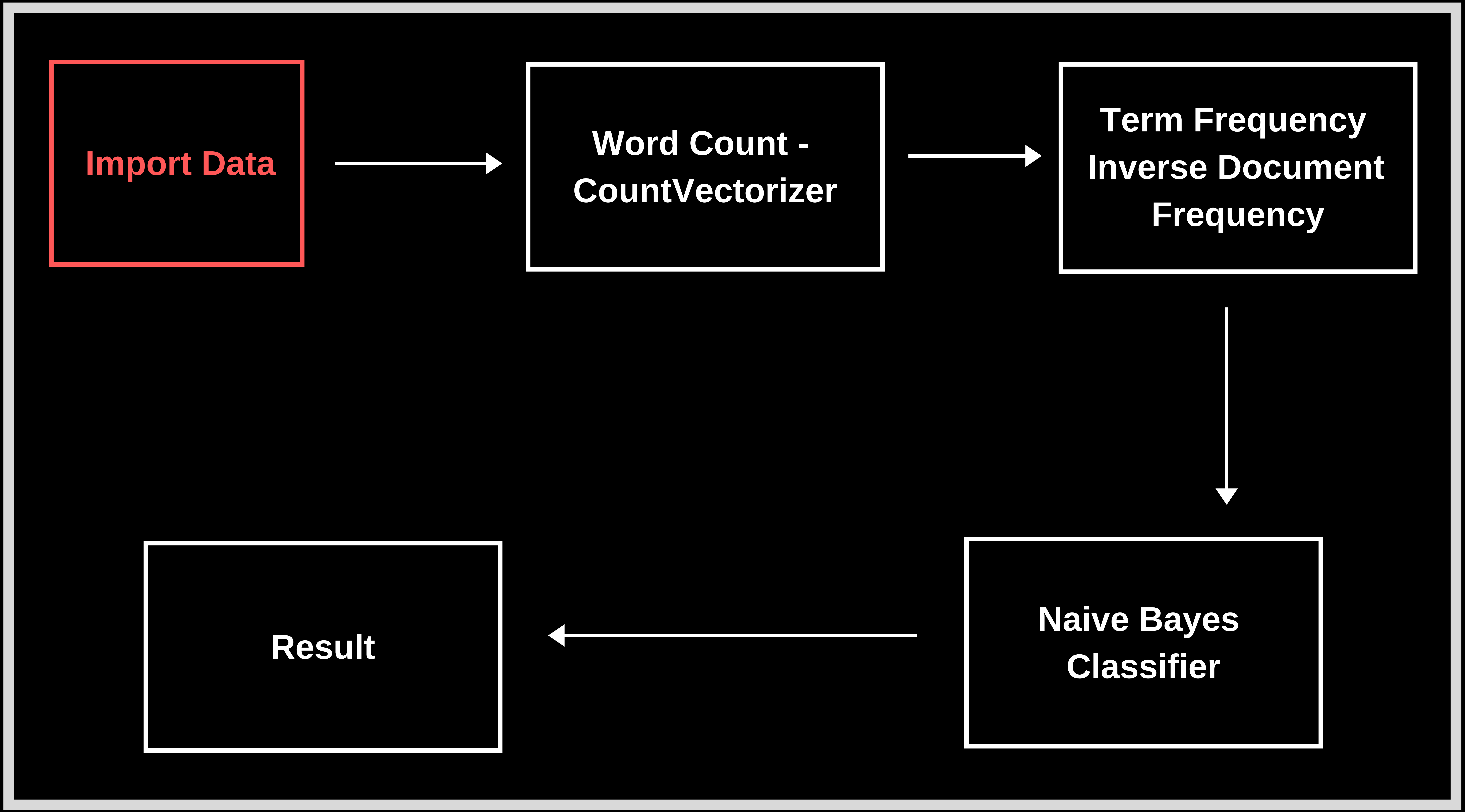

Import Data
Word Count - CountVectorizer
Term Frequency Inverse Document Frequency
Naive Bayes Classifier
Result

Understanding the structure of our data

We are using a 20 Newsgroups dataset




Import Data
Word Count - CountVectorizer
Term Frequency Inverse Document Frequency
Naive Bayes Classifier
Result

