

Report Final Machine Learning

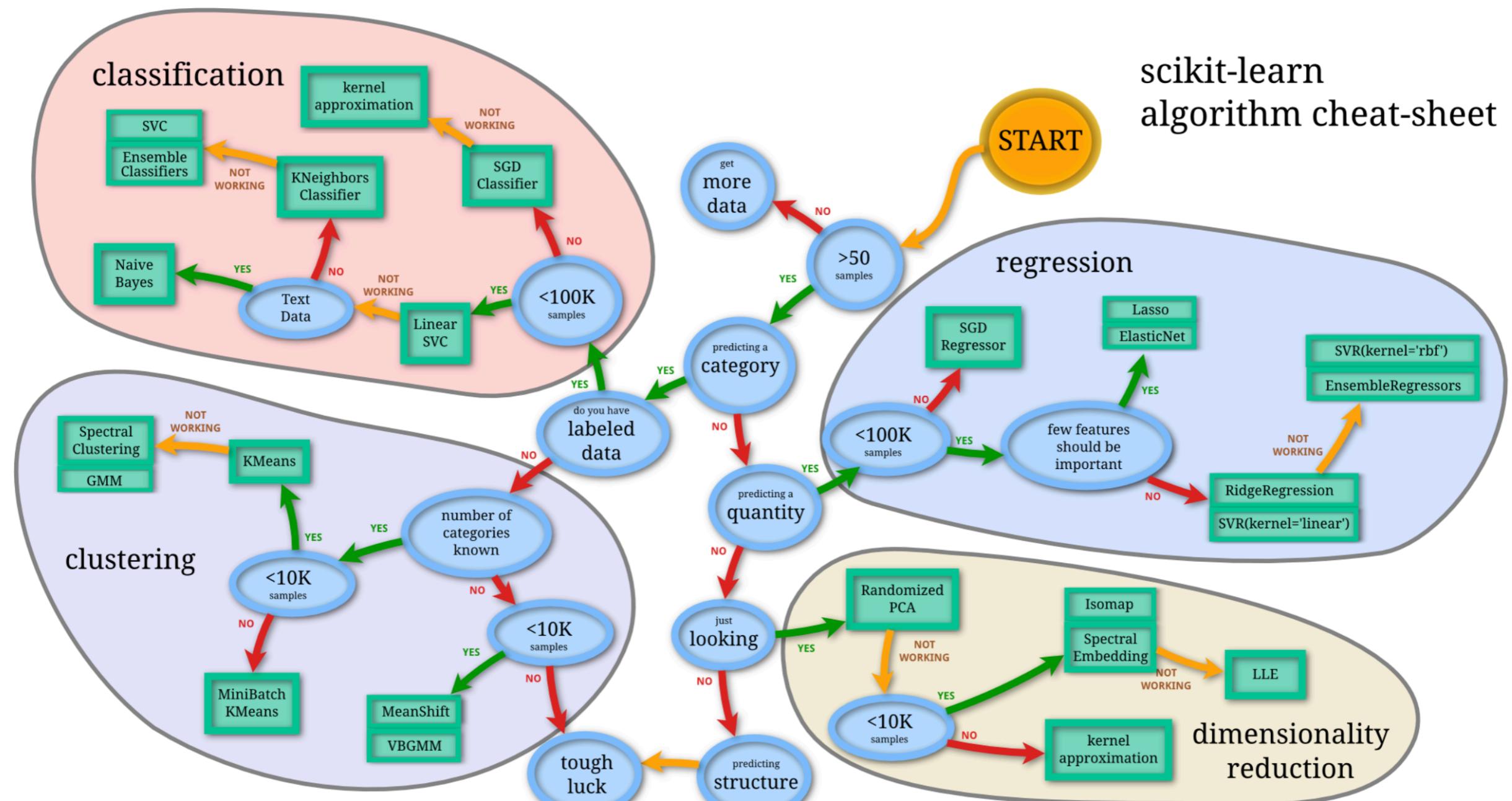
TEXT CLASSIFICATION

Người hướng dẫn: PGS TS Lê Anh Cường

Người thực hiện: Trần Ngọc Bảo Duy - 51702091

Phạm Anh Duy - 51702088

scikit-learn algorithm cheat-sheet



1. Bag of word

Khái niệm

- Bag of Words là một thuật toán hỗ trợ xử lý ngôn ngữ tự nhiên và mục đích của BoW là phân loại text hay văn bản.
- Bag-of-word sẽ gồm một bộ từ vựng từ tất cả các văn bản , rồi model các văn bản bằng cách đếm số lần xuất hiện của mỗi từ trong văn bản đó .
- Ví dụ:
 - Ông mặt trời có cái mặt thật to
 - Bầu trời có mặt trời

=> Từ hai câu trên ta sẽ có một bộ từ vựng như sau

[“ông”, “mặt”, “trời”, “có”, “cái”, “thật”, “to”, “bầu”]

