Text Mining Tutorial 6:

Text Categorization & Clustering

Prof. Hsing-Kuo Pao
Teaching Assistant: Ghaluh Indah Permata Sari

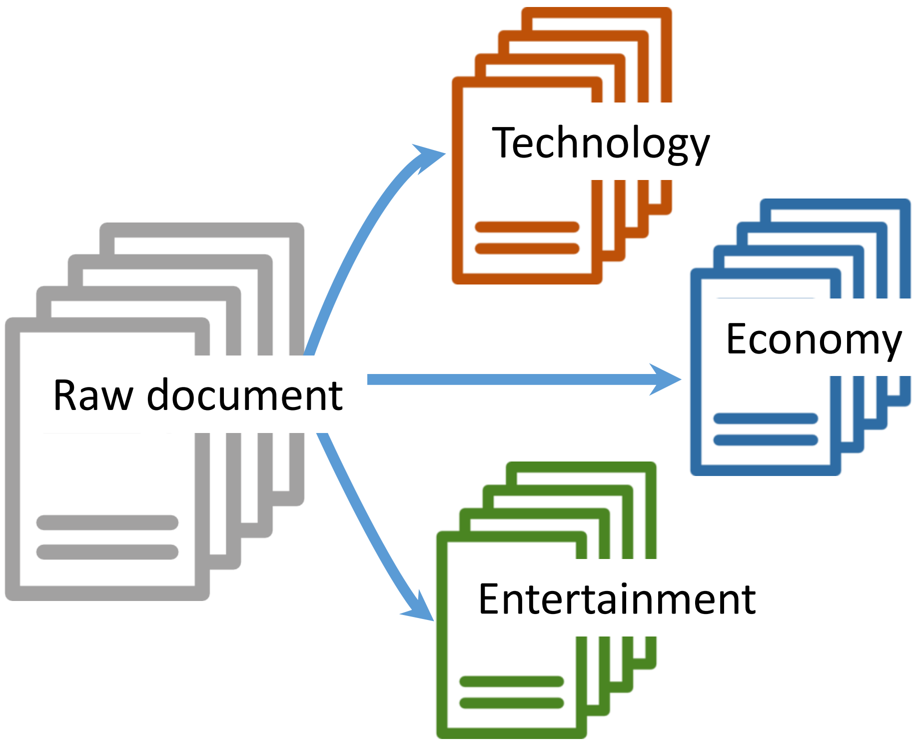
TAIWAN TECH
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Outline

- Introduction to Text Categorization: applications, pipeline
- Machine Learning Insight
- Introduction to Sentiment Analysis
- Sentiment analysis task: Supervised and Unsupervised case
- Dimension Reduction

Introduction

- Text categorization is a classification task in machine learning. Thus, we called this technique Text Classification.
- Objective: automatically classify the text documents into one or more defined categories.
- Purpose: Labeling natural language texts with relevant categories from a predefined set. Classifying your content and products into categories helps users to easily search and navigate within a website or application.



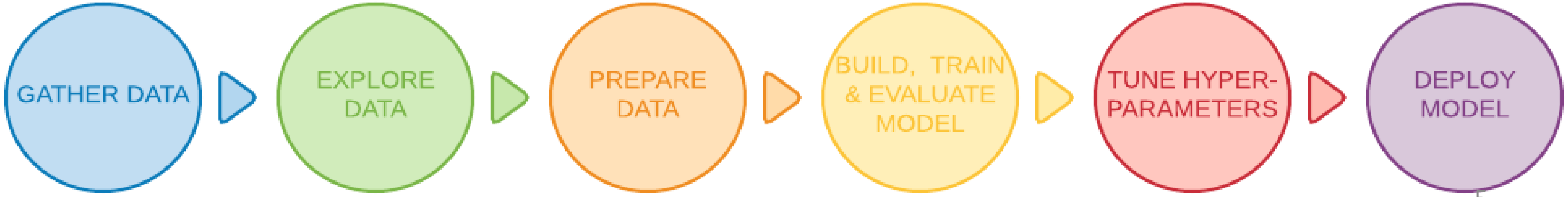
Text Classification: application

- Sentiment analysis.
- E-commerce, news agencies, content curators, blogs, directories, and likes.
- Automated CRM tasks.

Text Classification Pipeline

Text Classification Steps:

1. Get the data
2. Data exploratory and Feature extraction
3. Choose a model: SVM, Naïve bayes, Multinomial Naïve bayes, Linear regression, K-Nearest Neighbors, Random forest, Decision Trees, Stochastic Gradient Descent.
4. Build, train, and evaluate the model
5. Tune Hyperparameter
6. Deploy the model

