◀ home (../)

Machine Learning - NLP: Text Classification sử dụng scikit-learn - python

Document/Text classification là 1 phần điển hình và quan trọng trong supervised machine learning. Phân loại các tài liệu(bài báo, tạp chí, trang web, hay là cả những status, comment trên MXH), nó có rất nhiều ứng dụng trong việc phân loại spam mail, email routing, sentiment analysis hay ứng dụng vào tạo chatbot. Trong bài này, tôi sẽ giới thiệu đến với các bạn cách ứng dụng python, cụ thể là sử dụng thư viện scikit-learn để tiến hành phân loại text để giải quyết bài toán thực tế là tạo nền tảng cho chatbot.

Bước 1: Chuẩn bị môi trường

Bài viết này tôi sử dụng Pycharm (https://www.jetbrains.com/pycharm/) để viết code. Để nhanh gọn lẹ hơn, bạn hoàn toàn có thể dùng jupyter notebook nhé. Thư viện thì bạn chỉ cần cài đặt anaconda (https://www.anaconda.com/download/) tương ứng với os bạn đang sử dụng.

Bước 2: Dữ liệu và tiền xử lý

Đầu tiên không thể không kể đến là việc chuẩn bị dữ liệu, do mình không tìm được nguồn dữ liệu nào thích hợp, đầy đủ và chính xác nên mình sẽ xây dựng chatbot theo hướng ít dữ liệu training, cụ thể như sau:

Ví dụ bạn muốn hỏi chatbot thời tiết ngày mai thế nào? Sẽ có vô vàn cách hỏi khác nhau phải không:

- Mai trời mưa không nhỉ ?
- Mai liệu nắng không nhỉ ?
- Ngày mai thời tiết thế nào em ơi ?
- Em nghe nói ngày mai thời tiết đẹp lắm phải không ?
- ....

Đó, cùng một nội dung là hỏi thời tiết ngày mai nhưng với mỗi người, mỗi hoàn cảnh, cảm xúc khác nhau, chúng

ta lại có những cách hỏi khác nhau. Với trường hợp như vậy chúng ta không thể viết code theo phương pháp truyền thống là sử dụng if else, switch case được phải không. OK, bạn đã hình dung ra được vấn đề chưa ?

Đây chính là lúc ứng dụng bài toán **Text Classification** để giải quyết. Công việc chúng ta là làm cách nào đó để với những câu hỏi như trên sẽ phân loại về 1 class có tên là "hoi_thoi_tiet" chẳng hạn.

Hướng giải quyết của mình là với mỗi nhóm câu hỏi chuẩn bị sẵn từ 3-5 mẫu câu hỏi để làm training data. Sau đó sử dụng phương pháp **Bag-of-words, TF-IDF**, và cuối cùng là sử dụng 1 model tiến hành classification. Nếu bạn chưa biết **Bag-of-words, TF-IDF** là gì thì hãy tham khảo tại bài viết trước (http://codetudau.com/bag-of-words-tf-idf-xu-ly-ngon-ngu-tu-nhien/) của mình nhé.

Tiếp theo là công đoạn tiền xử lý tiếng việt, mình cũng đã có 1 bài viết nói về vấn đề này tại đây (http://codetudau.com/gioi-thieu-tien-xu-ly-trong-xu-ly-ngon-ngu-tu-nhien/), điều khó khăn nhất là:

Trong tiếng Việt, dấu cách (space) không được sử dụng như 1 kí hiệu phân tách từ, nó chỉ có ý nghĩa phân tách các âm tiết với nhau. Vì thế, để xử lý tiếng Việt, công đoạn tách từ (word segmentation) là 1 trong những bài toán cơ bản và quan trọng bậc nhất.

Ví dụ : từ “đất nước” được tạo ra từ 2 âm tiết “đất” và “nước”, cả 2 âm tiết này đều có nghĩa riêng khi đứng độc lập, nhưng khi ghép lại sẽ mang một nghĩa khác.