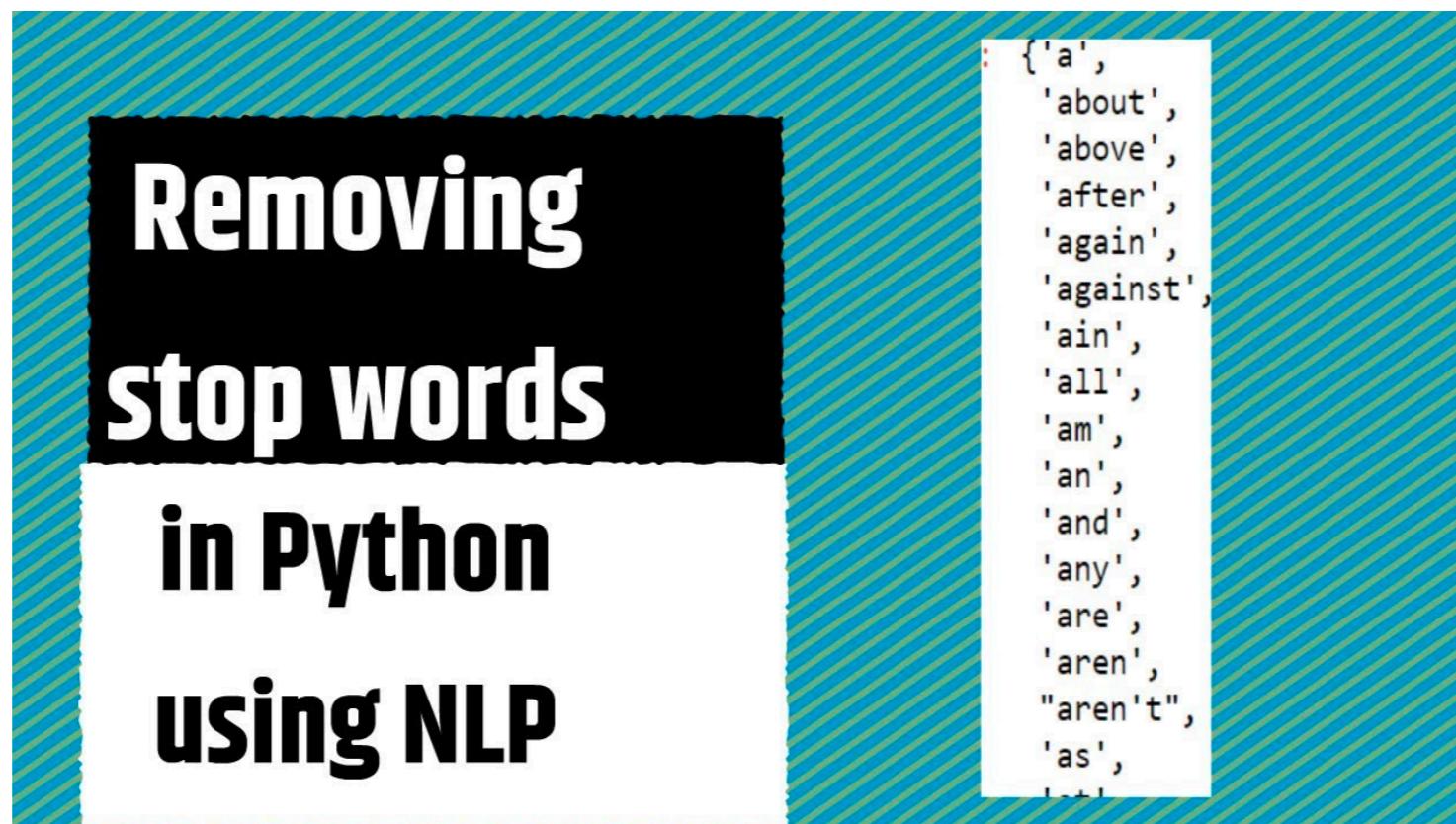
=> câu 1: [1, 2, 1, 1, 1, 1, 1, 0]

câu 2: [0, 1, 2, 1, 0, 0, 0, 1]

2. Stop word

Khái niệm

- Stop word: trong ngôn ngữ, thường sẽ có những từ loại dùng để biểu cảm, cảm thán, không có ý nghĩa trong văn viết mà thường dùng để làm động từ, bổ ngữ cho câu nhưng lại không có ý nghĩa. Vì vậy nó làm cho đặc trưng của câu bị loãng có thể gây mất đặc trưng, có các từ như "là", "của", "cứ", ...
- Chính vì vậy nếu chỉ xét theo tần số xuất hiện của từng từ thì việc phân loại văn bản rất có thể cho kết quả sai dẫn tỷ lệ chính xác sẽ thấp.



3. TD-IDF

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Term x within document y

$tf_{x,y}$ = frequency of x in y

df_x = number of documents containing x

N = total number of documents

3. TD-IDF

Khái niệm

- TF(Term Frequency) là tần số xuất hiện của 1 từ trong 1 văn bản có cách tính như sau:

$$tf = \frac{f_{t,d}}{\sum_t^d f_{t,d}}$$

$f_{t,d}$: số lần xuất hiện từ t trong văn bản d .

$\sum_t^d f_{t,d}$: mẫu số là tổng số từ trong văn bản d

- IDF (Inverse Document Frequency): Tần số nghịch của 1 từ trong tập văn bản (corpus).

$$idf = \log \frac{|D|}{|d \in D : for(t \in d)|}$$

Tử số là tổng số từ trong văn bản d

Mẫu số là số văn bản có chứa từ t

=> Nếu từ đó không xuất hiện ở bất cứ 1 văn bản nào trong tập thì mẫu số sẽ bằng 0 => phép chia cho không hợp lệ, vì thế với trường hợp này thường cộng thêm 1 vào mẫu số, để cho hợp lệ.

