UPEC - AEI | 25/04/2023

Big Data Analysis

# Chapter 3 : Applications for Comment Classification

Python Artificial Intelligence Projects for Beginners

**Name of Project:**

Text Classification on social media

**Presented By:**

VISETHJIT Anyanee

AL SID CHIKH Naima

PARAMESWARAN Thuluckshi

**Presented To:**

Mr. KHALFALLAH Malik

# Agenda

X Introduction

X Use case presentation & problem definition

X Data Analytics approach definition and explanation

X Solution development and illustration

X Evaluation and feedback

X Conclusion

# Introduction

- Managing and moderating comments on social media platforms
- Use of automated techniques
- Focus on using Python
- Understanding how comment classification can improve the user experience

python™

# Agenda

| | |
|---|---|
| X | Introduction |
| X | Use case presentation & problem definition |
| X | Data Analytics approach definition and explanation |
| X | Solution development and illustration |
| X | Evaluation and feedback |
| X | Conclusion |

# Use case presentation & problem definition

## Text Classification on social media

### Problem # 1

- massive comments > difficult to moderate and manage the content
- negative environment on the platform.

### Why # 2

- manage the content more effectively
- to promote a safer and more inclusive online environment
- By using automated techniques such as spam detection and sentiment analysis

### Benefit # 3

- improve the efficiency of content moderation
- reduce the workload of moderators
- promote meaningful discussions among users
- more inclusive online environment

# Agenda

| | |
|---|---|
| X | Introduction |
| X | Use case presentation & problem definition |
| X | Data Analytics approach definition and explanation |
| X | Solution development and illustration |
| X | Evaluation and feedback |
| X | Conclusion |

# Data Analytics approach definition and explanation

USING THE APPROACHES CALLED:

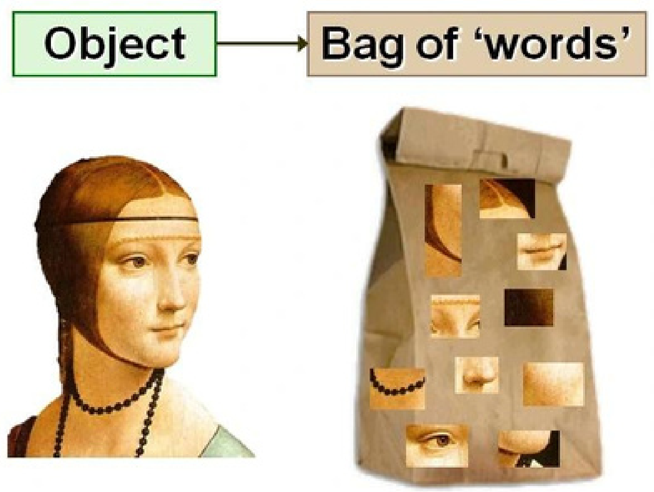
## Bag of words
## TF idf
## Random Forest
## Word2Vec
## Doc2Vec

The different techniques for natural language processing used to analyze and classify text data in various applications, including social media analysis and comment moderation.

# Data Analytics approach definition and explanation

"Machine learning"

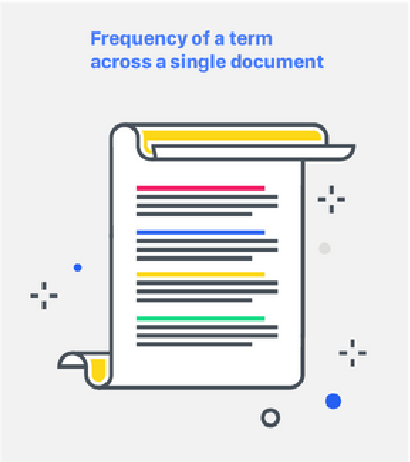
How can we detect the spam comment?