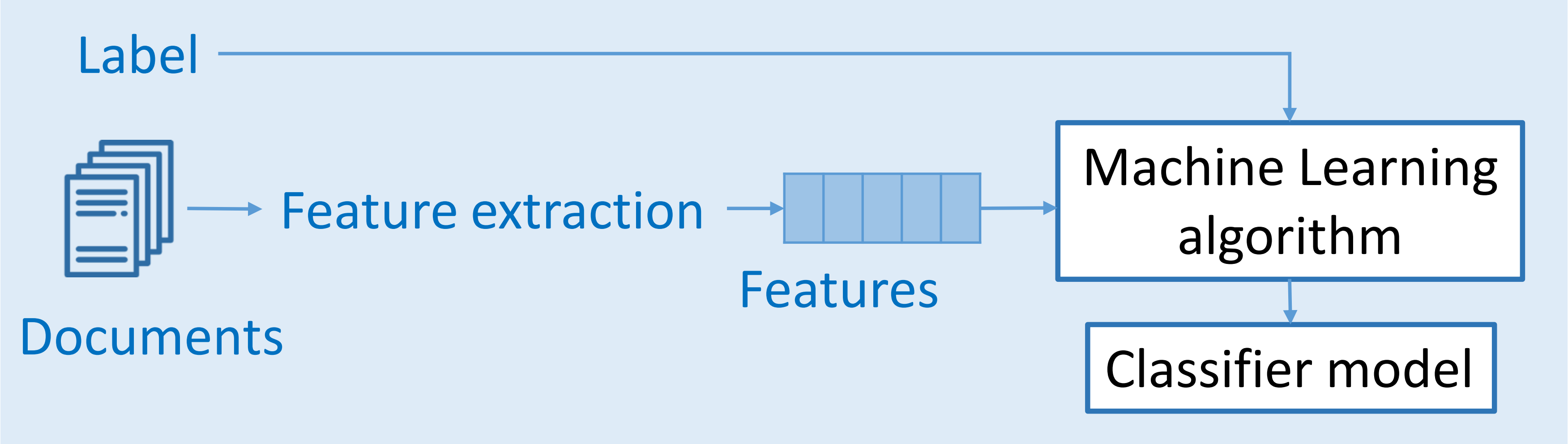


TAIWAN TECH
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

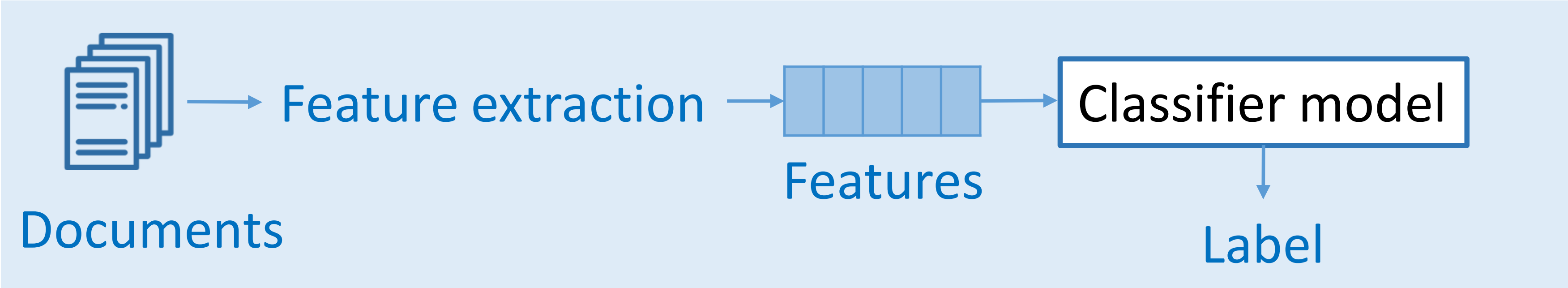
Source: developer.google.com

Text Classification Pipeline

Train



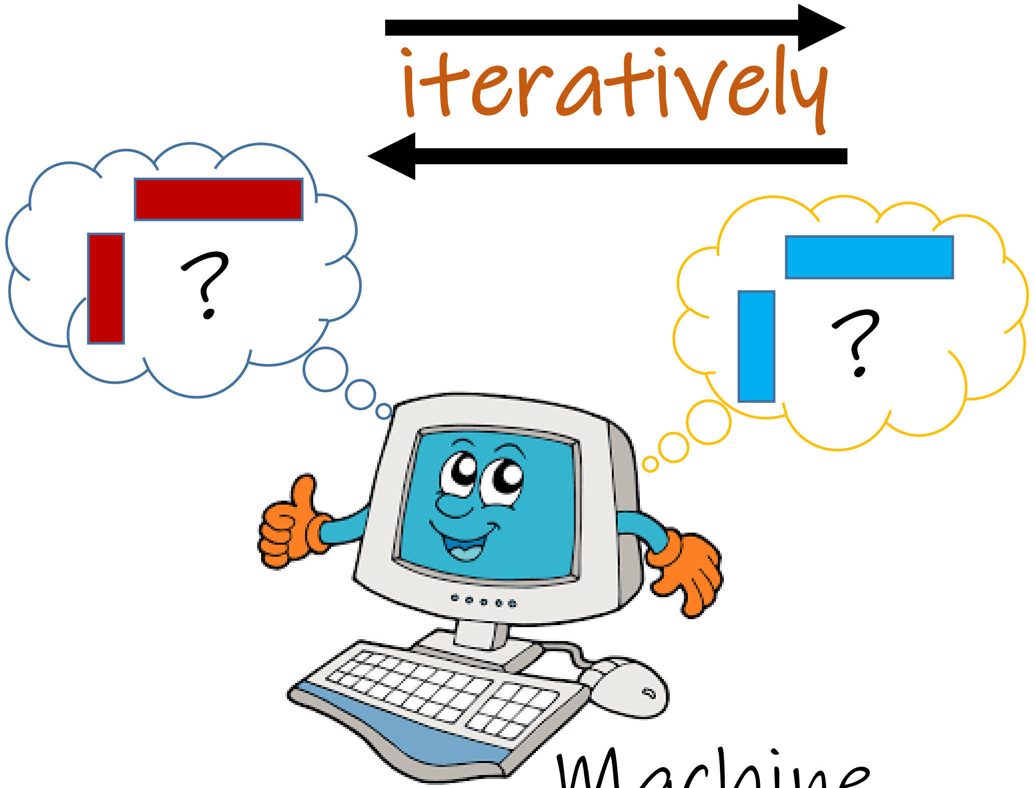
Test/ prediction



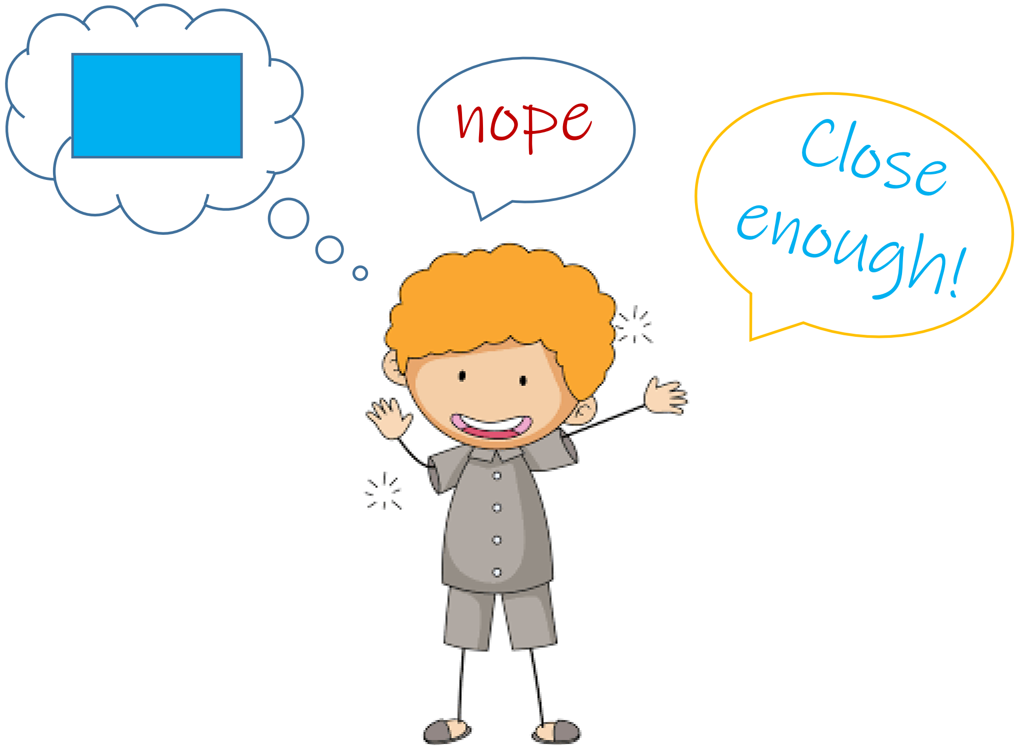
Machine Learning Insights

Training models

iteratively



Machine



Human

Machine Learning Insights

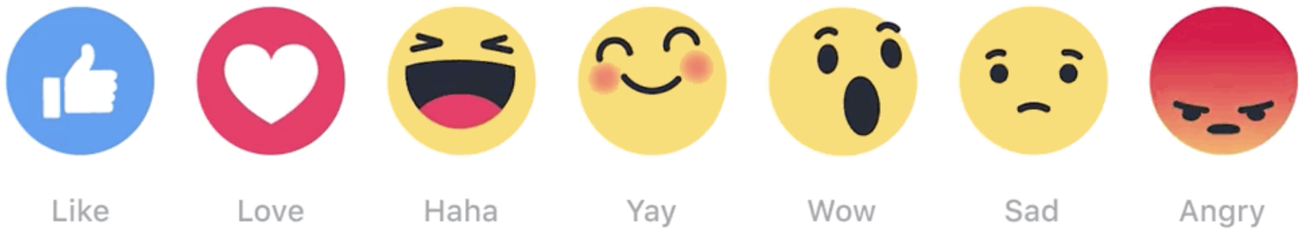
Learning = Optimization → Objective function

Supervised	Unsupervised
• Working with labeled data	• Working with unlabeled data
• Objective: prediction of outcomes for new data that is introduced	• Objective: getting new insights from massive amounts of new data
• Costly but robust, and need human annotation/ judgement	• No need human annotation. Human only help for validate the variable outputs.
• Classification, Regression	• Clustering, Association, Dimensional Reduction

TAIWAN TECH
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Introduction to Sentiment Analysis

