Train and Test datasets

the train data sets have 1500 data with column (reviews_content, category), and the test data set with column (rewiews_content, category). Our task is to train any proper classification model to predict what the category of the test datasets, on this project we will use **Support Vector Classifier with linear kernel** to predict the test datasets.

```
[6]: df_train
```

[6]:

	reviews_content	category
0	airplane ! is considered among many to be the ...	positive
1	you've got to love disney . \nno matter what t...	positive
2	" the tailor of panama " is a different kind ...	positive
3	the characters in jonathan lynn's " the whole ...	negative
4	vikings v . bears ? \nno , this isn't the line...	negative
...	...	...
1495	trekkies , roger nygard's energetic and hilari...	positive
1496	" dangerous beauty " is a really nothing more...	positive
1497	starring shawnee smith ; donovan leitch ; rick...	negative
1498	man , this was one wierd movie . \nsimilar to ...	negative
1499	review : ghost dog : the way of the samurai (...	positive

1500 rows × 2 columns

train datasets

```
[7]: df_test
```

[7]:

	reviews_content
0	towards the middle of " the sweet hereafter , ...
1	wild things is a suspenseful thriller starring...
2	hong kong cinema has been going through a bad ...
3	while alex browning (devon sawa) waits at jf...
4	sometimes i find 19th century british costume ...
...	...
495	luckily , some people got starship troopers . ...
496	trailing the success of brit humour in the mov...
497	seen february 15 , 1998 on home video (borrow...
498	matthew broderick and high school comedy . \nt...
499	the 1998 summer movie season is still in its i...

500 rows × 1 columns

test datasets

Feature Extraction and Selection

the text representation is not able to proccessed by computer, computer can only process number, to deal that we have a very common methods to represent the corpus into a bag of Words model i.e using TFIDF to extract feature, TF-IDF is an abbreviation for Term Frequency Inverse Document Frequency. This algorithm is to transform text into a meaningful representation of numbers which is used to fit machine algorithm for prediction. the mathematical representation of the TF-IDF is given by

$$\begin{aligned} TFIDF(d,t) &= TF(d,t) * IDF(t) \\ &= TF(d,t) * \log\left(\frac{N}{n_t}\right) \end{aligned}$$

with :

- $TF(t,d)$ is frequency of doc d with term t divided by frequency of term t on all doc.
- $IDF(t)$ is number of document in corpus (1500 for train) divided by number of document with term t

on this part will be use smooth idf instead to prevents zero division when a document don't have a words given a vocabulary the TF-IDF is given by

$$TFIDF(d,t) = TF(d,t) * \left[log\left(\frac{N}{n_t+1}\right) + 1\right]$$

the normalization using l2 normalization so after TFIDF calculation each features are divided by the sum of squares of each features

$$w_{ij} = \frac{TFIDF(d,t)}{\sqrt{\sum_{d \in D} TFIDF(d,t)}}$$

before input our data into TFIDF we need to perform some preprocessing, on this part the reprocessing is only lower casing, extract alphabetic, and lemmatization using wordnet to bring a word into the root words

```
df_train['processed text'] = df_train['reviews_content'].apply(extract_alphabetic)
df_train['processed text'] = df_train['processed text'].apply(wordnet_lemmatizer)

df_test['processed text'] = df_test['reviews_content'].apply(extract_alphabetic)
df_test['processed text'] = df_test['processed text'].apply(wordnet_lemmatizer)
```

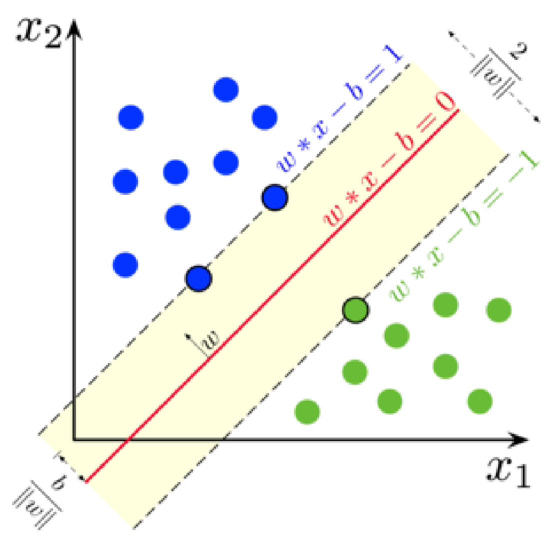
after the reprocessing perform TFIDF using sklearn, the parameters is the key to get best accuracy, on this part we must strict the features selection using parameters min_df and max_df to strict the features by counting the frequency it means should be between min_df and max_df, and ngram_range parameters try to capture 1-gram, 2-gram, and 3-gram words.

```
from sklearn.feature_extraction.text import TfidfVectorizer as vec
vect = vec(ngram_range = (1,3),min_df = 8,max_df = 1000)
vect.fit(df_train['processed text'])
X_train = vect.transform(df_train['processed text'])
y_train = df_train['category']
X_test = vect.transform(df_test['processed text'])
```

actually we can reparameterize the min_df and max_df but at here, I choose the value instead.

SVM Classifier model

Support Vector Machine model can be used for classification and regression tasks, it finds out the hyperplane that distincly coassifies the data points. the main objective is to discover such a plane that has the maximum distance between the points of both classes. Although designing the perfect kernel is difficult, SVM is highly preferred because of its higher accuracy with less computation.