countvectorizer

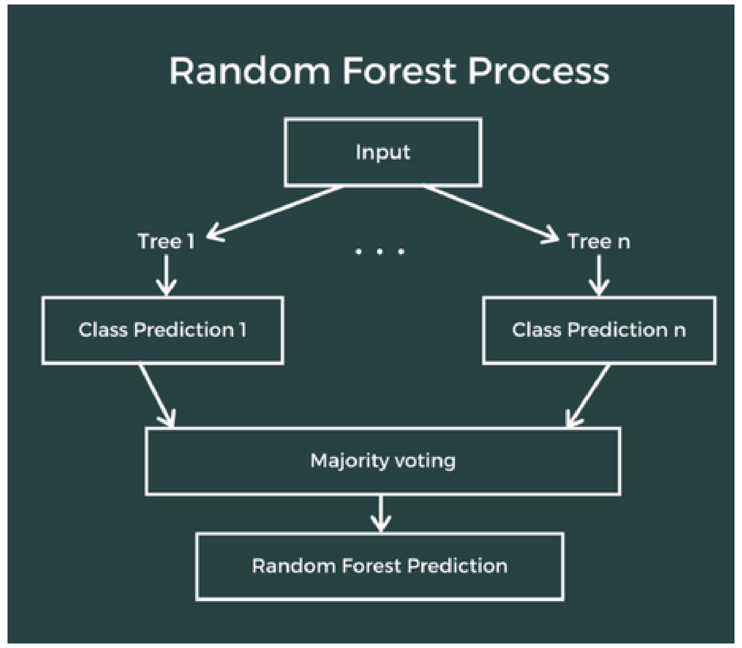
## Bag of words

-> gives us the frequency of words present in a text or document into numbers without regard to the order, structure or use of words,


## TF-IDF

-> to weigh the importance of words in a document or a collection of documents
-> Identify toxic comments