Import Data

```python
# Importing dataset directly from Internet

from sklearn.datasets import fetch_20newsgroups

categories = ['alt.atheism', 'soc.religion.christian','comp.graphics', 'sci.med']
news_train = fetch_20newsgroups(subset='train',categories=categories,shuffle=True)
news_test = fetch_20newsgroups(subset='test',categories=categories,shuffle=True)
```

comp.os.ms-windows.misc comp.sys.ibm.pc.hardware comp.sys.mac.hardware comp.windows.x	rec.motorcycles rec.sport.baseball rec.sport.hockey	sci.electronics sci.med sci.space
misc.forsale	talk.politics.misc talk.politics.guns talk.politics.mideast	talk.religion.misc alt.atheism soc.religion.christian

Data

The data available here are in .tar.gz bundles. You will need tar and gunzip to open them. Each subdirectory in the bundle represents a newsgroup; eac newsgroup document that was posted to that newsgroup.

Below are three versions of the data set. The first ("19997") is the original, unmodified version. The second ("bydate") is sorted by date into training(6 posts (duplicates) and does not include newsgroup-identifying headers (Xref, Newsgroups, Path, Followup-To, Date). The third ("18828") does not inc and "Subject" headers.

- 20news-19997.tar.gz - Original 20 Newsgroups data set
- 20news-bydate.tar.gz - 20 Newsgroups sorted by date; duplicates and some headers removed (18846 documents)
- 20news-18828.tar.gz - 20 Newsgroups; duplicates removed, only "From" and "Subject" headers (18828 documents)

I recommend the "bydate" version since cross-experiment comparison is easier (no randomness in train/test set selection), newsgroup-identifying infor because the train and test sets are separated in time.

[7/3/07] I had originally listed the bydate version as containing 18941 documents. I've discovered that the correct count is 18846, of which rainbow sk 18824 documents. However, my rainbow2matlab.py script drops empty and single-word documents, of which there are 50 post-rainbow-processing, s matlab/octave version.

Matlab/Octave

Import Data

Alternate approach

```python
# Importing Dataset offline

import sklearn.datasets as skd

categories = ['alt.atheism', 'soc.religion.christian','comp.graphics', 'sci.med']
news_train = skd.load_files('/home/ayush/Desktop/Fetch20newsgroup/train',
                                categories= categories, encoding= 'ISO-8859-1')

news_test = skd.load_files('/home/ayush/Desktop/Fetch20newsgroup/test/',
                                categories= categories, encoding= 'ISO-8859-1')
```

- **After loading the data, the variable news_train and news_test stores as a dictionary**

In [80]:
```
sample_dict = {'key1':'value1',
               'key2':['see', 'I','contain', 'list'],
               'key3':{'sub_key1':'nested','sub_key2':'dictionary'}
              }
```