3. TD-IDF

$$tfidf(t, d, D) = tf_{(t,d)} * idf_{(t,D)}$$

- Những từ có giá trị TF-IDF cao là những từ xuất hiện nhiều trong văn bản này, và xuất hiện ít trong các văn bản khác. Việc này giúp lọc ra những từ phổ biến và giữ lại những từ có giá trị cao (từ khoá của văn bản đó).

	"Trường	000	1	10	120	150	18	19/4	200	...	đầu năm	đảng	đề	đều	để	đồng	đồng",	đợt	ưu	ở	
0	11	1	4	1	2	1	1	1	1	...	1	1	1	7	1	1	3	1	2	1	1
1	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0	0

	"Trường	000	1	10	120	150	18	19/4	200	...	đầu năm	đảng
0	0.157611	0.016162	0.064648	0.016162	0.032324	0.016162	0.016162	0.016162	0.016162	...	0.016162	0.016162
1	0.773729	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000

```
maxA = max(tfidf_bowA.values())
maxB = max(tfidf_bowB.values())
a = max(dict(tfidf_bowA).items(),key = lambda x: x[1])
b = max(dict(tfidf_bowB).items(),key = lambda x: x[1])
# dict(tfidf_bowA.items())
print("Max a",a)
print("Max b",b)
```

```
Max a ('thi', 0.21010639834903902)
Max b ('thách', 0.8727496546806236)
```

3. TD-IDF

- Có thể sử dụng thư viện sklearn để tính td-idf

```
from sklearn.feature_extraction.text import TfidfTransformer  
from sklearn.feature_extraction.text import CountVectorizer
```

dùng CountVectorizer để đếm số lượng từ trong câu

dùng TfidfTransformer để tính td và idf => td-idf

```
tfidf_transformer=TfidfTransformer(smooth_idf=True,use_idf=True)  
tfidf_transformer.fit(word_count_vector)  
  
# print idf values  
df_idf = pd.DataFrame(tfidf_transformer.idf_, index=cv.get_feature_names(),  
  
# count matrix  
count_vector=cv.transform(docs)  
  
# tf-idf scores  
tf_idf_vector=tfidf_transformer.transform(count_vector)
```

Hoặc đơn giản hơn chúng ta sử dụng Tfidfvectorizer để chuyển sang td-idf trong 1 bước

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

```
tfidf_vectorizer=TfidfVectorizer(use_idf=True)  
  
# just send in all your docs here  
fitted_vectorizer=tfidf_vectorizer.fit(docs)  
tfidf_vectorizer_vectors=fitted_vectorizer.transform(docs)
```