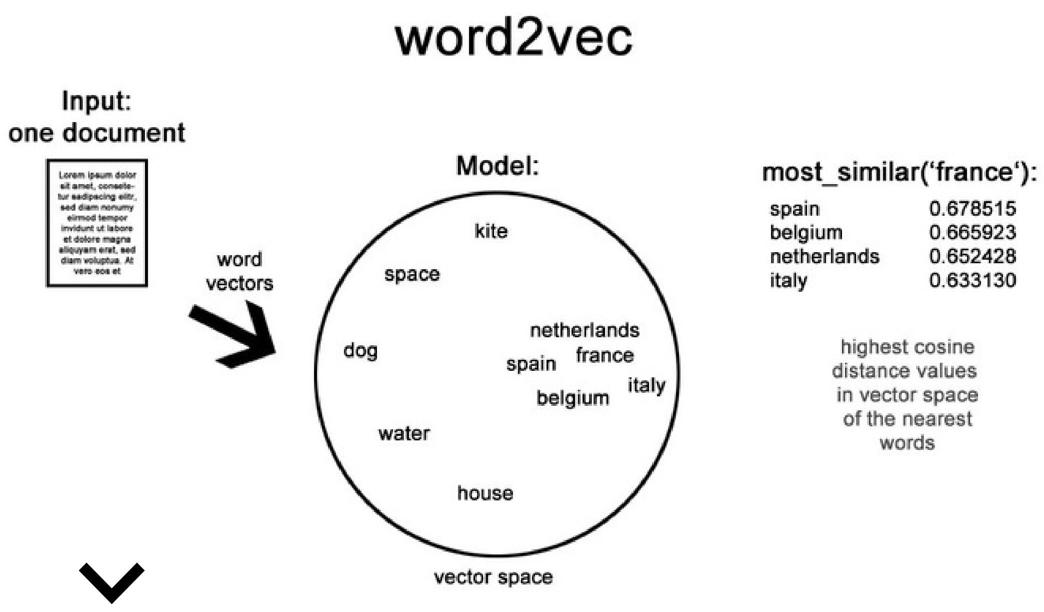

## Random Forest

-> use many decision trees to make predictions

# Data Analytics approach definition and explanation

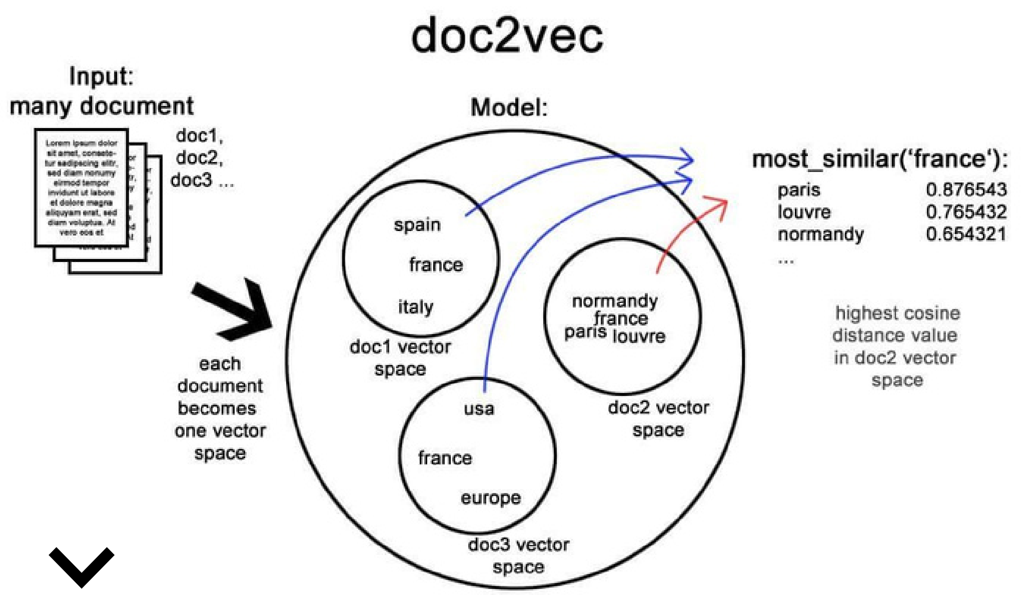
"Machine learning"

**How can we analyze the sentiment of the text?**




## Word2Vec

generates vector representations of words , called word embeddings. These embeddings capture the relationships between words




## Doc2Vec

like "Word2Vec," but it can understand not only individual words, but entire documents of text.

