Tweet Sentiment classifier

1. Introduction

Twitter is a popular microblogging service and it is also the name of a website. In comparison tweets are the messages that are sent over twitter, with no more than 140 characters. Each tweet can be classified as “positive”, “neutral” and “negative”. For an example, a tweet like “http://t.co/QV4m1Un9 Forget the phone.. Nice UI. Liking the Scroll Feature #android #google #nexus” and its sentiment is “positive”. Here I introduce a series of ideas and methods to automatically classify the sentiment.

Big challenges can be faced in tweet sentiment classifier: (i) neutral tweets are more common than positive and negative ones so its an imbalanced label problem. This is different from other sentiment classifier domains(e.g. product reviews), which tend to be predominantly positive or negative; (ii) tweets are very short and often show limited sentiment cues.

There are many traditional classifiers like Naive Bayes, Random Forest, and Support Vector Machines to solve such problems. Not only that, we can also use some methods of neural networks which are more complicated.

2. Main work

a. Data analysis

From the introduction, we have known the maximum of characters is 140 and its a very important value. We can limit our feature map to a limited number rather than a dynamic number. (i) After we get the dataset, we can draw the hist of class distribution. in most cases, “neutral” is domain to “positive” and “negative”. (ii) we can count all the words that appear in the dataset.

b. Data preprocess

It's important to do the data preprocessing because there are some irrelevant and redundant information. These noises will take considerable amount of processing time and make it difficlut to train a classifier. All in all, data preprocessing will help you get a better result.

In the train and test set, there are many URLS and some names followed by #@. Besides stop words, links, punctuation are also need to be removed. These words will not do help to our result so we need to delete them. It’s convenient to use regular expression.

Here is a trick that we can use the opinion lexicon which is a list including negative and positive words. Examples of positive opinion words are beautiful, wonderful, good, and amazing and examples of negative opinion words are bad, poor, and terrible.

c. Tweets representation

There are three strategies to represent the tweets: (i) bag-of-words. Tweets are represented by a table in which the columns represent the terms in the tweets. Therefore, we get a collection of tweets which contains n tweets and m terms. Each tweet is represented as $(a_{i1}, a_{i2}, \ldots, a_{im})$,where a_{ij} is the frequency of term t_j in the $tweet_i$. (ii) feature hashing. This method can reduce the number of features provided as input to a learning algorithm. The original high-dimensional space is reduced by hashing the features into a lower-dimensional space. (iii) word-vector. It is a row of real-valued numbers where each point captures a dimension of the word's meaning and where semantically similar words have similar vectors. We can say that word vectors represent the meaning of a word and it's different to the other strategies before.

However, the simple bag-of-words use term frequency to represent the words and a better way is adding inverse document frequency. This TD-IDF will help us get the real importance.

d. Classifier training

It's so happy that sklearn has offered many existing classifier and we can use them directly. I have delved into Naive Bayes model and support vector machine model. After we train the classifier, we should evaluate the quality of this classifier. It's better for us to evaluate the result on the test set. Unfortunately, we can only submit our result two times everyday. So I choose the method of cross_val_score. Then, the parameters will determine the effect of the classifier. We can use GridSearchCV to help us choose best parameters automatically.

e. Model stacking

We can find that each classifier has a different result on the test set. It will be better to combine these classifier to get a combined classifier. The easiest way is to use the method of voting. For an example, if we have trained three classifiers and we get the result of 2 "positive" 1 "negative", then we will vote to "positive". Unfortunately, it's okay to choose randomly if we get three different results.

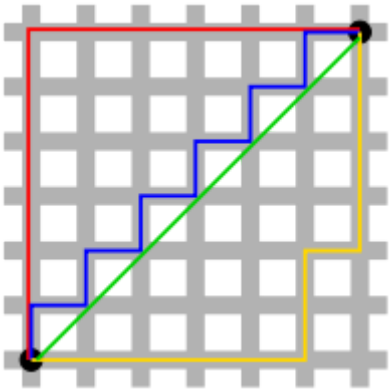
3. Conclusion

I have tried many ways to improve the performance. I consulted the data sheet over and over and read several related papers. What makes me desperate is that many ways do not improve the result, but the simplest model is most effective. Here I will explain in theory what I think should be effective.