4. Feature extraction

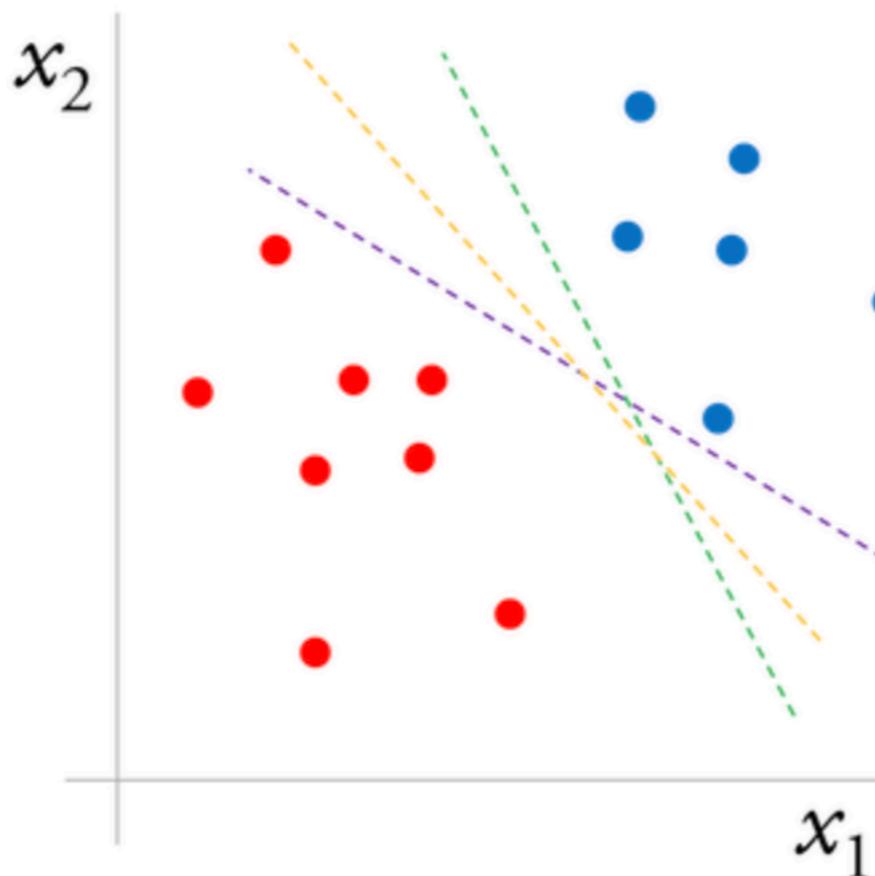
Khái niệm

- Ngoài việc, remove stop word để câu có thể chính xác hơn chúng ta còn cần phải lấy ra ngữ nghĩa chính xác, ví dụ ta có câu: “Bệnh viện là nơi giúp mọi người chữa bệnh”
 - Với ví dụ trên, nếu như theo bag of word thì ta sẽ có một vocabulary là:
 - [“bệnh”, “viện”, “là”, “nơi”, “giúp”, “mọi”, “người”, “chữa”, “bệnh”]
 - Ta thấy rằng từ bệnh viện hay mọi người đều là một từ kép, nếu ta tách riêng ra thì nó sẽ có nghĩa khác nhau, thậm chí sẽ không có ý nghĩa gì cho câu, chẳng hạn chữ bảo vệ, cửa sổ, ...
 - Cho nên chúng ta cần phải tokenize nó thành những từ khoá có nghĩa, bằng cách sử dụng các model đã được xây dựng sẵn như pyvi, underthesea, PhoBERT cũng vừa công bố bộ data pretrained dành cho tiếng việt.
- Với câu Tôi là sinh viên trường Đại học Công Nghệ
chúng ta sẽ tách ra được:

```
# INPUT TEXT IS WORD-SEGMENTED!
line = "Tôi là sinh_viện trường đại_học Công_nghệ ."
```

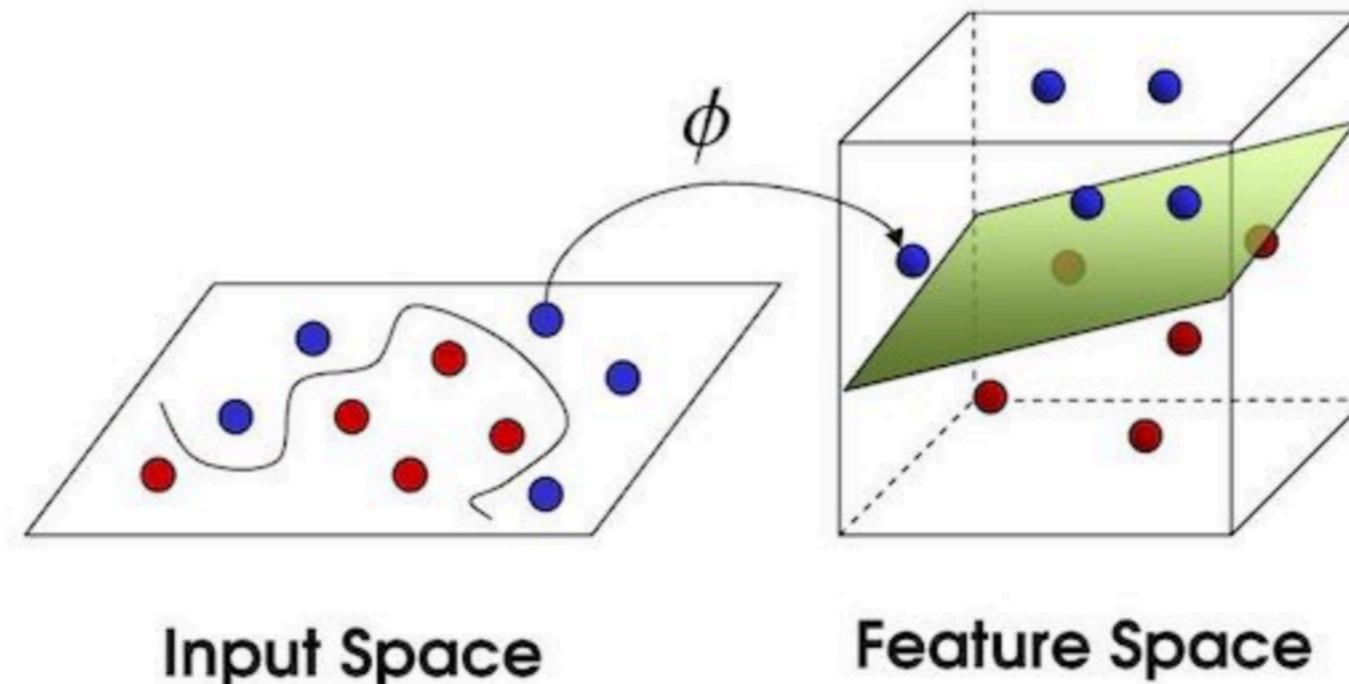
5. Support Vector Machine - SVM

- Support Vector Machine (SVM) là một thuật toán thuộc nhóm Supervised Learning (Học có giám sát) dùng để phân chia dữ liệu (Classification) thành các nhóm riêng biệt và tiên đoán giá trị (Regression). Tuy nhiên là nó được dùng classify nhiều hơn là regress



5. Support Vector Machine - SVM

- Ta cần dùng một phép biến đổi để có thể tìm ra được một siêu phẳng để phân chia các điểm dữ liệu
- => cần dùng thuật toán để ánh xạ bộ data đó vào không gian nhiều chiều hơn (n chiều), từ đó có thể tìm ra được siêu mặt phẳng (hyperplane) để phân chia các dữ liệu