In [81]:
```
sample_dict.keys()
```

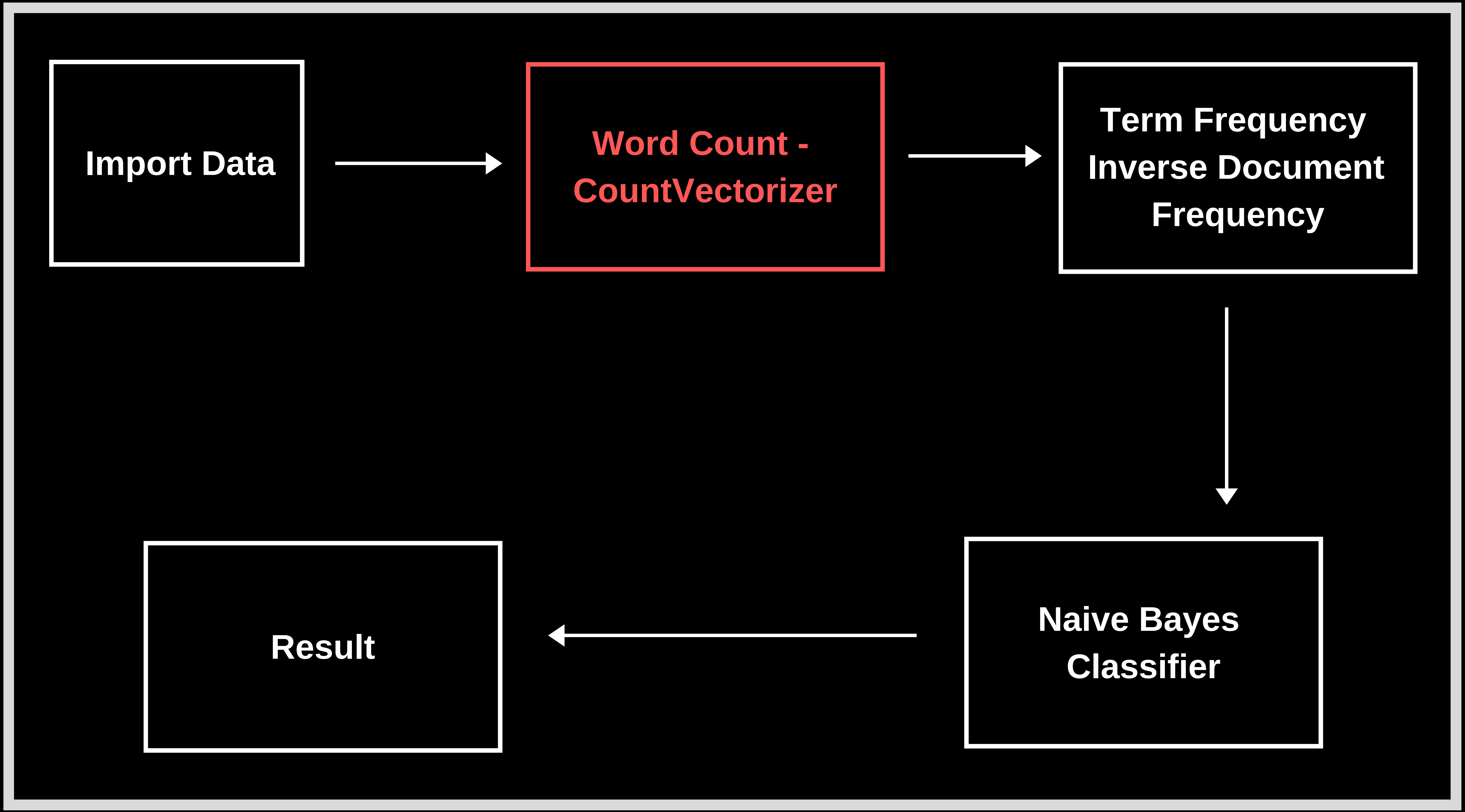
Out[81]:
```
dict_keys(['key1', 'key3', 'key2'])
```

In [82]:
```
sample_dict.values()
```

Out[82]:
```
dict_values(['value1', {'sub_key2': 'dictionary', 'sub_key1': 'nested'}, ['see', 'I', 'conta
in', 'list']])
```

In [83]:
```
sample_dict['key1']
```

Out[83]:
```
'value1'
```



Import Data
Word Count -
CountVectorizer
Term Frequency
Inverse Document
Frequency
Naive Bayes
Classifier
Result

Word Count - CountVectorizer

- **We have a text:** **"The quick brown fox jumped over the lazy dog."**
- **Assign a unique number to each word as: also known as "Tokenize"**

{'the': 7, 'lazy': 4, 'jumped': 3, 'brown': 0, 'over': 5, 'quick': 6, 'dog': 1, 'fox': 2}

- **Features are: (Vocabulary) [8 features]**

['brown', 'dog', 'fox', 'jumped', 'lazy', 'over', 'quick', 'the']

- **In ML terms: Learn a vocabulary dictionary of all tokens in the raw documents, and it is done by using CountVectorizer.fit()**

