Classification and Analysis of text data

Copyright 2018 The BUPT Zhengyuan Zhu.

Licensed under the Apache License, Version 2.0 (the "License").

View all sources on GitHub

Affilication: BUPT

Author1:824zzy(计算机学院-2018140455-朱正源)

Author2:Regulusyy(计算机学院-2018140506-杨莹)

References

- CNN wiki
- RNN wiki
- Support vector machine wiki
- python3:csv文件的读写
- pyhanlp 分词与词性标注
- python结巴分词、jieba加载停用词表
- 中文常用停用词表
- python读取和存储dict()与.json格式文件
- Python爬虫之爬取动态页面数据
- 824zzy（朱正源）的微博爬虫
- 6 Easy Steps to Learn Naive Bayes Algorithm (with codes in Python and R)
- https://medium.com/jatana/report-on-text-classification-using-cnn-rnn-han-f0e887214d5f
- Naive Bayes Tutorial: Naive Bayes Classifier in Python
- Let's implement a Gaussian Naive Bayes classifier in Python
- Support Vector Machines with Scikit-learn
- python中sklearn实现交叉验证
- Practical Text Classification With Python and Keras
- Text Preprocessing - Keras

Crawler demo：

In this part, we will show you how to build a crawler from scratch. It is a little tricky but not difficult enough.

Setup packages to Colab Virtual Machine.

- scrapy: especially for using XPATH to parse the html tree
- tqdm: a common tool for displaying the processing of ForLoop
- retrying: package for preventing connetct lose
- grequests: speed up for efficiency of crawler
- requests: basic package to get html

```
!pip install scrapy
!pip install tqdm
!pip install retrying
!pip install grequests
!pip install requests
```

```
Collecting scrapy
  Downloading https://files.pythonhosted.org/packages/5d/12/a6197eaf97385e96
    100% |████████████████████████████████| 256kB 11.7MB/s
```

Import packages

```
Collecting Twisted>=13.1.0 (from scrapy)
```

```python
import requests
import grequests
import scrapy
from tqdm import tqdm
from retrying import retry
import time
import random
import json
import csv
import pickle
```

```
/usr/local/lib/python3.6/dist-packages/grequests.py:21: MonkeyPatchWarning: M
  curious_george.patch_all(thread=False, select=False)

Requirement already satisfied: lxml in /usr/local/lib/python3.6/dist-packages
```

Auxiliary functions for crawler

Generate Headers Randomly

```
Downloading https://files.pythonhosted.org/packages/19/17/1d198a6aaa9aa459
```

```python
userAgent_file = [
"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.1 (KHTML, like Gecko) Chro
"Mozilla/5.0 (Windows NT 6.1; WOW64; rv:6.0) Gecko/20100101 Firefox/6.0",
"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/534.50 (KHTML, like Gecko) Ver
"Opera/9.80 (Windows NT 6.1; U; zh-cn) Presto/2.9.168 Version/11.50",
"Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0; .N
"Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2;
"Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; GTB7.0)",
"Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)",
"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)",
"Mozilla/5.0 (Windows; U; Windows NT 6.1; ) AppleWebKit/534.12 (KHTML, like Gec
"Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/5.0; SLCC2;
"Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/5.0; SLCC2;
"Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) AppleWebKit/534.3 (KHTML, like
"Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0; SLCC2;
"Mozilla/5.0 (Windows NT 6.1) AppleWebKit/535.1 (KHTML, like Gecko) Chrome/13.0
"Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/5.0; SLCC2;
"Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)",
]
class Headers:
    @staticmethod
    def getHeaders():
        userAgentList = []
        for line in userAgent_file:
            userAgentList.append({
                'User-Agent': line.strip(),
                #'User-Agent': 'Mozilla/5.0 (compatible; Googlebot/2.1; +http:/
                'Referer': 'http://cn.bing.com/',
                'X-Forwarded-For': '%s.%s.%s.%s' % (
                random.randint(50, 250), random.randint(50, 250), random.randin
                'CLIENT-IP': '%s.%s.%s.%s' % (
                random.randint(50, 250), random.randint(50, 250), random.randin
            })
        userAgent = random.sample(userAgentList, 1)
        return userAgent[0]

# test case
print(Headers.getHeaders())
```

```
{'User-Agent': 'Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Tri
```

Base class for Crawlers

```python
import json



class base_class(object):
  def __init__(self, website, website_url, header=None):
    """

    """
    self.cookies = {}
    self.website = website
    self.website_url = website_url
    self.header = header
    self.entrance_html = requests.get(self.website_url).content
    self.sections = self.section_list()

  def section_list(self):
    pass

  def section_urls(self):
    pass

  def parse_body(self):
    pass

  def ajax_url(self):
    pass

  def article_urls_dict(self):
    pass

  def display(self):
    pass
```

Jiandan Crawler

```python
class Jiandan(base_class):
  def __init__(self, **kwargs):
    super(Jiandan, self).__init__(**kwargs)

  def sec_subsec_dict(self, sections, sub_sections):
    """build a dictionary according to sections and subsections

    # Arguments:
      sections: a list contains sections' string.
      sub_sections: a list contains sub_sections' string.

    # Returns:
      sec_subsec_dict: a dictionary whose section as key and sub_sections as va

    """
    sec_subsec_dict = {}
    for index, section in enumerate(sections):
      sec_subsec_dict[section] = [sub_sections[index + 7 * i] for i in range(6)
    return sec_subsec_dict


  def section_list(self):
    """get section from each website. Sections could be seem as the label of
       classification in the case of saving manpower.

    # Arguments:
```

```
      None:

    # Returns:
      section: a list contains section name that is a str
    """
    section = scrapy.Selector(text=self.entrance_html).xpath('//*[@id="header"]
    return section

  def sub_section_list(self):
    sub_sections = scrapy.Selector(text=self.entrance_html).xpath('//*[@id="hea
    return sub_sections

  def section_urls(self, sub_sections):
    """get urls from each section in Jiandan website

    # Arguments:
      sub_sections: a list contains all the urls to be concatenated.

    # Returns:
      section_urls: a list contains all the urls after concatenate.
    """
    section_urls = ["http://jandan.net/tag/"+item for item in sub_sections]
    return section_urls

  @retry(stop_max_attempt_number=10)
  def retry_dict_request(self, section_url):
    reps = (grequests.get(section_url + "/page/" + str(page)) for page in range
    atk_urls = []
    for rep in grequests.map(reps):
      try:
        article_urls = [scrapy.Selector(text=rep.text).xpath('//*[@id="content"
        atk_urls.append(article_urls)
      except:
        return atk_urls

  def sub_sec_atk_dict(self, section_dict):
    sec_atk_dict = {}
    for section_name, section_url in section_dict.items():
      start_time = time.time()
      atk_urls = self.retry_dict_request(section_url)
      sec_atk_dict[section_name] = atk_urls
      print("Spended ", str(int(time.time() - start_time)), " seconds on subsec
            section_name, " to get dictionary that section as key and article u
    return sec_atk_dict

  @retry(stop_max_attempt_number=10)
  def retry_page_urls(self, page_urls, section, sub_section):
    page_parsed_data = []
    reps = (grequests.get(url) for url in page_urls)
    for rep in grequests.map(reps):
      title = scrapy.Selector(text=rep.text).xpath('//*[@id="content"]/div[2]/h
      main_body = scrapy.Selector(text=rep.text).xpath('//*[@id="content"]/div[
      body_str = ''
      for sen in main_body:
        body_str += sen
      meta_tuple = (section[0], sub_section, title[0], body_str)
      page_parsed_data.append(meta_tuple)
    return page_parsed_data

  def parsing_dict(self, sec_atk_dict, section_subsection_dict):
    """ deal with ajax dynamic loading problem.

    # Arguments:
      sections: a list contains all the sections in website
      website: a str represents target website
      headers: a str we generate in previous stage

    # Returns:
      article_urls_dict: a dict whose keys are sections and values
        are articles' urls for this section
    """
    parsed_data = []
    for sub_section, article_urls in sec_atk_dict.items():
      section = [k for k, v in section_subsection_dict.items() if sub_section i
```

```
      print("we are parsing section: ", sub_section, " now\n")
      start_time = time.time()
      for page_urls in tqdm(article_urls):
        page_parsed_data = self.retry_page_urls(page_urls, section, sub_sectior
        parsed_data.append(page_parsed_data)
      print("Spended ", str(int(time.time()-start_time)), " seconds to parse fi
            sub_section)

    return parsed_data


  def display(self):
    """display result of each function in class.
    """


    sections = self.section_list()
    print("Sections(class label) are:", sections)
    print('\n' + '-'*50 + '\n')

    sub_sections = self.sub_section_list()
    print("Subsections are:", sub_sections)
    print('\n' + '-'*50 + '\n')

    section_subsection_dict = self.sec_subsec_dict(sections, sub_sections)
    print("The Dictionary for section as key and subsction as value is ", secti
    print('\n' + '-'*50 + '\n')

    sub_section_urls = self.section_urls(sub_sections)
    print("Subsections urls are: ", sub_section_urls)
    print('\n' + '-'*50 + '\n')

    section_dict = dict(zip(sub_sections, sub_section_urls))
    section_article_dict = self.sub_sec_atk_dict(section_dict)

    print(section_article_dict)

    print('\n' + '-'*50 + '\n')
    parsed_data = self.parsing_dict(section_article_dict, section_subsection_di

    print("Some examples of tuple data")
    for index in range(10):
      print(parsed_data[index])

    return parsed_data


