School of Data and Computer Science, Sun Yat-sen University

SVM – Text Classification

17341111 Xuehai Liu

2020 年 6 月 30 日

目录

1 Acquirement

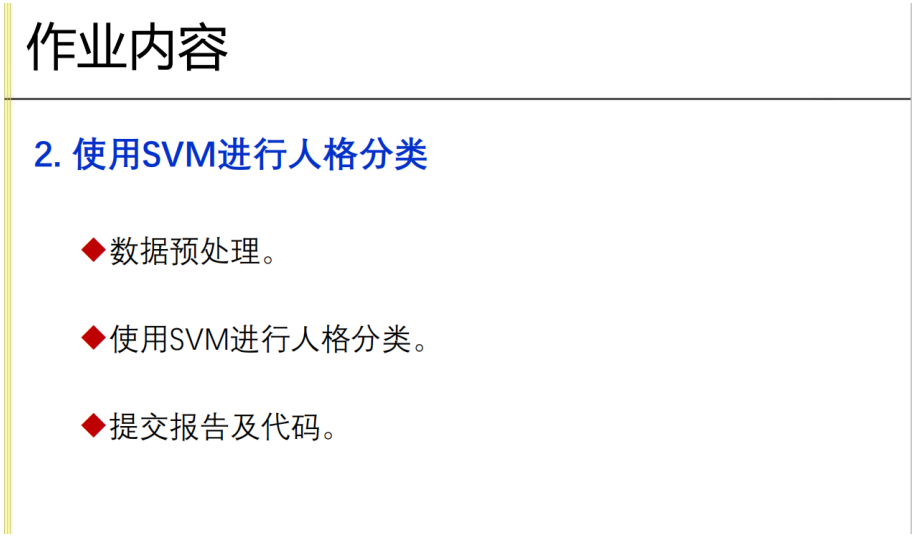

图 1:

This experiment needs to use SVM model to classify users' personalities according to their speech documents. Firstly, the data needs to be preprocessed, including denoising and vectorizing. Then, the SVM model is built and trained by the preprocessed data. Finally, the experimental results are verified and the acc and F1 score are calculated to evaluate the model.

2 Preprocessing–denoising and vectorizing

2.1 denoising

First of all, we can see that the data is in the form of a column of labels corresponding to a line of user's documents, in which the user's documents are mixed with irrelevant information (such as websites). Therefore, we need to carry out certain denoising processing first. In addition, many words in the user's speech have nothing to do with the predicted results of the model and need to be removed (this includes stop words and words that are often used by all types of personality, such as 'think', 'people'). In the experiment, I use nltk's stopword set and expand it.

2.2 vectorizing

After getting the denoised document, to use the document for training SVM model, it is necessary to extract the features of the document and quantify it. The word bag model is used in vectorization, and tfdif is used to extract the key features of documents, where TF stands for term frequency, and TFIDF stands for word frequency times inverse document frequency.

- **TF-IDF**: For a given keyword set, we need to define an evaluation index of the relevance

(importance) of the word set to the document. In the first step, we can determine the word frequency (TF). The most words in a document are related to the topic, so word frequency is an important feature evaluation criterion. However, even if some words appear as many, they are of different importance to document classification. If a word is rare, but it appears many times in this article, then it is likely to reflect the characteristics of this article, which is the key word we need.

Therefore, it is necessary to define a new indicator to measure the common degree of feature words, which is called inverse document frequency (IDF). Its size is inversely proportional to the common degree of a word. After knowing the word frequency (TF) and inverse document frequency (IDF), multiply the two values to get the TF-IDF value of a word. The more important a word feature is to the article, the greater its TF-IDF value is. Therefore, the set of several words with the largest TF-IDF value is the key feature subset of this paper.