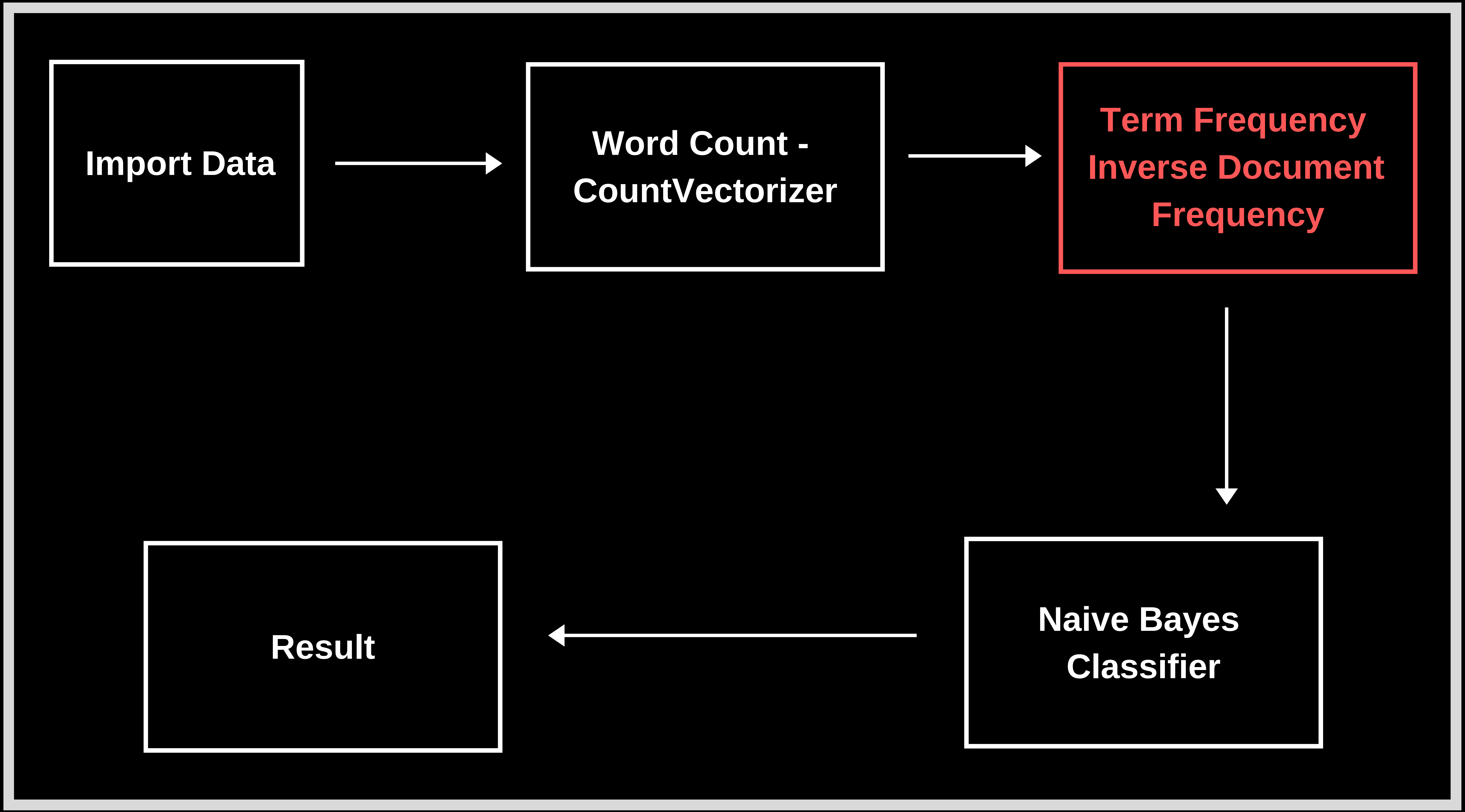
Word Count - CountVectorizer

- We have a text: "The quick brown fox jumped over the lazy dog."
- Count the occurence of each word: basically in ML terms "encoding documents"
- It is done by using CountVectorizer.transform()

```
[[1 1 1 1 1 1 1 2]]
```

```
['brown', 'dog', 'fox', 'jumped', 'lazy', 'over', 'quick', 'the']
```

- It stores it as an array and its shape is: (1,8) i.e 1 no. of sample and 8 no. of features.


Import Data
Word Count -
CountVectorizer
Term Frequency
Inverse Document
Frequency
Naive Bayes
Classifier
Result

Term Frequency (TF)
Inverse Document Frequency (IDF)

Term Frequency: This summarizes how often a given word appears within a document.

Inverse Document Frequency: This downscales words that appear a lot across documents.