```

```
header = Headers.getHeaders()
spider = Jiandan(website="煎蛋网", website_url="http://jandan.net/", header=head
print('\n'*2 + '='*50 + '\n'*2)
parsed_data = spider.display()
print('\n'*2 + '='*50 + '\n'*2)
```

```
==================================================

Sections(class label) are: ['科学', '技术', '极客', '脑洞', '故事', '人类', '折腾

--------------------------------------------------

Subsections are: ['走进科学', 'TECH', 'GEEK', 'DIY', '冷新闻', '女性', '减肥', '

--------------------------------------------------

The Dictionary for section as key and subsction as value is  {'科学': ['走进科

--------------------------------------------------

Subsections urls are:  ['http://jandan.net/tag/走进科学', 'http://jandan.net/t

--------------------------------------------------

Spended  56  seconds on subsection  走进科学  to get dictionary that section a
Spended  10  seconds on subsection  TECH  to get dictionary that section as k
Spended  64  seconds on subsection  GEEK  to get dictionary that section as k
Spended  10  seconds on subsection  DIY  to get dictionary that section as ke
Spended  52  seconds on subsection  冷新闻  to get dictionary that section as
Spended  52  seconds on subsection  女性  to get dictionary that section as k
Spended  17  seconds on subsection  减肥  to get dictionary that section as k
Spended  35  seconds on subsection  无厘头研究  to get dictionary that section
Spended  10  seconds on subsection  人工智能  to get dictionary that section a
Spended  6  seconds on subsection  MEME  to get dictionary that section as ke
Spended  23  seconds on subsection  艺术  to get dictionary that section as k
Spended  68  seconds on subsection  爷有钱  to get dictionary that section as
Spended  42  seconds on subsection  熊孩子  to get dictionary that section as
Spended  6  seconds on subsection  整形  to get dictionary that section as ke
Spended  10  seconds on subsection  天文  to get dictionary that section as k
Spended  6  seconds on subsection  无人机  to get dictionary that section as k
Spended  10  seconds on subsection  QUORA  to get dictionary that section as
Spended  26  seconds on subsection  设计  to get dictionary that section as k
Spended  21  seconds on subsection  致富信息  to get dictionary that section a
Spended  9  seconds on subsection  大丈夫  to get dictionary that section as k
Spended  8  seconds on subsection  旅游  to get dictionary that section as ke
Spended  7  seconds on subsection  NASA  to get dictionary that section as ke
```

Write parsed data into CSV format

```
Spended  8  seconds on subsection  安全警示  to get dictionary that section as
```

```python
import csv
import codecs

class File_IO(object):
  def __init__(self, read_file=None, write_file=None, headers=None, data=None):
    self.read_file = read_file
    self.write_file = write_file
    self.headers = headers
    self.parsed_data = data

  def read_csv(self):
    """ reading CSV Files with Pandas

    # Arguments:
      name: file name of website without suffix, str

    # Returns:
```

```
      df: data frame contains
    """

    with codecs.open(self.read_file + ".csv", 'r', encoding='utf-8') as csv_fil
      csv_reader = csv.reader(csv_file)
      csv_data = [row for row in csv_reader]
      csv_file.close()
    return csv_data

  def write_csv(self, seg=False):
    """ writing CSV Files with Pandas

    Arguments:
      name: file name of website text data without suffix, str

    Returns:
      f_csv:

    """
    with codecs.open(self.write_file + '.csv', 'w', encoding='utf-8') as f:
      f_csv = csv.writer(f)
      if self.headers:
        f_csv.writerow(self.headers)
      for row in self.parsed_data:
        if not seg:
          for item in row:
            f_csv.writerow(item)
        else:
          f_csv.writerow(row)

      f.close()
    return f_csv

  def read_json(self):
    """
    Arguments:

    Return:

    """
    with codecs.open(self.read_file + '.json', 'r') as f:
      data = list()
      for line in f:
        # load values of key
        data.append(json.load(line))
      f.close()
    return data



  def write_json(self):
    with codecs.open(self.write_file + '.json', 'w') as f:
      json.dump(self.parsed_data, f, ensure_ascii=False)
      f.close()
    return f

```

```
header = ('类别', '子类别', '标题', '正文')
writer = File_IO(write_file="Jiandan_final", headers=header, data=parsed_data)
Jiandan = writer.write_csv()
```

```
100%|██████████| 1309/1309 [00:00<00:00, 1613.96it/s]
```

Data Preprocessing

In this section we are going to use Jieba to deleting stop word and segementation.