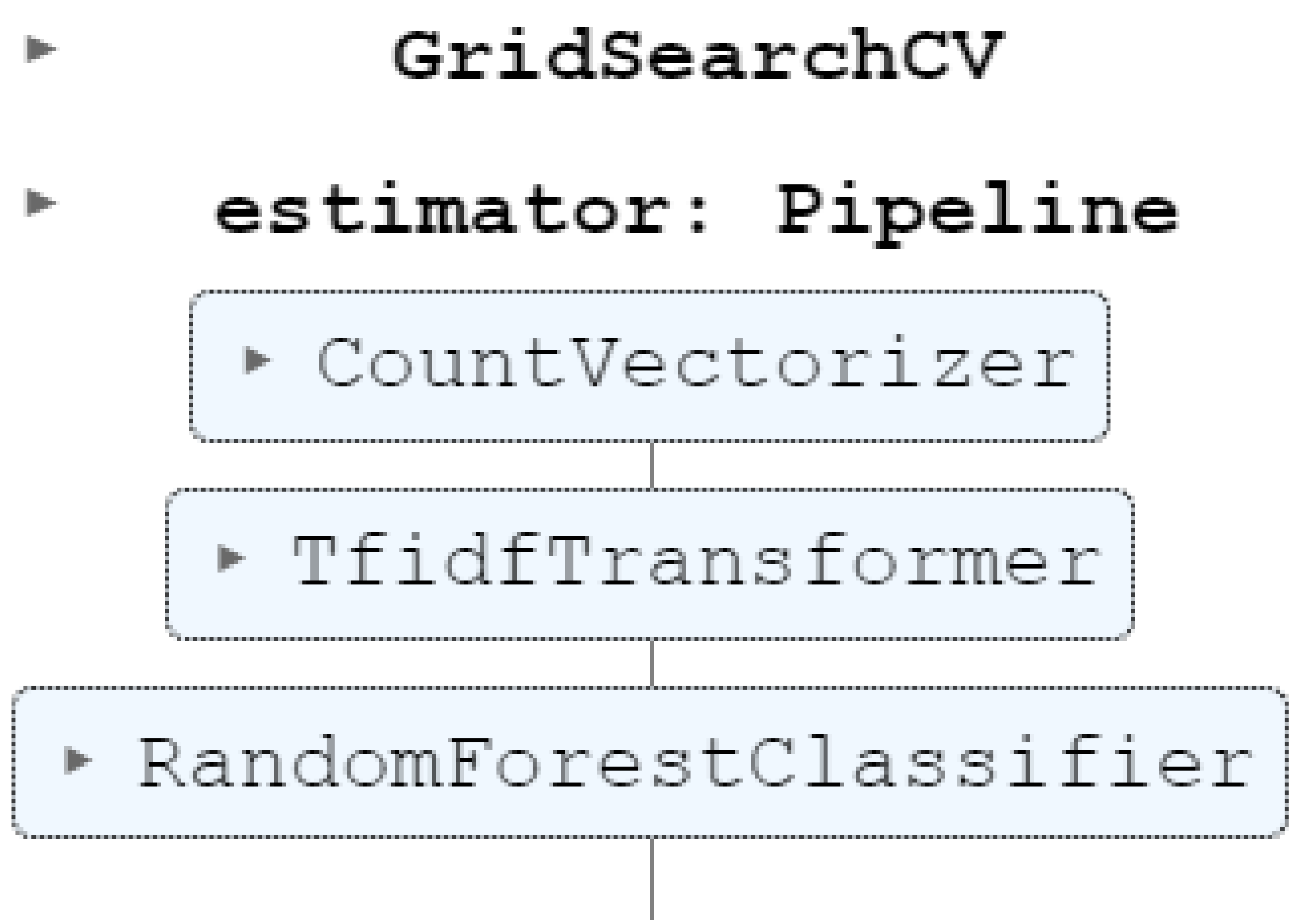
# Agenda

| | |
|---|---|
| X | Introduction |
| X | Use case presentation & problem definition |
| X | Data Analytics approach definition and explanation |
| X | Solution development and illustration |
| X | Evaluation and feedback |
| X | Conclusion |

# Solution development and illustration

**Dectecting spam with machine learning**

| Step #1 | Step #2 | Step #3 | Step #4 |
|---|---|---|---|
| select and prepare a training data set | select an algorithm to run on the training data set | train the algorithm | test and validate |



# Countvectorizer

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
```
[8] ✓ 3.8s

```
dvec = vectorizer.fit_transform(d['CONTENT'])
```
[15] ✓ 0.0s

```
dvec
```
[10] ✓ 0.0s

```
<350x1418 sparse matrix of type '<class 'numpy.int64'>'
	with 4354 stored elements in Compressed Sparse Row format>
```

Result : 350 rows (comments) and 1418 columns or features ( words)

# Analyzing comment

```
analyze = vectorizer.build_analyzer()
[14] ✓ 0.0s

print(d['CONTENT'][349])
analyze(d['CONTENT'][349])
[12] ✓ 0.0s

... The first billion viewed this because they thought it was really cool, the  other billion and a half came to see how stupid the first billion were.

['the',
 'first',
 'billion',
 'viewed',
 'this',
 'because',
 'they',
 'thought',
 'it',
 'was',
 'really',
 'cool',
 'the',
 'other',
 'billion',
 'and',
 'half',
 'came',
```

Analyzing a single comment:

“The first billion viewed this because they thought it was really cool... “

# Data train, Data test process:

D-train, D-test process: evaluate the accuracy of our algorithm

```
d_train = dshuf[:300]
d_test = dshuf[300:]
d_train_att = vectorizer.fit_transform(d_train['CONTENT']) # fit bag-of-words on training set
d_test_att = vectorizer.transform(d_test['CONTENT']) # reuse on testing set
d_train_label = d_train['CLASS']
d_test_label = d_test['CLASS']
```
[21] ✓ 0.0s

```
d_train_att
```
[22] ✓ 0.0s

```
<300x1269 sparse matrix of type '<class 'numpy.int64'>'
    with 3625 stored elements in Compressed Sparse Row format>
```

```
d_test_att
```
[23] ✓ 0.0s

```
<50x1269 sparse matrix of type '<class 'numpy.int64'>'
    with 576 stored elements in Compressed Sparse Row format>
```

# RandomForest Method

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=80)
```
[24] ✓ 1.4s