2 chiều => 3 chiều

5. SVM - Maximize margin

- Giờ ta cần phải tìm ra 2 đường thẳng đó là:

$$w_1 * x_1 + w_2 * x_2 + b = 1$$

$$w_1 * x_1 + w_2 * x_2 + b = -1$$

$$\Rightarrow \text{Và tối đa Margin} = \frac{2}{\|W\|^2}$$

$$\Rightarrow \mathbf{w}^T \mathbf{x}_2 + b = 1 \text{ where } \mathbf{x}_2 = \mathbf{x}_1 + \lambda \mathbf{w}$$

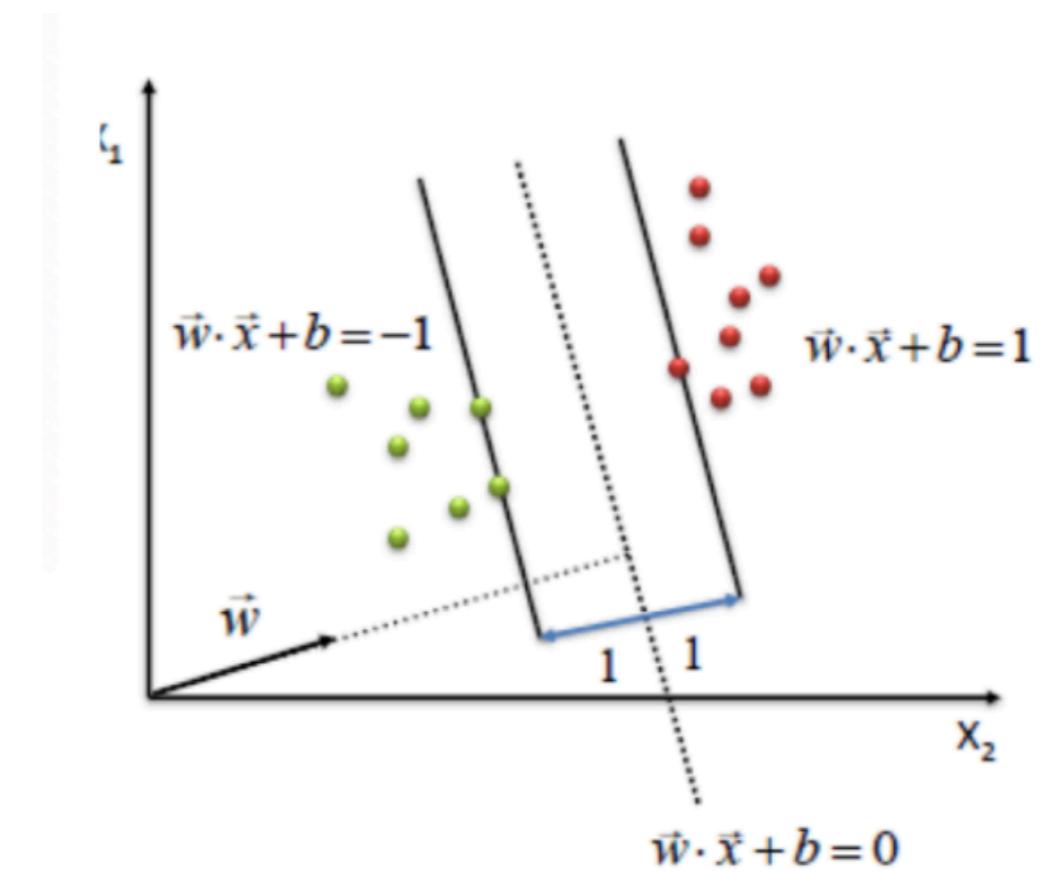
$$\Rightarrow \mathbf{w}^T (\mathbf{x}_1 + \lambda \mathbf{w}) + b = 1$$

$$\Rightarrow \mathbf{w}^T \mathbf{x}_1 + b + \lambda \mathbf{w}^T \mathbf{w} = 1 \text{ where } \mathbf{w}^T \mathbf{x}_1 + b = -1$$

$$\Rightarrow -1 + \lambda \mathbf{w}^T \mathbf{w} = 1$$

$$\Rightarrow \lambda \mathbf{w}^T \mathbf{w} = 2$$

$$\Rightarrow \lambda = \frac{2}{\mathbf{w}^T \mathbf{w}} = \frac{2}{\|\mathbf{w}\|^2}$$



- Lúc đánh nhãn (predict) dữ liệu thì ta chỉ cần xét dấu của phương trình:

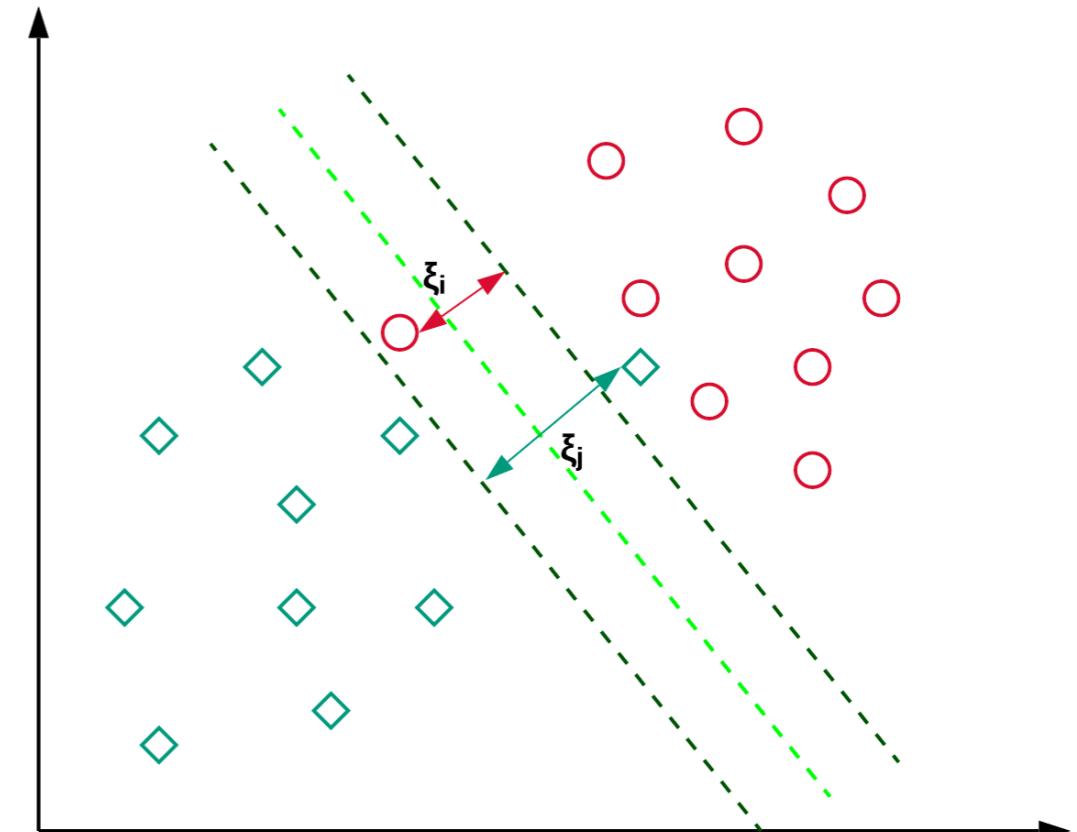
$$\mathbf{w}^T * \mathbf{x} + b \geq 1 \epsilon x_1$$