Mount to Google Drive

```
# Install the PyDrive wrapper & import libraries.
# This only needs to be done once in a notebook.
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

# Authenticate and create the PyDrive client.
# This only needs to be done once in a notebook.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
up_drive = GoogleDrive(gauth)

from google.colab import drive
down_drive = drive.mount('/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client

Enter your authorization code:
··········
Mounted at /gdrive
```

```
# check out mount result
print("Origin dir is:")
!ls
import os
os.chdir("../gdrive/My Drive")
print("Now the dir has changed to:")
!ls
```

```
Origin dir is:
adc.json  sample_data
Now the dir has changed to:
 action_data_car_racing_0.npy   MLP_model.h5
 action_data_car_racing_1.npy   NB_model.m
 action_data_car_racing_2.npy   papers
 auth.ipynb                     pdf
 BiLSTM.h5                      SVM_model.m
'CAIC Shared Code'              Tensorflow_eager_mode.ipynb
'Code Chips'                    test_case2.ipynb
'Colab Notebooks'               testcase3.ipynb
 Conv_model.h5                  Untitled0.ipynb
 Dconv_model.h5                 vae_weights.h5
 homeworks                      中文停用词表.txt
 Jiandan_final.csv              哈工大停用词表.txt
 Jiandan_re_segment.csv         四川大学机器智能实验室停用词库.txt
 Jiandan_segment.csv            百度停用词表.txt
```

Segmentation and Delete Stop Words

Download stop word dictionary from Internet

```
!pip install jieba
# download a stop word dictionary from network
!wget https://raw.githubusercontent.com/goto456/stopwords/master/%E7%99%BE%E5%B
```

```
!wget https://raw.githubusercontent.com/goto456/stopwords/master/%E5%9B%9B%E5%B
!wget https://raw.githubusercontent.com/goto456/stopwords/master/%E5%93%88%E5%B
!wget https://raw.githubusercontent.com/goto456/stopwords/master/%E4%B8%AD%E6%9
!ls
```

```
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.0.
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.(
```

Apply dictionary to corpus

```python
## basic demo for segmentation
import jieba
from tqdm import tqdm
import re

f_o = File_IO(read_file='Jiandan_final')
corpus = f_o.read_csv()
print("Example of corpus: ", corpus[1])

print("----------")
docs = [doc[1]+doc[2]+doc[3] for doc in corpus[1:]]
labels = [doc[0] for doc in corpus[1:]]

print("Example of Segment result: ", [word for word in jieba.cut(docs[3])])
print("----------")
segmented_corpus = [jieba.cut(doc) for doc in docs]

stop_word_dict = ["中文停用词表.txt", "哈工大停用词表.txt",
                  "四川大学机器智能实验室停用词库.txt", "百度停用词表.txt"]

stop_words_list = []
# delete stopping word
for each_dict in stop_word_dict:
  with open(each_dict, 'r') as word_file:
    stop_words = word_file.readlines()
    each_stop_words = [item[:-1] for item in stop_words]
    stop_words_list += each_stop_words


stop_words_list = list(set(stop_words_list))
print("\n停用词表: ", stop_words_list)
print("----------")

final_corpus = []
for index, segment_list in enumerate(segmented_corpus):
  final = ''
  for seg in segment_list:
    if seg not in stop_words_list:
        final = final + seg + ' '
  final_corpus.append([final[:-1], labels[index]])


print("Example of final corpus: ", final_corpus[1])
print("----------")
print(len(final_corpus))
f_i = File_IO(write_file='Jiandan_segment', data=f, headers=['context', 'label'
seg_corpus = f_i.write_csv(seg=True)

```

```
Example of corpus:  ['科学', '走进科学', '为什么袋熊的便便是方块状的', '有屎以来，只有
----------
Example of Segment result:  ['走进', '科学', '超级', '沙漠', '里', '的', '生命',
----------

停用词表:  ['替', '上去', '长话短说', '若非', '×', '越是', '从古至今', '据悉', "don'
----------
Example of final corpus:  ['走进 科学 走近 科学 为啥 加州 山火 烧足 一个月 11 月 号
----------
942480
```

Extract processed data as DataFrame

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.multiclass import OneVsRestClassifier

print("----------")

df = pd.read_csv("Jiandan_segment.csv")
```

```
----------
```

Naive Bayes implementation

In this section, we implement Naive Bayes as Baseline.

Import Libraries

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn import svm
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
from sklearn import metrics
from sklearn.model_selection import StratifiedKFold
from sklearn.decomposition import KernelPCA
from sklearn.externals import joblib
```

Implementation of Naive Bayes

```
from collections import Counter
from collections import defaultdict
from math import log


class MultinomialNB:
    """Hybrid implementation of Naive Bayes.
    Supports discrete and continuous features.
    """

    def __init__(self, extract_features, use_smoothing=True):
        """Create a naive bayes classifier.
        :param extract_features: Callback to map a feature vector to discrete a
        :param use_smoothing: Whether to use smoothing when calculating probabi
        """
        self.priors = defaultdict(dict)

        self.label_counts = Counter()
        self.discrete_feature_vectors = DiscreteFeatureVectors(use_smoothing)
        self.continuous_feature_vectors = ContinuousFeatureVectors()
        self._extract_features = extract_features
        self._is_fitted = False

    def fit(self, design_matrix, target_values):
        """Fit model according to design matrix and target values.
        :param design_matrix: Training examples with dimension m x n,
                              where m is the number of examples,
                              and n is the number of features.
        :param target_values: Target values with dimension m,
                              where m is the number of examples.
        :return: self
        """
        for i, training_example in enumerate(design_matrix):
```

```
            label = target_values[i]
            self.label_counts[label] += 1
            features = self._extract_features(training_example)
            for j, feature in enumerate(features):
                if feature.is_continuous():
                    self.continuous_feature_vectors.add(label, j, feature)
                else:
                    self.discrete_feature_vectors.add(label, j, feature)

        total_num_records = len(target_values)
        for label in set(target_values):
            self.priors[label] = self.label_counts[label] / total_num_records
            self.continuous_feature_vectors.set_mean_variance(label)

        self._is_fitted = True
        return self

    def predict(self, test_set):
        """Predict target values for test set.
        :param test_set: Test set with dimension m x n,
                         where m is the number of examples,
                         and n is the number of features.
        :return: Predicted target values for test set with dimension m,
                 where m is the number of examples.
        """
        self._check_is_fitted()

        predictions = []
        for i in range(len(test_set)):
            result = self.predict_record(test_set[i])
            predictions.append(result)
        return predictions

    def predict_record(self, test_record):
        """Predict the label for the test record.
        Maximizes the log likelihood to prevent underflow.
        :param test_record: Test record to predict a label for.
        :return: The predicted label.
        """
        self._check_is_fitted()

        log_likelihood = {k: log(v) for k, v in self.priors.items()}
        for label in self.label_counts:
            features = self._extract_features(test_record)
            for i, feature in enumerate(features):
                probability = self._get_probability(i, feature, label)
                try:
                    log_likelihood[label] += log(probability)
                except ValueError as e:
                    pass
        return max(log_likelihood, key=log_likelihood.get)

    def _check_is_fitted(self):
        if not self._is_fitted:
            raise NotFittedError(self.__class__.__name__)

    def _get_probability(self, feature_index, feature, label):
        if feature.is_continuous():
            probability = self.continuous_feature_vectors.probability(label,
                                                                      feature_i
        else:
            probability = self.discrete_feature_vectors.probability(label,
                                                                    feature_ind
                                                                    feature,
                                                                    self.label_
        return probability
```

Build Pipeline of Naive Bayes with TF-IDF as Text feature extraction

In information retrieval or text mining, the term frequency – inverse document frequency (also called tf-idf), is a well know method to evaluate how important is a word in a document. tf-idf are is a very

interesting way to convert the textual representation of information into a Vector Space Model (VSM), or into sparse features.

```
# Define a pipeline combining a text feature extractor with multi lable classif
NB_pipeline = Pipeline([
                ('tfidf', TfidfVectorizer()),
                ('clf', OneVsRestClassifier(MultinomialNB(fit_prior=True, class
            ]).
```

Fit & Predict

K-Fold Cross Validation

The general idea of K cross tests is to roughly divide the data into K sub-samples. One sample is taken as the verification data and the remaining k-1 samples are taken as the training data.