+ Code + Markdown

```
#train the model
clf.fit(d_train_att, d_train_label)
```
[ ]

```
#Check how well it performs
clf.score(d_test_att, d_test_label)
```
[26] ✓ 0.0s

```
0.94
```

```
from sklearn.metrics import confusion_matrix
pred_labels = clf.predict(d_test_att)
confusion_matrix(d_test_label, pred_labels)
```
[27] ✓ 0.0s

```
array([[20,  1],
       [ 2, 27]], dtype=int64)
```

80 decision trees

3 steps :

- split data randomly,
- make training on decision trees
- finally give a result.

confusion matrix

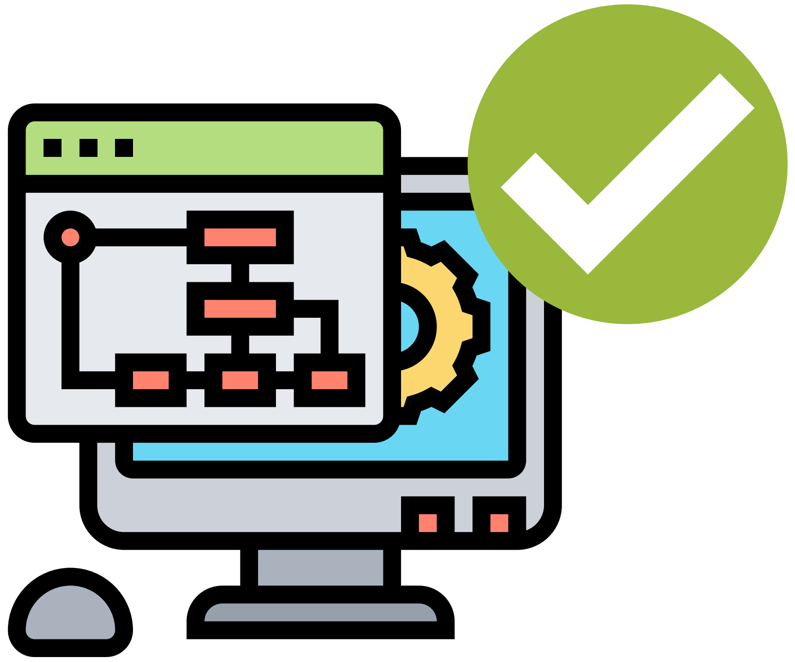
# Crossvalidation

```python
from sklearn.model_selection import cross_val_score
scores = cross_val_score(clf, d_train_att, d_train_label, cv=5)
# show average score and +/- two standard deviations away (covering 95% of scores)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

[28] ✓ 1.9s

```
Accuracy: 0.96 (+/- 0.02)
```

+ Code + Markdown



# Testing with the 5 videos

```
# load all datasets and combine them
d = pd.concat([pd.read_csv("Youtube01-Psy.csv"),
               pd.read_csv("Youtube02-KatyPerry.csv"),
               pd.read_csv("Youtube03-LMFAO.csv"),
               pd.read_csv("Youtube04-Eminem.csv"),
               pd.read_csv("Youtube05-Shakira.csv")])
```
✓ 0.2s

```
len(d)
```
✓ 0.1s

Total number of comments with the 5 videos

```
1956
```

```
len(d.query('CLASS == 1'))
```
✓ 0.1s

Spams

```
1005
```

```
len(d.query('CLASS == 0'))
```
✓ 0.0s

Usefull comments

```
951
```

# Pipeline setting

-> Pipeline is a feature of the Scikit-learn that bring together steps.

-> Pipeline contains :

- The Countvectoriser
- The Random Forest

Pipeline stetting => analyze from the the first shuffled 1500 comments

```
# set up a pipeline
from sklearn.pipeline import Pipeline, make_pipeline
pipeline = Pipeline([
    ('bag-of-words', CountVectorizer()),
    ('random forest', RandomForestClassifier()),
])
pipeline
```

```
# or: pipeline = make_pipeline(CountVectorizer(), RandomForestClassifier())
make_pipeline(CountVectorizer(), RandomForestClassifier())
```