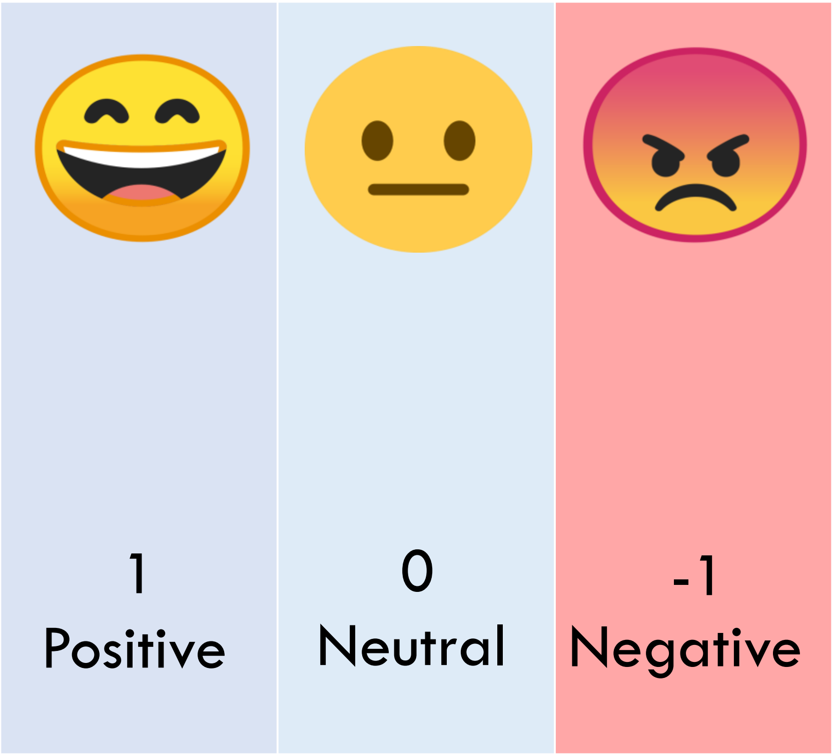
- Opinion mining, opinion extraction, sentiment mining, subjectivity analysis.
- Objective: to accurately extract people's opinions from a large number of unstructured review texts and classifying them into sentiment classes, i.e., positive, negative, or neutral.
- Purpose: Identifying and categorizing opinions from text. Determine the writer’s attitude towards a particular topic or the product.



Introduction to Sentiment Analysis task

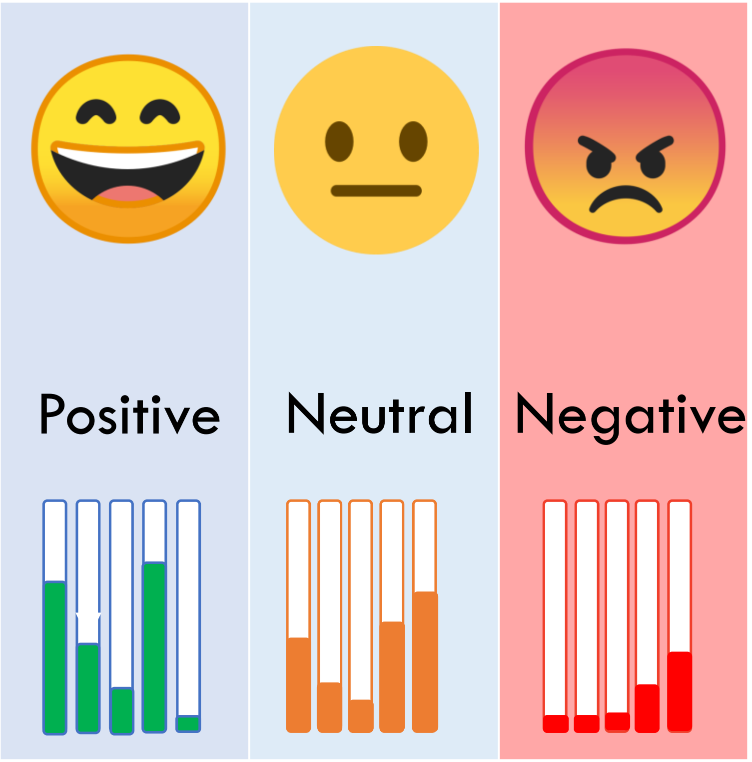
Simple task

Is the attitude of this text positive or negative?



Complex task

Rank the attitude of this text from 1~5



How it works?

Using AFINN Lexicons

AFINN → 3000+ words with polarity score

By Finn Arup Nielsen

```
abandon -2
abandoned       -2
abandons        -2
abducted        -2
abduction       -2
abductions      -2
abhor   -3
abhorred        -3
abhorrent       -3
abhors  -3
abilities       2
ability 2
aboard  1
```

e.g. I love dog, but I am allergic to them

+3 -2

+1 Mildly positive

Source: https://gist.githubusercontent.com/damianesteban/06e8be3225f641100126/raw/a51c27d4e9cc242f829d895e23b4435021ab55e5/afinn-111.txt

TAIWAN TECH
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

How it works?

Using TextBlob Lexicons

- As TextBlob is a Lexicon-based sentiment analyzer.
- It has some predefined rules or we can say word and weight dictionary, where it has some scores that help to calculate a sentence's polarity.
- That's why the Lexicon-based sentiment analyzers are also called "Rule-based sentiment analyzers".
- The output of TextBlob is **polarity** and **subjectivity.**

```
pip install textblob
```

```
from textblob import TextBlob
```


Source: https://textblob.readthedocs.io/en/dev/

How it works?

Using Vader Lexicons

- Vader (Valence Aware Dictionary and Sentiment Reasoner))
- VADER not only tells the lexicon is positive, negative, or neutral, it also tells how positive, negative, or neutral a sentence is.
- The output from VADER comes in a Python dictionary in which we have four keys and their corresponding values. ‘neg’, ‘neu’, ‘pos’, and ‘compound’ which stands for Negative, Neutral, and Positive respectively.
- The Compound score is an indispensable score that is calculated by normalizing the other 3 scores (neg, neu, pos) between -1 and +1.

by Eric Gilbert and C. Hutto

```
pip install vaderSentiment
```

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
```

TAIWAN TECH
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Sentiment types

- Document level
  - To classify whether a whole opinion document expresses a positive, or negative sentiment.

*"I bought an **iPhone** a few days ago. It was such a **nice** phone. The **touch screen** was **cool**. The **voice quality** was **clear** too. However, my mother was angry with me as I did not tell her before buying it. She also thought the phone was too expensive and wanted me to return it to the shop."*

TAIWAN TECH
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Sentiment types

- Sentence level
  - To classify whether a whole opinion sentence expresses a positive, or negative sentiment.
  - This level of analysis is closely related to subjectivity classification, which decides sentences that express factual information.

1. *I bought an **iPhone** a few days ago.*
2. *It was such a **nice** phone.*
3. *The **touch screen** was **cool**.*
4. *The **voice quality** was **clear** too.*
5. *Although the battery life was not long, that is ok for me.*
6. *However, my mother was angry with me as I did not tell her before buying it.*
7. *She also thought the phone was too expensive and wanted me to return it to the shop.*

Applications & Data

Applications:

- Analyzing customer feedback
- Campaign monitoring: Cambridge Analytica Scandal
- Brand monitoring: KFC
- Stock market analyzing
- Compliance monitoring

Data:

- Twitter API: https://www.tweepy.org/
- ScrapingHub: https://github.com/scrapinghub
- Brand monitoring tools: https://www.brandwatch.com/

Logistic Regression: Sentiment Analysis

Case: Simple Features - Simple Model

Dataset: https://archive.ics.uci.edu/ml/datasets/Sentiment+Labelled+Sentences

Create data frame

```
import pandas as pd

filepath_dict = {'yelp':   'teaching/sentiment labelled sentences/yelp_labelled.txt',
                 'amazon': 'teaching/sentiment labelled sentences/amazon_cells_labelled.txt',
                 'imdb':   'teaching/sentiment labelled sentences/imdb_labelled.txt'}

df_list = []
for source, filepath in filepath_dict.items():
    df = pd.read_csv(filepath, names=['sentence', 'label'], sep='\t')
    df['source'] = source  # Add another column filled with the source name
    df_list.append(df)

df = pd.concat(df_list)
print(df.iloc[0])
```

Output:

```
sentence    Wow... Loved this place.
label                              1
source                          yelp
Name: 0, dtype: object
```

TAIWAN TECH
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Sentiment analysis task: Supervised (Logistic Regression)

Feature Extraction
Scikit-Learn (Sklearn)

```
sentences = ['John likes ice cream', 'John hates chocolate.']
```

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

```
vectorizer = CountVectorizer(min_df=0, lowercase=False)
```

```
vectorizer.fit(sentences)
```

```
CountVectorizer(lowercase=False, min_df=0)
```

Model Construction

Documentation: *https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html*

Sentiment analysis task: Supervised (Logistic Regression)

Feature Extraction

Scikit-Learn (Sklearn)

Mapping term to features index

```
vectorizer.vocabulary_
```

Output: {'John': 0, 'likes': 5, 'ice': 4, 'cream': 2, 'hates': 3, 'chocolate': 1}

