Report

学号	姓名	方向
18340013	陈琮昊	人工智能与大数据

一、Experimental Task

This experiment requires us use `SVM (Support Vector Machine)` method to classify the `MBTI` personality in the data set of `Kaggle`. The data set gives a total of 8675 personality tags and corresponding posts text data. We need to analyze the text content of posts and extract features to finish this classification task.

Personality tags are as follows:

1. Extroversion (E) - Introversion (I)
2. Sensing (S) - Intuition (N)
3. Thinking (T) - Feeling (F)
4. Judging (J) – Perceiving (P)

After the combination, we can get 16 personality types. For convenience, a string of abbreviations is used to represent a personality type. For example, `INTP` represent `'Introversion, Intuition, Thinking, Perceiving'`.

二、Experimental Principle

1. SVM：

`SVM` is a classifier to find a hyperplane to divide the samples into two categories, which has the maximum `margin`, that is, the distance from the hyperplane to the samples(*i.e.* **Support Vector**) is the largest. With the knowledge of mathematics, we can get a convex optimization problem：

$$\begin{aligned} min \quad & \frac{1}{2}||w||^2 \\ s.t. \quad & y_i(wx_i+b) \geq 1 \end{aligned}$$

For this optimization problem, we can use Lagrangian multiplier method to transform it into its dual problem. Also, we have the concept of `soft margin`, which allows some samples not to satisfy the constraints. There is a penalty parameter called *C*. The larger the value of *C*, the greater the penalty for classification mistakes:

$$\begin{aligned} min \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i \\ s.t. \quad & y_i(wx_i+b) \geq 1-\xi_i, \quad \xi_i \geq 0 \end{aligned}$$

However, if the given data is not linearly separable, the above method can't be used directly. Therefore, we need to map the data to a high-dimensional space, where the data is linearly separable. Then we use `SVM` in this high-dimensional space to finish classification, also called `Kernel SVM`. Specifically, we replace inner product with `kernel function`:

$$K(x, z) = \phi(x)\phi(z)$$

We have the following kernel functions to choose: `sigmoid`, `rbf`, `poly` and `linear`(*i.e.* without kernel function).

---

2. TF-IDF:

`TF-IDF` is a statistical method to evaluate the importance of a word to a file set or one of the files in a corpus. The importance of a word increases with the frequency of its appearance in the document, but decreases with the frequency of its appearance in the corpus.

The more times a word appears in an article, the less it appears in all documents, the more it can represent the article. We have the following formulas:

$$TF-IDF(t,d,D) = TF(t,d) * IDF(t,D)$$
$$TF(t,d) = log(1 + freq(t,d))$$
$$IDF(t,D) = log(\frac{N}{count(d \in D : t \in d)})$$

From above formulas, we can know that the high frequency of words in a specific file and the low frequency of words in the whole file set can produce high weight `TF-IDF`. Therefore, `TF-IDF` tends to filter out common words and retain important words.

In this experiment, we use related functions(*e.g.* `svm.SVC()`, `TfidfTransformer`) provided by `sklearn` to finish NLP and classify the given data.

三、Experiment Content

---

1. Execution Environment:

OS: `Windows 10`.

Python Version: `Python 3.6`.

Packages: `numpy`, `pandas`, `re`, `csv`, `sklearn`, `matplotlib`.

---

2. Experimental Steps:

First, we need to process the given data to obtain word vector, then we use `SVM` to process word vector features for classification:

```
if __name__ == "__main__":
    data = pd.read_csv('mbti_1.csv')  # read data
    process(data)  # preprocess the data
    mbti,info=read() # read the data after preprocessing
    arg1,arg2=TFIDF(mbti,info)  # NLP
    SVM(arg1,arg2)  # SVM
```

Here I follow the above steps to introduce the code structure in details:

2.1 Data Preprocessing:

By observing the data, we can see that: each row is a sample, the first column is their personality type, and the second column is their posts text content, which has multiple pieces, separated by |||.

In data preprocessing, following points to pay attention:

First, we need to encode the personality type, the method I choose is to code the overall 16 personality types with the numbers 0-15, a number represents a personality type.

Second, there are many links in the data, they can't give us useful information, so we need to remove them.

Then, the string in posts related to personality types, such as 'intp', which we need to remove because it may prevent the classifier from learning nothing.