```
pipeline.fit(d_content[:1500],d_label[:1500])
```

```
pipeline.score(d_content[1500:], d_label[1500:])
```

```
0.9605263157894737
```

# Test of the Pipeline setting : pipeline predict

2 comments as example :

"What a neat video!"
 result : 0

"Please subscribe to my channel"
result : 1

```
pipeline.predict(["what a neat video!"])
[40] ✓ 0.1s
array([0], dtype=int64)

pipeline.predict(["please subscribe to my channel"])
[53] ✓ 0.3s
array([1], dtype=int64)

scores = cross_val_score(pipeline, d_content, d_label, cv=5)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
[44] ✓ 12.1s
Accuracy: 0.96 (+/- 0.02)
```

# Adding a second pipeline with TF-IDF

For more precise result

Proposal of 5 parameters

```
# parameter search
parameters = {
    'countvectorizer__max_features': (None, 1000, 2000),
    'countvectorizer__ngram_range': ((1, 1), (1, 2)),  # unigrams or bigrams
    'countvectorizer__stop_words': ('english', None),
    'tfidftransformer__use_idf': (True, False), # effectively turn on/off tfidf
    'randomforestclassifier__n_estimators': (20, 50, 100)
}
from sklearn.model_selection import GridSearchCV
grid_search = GridSearchCV(pipeline2, parameters, n_jobs=-1, verbose=1)
```

# Final result

```
print("Best score: %0.3f" % grid_search.best_score_)
print("Best parameters set:")
best_parameters = grid_search.best_estimator_.get_params()
for param_name in sorted(parameters.keys()):
    print("\t%s: %r" % (param_name, best_parameters[param_name]))
```

[48]

```
Best score: 0.960
Best parameters set:
	countvectorizer__max_features: 2000
	countvectorizer__ngram_range: (1, 1)
	countvectorizer__stop_words: 'english'
	randomforestclassifier__n_estimators: 100
	tfidftransformer__use_idf: False
```

# Sentiment Analysis

### 1st Step

Doc2Vec

### 2nd Step

K-nearest neighbors
RandomForest

### 3rd Step

Compare to other techniques

## Doc2Vec

# Training of Word2Vec and Doc2Vec model

```
from gensim.models.doc2vec import TaggedDocument
from gensim.models import Doc2Vec
```

✓ 0.0s

```
def extract_words(sent):
    sent = sent.lower()
    sent = re.sub(r'<[^>]+>', ' ', sent) # strip html tags
    sent = re.sub(r'(\w)\'(\w)', '\1\2', sent) # remove apostrophes
    sent = re.sub(r'\W', ' ', sent) # remove punctuation
    sent = re.sub(r'\s+', ' ', sent) # remove repeated spaces
    sent = sent.strip()
    return sent.split()
```

✓ 0.0s

-> Gensim library

-> TaggedDocument: to make the computer understand better

-> Extract_words function: to separate unnecessary words from the whole words.

# Training of our training set

-> As we need more data set

-> 175,000 examples for this training

```
for fname in sorted(os.listdir(dirname)):
    if fname[-4:] == '.txt':
        with open(dirname + "/" + fname, encoding='UTF-8') as f:
            for i, sent in enumerate(f):
                words = extract_words(sent)
                unsup_sentences.append(TaggedDocument(words, ["%s/%s-%d" % (dirname, fname, i)]))

# source: https://nlp.stanford.edu/sentiment/, data from Rotten Tomatoes
with open("stanfordSentimentTreebank/original_rt_snippets.txt", encoding='UTF-8') as f:
    for i, line in enumerate(f):
        words = extract_words(sent)
        unsup_sentences.append(TaggedDocument(words, ["rt-%d" % i]))
```

✓ 16m 26.4s

```
len(unsup_sentences)
```

✓ 0.1s

175325

# Similarity measurments

```
model.infer_vector(extract_words("This place is not worth your time, let alone Vegas."))
```

✓ 0.0s