```
vectorizer.transform(sentences).toarray()
```

Output:
```
array([[1, 0, 1, 0, 1, 1],
       [1, 1, 0, 1, 0, 0]])
```

Vector Space

TAIWAN TECH
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Sentiment analysis task: Supervised (Logistic Regression)

```
from sklearn.model_selection import train_test_split
```

```
df_yelp = df[df['source'] == 'yelp']
```

```
sentences = df_yelp['sentence'].values
y = df_yelp['label'].values
sentences_train, sentences_test, y_train, y_test = train_test_split(
    sentences, y, test_size=0.25, random_state=1000)
```

Test and Train Splitting

Documentation: *https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html*

Sentiment analysis task: Supervised (Logistic Regression)

Yet another examples,

Test and train
With K-Fold

```
# scikit-learn k-fold cross-validation
from numpy import array
from sklearn.model_selection import KFold
# data sample
data = array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6])
# prepare cross validation
kfold = KFold(3, True, 1)
# enumerate splits
for train, test in kfold.split(data):
    print('train: %s, test: %s' % (data[train], data[test]))
```

Sentiment analysis task: Supervised (Logistic Regression)

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

```
vectorizer = CountVectorizer()
vectorizer.fit(sentences_train)
```

```
CountVectorizer()
```

Learn a vocabulary dictionary of all tokens in the raw documents

```
X_train = vectorizer.transform(sentences_train)
X_test  = vectorizer.transform(sentences_test)
X_train
```

```
<750x1714 sparse matrix of type '<class 'numpy.int64'>'
	with 7368 stored elements in Compressed Sparse Row format>
```

Transform documents to document-term matrix

Sentiment analysis task: Supervised (Logistic Regression)

[Logistic Regression model]

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(X_train, y_train)
score = classifier.score(X_test, y_test)
print("Accuracy:", score)
```

Classifier

Output: Accuracy: 0.796

Sentiment analysis task: Supervised (Logistic Regression)

Logistic Regression model

```
for source in df['source'].unique():
    df_source = df[df['source'] == source]
    sentences = df_source['sentence'].values
    y = df_source['label'].values

    sentences_train, sentences_test, y_train, y_test = train_test_split(
        sentences, y, test_size=0.25, random_state=1000)

    vectorizer = CountVectorizer()
    vectorizer.fit(sentences_train)
    X_train = vectorizer.transform(sentences_train)
    X_test  = vectorizer.transform(sentences_test)

    classifier = LogisticRegression()
    classifier.fit(X_train, y_train)
    score = classifier.score(X_test, y_test)
    print('Accuracy for {} data: {:.4f}'.format(source, score))
```

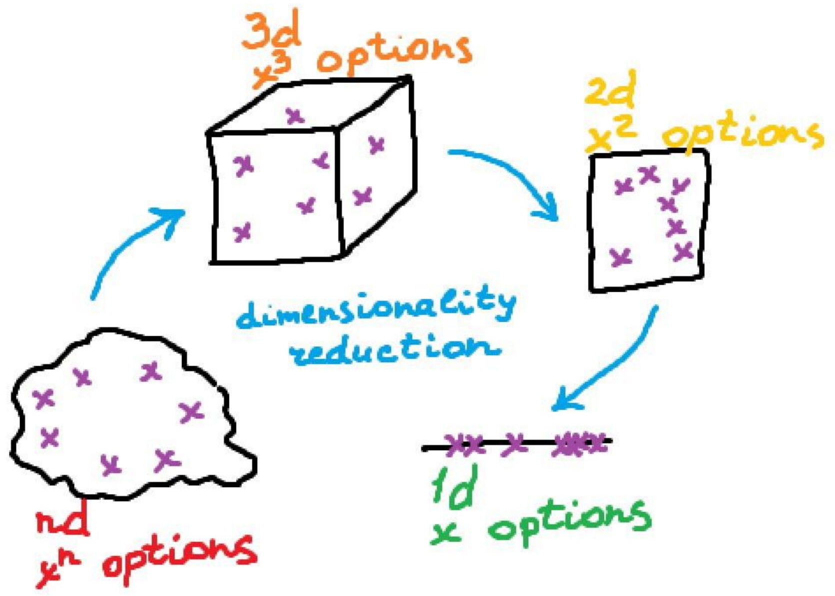
Output:

```
Accuracy for yelp data: 0.7960
Accuracy for amazon data: 0.7960
Accuracy for imdb data: 0.7487
```