Term Frequency (TF)
Inverse Document Frequency (IDF)

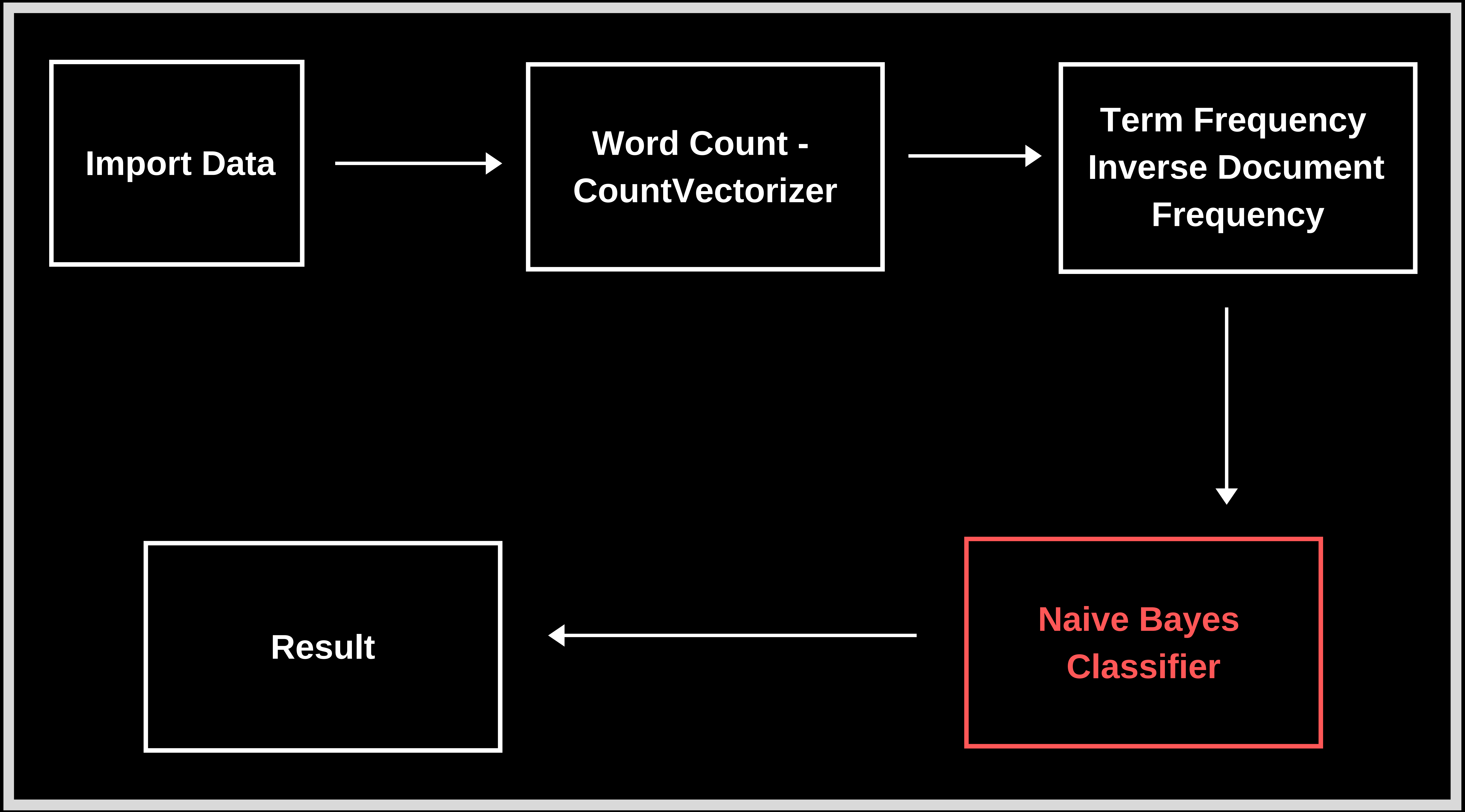
- We have a text: [" The quick brown fox jumped over the lazy dog " ,
  " The dog " ,
  " The fox "]

- Here it learns the IDF from the count matrix obtained from CountVectorizer

- So for above text, the IDF obtained is:

[1.69314718, 1.28768207, 1.28768207, 1.69314718, 1.69314718, 1.69314718, 1.69314718, 1]

- The inverse document frequencies are calculated for each word in the vocabulary, assigning the lowest score of 1.0 to the most frequently observed word: “the” at index 7.