```
array([ 0.15839541,  0.02213576,  0.26376584, -0.48066616, -0.03137077,
       -0.25015157,  0.26324415, -0.13032252,  0.01211821,  0.24326684,
       -0.16611272, -0.17989054, -0.33873653, -0.07376021,  0.62292314,
        0.35614946, -0.16246454,  0.08911735,  0.01923273,  0.33431724,
        0.44906926,  0.04037865,  0.06475811,  0.2848666 ,  0.32015496,
        0.49359483, -0.31922275, -0.01975652,  0.5144004 , -0.14519018,
       -0.20464548,  0.09204558,  0.28979242,  0.11151716, -0.26435006,
        0.09226024,  0.5630968 ,  0.11117744,  0.6047081 ,  0.09140925,
       -0.07361961,  0.1250164 , -0.3000361 , -0.15348944, -0.08545798,
       -0.44060743, -0.05337591, -0.09558962,  0.1370779 , -0.0928385 ],
      dtype=float32)
```

Result: 50 dimensional vector

```
from sklearn.metrics.pairwise import cosine_similarity
cosine_similarity(
    [model.infer_vector(extract_words("This place is not worth your time, let alone Vegas."))],
    [model.infer_vector(extract_words("Service sucks."))])
```

✓ 0.6s

```
array([[0.39466345]], dtype=float32)
```

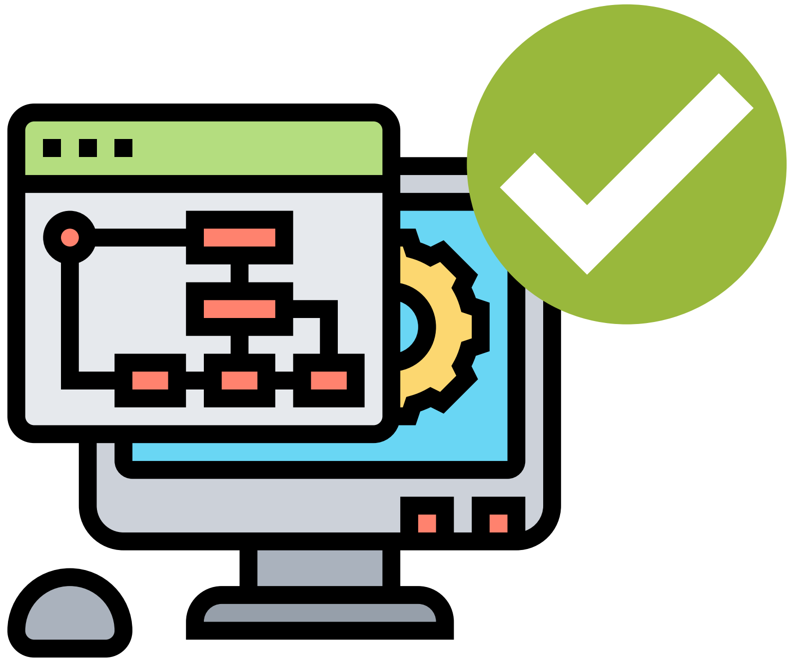
score : similarity 39%

# Similarity measurments

```
cosine_similarity(
    [model.infer_vector(extract_words("Highly recommended."))],
    [model.infer_vector(extract_words("Service sucks."))])
✓ 0.0s

array([[0.21989854]], dtype=float32)
```

score : similarity 21%



# K-nearest neighbors

K-NN will look for the K closest data points to this new data point, and if the majority of them are classified as "positive," then the new data point will also be classified as "positive."

```python
sentences = []
sentvecs = []
sentiments = []
for fname in ["yelp", "amazon_cells", "imdb"]:
    with open("sentiment labelled sentences/%s_labelled.txt" % fname, encoding='UTF-8') as f:
        for i, line in enumerate(f):
            line_split = line.strip().split('\t')
            sentences.append(line_split[0])
            words = extract_words(line_split[0])
            sentvecs.append(model.infer_vector(words, epochs =10)) # create a vector for this document
            sentiments.append(int(line_split[1]))

# shuffle sentences, sentvecs, sentiments together
combined = list(zip(sentences, sentvecs, sentiments))
random.shuffle(combined)
sentences, sentvecs, sentiments = zip(*combined)
```
✓ 3.1s

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
import numpy as np

clf = KNeighborsClassifier(n_neighbors=9)
clfrf = RandomForestClassifier()
```
✓ 0.4s