$$feature\_importance = TF * IDF$$

The following codes is the code of proprocessing. Here, I extract several features of the document by ordering the TF-IDF score. The final result of the preprocessing is a dataframe in the shape of 8675 * (feature_num + 1) , where feature_num refers to the number of key words choosed by TF-IDF score, and the last column refers to labels.

```
# Data Aggregate
import numpy as np
import pandas as pd
import sklearn
import re
import nltk
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import Normalizer
from sklearn.pipeline import make_pipeline

#扩充停用词以进行去噪
stopwords = nltk.corpus.stopwords.words('english')
stopwords.append("think")
stopwords.append("people")
stopwords.append("thing")
stopwords.append("http")
stopwords.append("https")
stopwords.append("youtube")

labeldict = {}
```

```
#创建数据集，同时将labels离散化
def createDataset(path):
    data = pd.read_csv(path, header=None)
    data = data.values.tolist()
    labels = []
    dataset = []
    i = 0
    for line in data[1:]:
        if(line[0] not in labeldict.keys()):
            labeldict[line[0]] = i
            labels.append(i)
            i += 1
        else:
            labels.append(labeldict[line[0]])
        dataset.append(line[1])
    return dataset,labels

mydataset, labels = createDataset("mbti_1.csv")

#使用tfidf进行去噪并提取关键特征
tfidf_vectorizer = TfidfVectorizer(max_features=1000, stop_words=stopwords, use_idf=True
                                                    )

X = tfidf_vectorizer.fit_transform(mydataset)

#使用SVD将特征降维到20维
svd = TruncatedSVD(20)
normalizer = Normalizer(copy=False)
lsa = make_pipeline(svd, normalizer)
mydataset = lsa.fit_transform(X)

#得到最终的dataframe形式的dataset，其最后一列为labels
mydataset = pd.DataFrame(mydataset)
mydataset['labels'] = labels
```

3 SVM Model Constructing, Training and Testing

3.1 Model Constructing

In this section, I would construct the SVM model. Here, I would analyze how certain parameters would influence the result's performance.Finally, the model would be trained with different parameters and scored by acc-score and F1-score.

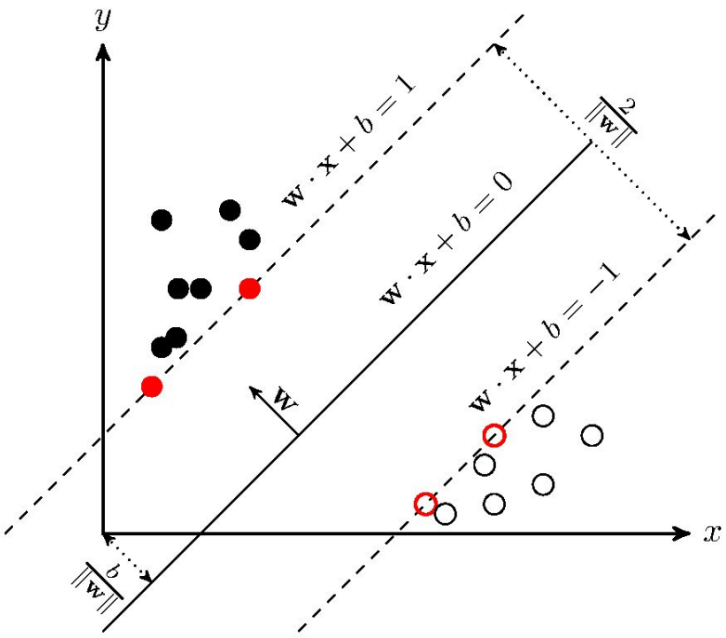

图 2: SVM

- **SVM**: Support vector machines (SVM) is a binary classification model. Its basic model is the linear classifier with the largest interval defined in the feature space. The maximum interval makes it different from the perceptron; SVM also includes kernel techniques, which makes it a non-linear classifier in essence. The learning strategy of SVM is to maximize the interval, which can be formalized as a convex quadratic programming problem, which is equivalent to the minimization of the regularized hinge loss function. The learning algorithm of SVM is the optimization algorithm to solve convex quadratic programming.