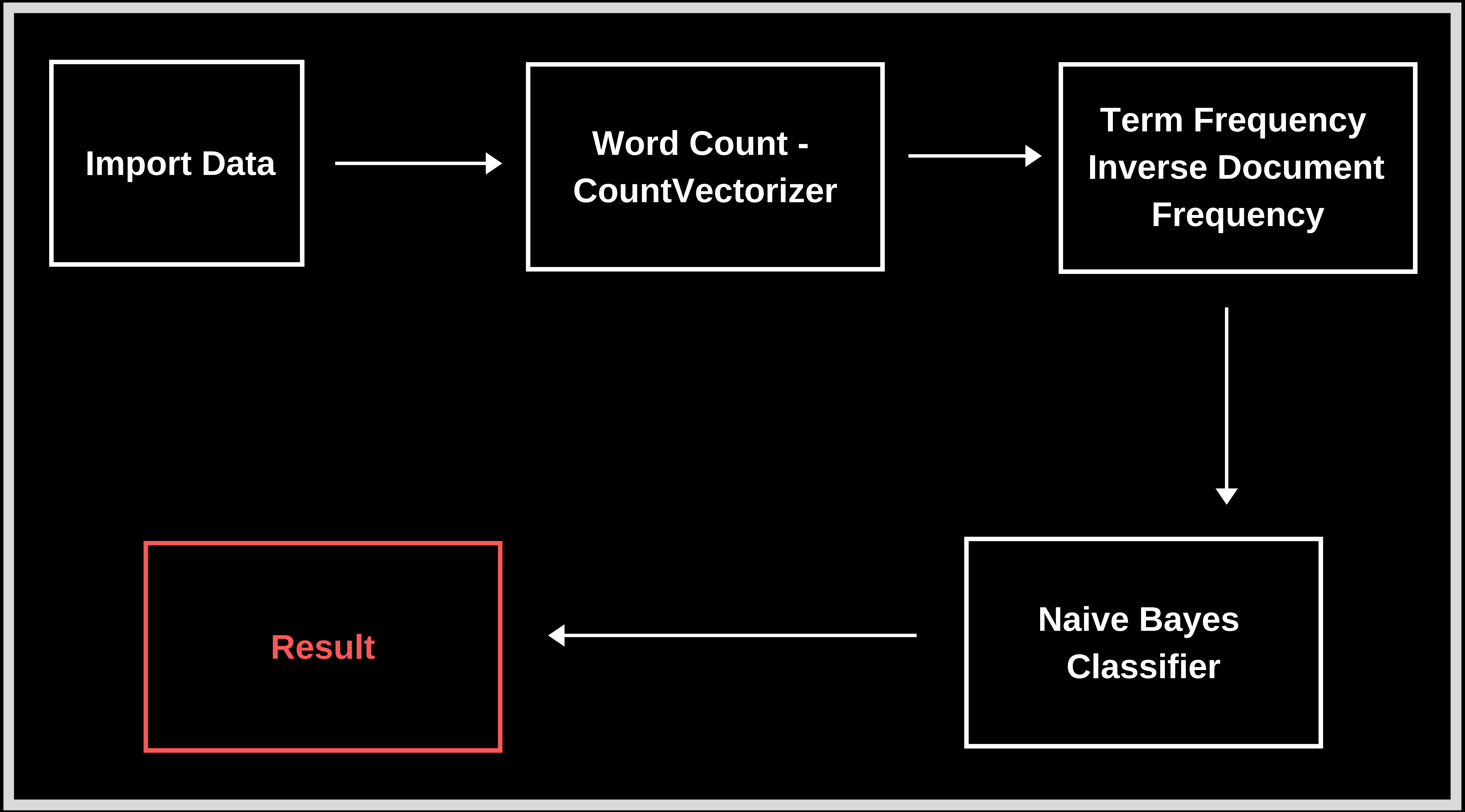

Import Data
Word Count - CountVectorizer
Term Frequency Inverse Document Frequency
Naive Bayes Classifier
Result

MultinomalNB

Naive Bayes Classifier for Multinomial models

- SKLEARN already has inbuilt Multinomial Naive Bayes Classiifer package
- Using this package we can directly train our model with the matrix obtained from Tdidf Transformer

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$


Import Data
Word Count - CountVectorizer
Term Frequency Inverse Document Frequency
Naive Bayes Classifier
Result