-> compare with the random forest to compare the performance

# K-nearest neighbors

```
scores = cross_val_score(clf, sentvecs, sentiments, cv=5)
np.mean(scores), np.std(scores)
```

✓ 1.2s

```
(0.7886666666666666, 0.018147543451754924)
```

```
scores = cross_val_score(clfrf, sentvecs, sentiments, cv=5)
np.mean(scores), np.std(scores)
```

✓ 14.5s

```
(0.7939999999999999, 0.010413666234542216)
```

-> to know the accuracy of the model

- K-nearest neigbors got 78% within 1.2s
- RandomForest got 79% within 15s

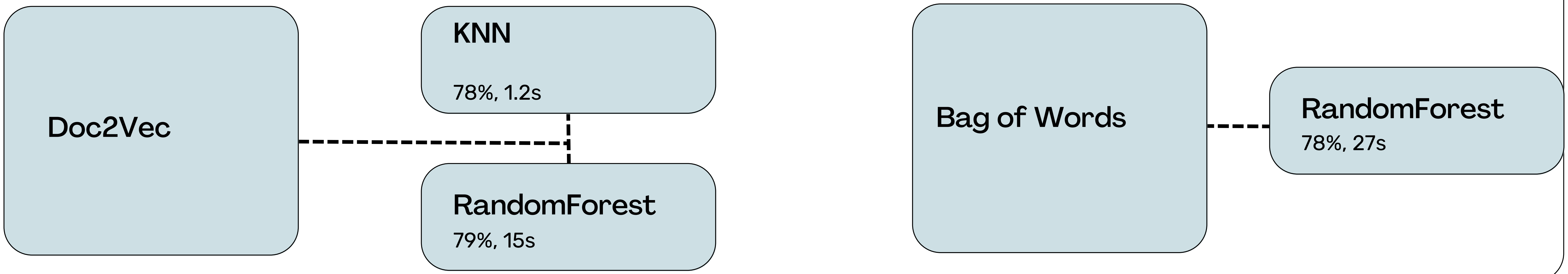
# Bag of words Comparison

Result from Bag of Words:
78% within 27s

```python
# bag-of-words comparison
from sklearn.pipeline import make_pipeline
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
pipeline = make_pipeline(CountVectorizer(), TfidfTransformer(), RandomForestClassifi
```
✓ 0.1s

```python
scores = cross_val_score(pipeline, sentences, sentiments, cv=5)
np.mean(scores), np.std(scores)
```
✓ 27.4s

```
(0.7843333333333333, 0.00806914562460681)
```



# Agenda

X Introduction

X Use case presentation & problem definition

X Data Analytics approach definition and explanation

X Solution development and illustration

X Evaluation and feedback

X Conclusion

# Evaluation and Feedback

## Spam Detector

identifying spam comments on YouTube

But needs to be complemented with : other techniques + manual moderation

## Sentiment Analysis

+ Doc2Vec

-> to understand the overall sentiment of a comment
-> classify the comment accordingly

**What it does not take :**

-> nuances and complexities of human language ( sarcasm, irony)

**What has to be considered :**

- quality & size of the traning datasets
- short sentences

# Agenda

| | |
|---|---|
| X | Introduction |
| X | Use case presentation & problem definition |
| X | Data Analytics approach definition and explanation |
| X | Solution development and illustration |
| X | Evaluation and feedback |
| X | Conclusion |

# Conclusion

## "Machine learning for Text Classification"

- Reduce time consuming
- Remove  irrelevant content
- Improve Business performance

# Appendix: software and datasets

"Machine learning for Text Classification"

- Our software

python

Visual Studio Code

GENSIM topic modelling for humans

scikits learn

pandas

- Our Datasets :

aclImdb
review_polarity
sentiment labelled sentences
stanfordSentimentTreebank
reviews.d2v
SentimentAnalysisnew.ipynb
Spam detector copy.ipynb
Spam detector.ipynb
Youtube01-Psy.csv
Youtube02-KatyPerry.csv
Youtube03-LMFAO.csv
Youtube04-Eminem.csv
Youtube05-Shakira.csv