$$\mathbf{w}^T * \mathbf{x} + b \leq -1 \epsilon x_2$$

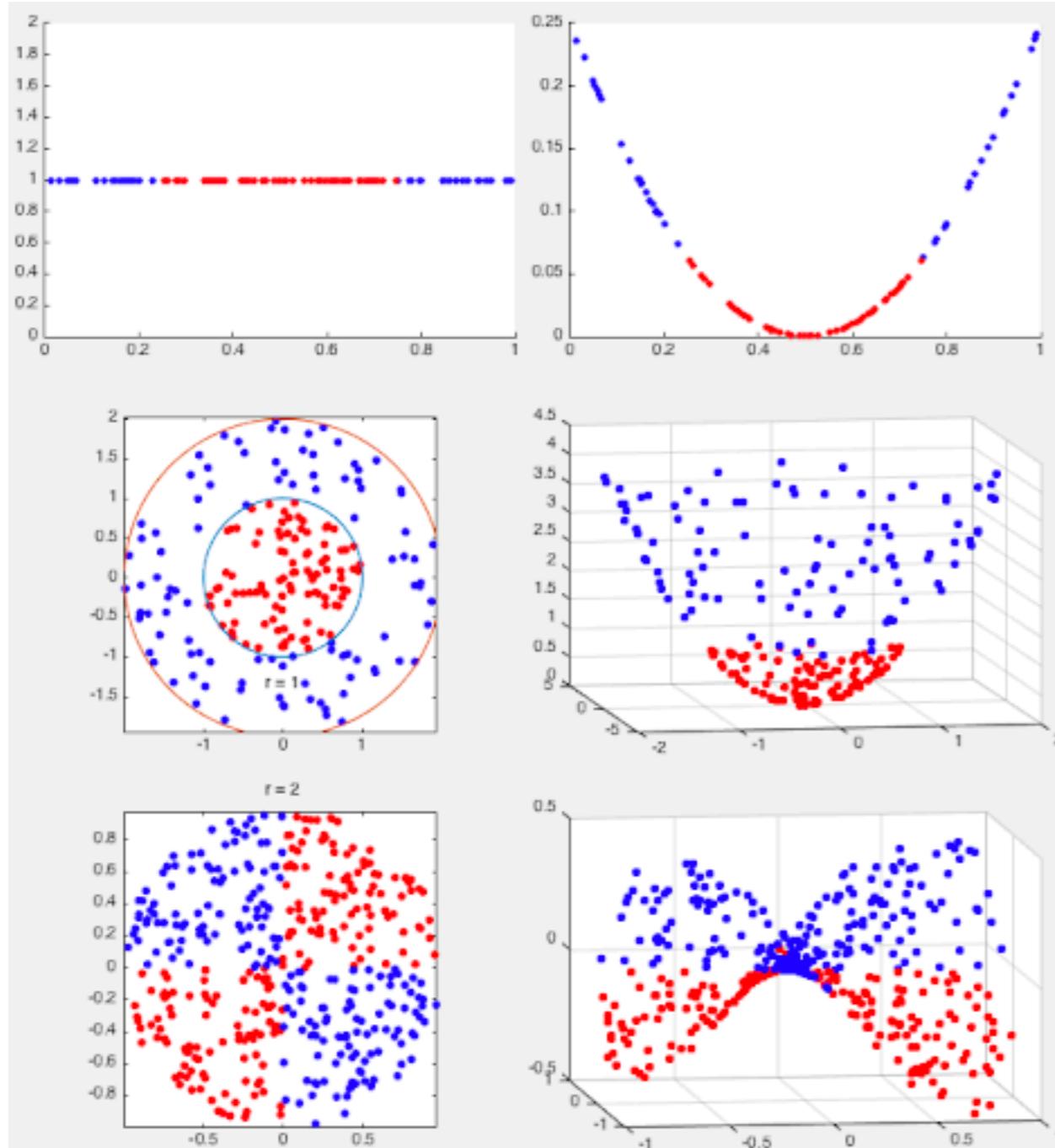
5. SVM - Soft Margin

- Để tránh overfitting, nhiều khi để muốn có margin càng lớn, sẽ có một vài data có thể không được chia chính xác (ví dụ như 1 bóng xanh bị lọt sang vùng của bóng đỏ). => noise data => Soft margin.
- Ngược lại là Hard margin có nghĩa là không cho 1 vài sai sót => 100%
- Cho nên chúng ta cần điều chỉnh tham số C : C càng lớn thì tiến dần về Hard Margin

- $C = \infty \Rightarrow$ Không cho phép sai lệch => Hard Margin.
- C lớn => sai lệch nhỏ => thu được Margin nhỏ.
- C nhỏ => sai lệch lớn => thu được Margin lớn.



5. SVM - Kernel



- Kernel SVM là việc đi tìm một hàm số biến đổi dữ liệu x từ không gian feature ban đầu thành dữ liệu trong một không gian mới bằng hàm số $\Phi(x)$

$$k(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$$

$$\phi: \mathbf{x} = (x_1, x_2) \mapsto \phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2) \in F = \mathbb{R}^3$$

$$\psi(\mathbf{x}) = (x_1^2, x_2^2, x_1x_2, x_2x_1) \in F = \mathbb{R}^4$$

5. SVM - Kernel thông dụng

Tên	Công thức	kernel	Thiết lập hệ số
linear	$\mathbf{x}^T \mathbf{z}$	'linear'	không có hệ số
polynomial	$(r + \gamma \mathbf{x}^T \mathbf{z})^d$	'poly'	d : degree, γ : gamma, r : coef0
sigmoid	$\tanh(\gamma \mathbf{x}^T \mathbf{z} + r)$	'sigmoid'	γ : gamma, r : coef0
rbf	$\exp(-\gamma \ \mathbf{x} - \mathbf{z}\ _2^2)$	'rbf'	$\gamma > 0$: gamma

- Radial Basic Function (RBF) kernel hay Gaussian kernel được sử dụng nhiều nhất trong thực tế vì độ chuẩn xác của nó. Nhưng không phải lúc nào cũng dùng rbf mà còn tuỳ vào dữ liệu và độ chính xác mà model áp dụng kernel mới hợp lí
- Việc tính toán trực tiếp hàm Φ đôi khi phức tạp và tốn nhiều bộ nhớ. Cho nên, ta có thể sử dụng kernel trick. Trong cách tiếp cận này, ta chỉ cần tính tích vô hướng của hai vector bất kỳ trong không gian mới:

$$k(\mathbf{x}, \mathbf{z}) = \Phi(\mathbf{x})^T \Phi(\mathbf{z})$$

6. Neural Network

- Ngoài việc chúng ta sử dụng SVM để phân loại text chúng ta cũng có thể dùng Neural Network để phân loại với hàm activation cuối là softmax - đầu ra là số lượng class (one hot encoding) chúng ta cần phân loại

```
# xây dựng mạng network
class ModelNLP(object):
    @staticmethod
    def build(SIZE_TRAIN,num_classes, final_activation):
        model = Sequential()
        model.add(Dense(500, input_shape=(SIZE_TRAIN,), activation='relu', kernel_initializer="uniform"))
        model.add(LeakyReLU(alpha=0.3))
        model.add(Dropout(rate=0.2))
        model.add(Dense(400))
        model.add(LeakyReLU(alpha=0.3))
        model.add(Dropout(rate=0.2))
        model.add(Dense(300))
        model.add(LeakyReLU(alpha=0.3))
        model.add(Dropout(rate=0.2))
        model.add(Dense(200))
        model.add(Dense(num_classes, activation=final_activation))
        return model
```