Dimension Reduction

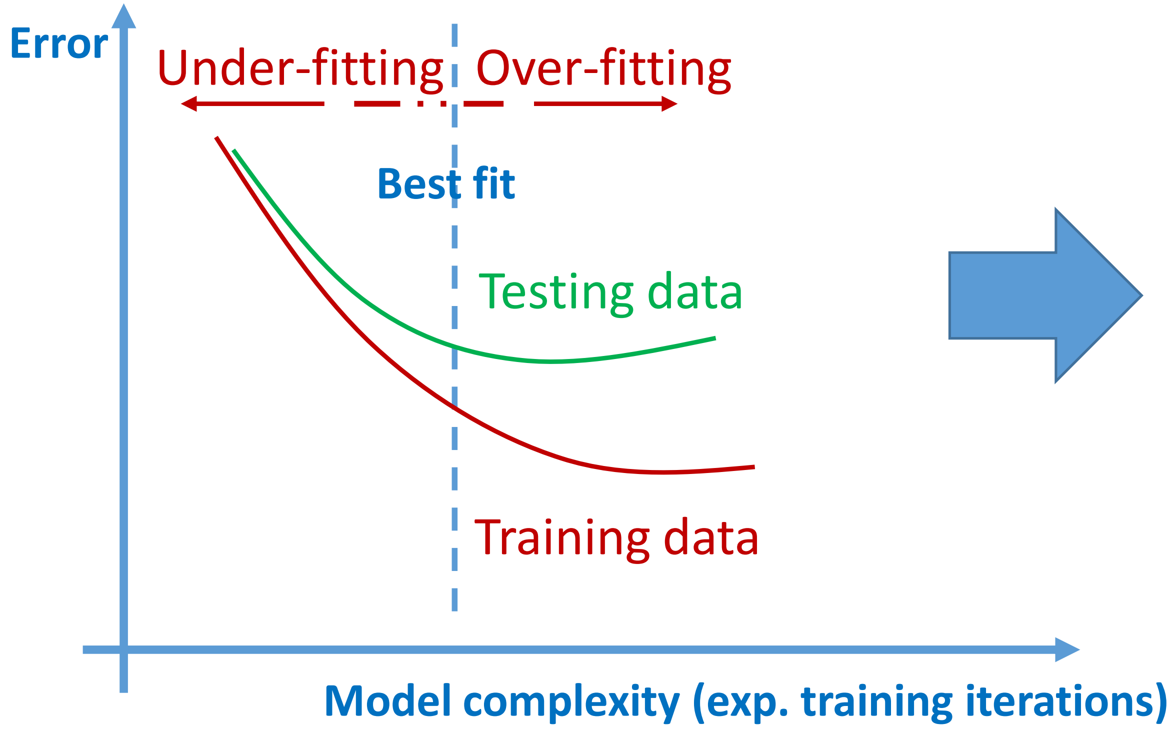
The importance of dimension reduction

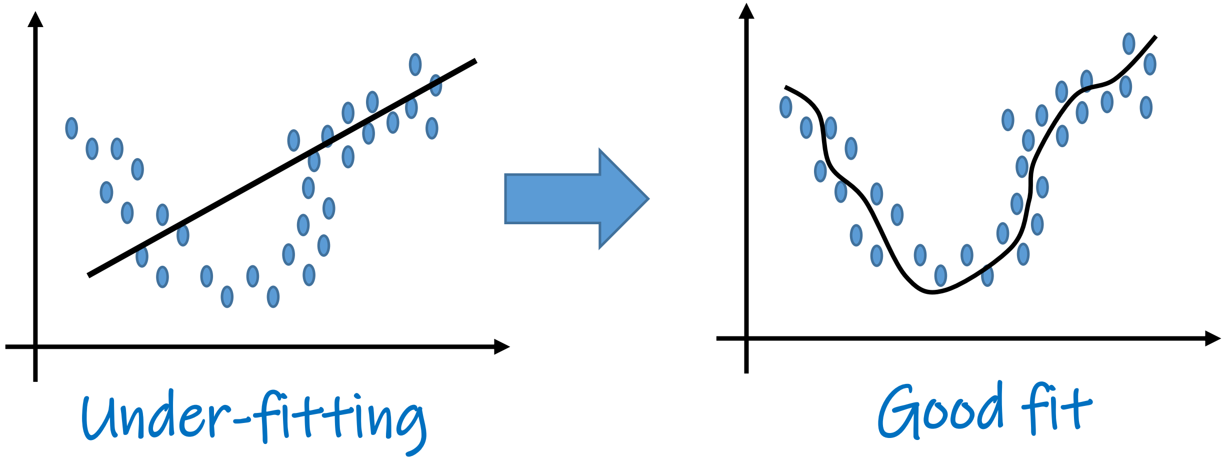
- A lower number of dimensions in data: Less training time and less computational resources, increases the overall performance of machine learning algorithms.
- Transform non-linear data into a linearly-separable form
- Avoids overfitting
- Data visualization
- Removes noise in the data
- Useful for image compression



TAIWAN TECH
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Model fitting: under-fitting





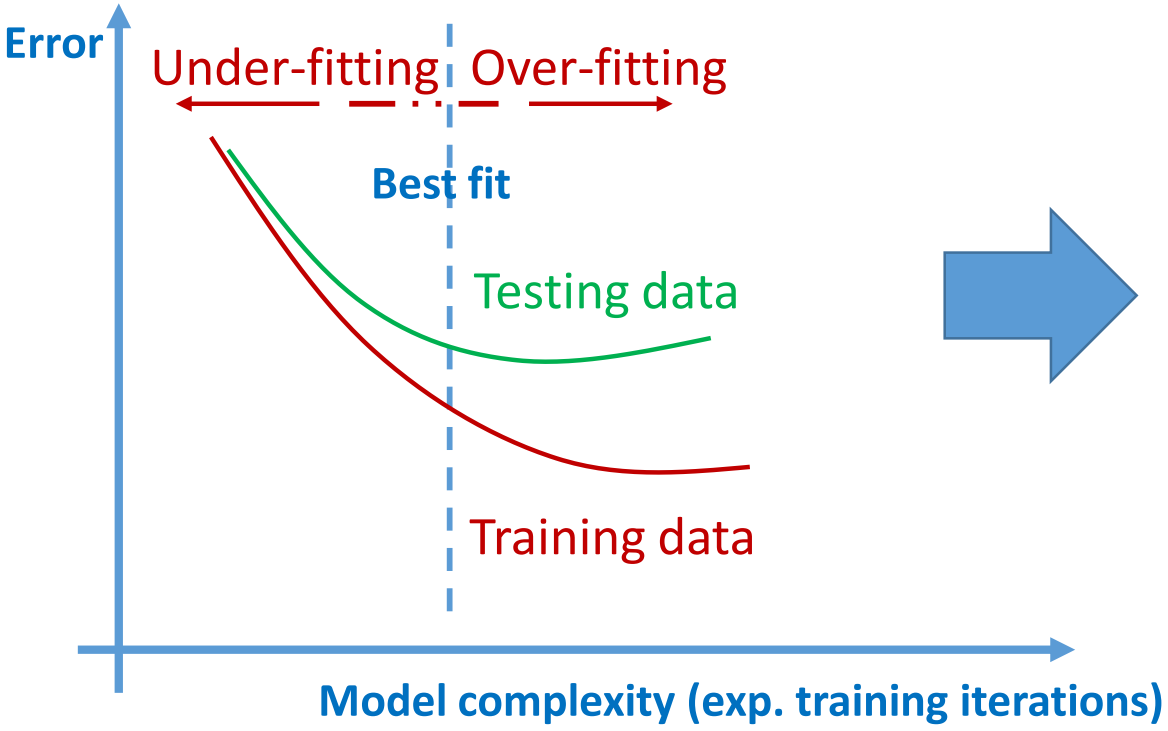
High training error & high testing error

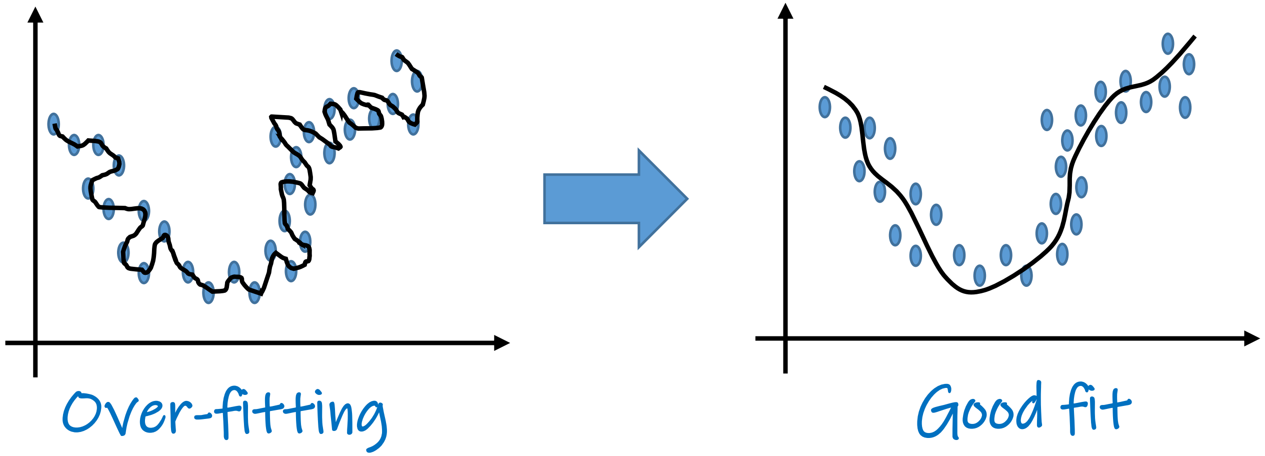
High bias + Low Variance

Solution: Find a more complex model

TAIWAN TECH
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Model fitting: over-fitting





Low training error & high testing error

Low bias + High Variance

Solution: **More training data**: Decrease the number of parameters in the model.
Regularization: penalize certain parts of the parameter or introduce additional constraints.

TAIWAN TECH
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Dimensionality reduction methods

Feature selection

- Backward elimination
- Forward selection
- Random forests
- Missing value ratio
- Low variance filter
- High correlation filter

Find combination of new features

Linear methods

- PCA
- Factor Analysis
- LDA
- Truncated SVD

Non-linear methods

- Kernel PCA
- MDS
- t-SNE
- Isomap
- LLE

Dimensionality reduction sources

Linear methods

PCA: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

FA: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.FactorAnalysis.html

LDA: https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html

Truncated SVD:
https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html#sklearn.decomposition.TruncatedSVD

LLE: https://scikit-learn.org/stable/modules/generated/sklearn.manifold.LocallyLinearEmbedding.html

Dimensionality reduction sources

Non-linear methods

Kernel PCA: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.KernelPCA.html

MDS: https://scikit-learn.org/stable/modules/generated/sklearn.manifold.MDS.html

t-SNE: https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

Isomap: https://scikit-learn.org/stable/modules/generated/sklearn.manifold.Isomap.html

Dimensionality reduction: case example-PCA

How to use PCA

- Define number of components to keep.

```
pca = PCA(n_components=100)
```

- Fit and get transformed feature

```
X_train_pca = pca.fit_transform(X_train.toarray())
```

- Get transformed feature by fitted model

```
X_test_pca = pca.transform(X_test.toarray())
```

Dimensionality reduction: case example-PCA

PCA & Logistic Regression

Case: Complex Features - Simple Model

Data

```
!unzip -q review_polarity.zip
```

```
data_path = "./review_polarity/txt_sentoken/"
```