The first is data preprocessing, URLs, numbers, irrelevant words, etc. are not helpful because they have no Sentiment meaning. They can be regarded as noise, so we must find a way to remove them. The second is feature extraction. In the bag-of-words model, tf-idf performs better than tf, which can be analyzed in principle. It is also better to use a word vector model on a larger data. Finally is the classifier training. The first thing we should do is to use as many classifiers as possible to compare the effects. Then we try to use as many parameters as possible to compare the effects. So that we have an optimal classifier with the best parameters. Similarly, for a large data set, neural networks can also be useful for better results. When encountering multiple classifiers, the effects of the classifier are not inferior, we can consider model stacking to make the classifier achieve better results.

L1和L2

> 范数定义这里不再讲解，只给出一个图



上图出自Wiki-Taxicab geometry，绿线就是两点之间的L2距离，红线蓝线黄线都是L1距离

L1 和 L2 范数在机器学习上主要应用于：

- 用作损失函数
- 用作正则化项

损失函数

回归问题中，我们需要在一组数据点中找到一条线，使得点到线的总距离最小

1. 使用距离的L1范数作为损失函数，又被称为 **least absolute deviation (LAD，最小绝对偏差)**

$$J = \sum_{i=1}^{n} |y_i - f(x_i)| \tag{1}$$

2. 使用距离的L2范数作为损失函数，又被称为 **least squares error (LSE，最小二乘误差)**:

$$J = \sum_{i=1}^{n} (y_i - f(x_i))^2 \tag{2}$$

> 问题1：为什么大多使用L2范数作为损失函数呢?

求解的时候L2损失函数可以**直接求导，计算更方便**

L1损失函数也是有优点的，它**鲁棒性更强，对异常值更不敏感**，L2对大数的惩罚更大，

正则化项

正则化主要用于解决拟合问题中的**过拟合**，降低模型的复杂度，减少特征的数量

衡量一个模型复杂度可以用VC维，而一般VC维和系数 w 的数量成正比；于是要降低模型的复杂度，减少系数 w 的数量就行了，即限制 w 中非零元素的数量。我们可以用一个优化问题来描述：

$$\min_w J(w;X,y)$$
$$\text{s.t. } \|w\|_0 \leq C$$

这里用的L0范数，但L0范数不易求解，于是使用L1和L2范数来近似求解，

$$\min_w J(w;X,y)$$
$$\text{s.t. } \|w\|_1 \leq C$$

$$\min_w J(w;X,y)$$
$$\text{s.t. } \|w\|_2 \leq C$$

这样我们就可以使用拉格朗日算子，就上述带约束条件的优化问题写成拉格朗日方程

$$L(w,\alpha) = J(w;X,y) + \alpha\left(\|w\|_1 - C\right) \tag{3}$$

我们假设 α 的最优解为 α^* ，则上式等价于

$$\min_w J(w;X,y) + \alpha^*\|w\|_1 \tag{4}$$

这样其实正则化就是在原优化目标函数中增加了一个约束条件

正则化的标准表示形式如下：

L1-regularization

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(y_j - \sum_i w_i h_i\left(\mathbf{x}_j\right)\right)^2 + \lambda \sum_{i=1}^{k} |w_i| \tag{5}$$

L2-regularization

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(y_j - \sum_i w_i h_i\left(\mathbf{x}_j\right)\right)^2 + \lambda \sum_{i=1}^{k} w_i^2 \tag{6}$$

两者特点总总结：

- L2 计算起来更方便， L1 在非稀疏向量上的计算效率很低
- L1 最重要的一个特点，输出稀疏，会把不重要的特征直接置零，而 L2 则不会；稀疏性在面对large-scale的问题来说很重要，因为可以减少存储空间；实际上L1也是一种妥协的做法，要获得真正sparse的模型，要用L0正则化
- L2 有唯一解，而 L1 没有唯一解

再多探讨一下L1和L2正则项的梯度差异：

L1的梯度是两个相反的固定值（例如1和-1），每次更新迭代的时候，它都是稳步向0前进。

L2的梯度越靠近0，会越来越小。