Predict the result

- Take your news test data
- Again CountVectorize it
- Use TD IDF transformer
- Then predict it using MultinomialNB()
- Analyse the result: METRICS.Classification_report()
    - Precision
    - Recall
    - F1-score
    - micro-avg
    - macro-avg
    - weighted-avg

```python
from sklearn.metrics import classification_report
y_true = [0, 1, 2, 2, 2]
y_pred = [0, 0, 2, 2, 1]
target_names = ['class 0', 'class 1', 'class 2']
print(classification_report(y_true, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

     class 0       0.50      1.00      0.67         1
     class 1       0.00      0.00      0.00         1
     class 2       1.00      0.67      0.80         3
```

$$\text{PRECISION} = \frac{\text{No. of correct result}}{\text{No. of total returned result}} \Rightarrow \frac{1}{2} \Rightarrow 0.5$$

```python
from sklearn.metrics import classification_report
y_true = [0, 1, 2, 2, 2]
y_pred = [0, 0, 2, 2, 1]
target_names = ['class 0', 'class 1', 'class 2']
print(classification_report(y_true, y_pred, target_names=target_names))
```

```
             precision    recall  f1-score   support

    class 0       0.50      1.00      0.67         1
    class 1       0.00      0.00      0.00         1
    class 2       1.00      0.67      0.80         3
```

RECALL $= \dfrac{\text{No. of correct result}}{\text{No. of correct result that should have been returned}} \Rightarrow \dfrac{1}{1} \Rightarrow 1.0.$

```python
from sklearn.metrics import classification_report
y_true = [0, 1, 2, 2, 2]
y_pred = [0, 0, 2, 2, 1]
target_names = ['class 0', 'class 1', 'class 2']
print(classification_report(y_true, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

     class 0       0.50      1.00      0.67         1
     class 1       0.00      0.00      0.00         1
     class 2       1.00      0.67      0.80         3
```

$$\text{F1 SCORE} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{2 \times 0.5 \times 1}{0.5 + 1} = 0.67$$

```python
from sklearn.metrics import classification_report
y_true = [0, 1, 2, 2, 2]
y_pred = [0, 0, 2, 2, 1]
target_names = ['class 0', 'class 1', 'class 2']
print(classification_report(y_true, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

     class 0       0.50      1.00      0.67         1
     class 1       0.00      0.00      0.00         1
     class 2       1.00      0.67      0.80         3

   micro avg       0.60      0.60      0.60         5
   macro avg       0.50      0.56      0.49         5
weighted avg       0.70      0.60      0.61         5
```

$$\text{micro-avg} = \frac{\text{Total no. of correct result}}{\text{Total no. of returned result}} = 3/5 = 0.6$$

```python
from sklearn.metrics import classification_report
y_true = [0, 1, 2, 2, 2]
y_pred = [0, 0, 2, 2, 1]
target_names = ['class 0', 'class 1', 'class 2']
print(classification_report(y_true, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

     class 0       0.50      1.00      0.67         1
     class 1       0.00      0.00      0.00         1
     class 2       1.00      0.67      0.80         3

   micro avg       0.60      0.60      0.60         5
   macro avg       0.50      0.56      0.49         5
weighted avg       0.70      0.60      0.61         5
```

$$\text{macro-avg} = \frac{\text{Add all precision values}}{\text{Total no. of labels}} = \frac{0.5+0+1.0}{3} = 0.5$$

```python
from sklearn.metrics import classification_report
y_true = [0, 1, 2, 2, 2]
y_pred = [0, 0, 2, 2, 1]
target_names = ['class 0', 'class 1', 'class 2']
print(classification_report(y_true, y_pred, target_names=target_names))
```

```
              precision    recall  f1-score   support

     class 0       0.50      1.00      0.67         1
     class 1       0.00      0.00      0.00         1
     class 2       1.00      0.67      0.80         3

   micro avg       0.60      0.60      0.60         5
   macro avg       0.50      0.56      0.49         5
weighted avg       0.70      0.60      0.61         5
```

$$\text{weighted avg} = \frac{\sum \text{precision of each label} \times \text{support of each label}}{\text{Total no. of support}} = \frac{0.5 \times 1 + 0 \times 1 + 1.0 \times 3}{5} = 0.7.$$

MAGIC

MAGIC → 1

Count Vectorizer

→ Word Count

TDIDF Transformer

→ Frequency

TDIDF Vectorizer

→ DOES BOTH

↳ Word Count

↳ Frequency.