Gensim Installation

```
!pip install gensim
```

TAIWAN TECH
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Data source: (http://www.cs.cornell.edu/people/pabo/movie-review-data/)

```python
import pandas as pd
import pathlib as pl
def _read_all_reviews_():
    all_reviews = []
    for p in pl.Path(data_path+'pos').iterdir():
        file = open(p, 'r')
        all_reviews.append({'reviews_content': file.read(), 'category': 'positive'})
        file.close()
    for p in pl.Path(data_path+'neg').iterdir():
        file = open(p, 'r')
        all_reviews.append({'reviews_content': file.read(), 'category': 'negative'})
        file.close()
    all_reviews_df = pd.DataFrame(all_reviews)
    return all_reviews_df
print(_read_all_reviews_())
```

Load the data

Output:

```
                                        reviews_content  category
0     the disney studios has its formula for annual ...  positive
1     with storytelling this compelling , who needs ...  positive
2     the only historical figure that has been writt...  positive
3     an astonishingly difficult movie to watch , th...  positive
4     moviemaking is a lot like being the general ma...  positive
...                                                 ...       ...
1995  have you ever been in an automobile accident w...  negative
1996  synopsis : a maniac , crazed by virulent micro...  negative
1997  making your first feature film ain't easy . \n...  negative
1998  it's now the anniversary of the slayings of ju...  negative
1999  whenever u . s . government starts meddling in...  negative

[2000 rows x 2 columns]
```

Build Doc2Vec

```
import pandas as pd
import pathlib as pl
import multiprocessing
import numpy as np
import sklearn.metrics as metrics

from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from gensim.models.doc2vec import TaggedDocument, Doc2Vec
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from gensim.parsing.preprocessing import preprocess_string
from sklearn import utils
from tqdm import tqdm
from sklearn.model_selection import GridSearchCV
```

Get the features (which is non-linear)

Model Packages

Build Doc2Vec (cont'd)

```
class Doc2VecTransformer(BaseEstimator):

    def __init__(self, vector_size=100, learning_rate=0.02, epochs=20):
        self.learning_rate = learning_rate
        self.epochs = epochs
        self._model = None
        self.vector_size = vector_size
        self.workers = multiprocessing.cpu_count()

    def fit(self, df_x, df_y=None):
        tagged_x = [TaggedDocument(preprocess_string(row['reviews_content']), [index]) for index, row in df_x.iterrows()]
        model = Doc2Vec(documents=tagged_x, vector_size=self.vector_size, workers=self.workers)

        for epoch in range(self.epochs):
            model.train(utils.shuffle([x for x in tqdm(tagged_x)]), total_examples=len(tagged_x), epochs=1)
            model.alpha -= self.learning_rate
            model.min_alpha = model.alpha

        self._model = model
        return self

    def transform(self, df_x):
        return np.asmatrix(np.array([self._model.infer_vector(preprocess_string(row['reviews_content']))
                                     for index, row in df_x.iterrows()]))
```

Defining Function

Train Epoch Documentation : *https://www.programcreek.com/python/?CodeExample=train+epoch*

Fitting a Logistic Regression Classifier

```
def train_and_build_model():
    all_reviews_df = _read_all_reviews_()
    train_x_df, test_x_df, train_y_df, test_y_df = train_test_split(all_reviews_df[['reviews_content']],
                                                                    all_reviews_df[['category']])

    pl = Pipeline(steps=[('doc2vec', Doc2VecTransformer(vector_size=220)),
                         ('pca', PCA(n_components=100)),
                         ('logistic', LogisticRegression())
                         ])
    pl.fit(train_x_df[['reviews_content']], train_y_df[['category']])
    predictions_y = pl.predict(test_x_df[['reviews_content']])
    print('Accuracy: ', metrics.accuracy_score(y_true=test_y_df[['category']], y_pred=predictions_y))
```

Machine Learning Pipeline

```
def train_short_range_grid_search():
    all_reviews_df = _read_all_reviews_()
    train_x_df, test_x_df, train_y_df, test_y_df = train_test_split(all_reviews_df[['reviews_content']],
                                                                     all_reviews_df[['category']])

    pl = Pipeline(steps=[('doc2vec', Doc2VecTransformer()),
                         ('pca', PCA()),
                         ('logistic', LogisticRegression())
                         ])

    param_grid = {
        'doc2vec__vector_size': [200, 220, 250],
        'pca__n_components': [50, 75, 100]
    }
    gs_cv = GridSearchCV(estimator=pl, param_grid=param_grid, cv=3, n_jobs=-1,
                         scoring="accuracy")
    gs_cv.fit(train_x_df[['reviews_content']], train_y_df[['category']])

    print("Best parameter (CV score=%0.3f):" % gs_cv.best_score_)
    print(gs_cv.best_params_)
    predictions_y = gs_cv.predict(test_x_df[['reviews_content']])
    print('Accuracy: ', metrics.accuracy_score(y_true=test_y_df[['category']], y_pred=predictions_y))
```

Tuning Hyper Parameters with Grid-Search

Result

```
train_and_build_model()
```

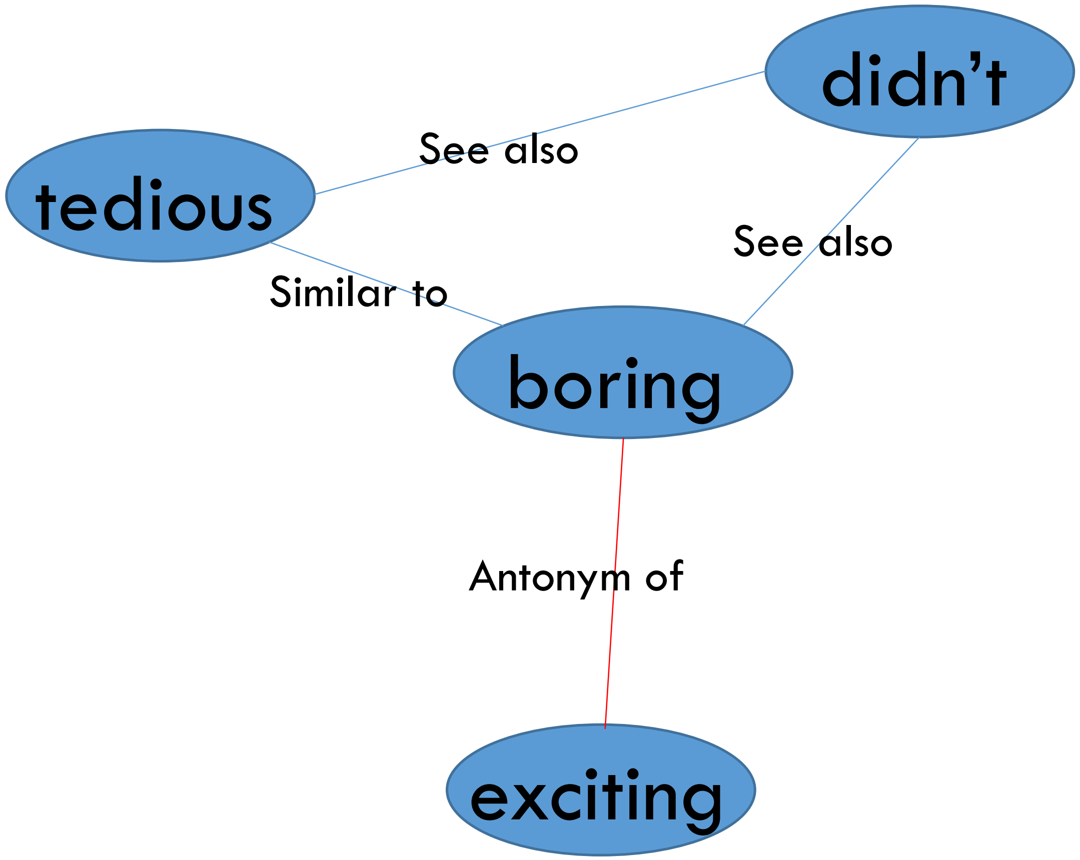
```
100%|████████████████████████████████████████████████████████████████| 1500/150
0 [00:00<00:00, 6297753.75it/s]
100%|████████████████████████████████████████████████████████████████| 1500/150
0 [00:00<00:00, 6198478.82it/s]
100%|████████████████████████████████████████████████████████████████| 1500/150
0 [00:00<00:00, 6310387.16it/s]
/home/deeplab307-170/miniconda3/envs/ghaluh/lib/python3.9/site-packages/sklearn/utils/validation.py:593: FutureWarning: np.matr
ix usage is deprecated in 1.0 and will raise a TypeError in 1.2. Please convert to a numpy array with np.asarray. For more info
rmation see: https://numpy.org/doc/stable/reference/generated/numpy.matrix.html
  warnings.warn(
/home/deeplab307-170/miniconda3/envs/ghaluh/lib/python3.9/site-packages/sklearn/utils/validation.py:993: DataConversionWarning:
A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using rav
el().
  y = column_or_1d(y, warn=True)
```

Output: Accuracy: 0.552

Sentiment analysis task: Unsupervised

- The main idea behind this approach is that negative and positive words usually are surrounded by similar words.