Vì đặc điểm này, bài toán tách từ trở thành 1 bài toán tiền đề cho các ứng dụng xử lý ngôn ngữ tự nhiên khác như phân loại văn bản, tóm tắt văn bản, máy dịch tự động, …

Về bài toán vietnamese word-segmentation:

Để giải quyết bài toán này cũng cần một lượng lớn dữ liệu. Dữ liệu đó đòi hỏi sự chính xác cao và tốn rất nhiều công sức. Tuy nhiên hiện nay cộng động xử lý ngôn ngữ tự nhiên tiếng việt khá mạnh nên có rất nhiều mã nguồn chất lượng. Với độ chính xác khá cao, dễ dàng sử dụng nên mình sử dụng thư viện pyvi (https://pypi.python.org/pypi/pyvi) có sẵn trên python. Bạn muốn tự làm công đoạn này hoặc muốn tìm hiểu sâu hơn về phương pháp thì hãy tham khảo thêm phần link đính kèm cuối bài viết nhé.

Bước 3: Trích xuất đặc trưng (Extracting features)

Để có thể sử dụng ML algorithms bạn cần vector hóa văn bản, đoạn text đầu vào. Như đã nói ở trên mình sử dụng **Bag-of-words**. Đầu vào của **Bag-of-words** là đoạn text đã được tách từ sử dụng **pyvi**.

Đây là đoạn code sử dụng **pyvi** thực hiện tách từ.

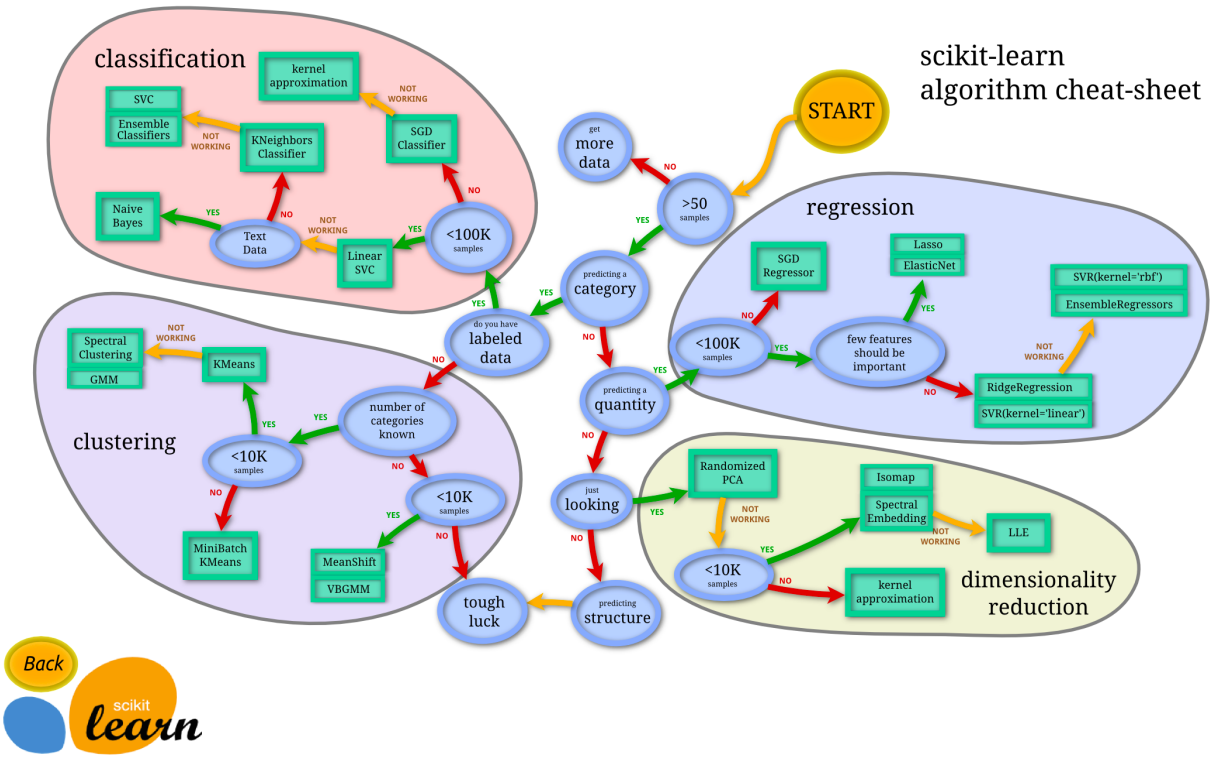
```
from pyvi.pyvi import ViTokenizer, ViPosTagger
from sklearn.base import TransformerMixin, BaseEstimator


class FeatureTransformer(BaseEstimator, TransformerMixin):
    def __init__(self):
        self.tokenizer = ViTokenizer()
        self.pos_tagger = ViPosTagger()

    def fit(self, *_):
        return self

    def transform(self, X, y=None, **fit_params):
        result = X.apply(lambda text: self.tokenizer.tokenize(text))
        return result
```

Sau đó sẽ dùng phương pháp **Bag-of-words** như mình đã giới thiệu ở bài trước để vector hóa từng câu sau khi

được tách từ.

