Rucha Dubbewar A20373886
Reona Cerejo A20375246

Keyword Extraction of News Dataset

Abstract

This project explores various techniques in managing a large dataset of news in order to produce a real practical point. The algorithms being focused are Naïve Bayes, SVC and Logistic Regression. Using these algorithms, news aggregator dataset will be analysed to obtain some metrics in order to demonstrate their efficiency and working.

Background & Motivation:

Now-a-days, a common problem faced by many is keyword extraction as there is a rapid growth of data. It so happens that people just read the headlines in order to find out what the news is about. Sometimes, the headlines cannot convey the news category. Life can be made easy if the news articles are categorized based on the category.

Introduction:

Keyword extraction is tasked with automatic identification of terms that best describe the subject or category of document. The same logic is used here to extract the category of news article using specific text extraction algorithms. Dataset used in this project is UCI News Aggregator. This dataset contains headlines, URLs, and categories for 422,937 news stories collected by a web aggregator between March 10th, 2014 and August 10th, 2014.

News categories included in this dataset include business; science and technology; entertainment; and health. Different news articles that refer to the same news item (e.g., several articles about recently released employment statistics) are also categorized together.

The project will help to analyse and as well as get the execution time required to extract news of a particular category. The dataset has four news category: Business, Technology, Entertainment and Health. Using Naïve Bayes, SVM and Logistic regression, time required to obtain necessary information by usage of classifiers is determined. Based on result, best algorithm giving good results in less time is selected. This will be the baseline of the project.

Data Manipulation:

The dataset which we have taken does not have category column. We analyze the data and based on the news headline and content we derive the category column and then based on the category, the news articles can be categorized. In order to determine and make analysis easy, dataset is divided into small sets depending on their news category. For training and testing purpose, the dataset is split into 80:20 ratio. A vectorized representation of document is used. The document is represented in the form of a vector, whose dimension is equal to number of all available words. The code has an enhanced version of the model. That takes into the account the relevance of the title.

Naïve Bayes

Using Naïve Bayes on the selected dataset having 422420 data rows, with cross validation, following results were obtained with our baseline being considered.

Category	Execution time
Business	1.89 s
Technology	1.56 s
Entertainment	2.71 s
Health	1.12 s

The total training time for Naïve Bayes with cross validation is 7.28 s

At glance at the table suggest that Naïve Bayes is a good predictor. Naïve Bayes is originally used for dataset with high dimensionality. Being relative robust, easy to implement, fast and accurate, naïve Bayes is used in used in many different fields. However, Multinomial Naïve Bayes shows more fast results. Below are results of Multinomial Naïve Bayes approach.

Category	Testing Time	Change in Testing Time
Business	136Ms	-1754 ms
Technology	138 ms	-1422 ms
Entertainment	185 ms	-2525 ms
Health	43.1 ms	-1076.9 ms

The total execution time for Multinomial Naïve Bayes is 502.1 ms

Noise was effectively removed with multinomial Naïve Bayes. Though the same machine was used for all runs, not all transient variables were accounted. Therefore, between the two different Naïve Bayes contours, their testing times were roughly equivalent, though the smaller dictionary in the filtered Naïve Bayes run did produce a noticeably faster training time compared to the baseline. However, it turns out that execution time of Multinomial Naïve Bayes is a good predictor.

SVC

SVC results are determined on the basis of accuracy. SVC is a Support Vector classification model with linear kernel. SVM algorithms work well with high dimensionality. SVC has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples. SVC is used implements an alternative multi-class strategy. Below are the results for SVC approach.

Logistic Regression:

Logistic Regression is basically a statistical method wherein we analyse the dataset based on 2 or more independent variables to get an output. In our dataset, the two predictor variables are Title and Category.

Here we perform logistic regression using one versus all strategy for fitting classifier. Foe each of the classifiers (in this case Business, Technology, Entertainment, Health), the class is fitted against all the other classes.

We calculated the execution time for each category using Logistic Regression on the selected dataset having 422420 data rows the following results were obtained with our baseline being considered.

Category	Execution Time
Business	1.36 ms
Technology	59 ms
Entertainment	75.1 ms
Health	32 ms

Logistic regression is fast and provides good accuracy. Results can be compared as follows based on accuracy.

Accuracy:

Naïve Bayes	Logistic Regression	SVC
1.0	1.0	0.95038

Future Improvement

For Naïve Bayes and Logistic Regression, we can create tokens based on tf-idf and calculate the top 10 words from the document. Baes on the words, we can then calculate the execution time and accuracy for each word/tag. Also, using LinearSVC, if we are given a headline then the system can predict the article related to the headline.

Conclusion:

Naive Bayes Classifier (NBC) and Support Vector Machine (SVM) have different options including the choice of kernel function for each. They are both sensitive to parameter optimization (i.e. different parameter selection can significantly change their output). So, if you have a result showing that NBC is performing better than SVM. This is only true for the selected parameters. However, for another parameter selection, you might find SVM is performing better. There is no single answer about which is the best classification method for a given dataset. Different kinds of classifiers should be always considered for a comparative study over a given dataset. Given the properties of the dataset, you might have some clues that may give preference to some methods. However, it would still be advisable to experiment with all, if possible.