Measure the model with Stratified k-fold

Than KFold StratifiedKFold () this function is used, the advantage of the k data data set on a percentage basis, the percentage for each category in the training set and test set are the same, so that can not have a certain categories of data in the training set and test set is not this kind of situation, also not in training all in the test set, this will lead to worse results.

```
X = df.context
y = df.label
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=824)
for train_index, test_index in skf.split(X, y):

    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    NB_pipeline.fit(X_train, y_train)
    prediction = NB_pipeline.predict(X_test)
    print(metrics.classification_report(y_test, prediction))
```

```
                precision    recall  f1-score   support

          人类       1.00      0.01      0.01       370
          技术       0.96      0.06      0.11       394
          折腾       0.84      0.02      0.05       653
          故事       0.57      0.73      0.64      1412
          极客       1.00      0.10      0.19       543
          科学       0.50      0.97      0.66      2136
          脑洞       0.91      0.29      0.44       778

   micro avg       0.54      0.54      0.54      6286
   macro avg       0.83      0.31      0.30      6286
weighted avg       0.70      0.54      0.45      6286

                precision    recall  f1-score   support

          人类       1.00      0.02      0.03       370
          技术       0.92      0.03      0.05       394
          折腾       0.93      0.02      0.04       653
          故事       0.58      0.75      0.65      1411
          极客       0.98      0.08      0.16       543
          科学       0.50      0.97      0.67      2136
          脑洞       0.92      0.30      0.45       778

   micro avg       0.55      0.55      0.55      6285
   macro avg       0.83      0.31      0.29      6285
weighted avg       0.71      0.55      0.45      6285

                precision    recall  f1-score   support

          人类       1.00      0.02      0.03       370
          技术       0.96      0.06      0.11       394
          折腾       1.00      0.01      0.03       653
```

One-Fold Validation for demonstrating

Save model to Google Drive

```
import time
X = df.context
y = df.label
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random
NB_pipeline.fit(X_train, y_train)
joblib.dump(NB_pipeline, "NB_model.m")
```

```
['NB_model.m']
```

```
          折腾       1.00      0.01      0.03       653
```

Test for one fold

```
          科学       0.50      0.97      0.66      2136
```

```
X = df.context
y = df.label
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random

NB_pipeline = joblib.load("NB_model.m")
prediction = NB_pipeline.predict(X_test)
print(metrics.classification_report(y_test, prediction))
```

	precision	recall	f1-score	support
人类	1.00	0.03	0.06	27715
技术	0.96	0.06	0.11	29630
折腾	0.97	0.02	0.04	48908
故事	0.61	0.75	0.67	106037
极客	1.00	0.10	0.18	40656
科学	0.50	0.98	0.67	160056
脑洞	0.96	0.34	0.50	58238
micro avg	0.56	0.56	0.56	471240
macro avg	0.86	0.33	0.32	471240

Support Vector Machine

In machine learning, support vector machines (SVMs, also support vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

When data is unlabelled, supervised learning is not possible, and an unsupervised learning approach is required, which attempts to find natural clustering of the data to groups, and then map new data to these formed groups. The support vector clustering algorithm, created by Hava Siegelmann and Vladimir Vapnik, applies the statistics of support vectors, developed in the support vector machines algorithm, to categorize unlabeled data, and is one of the most widely used clustering algorithms in industrial applications

```
# Define a pipeline combining a text feature extractor with multi lable classif
l_svm = Pipeline([
                ('tfidf', TfidfVectorizer()),
                ('clf', OneVsRestClassifier(LinearSVC()))
            ])
```

Test Performance of Basic Linear Kernel

```
for train_index, test_index in skf.split(X, y):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    l_svm.fit(X_train, y_train)
    prediction = l_svm.predict(X_test)
    print(metrics.classification_report(y_test, prediction))
```

```
weighted avg       0.79      0.79      0.79      6286

              precision    recall  f1-score   support

          人类       0.85      0.62      0.71       370
          技术       0.71      0.60      0.65       394
          折腾       0.68      0.63      0.65       653
          故事       0.84      0.91      0.87      1411
          极客       0.73      0.56      0.63       543
          科学       0.80      0.87      0.83      2136
          脑洞       0.82      0.83      0.83       778

   micro avg       0.79      0.79      0.79      6285
   macro avg       0.77      0.72      0.74      6285
weighted avg       0.79      0.79      0.79      6285

              precision    recall  f1-score   support

          人类       0.89      0.69      0.78       370
          技术       0.74      0.60      0.66       394
          折腾       0.68      0.64      0.66       653
          故事       0.85      0.91      0.88      1411
          极客       0.70      0.55      0.62       542
          科学       0.81      0.89      0.85      2136
          脑洞       0.85      0.87      0.86       778

   micro avg       0.80      0.80      0.80      6284
   macro avg       0.79      0.73      0.76      6284
weighted avg       0.80      0.80      0.80      6284

              precision    recall  f1-score   support

          人类       0.85      0.64      0.73       369
          技术       0.71      0.64      0.67       393
          折腾       0.67      0.63      0.65       653
          故事       0.83      0.90      0.86      1411
          极客       0.73      0.57      0.64       542
          科学       0.81      0.88      0.84      2136
          脑洞       0.84      0.86      0.85       777

   micro avg       0.80      0.80      0.80      6281
   macro avg       0.78      0.73      0.75      6281
weighted avg       0.79      0.80      0.79      6281

              precision    recall  f1-score   support

          人类       0.88      0.69      0.78       369
          技术       0.70      0.58      0.64       393
          折腾       0.70      0.59      0.64       652
          故事       0.83      0.91      0.87      1411
          极客       0.68      0.54      0.60       542
          科学       0.79      0.87      0.83      2136
          脑洞       0.83      0.86      0.85       777
```

Further More: Deep Learning Method!

```
weighted avg       0.79      0.79      0.79      6280
```

Word Embedding as Feature Extraction

Text is considered a form of sequence data similar to time series data that you would have in weather data or financial data. In the previous BOW model, you have seen how to represent a whole sequence of words as a single feature vector. Now you will see how to represent each word as vectors. There are various ways to vectorize text, such as:

- Words represented by each word as a vector
- Characters represented by each character as a vector
- N-grams of words/characters represented as a vector (N-grams are overlapping groups of

Preprocessing text with Keras Tokenizer

This class allows to vectorize a text corpus, by turning each text into either a sequence of integers (each integer being the index of a token in a dictionary) or into a vector where the coefficient for each token could be binary, based on word count, based on tf-idf...

```
import pandas as pd
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
from keras.models import load_model

df = pd.read_csv("Jiandan_segment.csv")
X = df.context
# convert Chinese to number
label_dict = {'人类': 0,
              '技术': 1,
              '折腾': 2,
              '故事': 3,
              '极客': 4,
              '科学': 5,
              '脑洞': 6,
             }
y = df.label
for i, l in enumerate(y):
  y[i] = label_dict[l]

print("check out numerize label:")
y_cate = to_categorical(y, num_classes=7)
print(y_cate[:10])
print("----------")


# Tokenize X_train data todo:change num_words
x_tokenizer = Tokenizer(num_words=5000)
x_tokenizer.fit_on_texts(X)
X_token = x_tokenizer.texts_to_sequences(X)
vocab_size = len(x_tokenizer.word_index) + 1

print("Examples of text after tokenized is: ", X_token[0])
```

check out numeric label:

[0. 0. 0. 0. 0. 1. 0.]