Bài này tôi cũng sử dụng Pipeline, tại sao nên sử dụng Pipeline và cách sử dụng nếu các bạn chưa rõ thì có thể xem tại bài viết trước của tôi tại đây (http://codetudau.com/lam-quen-pipeline-sklearn-python/).

Bước 4: Lựa chọn ML algorithms



Trên đây là tổng hợp các trường hợp thực tế, đối với lượng data nhiều hay ít và đối tượng bài toán chúng ta sẽ chọn được model phù hợp.

Trong trường hợp của mình vì lượng data rất ít nên khi tiến hành **Text classification** được khuyên nên sử dụng **Naive Bayes**. Thuật toán này ra sao thì mình không nhắc lại nữa. Thuật toán này cài đặt khá đơn giản, dễ hiểu và độ chính xác rất cao. Bạn có thể tham khảo link wiki tại đây (https://en.wikipedia.org/wiki/Naive_Bayes_classifier) , có ví dụ rất dễ hiểu.

Với việc sử dụng **Pipeline** và thư viện sklearn, việc viết code trở nên rất đơn giản.

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from src.transformer.feature_transformer import FeatureTransformer


class NaiveBayesModel(object):
    def __init__(self):
        self.clf = self._init_pipeline()

    @staticmethod
    def _init_pipeline():
        pipe_line = Pipeline([
            ("transformer", FeatureTransformer()),#sử dụng pyvi tiến hành word segment
ation
            ("vect", CountVectorizer()),#bag-of-words
            ("tfidf", TfidfTransformer()),#tf-idf
            ("clf", MultinomialNB())#model naive bayes
        ])

        return pipe_line
```

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import pandas as pd
from model.svm_model import SVMModel
from model.naive_bayes_model import NaiveBayesModel


class TextClassificationPredict(object):
    def __init__(self):
        self.test = None

    def get_train_data(self):
        # Tạo train data
        train_data = []
        train_data.append({"feature": u"Hôm nay trời đẹp không ?", "target": "hoi_thoi
_tiet"})
        train_data.append({"feature": u"Hôm nay thời tiết thế nào ?", "target": "hoi_t
hoi_tiet"})
        train_data.append({"feature": u"Hôm nay mưa không ?", "target": "hoi_thoi_tiet
"})
        train_data.append({"feature": u"Chào em gái", "target": "chao_hoi"})
        train_data.append({"feature": u"Chào bạn", "target": "chao_hoi"})
        train_data.append({"feature": u"Hello bạn", "target": "chao_hoi"})
        train_data.append({"feature": u"Hi kimi", "target": "chao_hoi"})
        train_data.append({"feature": u"Hi em", "target": "chao_hoi"})
        df_train = pd.DataFrame(train_data)

        # Tạo test data
        test_data = []
        test_data.append({"feature": u"Nóng quá, liệu mưa không em ơi?", "target": "ho
i_thoi_tiet"})
        df_test = pd.DataFrame(test_data)

        # init model naive bayes
        model = NaiveBayesModel()

        clf = model.clf.fit(df_train["feature"], df_train.target)

        predicted = clf.predict(df_test["feature"])

        # Print predicted result
        print predicted
        print clf.predict_proba(df_test["feature"])


if __name__ == '__main__':
    tcp = TextClassificationPredict()
    tcp.get_train_data()
```

Như code mình viết ở trên, tạm thời có 2 class cần phân loại là **hoi_thoi_tiet** và **chao_hoi**, mỗi class mình có vài câu sử dụng làm training. Test data chính là câu **"Nóng quá, liệu mưa không em ơi?"** Thử xem có phân loại đúng không nhé.

Kết quả:

```
#predicted result
['hoi_thoi_tiet']
#predict_proba
[[ 0.05407486  0.94592514]]
```

Nhận thấy kết quả rất chính xác, có đến 94,59% dự đoán về class **hoi_thoi_tiet**. Có 1 lưu ý là model trên chưa tiến hành tuning hyper parameter. Nếu bạn chưa biết về quá trình tuning parameter thì có thể tham khảo bài viết của mình tại đây (http://codetudau.com/tim-kiem-high-parameter-voi-scikit-learn/).