References

http://cs229.stanford.edu/proj2013/Yan-ApplicationsofMachineLearningonKeywordExtractionofLargeDatasets.pdf

Bird, Steven, Edward Loper, and Ewan Klein (2009). O'Reilly Media Inc.

177 178 R.-E. Fan, K.-W. Change, C.-J. Hsieh, C.-R. Wang, and C.-J. Lin. LIBLINEAR: A Library for Large Linear Classification, Journal of Machine Learning Research 9(2008), 1871-1874.

https://archive.ics.uci.edu/ml/datasets/News+Aggregator

Appendix:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re
import string

from IPython.core.display import display
from sklearn.svm import LinearSVC
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import accuracy_score

import requests
import pandas as pd
from datetime import datetime
from tqdm import tqdm
from matplotlib import pyplot as plt
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords
from string import punctuation
rom collections import Counter
stop = set(stopwords.words('english'))

data = pd.read_csv("C:/Users/7000/Desktop/Sem3/Machine Learning/project/News-
aggregator.csv")
data.head()

dataset_b = data[data['CATEGORY'].str.contains('b')]
dataset_t = data[data['CATEGORY'].str.contains('t')]
dataset_e = data[data['CATEGORY'].str.contains('e')]
dataset_m = data[data['CATEGORY'].str.contains('m')]

s_b = dataset_b['HOSTNAME'].value_counts()
for hostname in s_b.keys():
        target_website = dataset_b.loc[dataset_b['HOSTNAME'].isin([hostname])]
        vc = target_website['CATEGORY'].value_counts()
        if 'm' in vc:
                if vc['m'] > 500:
                        print('Hostname: %s , %s' %(hostname, str(vc)))

s_t = dataset_t['HOSTNAME'].value_counts()
for hostname in s_t.keys():
        target_website = dataset_t.loc[dataset_t['HOSTNAME'].isin([hostname])]
         vc = target_website['CATEGORY'].value_counts()
        if 'm' in vc:
                 if vc['m'] > 500:
                        print('Hostname: %s , %s' %(hostname, str(vc)))

s_e = dataset_e['HOSTNAME'].value_counts()
for hostname in s_e.keys():
        target_website = dataset_e.loc[dataset_e['HOSTNAME'].isin([hostname])]
         vc = target_website['CATEGORY'].value_counts()
        if 'm' in vc:
                 if vc['m'] > 500:
                        print('Hostname: %s , %s' %(hostname, str(vc)))



s_m = dataset_m['HOSTNAME'].value_counts()
for hostname in s_m.keys():
        target_website = dataset_m.loc[dataset_m['HOSTNAME'].isin([hostname])]
        vc = target_website['CATEGORY'].value_counts()
        if 'm' in vc:
                if vc['m'] > 500:
                        print('Hostname: %s , %s' %(hostname, str(vc)))

dataset_b['CATEGORY'].unique()
s_b = dataset_b['HOSTNAME'].value_counts()
print(s_b.keys())
target_website = dataset_b.loc[dataset_b['HOSTNAME'].isin([r'www.marketwatch.com'])]
target_website

dataset_t['CATEGORY'].unique()
s_t = dataset_t['HOSTNAME'].value_counts()
print(s_t.keys())
target_website = dataset_t.loc[dataset_t['HOSTNAME'].isin([r'www.marketwatch.com'])]
target_website

dataset_e['CATEGORY'].unique()
s_e = dataset_e['HOSTNAME'].value_counts()
print(s_e.keys())
target_website = dataset_e.loc[dataset_e['HOSTNAME'].isin([r'www.marketwatch.com'])]
target_website

dataset_m['CATEGORY'].unique()
s_m = dataset_m['HOSTNAME'].value_counts()
print(s_m.keys())
target_website = dataset_m.loc[dataset_m['HOSTNAME'].isin([r'www.marketwatch.com'])]
target_website

def clean_text(s_b):
        s_b = s_b.lower()
        for ch in string.punctuation:
                s_b = s_b.replace(ch, " ")
        s_b = re.sub("[0-9]+", "||DIG||",s_b)
        s_b = re.sub(' +',' ', s_b)
        return s_b

dataset_b['TEXT'] = [clean_text(s_b) for s_b in dataset_b['TITLE']]

def clean_text(s_t):
        s_t = s_t.lower()
        for ch in string.punctuation:
                s_t = s_t.replace(ch, " ")
        s_t = re.sub("[0-9]+", "||DIG||",s_t)
        s_t = re.sub(' +',' ', s_t)
        return s_t

dataset_t['TEXT'] = [clean_text(s_t) for s_t in dataset_t['TITLE']]

def clean_text(s_e):
        s_e = s_e.lower()
        for ch in string.punctuation:
                s_e = s_e.replace(ch, " ")
        s_e = re.sub("[0-9]+", "||DIG||",s_e)
         s_e = re.sub(' +',' ', s_e)
        return s_e