Wordnet



Source: https://www.nltk.org/howto/wordnet.html

Sentiment analysis task: case example (K-Means)

Pre-processing and data cleaning steps:

- Dropping rows with missing (NaN) values.
- Dropping duplicated rows.
- Removing rows with rate equal to 0, as it contained some error, probably from the data gathering phase.
- Replacing polish letters with use of unidecode package.
- Replacing all non-alphanumeric signs, punctuation signs, and duplicated white spaces with a single white space.
- Retaining all rows with sentences with a length of at least 2 words.

Sentiment analysis task: case example (K-Means)

```
import re
import logging
import numpy as np
import pandas as pd
import multiprocessing

from re import sub
from time import time
from unidecode import unidecode
from gensim.models import Word2Vec
from collections import defaultdict
from gensim.models import KeyedVectors
from gensim.test.utils import get_tmpfile
from gensim.models.phrases import Phrases, Phraser

logging.basicConfig(format="%(levelname)s - %(asctime)s: %(message)s", datefmt= '%H:%M:%S', level=logging.INFO)
```

Import package

Inspired by: https://www.kaggle.com/code/pierremegret/gensim-word2vec-tutorial/notebook

TAITWAN TECH
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Sentiment analysis task: case example (K-Means)

Data cleaning

```
file = pd.read_csv("polish_sentiment_dataset.csv")
file_cleaned = file.dropna().drop_duplicates().reset_index(drop=True).rename(columns={'description':'title'})
```

```
file_cleaned.rate.value_counts()/len(file_cleaned)
```

```
file_cleaned[file_cleaned.rate==0]
```

```
file_cleaned = file_cleaned[file_cleaned.rate!=0]
```

```
file_cleaned.rate.value_counts()/len(file_cleaned)
```

Sentiment analysis task: case example (K-Means)

```
def text_to_word_list(text, remove_polish_letters):
    ''' Pre process and convert texts to a list of words
    method inspired by method from eliorc github repo: https://github.com/eliorc/Medium/blob/master/MaLSTM.ipynb'''
    text = remove_polish_letters(text)
    text = str(text)
    text = text.lower()

    # Clean the text
    text = sub(r"[^A-Za-z0-9^,!?.\/'+]", " ", text)
    text = sub(r"\+", " plus ", text)
    text = sub(r",", " ", text)
    text = sub(r"\.", " ", text)
    text = sub(r"!", " ! ", text)
    text = sub(r"\?", " ? ", text)
    text = sub(r"'", " ", text)
    text = sub(r":", " : ", text)
    text = sub(r"\s{2,}", " ", text)

    text = text.split()

    return text
```

Morfologik (stemming process for Polish language): https://github.com/dmirecki/pyMorfologik

Spell checker: https://thomasdecaux.medium.com/build-a-spell-checker-with-word2vec-data-with-python-5438a9343afd

Sentiment analysis task: case example (K-Means)

```
file_cleaned.title = file_cleaned.title.apply(lambda x: text_to_word_list(x, unidecode))
```

```
file_model = file_cleaned.copy()
file_model = file_model[file_model.title.str.len()>1]
```

```
sent = [row for row in file_model.title]
phrases = Phrases(sent, min_count=1, progress_per=50000)
bigram = Phraser(phrases)
sentences = bigram[sent]
sentences[1]
```

As Phrases() takes a list of list of words as input

Creates the relevant phrases from the list of sentences

Transform the corpus based on the bigrams detected

TAIWAN TECH
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Sentiment analysis task: case example (K-Means)

Build the vocab

```
w2v_model = Word2Vec(min_count=3,
                     window=4,
                     vector_size=300,
                     sample=1e-5,
                     alpha=0.03,
                     min_alpha=0.0007,
                     negative=20,
                     workers=multiprocessing.cpu_count()-1)

start = time()

w2v_model.build_vocab(sentences, progress_per=50000)

print('Time to build vocab: {} mins'.format(round((time() - start) / 60, 2)))
```

Sentiment analysis task: case example (K-Means)

Note for build the vocab

- min count = 3 - remove most unusual words from training embeddings, like words 'ssssuuuuuuuppppppeeeeeerrrr', which actually stands for 'super', and doesn't need additional training.
- window = 4 - Word2Vec model will learn to predict given word from up to 4 words to the left, and up to 4 words to the right.
- vector_size = 300 - size of hidden layer used to predict surroundings of embedded word, which also stands for dimensions of trained embeddings.
- sample = 1e-5 - probability baseline for subsampling most frequent words from surrounding of embedded word.
- negative = 20 - number of negative words that will have their corresponding weights updated while training on specific training example, along with positive word (ones that shouldn't have been predicted while modeling selected pair of words).
- negative sampling aims at maximizing the similarity of the words in the same context and minimizing it when they occur in different contexts.

Sentiment analysis task: case example (K-Means)

Train the model

```
start = time()

w2v_model.train(sentences, total_examples=w2v_model.corpus_count, epochs=30, report_delay=1)

print('Time to train the model: {} mins'.format(round((time() - start) / 60, 2)))

w2v_model.init_sims(replace=True)
```

```
w2v_model.save("word2vec.model")
```

Sentiment analysis task: case example (K-Means)

Exporting preprocessed dataset for further steps (with replaced bigrams)

```
file_export = file_model.copy()
file_export['old_title'] = file_export.title
file_export.old_title = file_export.old_title.str.join(' ')
file_export.title = file_export.title.apply(lambda x: ' '.join(bigram[x]))
file_export.rate = file_export.rate.astype('int8')
```

```
file_export[['title', 'rate']].to_csv('cleaned_dataset.csv', index=False)
```

TAIWAN TECH
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Sentiment analysis task: case example (K-Means)

K-Means Clustering

```
import numpy as np
import pandas as pd
from gensim.models import Word2Vec
from sklearn.cluster import KMeans
```

```
word_vectors = Word2Vec.load("../preprocessing_and_embeddings/word2vec.model").wv
```

```
model = KMeans(n_clusters=2, max_iter=1000, random_state=True, n_init=50).
fit(X=word_vectors.vectors.astype('double'))
```

Note: n_init= 50, to presumably prevent the algorithm from choosing wrong starting centroid coordinates, that would lead the algorithm to converge to not optimal clusters.
1000 iterations of reassigning points to clusters.

Sentiment analysis task: case example (K-Means)

```
word_vectors.similar_by_vector(model.cluster_centers_[1], topn=10, restrict_vocab=None)
```

Checking the word vectors are most similar in terms of cosine similarity to coordinates of first cluster.