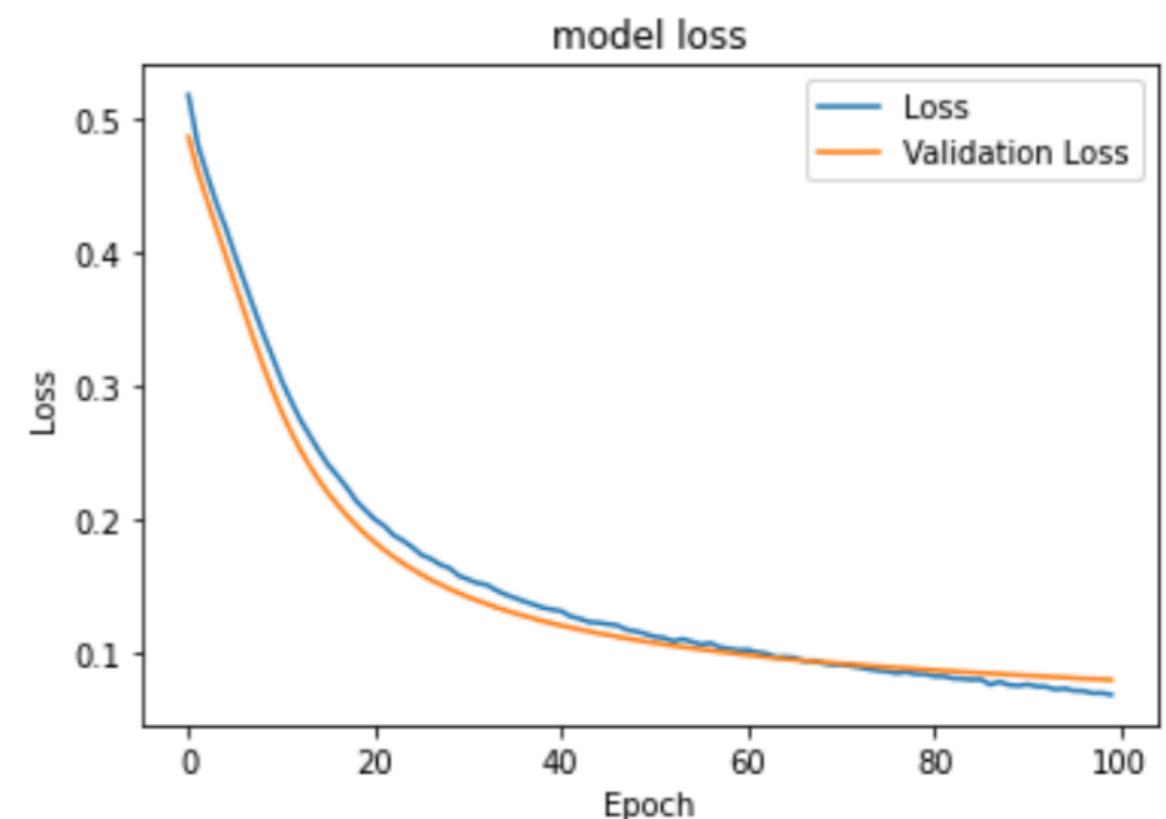
6. Neural Network

- Mô hình + loss khi dùng neural network

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 500)	2199000
leaky_re_lu_1 (LeakyReLU)	(None, 500)	0
dropout_1 (Dropout)	(None, 500)	0
dense_2 (Dense)	(None, 400)	200400
leaky_re_lu_2 (LeakyReLU)	(None, 400)	0
dropout_2 (Dropout)	(None, 400)	0
dense_3 (Dense)	(None, 300)	120300
leaky_re_lu_3 (LeakyReLU)	(None, 300)	0
dropout_3 (Dropout)	(None, 300)	0
dense_4 (Dense)	(None, 200)	60200
dense_5 (Dense)	(None, 5)	1005

```
Total params: 2,580,905
Trainable params: 2,580,905
Non-trainable params: 0
```



7. Reference

Support vector machine (SVM)

<https://ongxuanhong.wordpress.com/2015/09/19/support-vector-machine-svm-hoi-gi-dap-nay/>

Support Vector Machine

<https://machinelearningcoban.com/2017/04/09/sm/>

Kernel Support Vector Machine

<https://machinelearningcoban.com/2017/04/22/kernelsmv/>

Support Vector Machine (SVM) là gì?

<https://1upnote.me/post/2018/10/ds-ml-svm/>

Support Vector Machine – Introduction to Machine Learning Algorithms

<https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>

An Introduction to Support Vector Machines (SVM)

<https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/>

<https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>

7. Reference

Text Classification

<https://monkeylearn.com/text-classification/>

Text Classification in Python

<https://towardsdatascience.com/text-classification-in-python-dd95d264c802>

Text Classification(NLP) using SVM and Naive Bayes

<https://medium.com/@bedigunjit/simple-guide-to-text-classification-nlp-using-svm-and-naive-bayes-with-python-421db3a72d34>

How to Use TfIdftransformer & TfIdfvectorizer?

<https://kavita-ganesan.com/tfidftransformer-tfidfvectorizer-usage-differences/#.XrEGbBMzbRZ>

Bag of Words (Bow) TF-IDF - Xử lý ngôn ngữ tự nhiên

<https://codetudau.com/bag-of-words-tf-idf-xu-ly-ngon-ngu-tu-nhien/index.html>

Github PhoBert, Underthesea, pyvi

<https://github.com/VinAIResearch/PhoBERT>

<https://github.com/undertheseanlp/classification>