dataset_e['TEXT'] = [clean_text(s_e) for s_e in dataset_e['TITLE']]

def clean_text(s_m):
        s_m = s_m.lower()
        for ch in string.punctuation:
                s_m = s_m.replace(ch, " ")
        s_m = re.sub("[0-9]+", "||DIG||",s_m)
        s_m = re.sub(' +',' ', s_m)
        return s_m

dataset_m['TEXT'] = [clean_text(s_m) for s_m in dataset_m['TITLE']]
```

Naïve Bayes:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer, TfidfTransformer
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.model_selection import cross_val_score

# Business

vectorizer = CountVectorizer()
xb = vectorizer.fit_transform(dataset_b['TEXT'])

encoder = LabelEncoder()
yb = encoder.fit_transform(dataset_b['CATEGORY'])

# train and test sets
xb_train, xb_test, yb_train, yb_test = train_test_split(xb, yb, test_size=0.2)

%%time
nbb = MultinomialNB()
nbb.fit(xb_train, yb_train)
```

```
%%time
results_nbb_cv = cross_val_score(nbb, xb_train, yb_train, cv=10)
print(results_nbb_cv.mean())

nbb.score(xb_test, yb_test)

xb_test_pred = nbb.predict(xb_test)
confusion_matrix(yb_test, xb_test_pred)

# Technology

xt = vectorizer.fit_transform(dataset_t['TEXT'])
yt = encoder.fit_transform(dataset_t['CATEGORY'])
xt_train, xt_test, yt_train, yt_test = train_test_split(xt, yt, test_size=0.2)

%%time
nbt = MultinomialNB()
nbt.fit(xt_train, yt_train)

%%time
results_nbt_cv = cross_val_score(nbt, xt_train, yt_train, cv=10)
print(results_nbt_cv.mean())

nbt.score(xt_test, yt_test)

xt_test_pred = nbt.predict(xt_test)
confusion_matrix(yt_test, xt_test_pred)

# Entertainment

xe = vectorizer.fit_transform(dataset_e['TEXT'])
ye = encoder.fit_transform(dataset_e['CATEGORY'])
xe_train, xe_test, ye_train, ye_test = train_test_split(xe, ye, test_size=0.2)

%%time
nbe = MultinomialNB()
nbe.fit(xe_train, ye_train)

%%time
results_nbe_cv = cross_val_score(nbe, xe_train, ye_train, cv=10)
print(results_nbe_cv.mean())

nbe.score(xe_test, ye_test)

xe_test_pred = nbe.predict(xe_test)
confusion_matrix(ye_test, xe_test_pred)

#Health

xm = vectorizer.fit_transform(dataset_m['TEXT'])
ym = encoder.fit_transform(dataset_m['CATEGORY'])
xm_train, xm_test, ym_train, ym_test = train_test_split(xm, ym, test_size=0.2)

%%time
nbm = MultinomialNB()
nbm.fit(xm_train, ym_train)
```

```
%%time
results_nbm_cv = cross_val_score(nbm, xm_train, ym_train, cv=10)
print(results_nbm_cv.mean())

nbm.score(xm_test, ym_test)

xm_test_pred = nbm.predict(xm_test)
confusion_matrix(ym_test, xm_test_pred)

# Logistic Regression:

from sklearn.linear_model import LogisticRegression
from sklearn.multiclass import OneVsRestClassifier

# Business

# Instantiate classifier
clfb = OneVsRestClassifier(LogisticRegression())

# Fit the classifier
clfb.fit(xb_train, yb_train)

print("Accuracy: {}".format(clfb.score(xb_test, yb_test)))


# Technology

# Instantiate classifier
clft = OneVsRestClassifier(LogisticRegression())

clft.fit(xt_train, yt_train)

print("Accuracy: {}".format(clft.score(xt_test, yt_test)))


# Entertainment

# Instantiate classifier
clfe = OneVsRestClassifier(LogisticRegression())

clfe.fit(xe_train, ye_train)

print("Accuracy: {}".format(clfe.score(xe_test, ye_test)))


# Health

# Instantiate classifier
clfm = OneVsRestClassifier(LogisticRegression())

clfm.fit(xm_train, ym_train)

print("Accuracy: {}".format(clfm.score(xm_test, ym_test)))

xb_test_clv_pred = clfb.predict(xb_test)
confusion_matrix(yb_test, xb_test_clv_pred)
```

```
xt_test_clv_pred = clft.predict(xt_test)
confusion_matrix(yt_test, xt_test_clv_pred)

xe_test_clv_pred = clfe.predict(xe_test)
confusion_matrix(ye_test, xe_test_clv_pred)

xm_test_clv_pred = clfm.predict(xm_test)
confusion_matrix(ym_test, xm_test_clv_pred)


# SVC

X_raw, Y = data['TITLE'], data['CATEGORY']

vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(X_raw)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

clf = LinearSVC()
clf.fit(X_train, Y_train)

Y_pred = clf.predict(X_test)

accuracy_score(Y_test, Y_pred)
```