MAGIC-2

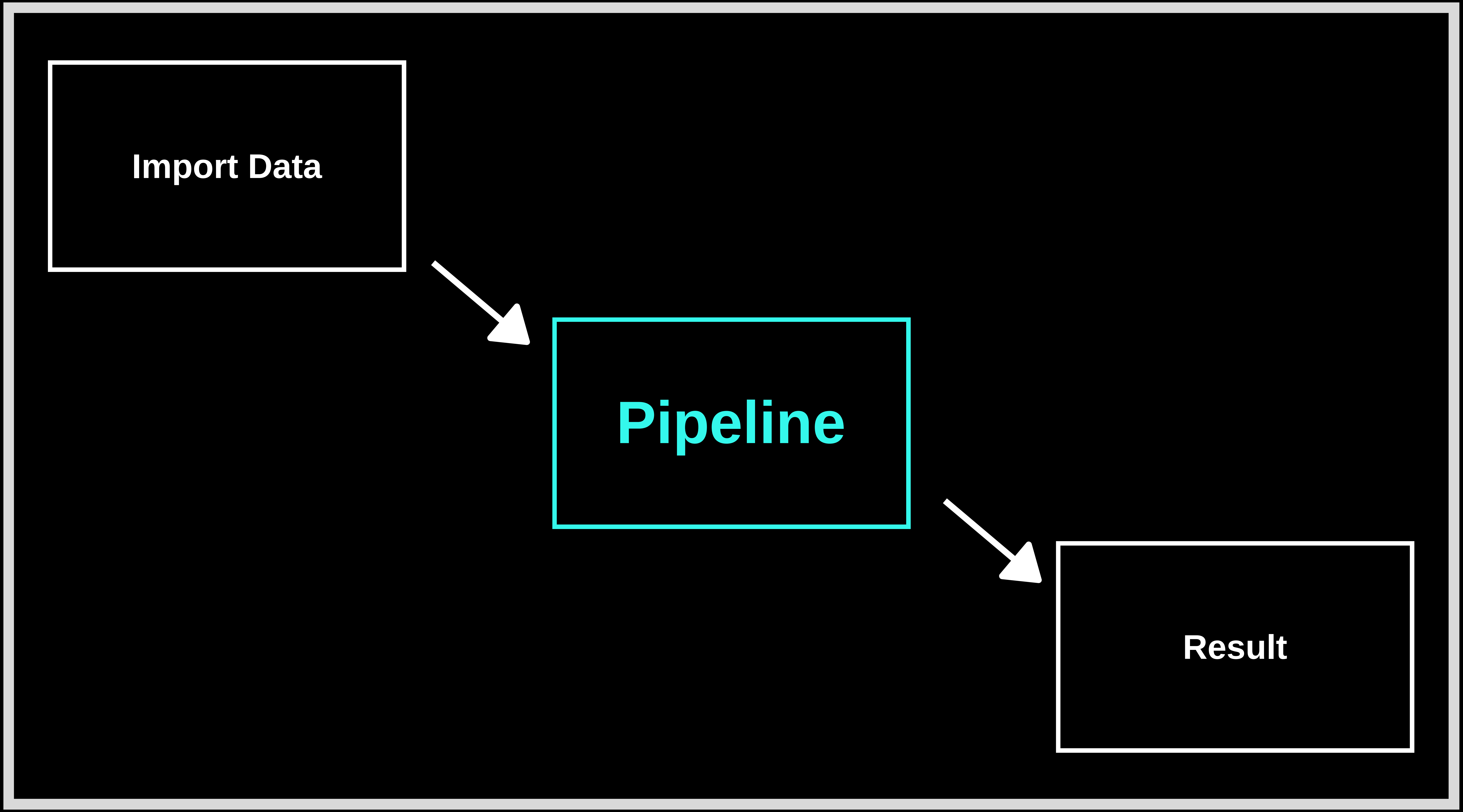
TDIDF Vectorizer

MultiNomial NB

#Pipeline

Object $\underline{a}$ = 'TDIDFVectorizer', 'MultiNomial NB()'

Perform operations with object 'a' of Pipeline


Import Data
Pipeline
Result