The Following code is about constructing the SVM model and training with certain parameters. Later I would show how the parameters would influence the result.

```
from __future__ import division
import pandas as pd
import numpy as np
import copy
import random
import math
from sklearn import svm
from preDeal import mydataset
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score

def main():
    #dataset 的形式是特征 + labels
    df = mydataset
```

```
    df_train, df_test = train_test_split(df, test_size=0.2)
    labels = df.columns.values.tolist()

    #获取训练和测试数据
    x_train = np.array(df_train.ix[:,0:df.shape[1]-2].values.tolist() )
    y_train = np.array(df_train.ix[:,'labels'].values.tolist() )
    x_test = np.array(df_test.ix[:, 0:df.shape[1]-2].values.tolist())
    y_test = np.array(df_test.ix[:, 'labels'].values.tolist())
    #训练SVM 模型
    clf = svm.SVC(C=0.8, kernel='rbf', gamma=20, decision_function_shape='ovr')
    clf.fit(x_train, y_train.ravel())

    #验证模型，取得分数
    print ("训练集精度：",clf.score(x_train, y_train))  # 精度
    y_pred = clf.predict(x_train)
    print ("测试集精度：",clf.score(x_test, y_test))
    y_pred = clf.predict(x_test)
    print("F1 score:",f1_score(y_test, y_pred, average='macro'))
main()
```

3.2 Model Training and Parameter analysis

Here, I mainly experiment 3 parameters of SVM model ,and record the result on both train set and test set. I would use table to show them briefly.Later, I would analyze how these parameters influence the model performance. The parameters of SVC are as follows.

$$clf = svm.SVC(C = 0.8, kernel =' rbf', gamma = 20)$$

- **C**:

C	kernel	gamma	acc on train	acc on test	F1 score
0.8	rbf	20	0.8444	0.5752	0.2979
0.9	rbf	20	0.8880	0.5424	0.2568
1	rbf	20	0.9154	0.5729	0.2872

  We can see that the larger C is, the better the classification effect is, but it may be over fitted. The best C here is 0.8.

- **kernel**:

C	kernel	gamma	acc on train	acc on test	F1 score
0.8	rbf	20	0.8444	0.5752	0.2979
0.8	sigmoid	20	0.1681	0.1648	0.041
0.8	linear	20	0.6327	0.6265	0.3081

  It's obvious that kernel function has an important influence on model performance. Here, RBF

kernel (default) correspond to Gaussian kernel. We can conclude that in such kind of linear classification problem, the linear kernel could get a better performance.

- **gamma**:

C	kernel	gamma	acc on train	acc on test	F1 score
0.8	rbf	20	0.8444	0.5752	0.2979
0.8	rbf	15	0.8222	0.5844	0.3011
0.8	rbf	10	0.778	0.5907	0.3317

The smaller the gamma value is, the more continuous the classification interface is; the larger the gamma value is, the more scattered the classification interface is, and the classification effect is better, but it may be over fitted.

4 Result Analysis and Model Comparison

4.1 Result

Finally, I would present one of the best parameters of the experiment and compare the SVM model with some popular models.The parameters are discribed as follow:

$$C = 0.8, kernel =' linear', gamma = 15, decision\_function\_shape =' ovr'$$

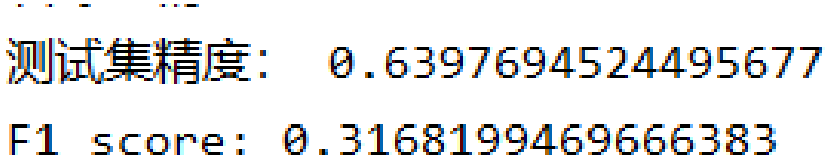

图 3: result

4.2 Model Comparison

Model	acc on test	F1 score
SVM	0.6398	0.3168
Random forest	0.4032	0.1012
Decision tree	0.3302	0.0901
logistic regression	0.6236	0.3007

We can see that SVM model gets the best performance among the four models. In SVM model, adding or reducing any points outside the support vector machine has no effect on the result, which means that SVM is not sensitive to outliers, comparing to logistic regression and dicision tree model.

What's more, when SVM is transformed into dual problem, only the distance between SVM and a few support vectors is needed. This method has obvious advantages in complex kernel function calculation and can greatly simplify the model and calculation. Therefore, SVM is an overall simple and effective model to do classification jobs.