Build basic MLP as Deep Learning Baseline

```
!apt-get install python-pydot python-pydot-ng graphviz
!pip install pydot graphviz pydot-ng
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
graphviz is already the newest version (2.40.1-2).
Suggested packages:
  python-pyparsing-doc
The following NEW packages will be installed:
  python-pydot python-pydot-ng python-pyparsing
0 upgraded, 3 newly installed, 0 to remove and 8 not upgraded.
Need to get 91.5 kB of archives.
After this operation, 443 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu bionic/main amd64 python-pyparsing all
Get:2 http://archive.ubuntu.com/ubuntu bionic/universe amd64 python-pydot all
Get:3 http://archive.ubuntu.com/ubuntu bionic/universe amd64 python-pydot-ng
Fetched 91.5 kB in 1s (161 kB/s)
Selecting previously unselected package python-pyparsing.
(Reading database ... 110377 files and directories currently installed.)
Preparing to unpack .../python-pyparsing_2.2.0+dfsg1-2_all.deb ...
Unpacking python-pyparsing (2.2.0+dfsg1-2) ...
Selecting previously unselected package python-pydot.
Preparing to unpack .../python-pydot_1.2.3-1_all.deb ...
Unpacking python-pydot (1.2.3-1) ...
Selecting previously unselected package python-pydot-ng.
Preparing to unpack .../python-pydot-ng_1.0.0-3_all.deb ...
Unpacking python-pydot-ng (1.0.0-3) ...
Setting up python-pyparsing (2.2.0+dfsg1-2) ...
Setting up python-pydot-ng (1.0.0-3) ...
Setting up python-pydot (1.2.3-1) ...
Requirement already satisfied: pydot in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: graphviz in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: pydot-ng in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: pyparsing>=2.1.4 in /usr/local/lib/python3.6/d
```

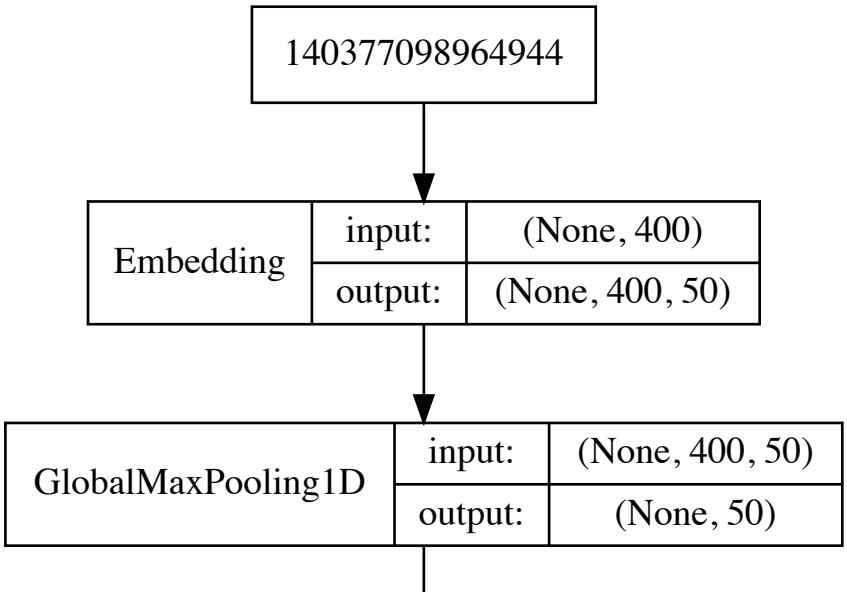
```
from keras.models import Sequential
from keras import layers
from keras.utils import plot_model
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot
from keras import regularizers
from keras.preprocessing.sequence import pad_sequences
import numpy as np
```

```
embedding_dim = 50

mlp = Sequential()
mlp.add(layers.Embedding(input_dim=vocab_size,
                         output_dim=embedding_dim,
                         input_length=400))
# mlp.add(layers.Flatten())
mlp.add(layers.GlobalMaxPool1D())
mlp.add(layers.Dense(128, activation='relu', kernel_regularizer = regularizers.
mlp.add(layers.Dropout(0.5))
mlp.add(layers.Dense(64, activation='relu', kernel_regularizer = regularizers.l
mlp.add(layers.Dropout(0.5))
mlp.add(layers.Dense(7, activation='softmax'))
mlp.compile(optimizer='adam',
```

```
            loss='categorical_crossentropy',
            metrics=['accuracy'],
           )
mlp.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_3 (Embedding)      (None, 400, 50)           4875050   
_________________________________________________________________
global_max_pooling1d_3 (Glob (None, 50)                0         
_________________________________________________________________
dense_7 (Dense)              (None, 128)               6528      
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_8 (Dense)              (None, 64)                8256      
_________________________________________________________________
dropout_6 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_9 (Dense)              (None, 7)                 455       
=================================================================
Total params: 4,890,289
Trainable params: 4,890,289
Non-trainable params: 0
_________________________________________________________________
```

```
SVG(model_to_dot(mlp, show_shapes=True, show_layer_names=False).create(prog='do
```