Tuy nhiên việc xác định độ chính xác thì cần 1 lượng kha khá dữ liệu thì kết quả mới chính xác nhé.

Từ ví dụ trên bạn hoàn toàn có thể tự xây dựng 1 hệ thống chatbot cho riêng mình, dữ liệu bạn càng nhiều thì chatbot của bạn càng có khả năng trả lời nhiều câu hỏi, trò chuyện được nhiều hơn. ^^ Đơn giản như việc hỏi thời tiết, sau khi predict về class **hoi_thoi_tiet** bạn code call api weather để lấy thông tin thời tiết và trả lời cho người dùng, rất hữu ích phải không.

Ví dụ như em này:

Bước 4b:

Trong trường hợp bạn có 1 lượng lớn data, như trên sơ đồ là hơn 100k samples, bạn có thể thử dùng Support Vector Machines. Tuy dữ liệu mình ít nhưng cũng thử xem sao:

```
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from src.transformer.feature_transformer import FeatureTransformer
from sklearn.linear_model import SGDClassifier


class SVMModel(object):
    def __init__(self):
        self.clf = self._init_pipeline()

    @staticmethod
    def _init_pipeline():
        pipe_line = Pipeline([
            ("transformer", FeatureTransformer()),
            ("vect", CountVectorizer()),
            ("tfidf", TfidfTransformer()),
            ("clf-svm", SGDClassifier(loss='log', penalty='l2', alpha=1e-3, n_iter=5,
random_state=None))
        ])

        return pipe_line
```

```
['hoi_thoi_tiet']
[[ 0.07582323  0.92417677]]
```

Kết quả phân loại cũng chính xác. Tuy nhiên mình chưa có data đủ lớn để đánh giá accuracy cho các model trên.

Cụ thể về SVM thì mình sẽ cập nhật vào bài viết tiếp theo.

Kết luận

Text Classification là 1 bài toán khá hay và nhiều ứng dụng trong thực tế. Ngoài Naive Bayes, SVM, bạn hoàn toàn có thể dùng Random Forest, SLDA, Logistic Regression, Decision Trees, Neural Networks, Boosting and Bagging algorithms ... Và tất nhiên không có model nào tốt nhất, trong thực tế bạn cần thử, đánh giá nhiều model và tiến hành tuning parameter để có thể đạt kết quả tốt nhất.

Mong bài viết đem lại lợi ích cho các bạn. Hẹn gặp lại các bạn ở các bài viết sau.

Full source code
https://github.com/thanhttvn/machinelearning_codetudau/tree/master/text_classification_example (https://github.com/thanhttvn/machinelearning_codetudau/tree/master/text_classification_example)

Link tham khảo

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.106.220&rep=rep1&type=pdf (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.106.220&rep=rep1&type=pdf)

https://www.scribd.com/fullscreen/342411823?access_key=key-UJOfjnzF9fZfZiAgnaR8 (https://www.scribd.com/fullscreen/342411823?access_key=key-UJOfjnzF9fZfZiAgnaR8)

https://github.com/magizbox/underthesea/wiki/Vietnamese-NLP-Tools#text-classification (https://github.com/magizbox/underthesea/wiki/Vietnamese-NLP-Tools#text-classification)

http://www.clc.hcmus.edu.vn/wp-content/uploads/2016/01/CLC_Vietnamese-Word-Segmentation.pdf (http://www.clc.hcmus.edu.vn/wp-content/uploads/2016/01/CLC_Vietnamese-Word-Segmentation.pdf)

ALSO ON CODETUDAU

Phân tích cây quyết định với scikit-learn | Code từ đâu - Code từ đầu

1 comment • a year ago

Hằng Phan Thúy — đúng thứ đang tìm, cảm ơn anh.

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

1 comment • 7 months ago

Phuc Coi — hay

Tìm hiểu phương pháp k-means | Code từ đâu - Code từ đầu

4 comments • 9 months ago

Cao Thinh — Cảm ơn bạn vì bài chia sẻ rất hữu ích. Cho mình hỏi một số parameters của Kmeans1) random state =0 nghĩa là gì?2)

Đi tìm cội nguồn của Deep Learning - Perceptron | Code từ đầu - machine learning

1 comment • 3 months ago

Gou Tain — Có thanh niên nào giải thích được vì sao 1 perceptron không biểu diễn được XOR ko.Tất nhiên là ngoài câu "Không tin bạn