SVM : Linear Model

$$w.x_i - b \geq 1 \text{ if } y_i = 1$$
$$w.x_i - b \geq -1 \text{ if } y_i = -1$$
$$y_i(w.x_i - b) \geq 1, y \in \{-1, 1\}$$

Loss function : Hinge Loss

$$l = \max(0, 1 - y_i(w.x_i - b))$$

The hinge loss penalizes misclassified points by the amount they fall on the wrong side of the margin (1 in this case). If the prediction $y_i(w_i)$ is correct and the margin is large enough (greater than 1), the hinge loss is 0. Otherwise, it increases linearly with the distance from the correct side of the margin. In SVMs, the goal is to minimize both the hinge loss and the regularization term, which encourages a smaller weight vector $||w||^2$ This optimization problem is typically solved using techniques like gradient descent.Hinge loss encourages the model to find a decision boundary with a larger margin, making it less sensitive to individual data points and more robust to overfitting. It's commonly used in linear SVMs and related models for binary classification tasks.

Regularization

Regularization is a technique used in machine learning to prevent overfitting by adding a penalty term to the loss function. In the context of Support Vector Machines (SVMs), regularization is often applied to the optimization problem to encourage a solution that generalizes well to unseen data.

In SVMs, regularization is typically applied to the margin and is commonly referred to as "soft-margin" SVM. The goal of soft-margin SVM is to find the decision boundary that maximizes the margin between classes while allowing for some misclassification (points falling within the margin or on the wrong side of the decision boundary). This is achieved by introducing a regularization parameter, usually denoted as C,

$$J = \lambda||w||^2 + \frac{1}{n}\sum_{i=1}^{n} max(0, 1 - y_i(w.x_i - b))$$

or simply

$$\text{if: } y_i.f(x) \geq 1 : J_i = \lambda||w||^2 \qquad \text{else: } \lambda||w||^2 + 1 - y_i(w.x_i - b)$$

the gradients is calculated for perform gradient descent in training steps

$$\text{if } y_i.f(x) \geq 1 : \frac{\partial J_i}{\partial w_k} = 2\lambda w_k \qquad \frac{\partial J_i}{db} = 0$$
$$else : \frac{\partial J_i}{\partial w_k} = 2\lambda w_k - y_i.x_{ik} \qquad \frac{\partial J_i}{db} = y_i$$

In practice, the choice of C is often determined using techniques like cross-validation, where different values of C are evaluated on a validation set to find the one that results in the best performance. Regularization in SVMs helps prevent overfitting by penalizing complex decision boundaries, resulting in models that generalize better to unseen data. It is a crucial component in building robust and effective SVM classifiers.

Training update rule

$$\text{if: } y_i.f(x) \geq 1 : w_t = w_{t-1} - \alpha\frac{\partial J_i}{\partial w_k} = w_{t-1} - \alpha.2\lambda w \qquad b_t = b_{t-1} - \alpha\frac{\partial J_i}{db} = b_{t-1}$$
$$\text{else: } w_t = w_{t-1} - \alpha\frac{\partial J_i}{\partial w_k} = w_{t-1} - \alpha(2\lambda w - y_i.x_i) \qquad b_t = b_{t-1} - \alpha\frac{\partial J_i}{\partial b} = b - \alpha.y_i$$

hence that we can create algorithm to train this model

1. Training (learn weights)

- initialize weights
- make sure $y \in -1, 1$
- apply update rules stop until n_iter

2. Prediction

- Calculate $y = sign(w.x - b)$

Train our model

finetuning C regularization parameter

```
for i in np.arange(0.1,2,0.1):
    model = LinearSVC(tol = 0.001, C = i,dual ='auto')
    result = cross_val_score(model, X_train,y_train,cv = 15)
    print(f'C = {i:.2f} avg - {np.mean(result):.2f} median - {np.median(result):.2f}')
```

the output is given

```
C = 0.10 avg - 0.84 median - 0.84
C = 0.20 avg - 0.85 median - 0.86
C = 0.30 avg - 0.86 median - 0.85
C = 0.40 avg - 0.86 median - 0.85
C = 0.50 avg - 0.86 median - 0.86
C = 0.60 avg - 0.86 median - 0.86
C = 0.70 avg - 0.86 median - 0.86
C = 0.80 avg - 0.86 median - 0.86
C = 0.90 avg - 0.87 median - 0.86
C = 1.00 avg - 0.87 median - 0.86
C = 1.10 avg - 0.87 median - 0.86
C = 1.20 avg - 0.87 median - 0.86
C = 1.30 avg - 0.87 median - 0.86
C = 1.40 avg - 0.87 median - 0.86
C = 1.50 avg - 0.87 median - 0.86
C = 1.60 avg - 0.87 median - 0.86
C = 1.70 avg - 0.87 median - 0.86
C = 1.80 avg - 0.87 median - 0.86
C = 1.90 avg - 0.87 median - 0.86
```

from that I choose C = 0.9 because we get increasing performance 0.87 first time, then train the model

```
model = LinearSVC(tol = 0.001, C = 0.9)
model.fit(X_train,y_train)
y_predict = model.predict(X_test)
```

now convert the y_predict into CSV files

```
output_file = pd.DataFrame({'Row': range(1,501), 'Label':y_predict})
output_file.to_csv('SVM_.csv',index = False
```