Output:

```
[('pelen_profesjonalim', 0.9740794897079468),
 ('superszybko_supersprawnie', 0.97325599193573),
 ('bardzon', 0.9731361865997314),
 ('duzu_wybor', 0.971358060836792),
 ('ladne_garnki', 0.9698898196220398),
 ('najlpszym_porzadku', 0.9690271615982056),
 ('wieloma_promocjami', 0.9684171676635742),
 ('cudowna_wspolpraca', 0.9679782390594482),
 ('pelen_profesjonaliz', 0.9675517678260803),
 ('przyzwoicie_cenowo', 0.9674378633499146)]
```

```
positive_cluster_index = 1
positive_cluster_center = model.cluster_centers_[positive_cluster_index]
negative_cluster_center = model.cluster_centers_[1-positive_cluster_index]
```

Sentiment analysis task: case example (K-Means)

Assigning each word sentiment score — negative or positive value (-1 or 1) based on the cluster to which they belong

```
words = pd.DataFrame(word_vectors.index_to_key)
words.columns = ['words']
words['vectors'] = words.words.apply(lambda x: word_vectors[f'{x}'])
words['cluster'] = words.vectors.apply(lambda x: model.predict([np.array(x)]))
words.cluster = words.cluster.apply(lambda x: x[0])
```

Sentiment analysis task: case example (K-Means)

```
words['cluster_value'] = [1 if i==positive_cluster_index else -1 for i in words.cluster]
words['closeness_score'] = words.apply(lambda x: 1/(model.transform([x.vectors]).min()), axis=1)
words['sentiment_coeff'] = words.closeness_score * words.cluster_value
```

weighting how potentially positive/negative they are

properly weighting the distance from both clusters

TAIWAN TECH
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Sentiment analysis task: case example (K-Means)

Words based on sentiment coefficient

```python
words.head(10)
```

output

	words	sentiment_coeff
8603	bezzwlocznie	-0.970751
61219	przyczyny_opoznienia	-1.275320
38906	delikatnie_uszkodzony	-1.276431
38670	chetnie_sluza	1.160198
50968	mniej_popularne	1.190393
11128	czterokrotnie	-1.038108
50111	pekniete	-1.144333
14846	expresowe_zalatwienie	1.534966
25455	/nie	-1.149102
5587	samoczynnie	-1.337816

```python
words[['words', 'sentiment_coeff']].to_csv('sentiment_dictionary.csv', index=False)
```

Case example: K-Means for prediction

1. Data Preparation
2. Choose the number of Cluster (K)
3. Apply K-Means clustering
4. Cluster assignment: assign each data point to the nearest cluster centroid
5. Feature Engineering (optional): use the cluster assignments as features for prediction
6. Train a prediction model
7. Evaluation
8. Prediction
9. Iterate & Refine

Case example: K-Means for prediction

Import package

```
import numpy as np
import pandas as pd
from IPython.display import display
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score, f1_score
```

Load the data

```
final_file = pd.read_csv('preprocessing_and_embeddings/cleaned_dataset.csv')
```

```
sentiment_map = pd.read_csv('KMeans_clustering//sentiment_dictionary.csv')
sentiment_dict = dict(zip(sentiment_map.words.values, sentiment_map.sentiment_coeff.values))
```

Case example: K-Means for prediction

TF*IDF Weighting

```
file_weighting = final_file.copy()
```

```
tfidf = TfidfVectorizer(tokenizer=lambda y: y.split(), norm=None)
tfidf.fit(file_weighting.title)
features = pd.Series(tfidf.get_feature_names())
transformed = tfidf.transform(file_weighting.title)
```

Case example: K-Means for prediction

```python
def create_tfidf_dictionary(x, transformed_file, features):
    '''
    create dictionary for each input sentence x, where each word has assigned its tfidf score
    x - row of dataframe, containing sentences, and their indexes,
    transformed_file - all sentences transformed with TfidfVectorizer
    features - names of all words in corpus used in TfidfVectorizer

    '''
    vector_coo = transformed_file[x.name].tocoo()
    vector_coo.col = features.iloc[vector_coo.col].values
    dict_from_coo = dict(zip(vector_coo.col, vector_coo.data))
    return dict_from_coo

def replace_tfidf_words(x, transformed_file, features):
    '''
    replacing each word with it's calculated tfidf dictionary with scores of each word
    x - row of dataframe, containing sentences, and their indexes,
    transformed_file - all sentences transformed with TfidfVectorizer
    features - names of all words in corpus used in TfidfVectorizer
    '''
    dictionary = create_tfidf_dictionary(x, transformed_file, features)
    return list(map(lambda y:dictionary[f'{y}'], x.title.split()))
```

Replacing words in sentences with their TF*IDF scores

TAIWAN TECH
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Case example: K-Means for prediction

Adding time to see the effectiveness

```
%%time
replaced_tfidf_scores = file_weighting.apply(lambda x: replace_tfidf_words(x, transformed, features), axis=1)
```

```
CPU times: user 1min 27s, sys: 120 ms, total: 1min 27s
Wall time: 1min 27s
```

Case example: K-Means for prediction

Replacing words in sentences with their sentiment score, to obtain 2 vectors for each sentence

```python
def replace_sentiment_words(word, sentiment_dict):
    '''
    replacing each word with its associated sentiment score from sentiment dict
    '''
    try:
        out = sentiment_dict[word]
    except KeyError:
        out = 0
    return out
```

```python
replaced_closeness_scores = file_weighting.title.apply(lambda x: list(
    map(lambda y: replace_sentiment_words(y, sentiment_dict), x.split())))
```

TAIWAN TECH
NATIONAL TAIWAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Case example: K-Means for prediction

Merging both previous steps and getting the predictions:

```
replacement_df = pd.DataFrame(data=[replaced_closeness_scores, replaced_tfidf_scores,
                                    file_weighting.title, file_weighting.rate]).T
replacement_df.columns = ['sentiment_coeff', 'tfidf_scores', 'sentence', 'sentiment']
replacement_df['sentiment_rate'] = replacement_df.apply(lambda x: np.array(x.loc['sentiment_coeff'])
                                                       @ np.array(x.loc['tfidf_scores']), axis=1)
replacement_df['prediction'] = (replacement_df.sentiment_rate>0).astype('int8')
replacement_df['sentiment'] = [1 if i==1 else 0 for i in replacement_df.sentiment]
```

Note: The dot product of such two sentence-vectors indicated whether the overall sentiment was positive or negative (if the dot product was positive, the sentiment was positive, and in the opposite case negative).

Case example: K-Means for prediction

Evaluation

We have predicted class to perform F1-score

```
predicted_classes = replacement_df.prediction
y_test = replacement_df.sentiment

conf_matrix = pd.DataFrame(confusion_matrix(replacement_df.sentiment, replacement_df.prediction))
print('Confusion Matrix')
display(conf_matrix)

test_scores = accuracy_score(y_test,predicted_classes), precision_score(y_test, predicted_classes),
recall_score(y_test, predicted_classes), f1_score(y_test, predicted_classes)

print('\n \n Scores')
scores = pd.DataFrame(data=[test_scores])
scores.columns = ['accuracy', 'precision', 'recall', 'f1']
scores = scores.T
scores.columns = ['scores']
display(scores)
```

The main reason using F1-score because classes in dataset were highly imbalanced.

Case example: K-Means for prediction

Output:

Confusion Matrix

	0	1
0	9529	300
1	121610	513792

	scores
accuracy	0.811060
precision	0.999416
recall	0.808609
f1	0.893945

Assignment Sentiment Analysis

Description

1. Dataset: Review_polarity dataset
   - Train
   - Test
2. Objective:
   - Implements a vector space model or any embedding model to build a data representation for the classification model. Provide the best word representation (vocabulary) to get a better performance of your model.
   - Select any kind of classification model to enhance your model performance.
   - Feature Engineering is a Plus +.
3. The evaluation metric is Accuracy.

Assignment Sentiment Analysis

Requirements

1. Print out the classification results.
2. Submit your works to Kaggle at the link below:
https://www.kaggle.com/competitions/ntust-text-classification/overview
3. The maximum of daily submissions is 20
4. Write your ID on Kaggle as a team name:
F11115111_NAME
5. Write your report, and it must contain: methods, source codes, and results. Please submit your report on the Moodle system.
6. Deadline: April 30, 2024

Assignment Sentiment Analysis

5. Scoring: Students only get a score when they complete the requirements.
   - We provide a baseline to the Kaggle, thus:
     - The 1st rank on the Leaderboard will receive 100 points.
     - The 2nd-10th ranks on the Leaderboard will receive 85 points.
     - Reach the baseline, and lower than 10 ranks will receive 70 points.
     - Below the baseline, get 0 point.

submission.csv

	A	B
1	Row	Label
2	1	negative
3	2	positive
4	3	negative
5	4	negative
6	5	negative
7	6	positive
8	7	positive
9	8	negative
10	9	negative

Thank you

Q & A