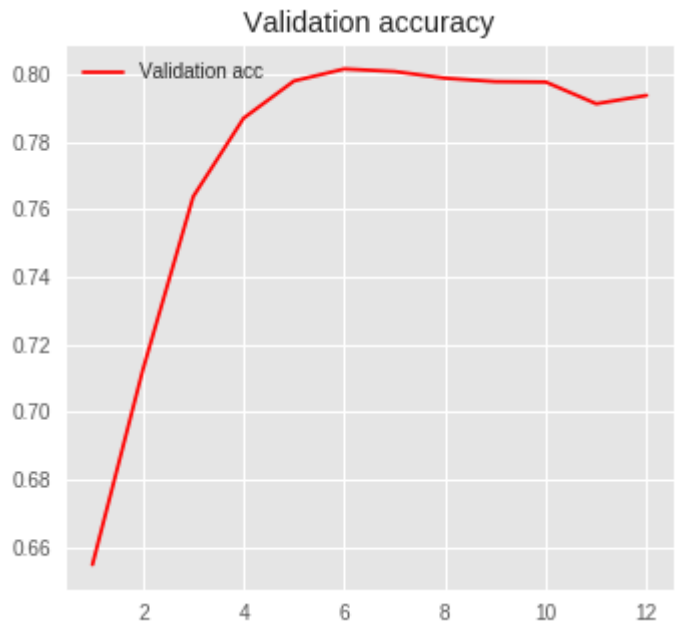


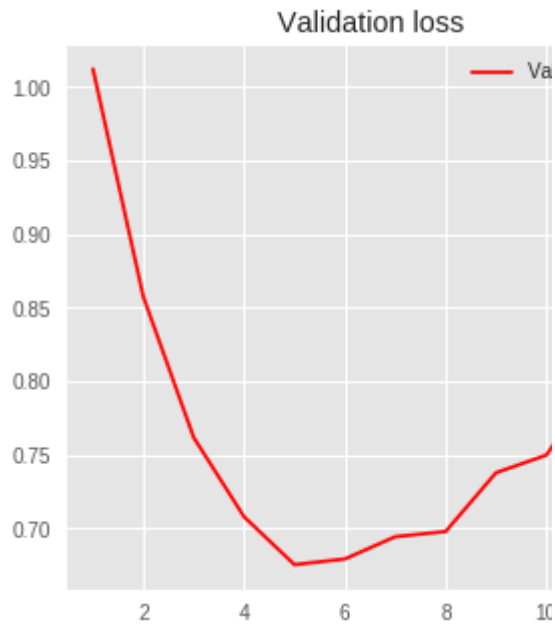
Measure performance of MLP

```
import matplotlib.pyplot as plt
plt.style.use('ggplot')

def plot_history(history):
    acc = history.history['acc']
    val_acc = history.history['val_acc']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    x = range(1, len(acc) + 1)

    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(x, val_acc, 'r', label='Validation acc')
    plt.title('Validation accuracy')
    plt.legend()
    plt.subplot(1, 2, 2)
    plt.plot(x, val_loss, 'r', label='Validation loss')
    plt.title('Validation loss')
    plt.legend()
```

```
X_train, X_test, y_train, y_test = train_test_split(X_token, y_cate, test_size=

X_train, X_test = pad_sequences(X_train, padding='post', maxlen=400),\
                  pad_sequences(X_test, padding='post', maxlen=400)

history = mlp.fit(X_train, y_train,
                  epochs=12,
                  verbose=0,
                  validation_data=(X_test, y_test),
                  batch_size=32)

mlp.save('MLP_model.h5')
loss, accuracy = mlp.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = mlp.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
plot_history(history)


mlp_train = mlp.predict_classes(X_train)
mlp_pred = mlp.predict_classes(X_test)
print("Training results:")
print(metrics.classification_report(mlp_train, np.argmax(y_train, axis=1)))
print("Testing results:")
print(metrics.classification_report(mlp_pred, np.argmax(y_test, axis=1)))

```

```
Training Accuracy: 0.9555
Testing Accuracy:  0.7937
Training results:
              precision    recall  f1-score   support

           0       0.92      0.97      0.94       866
           1       0.86      0.88      0.87       994
           2       0.92      0.94      0.93      1594
           3       0.99      0.98      0.99      3618
           4       0.86      0.87      0.87      1342
           5       0.99      0.97      0.98      5392
           6       0.97      0.97      0.97      1902

   micro avg       0.96      0.96      0.96     15708
   macro avg       0.93      0.94      0.94     15708
weighted avg       0.96      0.96      0.96     15708

Testing results:
              precision    recall  f1-score   support

           0       0.73      0.86      0.79       791
           1       0.45      0.51      0.48       834
           2       0.59      0.69      0.63      1397
           3       0.95      0.91      0.93      3633
           4       0.59      0.57      0.58      1406
           5       0.87      0.81      0.84      5768
           6       0.81      0.85      0.83      1879

   micro avg       0.79      0.79      0.79     15708
   macro avg       0.71      0.74      0.73     15708
weighted avg       0.80      0.79      0.80     15708
```





CNN: Convolutional Neural Network

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery.

CNNs use a variation of multilayer perceptrons designed to require minimal preprocessing.[1] They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their

shared-weights architecture and translation invariance characteristics.[2][3]

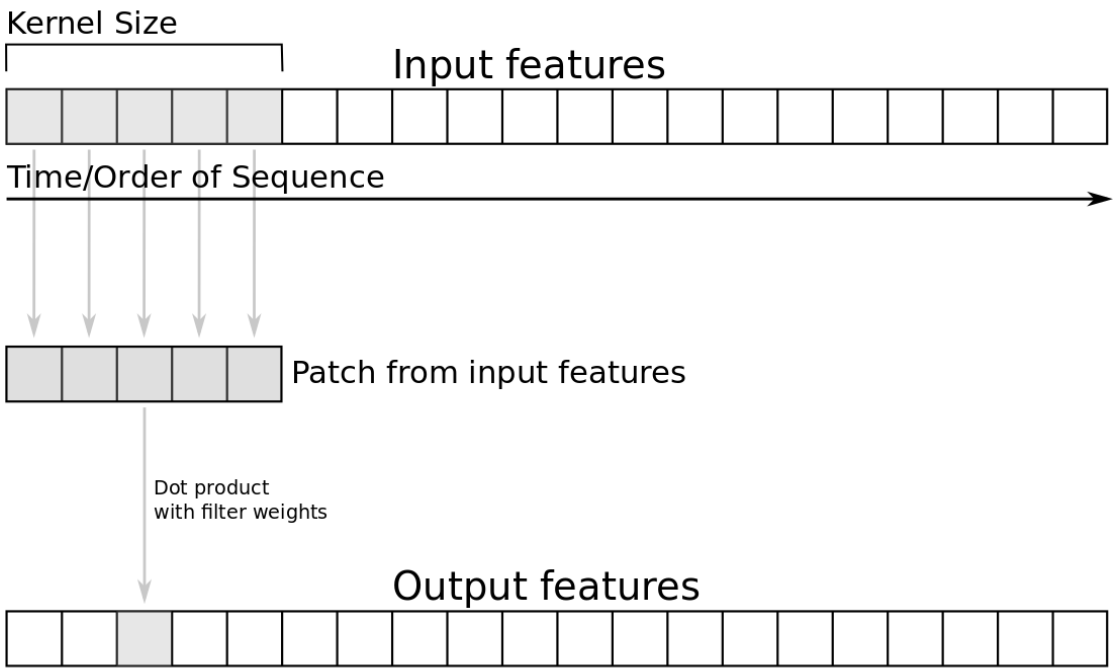
Convolutional networks were inspired by biological processes[4] in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage.

They have applications in image and video recognition, recommender systems,[5] image classification, medical image analysis, and natural language processing.

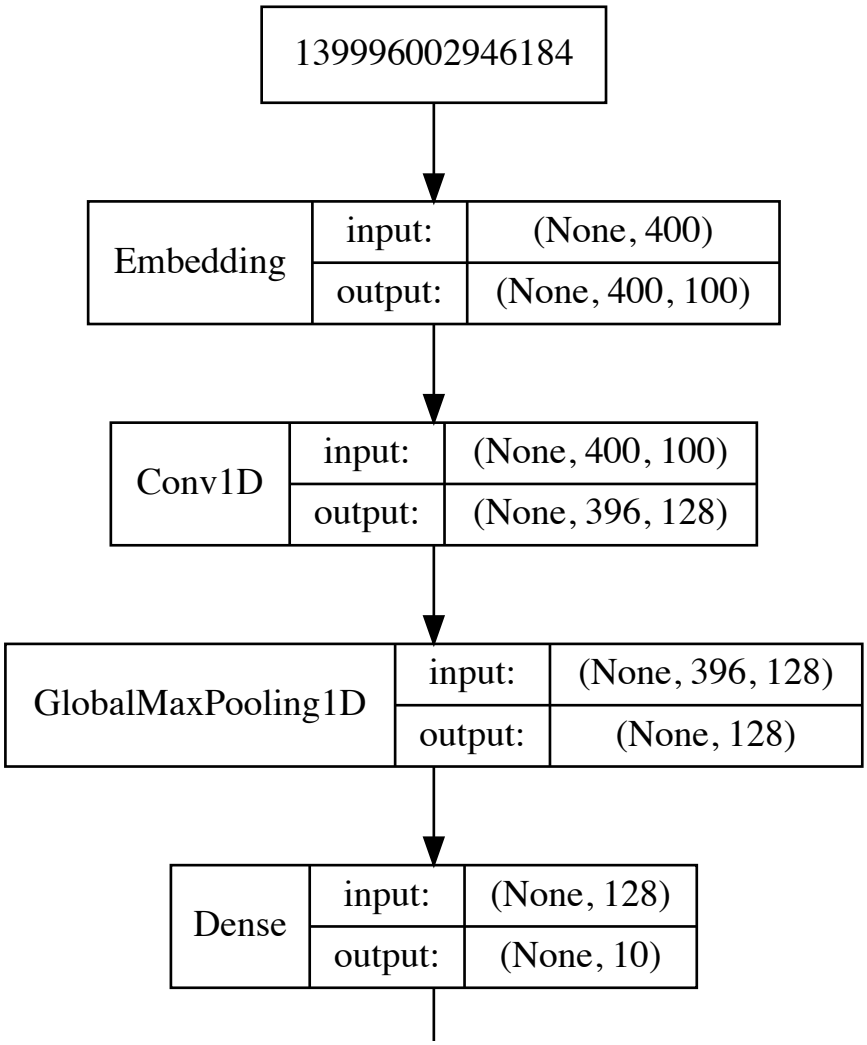
Conv1D

Keras offers again various Convolutional layers which you can use for this task. The layer you'll need is the Conv1D layer. This layer has again various parameters to choose from.




```
embedding_dim = 100

Conv = Sequential()
Conv.add(layers.Embedding(vocab_size, embedding_dim, input_length=400))
Conv.add(layers.Conv1D(128, 5, activation='relu'))
Conv.add(layers.GlobalMaxPooling1D())
Conv.add(layers.Dense(10, activation='relu'))
Conv.add(layers.Dense(7, activation='softmax'))
Conv.compile(optimizer='adam',
             loss='categorical_crossentropy',
             metrics=['accuracy'])
Conv.summary()
SVG(model_to_dot(Conv, show_shapes=True, show_layer_names=False).create(prog='d
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 400, 100)          9750100
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 396, 128)          64128
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 128)               0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1290
_________________________________________________________________
dense_7 (Dense)              (None, 7)                 77
=================================================================
Total params: 9,815,595
Trainable params: 9,815,595
Non-trainable params: 0
_________________________________________________________________
```



Measure the performance of basic CNN

```
Dense
```

```
from keras.preprocessing.sequence import pad_sequences
import numpy as np

# X_train, X_test, y_train, y_test = train_test_split(X_token, y_cate, test_siz

# X_train, X_test = pad_sequences(X_train, padding='post', maxlen=400),\
#                   pad_sequences(X_test, padding='post', maxlen=400)

history = Conv.fit(X_train, y_train,
                   epochs=15,
                   verbose=0,
```

```
                    validation_data=(X_test, y_test),
                    batch_size=32)

Conv.save('MLP_model.h5')
loss, accuracy = Conv.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = Conv.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
plot_history(history)


Conv_train = Conv.predict_classes(X_train)
Conv_pred = Conv.predict_classes(X_test)
print("Training results:")
print(metrics.classification_report(Conv_train, np.argmax(y_train, axis=1)))
print("Testing results:")
print(metrics.classification_report(Conv_pred, np.argmax(y_test, axis=1)))

```

```
Training Accuracy: 0.9999
Testing Accuracy:  0.8516
Training results:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       912
           5       1.00      1.00      1.00      5292
```

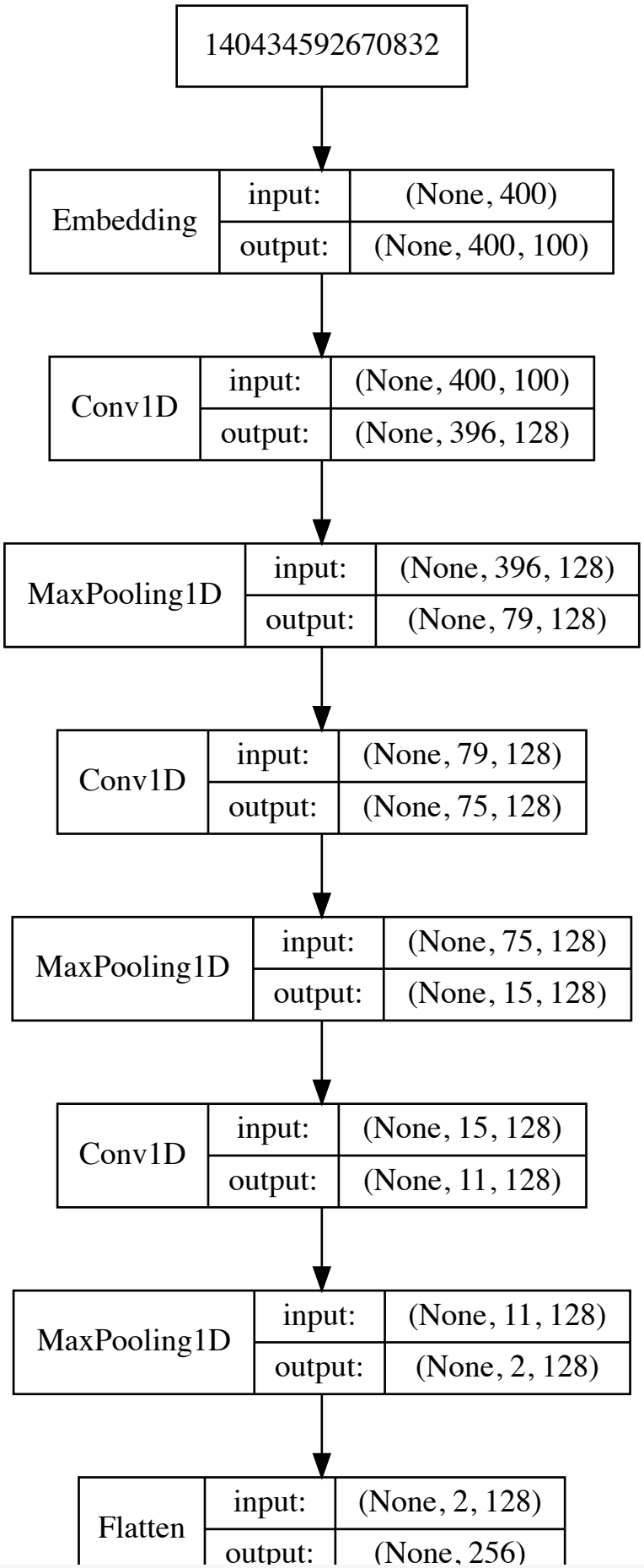
Deeper CNN

Let us check out whether a deeper CNN can improve the performance of text classification.

```
embedding_dim = 100

D_Conv = Sequential()
D_Conv.add(layers.Embedding(vocab_size, embedding_dim, input_length=400))
D_Conv.add(layers.Conv1D(128, 5, activation='relu'))
D_Conv.add(layers.MaxPooling1D(5))
D_Conv.add(layers.Conv1D(128, 5, activation='relu'))
D_Conv.add(layers.MaxPooling1D(5))
D_Conv.add(layers.Conv1D(128, 5, activation='relu'))
D_Conv.add(layers.MaxPooling1D(5))
D_Conv.add(layers.Flatten())
D_Conv.add(layers.Dense(32, activation='relu'))
D_Conv.add(layers.Dense(7, activation='softmax'))
D_Conv.compile(optimizer='adam',
            loss='categorical_crossentropy',
            metrics=['accuracy'])
D_Conv.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 400, 100)          9750100
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 396, 128)          64128
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 79, 128)           0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 75, 128)           82048
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 15, 128)           0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 11, 128)           82048
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 2, 128)            0
_________________________________________________________________
flatten_1 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                8224
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 231
=================================================================
Total params: 9,986,779
Trainable params: 9,986,779
Non-trainable params: 0
_________________________________________________________________
```

```
SVG(model_to_dot(D_Conv, show_shapes=True, show_layer_names=False).create(prog=
```




```
from keras.preprocessing.sequence import pad_sequences
import numpy as np

history = D_Conv.fit(X_train, y_train,
                     epochs=1,
                     verbose=0,
                     validation_data=(X_test, y_test),
```

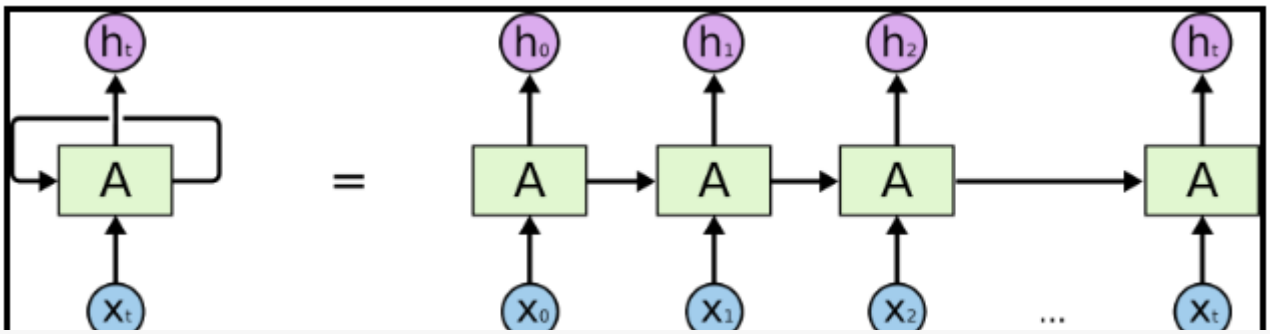
```
                    batch_size=32)
```

```
D_conv = load_model('Dconv_model.h5')
loss, accuracy = D_Conv.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = D_Conv.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
# plot_history(history)


D_Conv_train = D_Conv.predict_classes(X_train)
D_Conv_pred = D_Conv.predict_classes(X_test)
print("Training results:")
print(metrics.classification_report(D_Conv_train, np.argmax(y_train, axis=1)))
print("Testing results:")
print(metrics.classification_report(D_Conv_pred, np.argmax(y_test, axis=1)))
```

```
Training Accuracy: 0.9221
Testing Accuracy:  0.8240
Training results:
              precision    recall  f1-score   support

           0       0.90      0.71      0.79      1154
           1       0.77      0.72      0.74      1092
           2       0.93      0.92      0.92      1640
           3       0.98      1.00      0.99      3526
           4       0.64      0.82      0.72      1056
           5       0.98      0.96      0.97      5355
           6       0.95      0.97      0.96      1885

   micro avg       0.92      0.92      0.92     15708
   macro avg       0.88      0.87      0.87     15708
weighted avg       0.93      0.92      0.92     15708

Testing results:
              precision    recall  f1-score   support

           0       0.77      0.50      0.61      1444
           1       0.56      0.49      0.52      1091
           2       0.74      0.73      0.73      1665
           3       0.98      1.00      0.99      3399
           4       0.38      0.57      0.46       901
           5       0.91      0.91      0.91      5424
           6       0.84      0.93      0.88      1784

   micro avg       0.82      0.82      0.82     15708
   macro avg       0.74      0.73      0.73     15708
weighted avg       0.83      0.82      0.82     15708
```

RNN

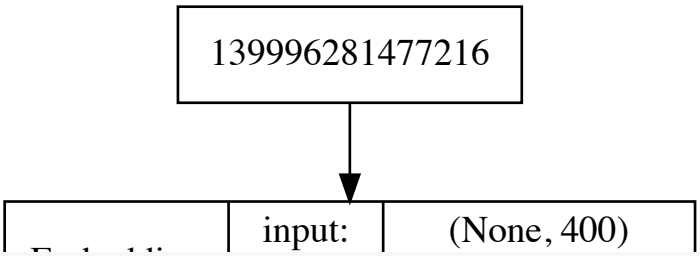
A recurrent neural network (RNN) is a class of artificial neural network where connections between nodes form a directed graph along a sequence. This allows it to exhibit temporal dynamic behavior for a time sequence. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition.




```
LSTM = Sequential()
LSTM.add(layers.Embedding(vocab_size, 100, input_length=400))
LSTM.add(layers.Bidirectional(layers.CuDNNLSTM(100)))
LSTM.add(layers.Dense(32, activation='relu'))
LSTM.add(layers.Dense(7, activation='softmax'))

LSTM.compile(optimizer='adam',
             loss='categorical_crossentropy',
             metrics=['accuracy'])
LSTM.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 400, 100)          9750100
_________________________________________________________________
bidirectional_1 (Bidirection (None, 200)               161600
_________________________________________________________________
dense_4 (Dense)              (None, 32)                6432
_________________________________________________________________
dense_5 (Dense)              (None, 7)                 231
=================================================================
Total params: 9,918,363
Trainable params: 9,918,363
Non-trainable params: 0
_________________________________________________________________
```

```
SVG(model_to_dot(LSTM, show_shapes=True, show_layer_names=False).create(prog='d
```




```
from keras.preprocessing.sequence import pad_sequences
import numpy as np

X_train, X_test, y_train, y_test = train_test_split(X_token, y_cate, test_size=

X_train, X_test = pad_sequences(X_train, padding='post', maxlen=400),\
                  pad_sequences(X_test, padding='post', maxlen=400)

history = LSTM.fit(X_train, y_train,
                   epochs=10,
                   verbose=0,
                   validation_data=(X_test, y_test),
                   batch_size=128)

loss, accuracy = LSTM.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = LSTM.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
plot_history(history)


LSTM_train = LSTM.predict_classes(X_train)
LSTM_pred = LSTM.predict_classes(X_test)
print("Training results:")
print(metrics.classification_report(LSTM_train, np.argmax(y_train, axis=1)))
print("Testing results:")
print(metrics.classification_report(LSTM_pred, np.argmax(y_test, axis=1)))

```

```
Training Accuracy: 0.9999
Testing Accuracy:  0.9004
Training results:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       912
           1       1.00      1.00      1.00      1014
           2       1.00      1.00      1.00      1632
           3       1.00      1.00      1.00      3585
           4       1.00      1.00      1.00      1353
           5       1.00      1.00      1.00      5292
           6       1.00      1.00      1.00      1920

   micro avg       1.00      1.00      1.00     15708
   macro avg       1.00      1.00      1.00     15708
weighted avg       1.00      1.00      1.00     15708

Testing results:
              precision    recall  f1-score   support

           0       0.86      0.84      0.85       968
           1       0.60      0.68      0.64       851
           2       0.91      0.89      0.90      1669
           3       0.99      0.99      0.99      3499
           4       0.66      0.72      0.69      1239
           5       0.94      0.93      0.94      5456
           6       0.94      0.91      0.92      2026

   micro avg       0.90      0.90      0.90     15708
   macro avg       0.84      0.85      0.85     15708
weighted avg       0.90      0.90      0.90     15708
```