The code of this stage is as follows:

```
def process(data):
    # encoding the personality type:
    encode = {'INTJ': 0, 'INTP': 1, 'INFJ': 2, 'INFP': 3, 'ISTJ': 4, 'ISTP': 5,
'ISFJ': 6, 'ISFP': 7, 'ENTJ': 8, 'ENTP': 9, 'ENFJ': 10, 'ENFP': 11, 'ESTJ': 12,
'ESTP': 13, 'ESFJ': 14, 'ESFP': 15}
    typeinposts = ['INFJ', 'ENTP', 'INTP', 'INTJ', 'ENTJ', 'ENFJ', 'INFP',
'ENFP', 'ISFP', 'ISTP', 'ISFJ', 'ISTJ', 'ESTP', 'ESFP', 'ESTJ', 'ESFJ', 'infj',
'entp', 'intp', 'intj', 'entj', 'enfj', 'infp', 'enfp', 'isfp', 'istp', 'isfj',
'istj', 'estp', 'esfp', 'estj', 'esfj']
    persontype = []   # store personality type
    postsinfo = []    # store posts
    for row in data.iterrows():
        # remove useless information
        posts = row[1].posts
        temp = re.sub('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|(?:%[0-9a-fA-F][0-
9a-fA-F]))+', ' ', posts)
        temp = re.sub("[^a-zA-Z]", " ", temp)
        temp = re.sub(' +', ' ', temp).lower()
        # remove the words in posts which are related to personality type,
prevent the classifier from learning nothing.
        for j in range(len(typeinposts)):
            temp = temp.replace(typeinposts[j], "")
        # get personality and posts after preprocessing:
        label = encode[row[1].type]
        persontype.append(label)
        postsinfo.append(temp)
```

```
    # write them into a new file:
    with open("pre.csv", "w", newline='') as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(["type", "posts"])
        for k in range(len(postsinfo)):
            writer.writerow([persontype[k], postsinfo[k]])

def read():
    # read the data after preprocessing, this function can reduce the call of
process(), so it save time.
    data = pd.read_csv("pre.csv")
    perlist = data['type']
    postlist = data['posts']
    perlist = np.array(perlist)
    postlist = np.array(postlist)
    return perlist, postlist
```

2.2 Nature Language Processing:

The code of this stage is as follows:

```
def TFIDF(perlist,postlist):
    # according to experiment and after several attempts to determine the
parameter:
    word2vec =
CountVectorizer(analyzer="word",max_features=2000,max_df=0.8,min_df=0.05)
    # calculate the frequency of words
    print("WAIT...")
    freq = word2vec.fit_transform(postlist)
    # print(freq)
    # calculate tf-idf matrix:
    tftrans = TfidfTransformer()
    tfidf = tftrans.fit_transform(freq).toarray()
    # print tf-idf matrix:
    # print(tfidf, len(tfidf),len(tfidf[0]))
    return tfidf, perlist
```

2.3 Cross Validation:

We use cross validation to find the best parameter *C*(`Penalty Parameter`). To save time, we use cross validation method to select *C* only when the kernel function is `linear`. The code is as follows:

```
def SVM(tfidf, perlist):
    typeclass =
['INTJ','INTP','INFJ','INFP','ISTJ','ISTP','ISFJ','ISFP','ENTJ','ENTP','ENFJ','E
NFP','ESTJ','ESTP','ESFJ','ESFP']
    X = tfidf
    Y = perlist
    randomseed = 42
    size = 0.3
    # split data into train set and test set with proportion 7:3(according to
experiment):
```

```
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=size,
random_state=randomseed)
    kernel = "linear"
    # code for cross validation(K-Fold,where K=10):
    Ctemp = [0.5, 1.0, 2.0]     # hyper-parameter, determined by cross
validation.
    for C in Ctemp:
        for m in range(len(typeclass)):
            train_Y, test_Y = tobinary(Y_train, Y_test, m)
                for kernel in kernels:
                    method = svm.SVC(decision_function_shape='ovr',
kernel=kernel, C=C)
                    model = method.fit(X_train, train_Y)
                    scores = cross_val_score(method, X_train, train_Y, cv=10,
scoring='accuracy')
                    print("C={},score:{}".format(C, scores.mean()))
```

2.4 SVM Classification:

After cross validation, we can use the function `svm.SVC()` provided by `sklearn` to classify. We need to mark the label of the personality type which to be predicted in this round as `1`, the other 15 types as `-1` at the beginning of classification. The code is as follows:

```
def tobinary(train, test, K):
    # (train, test, K)==(y_train, y_test, the label of positive samples)
    # transfer muliti-class into binary-class, convert the label of positive
samples into 1, the label of other 15 types into -1:
    temp1 = np.zeros(len(train))
    for i in range(len(train)):
        temp1[i] = 1 if train[i]==K else -1
    train_Y = np.array(temp1, dtype=np.int)
    temp2 = np.zeros(len(test))
    for i in range(len(test)):
        temp2[i] = 1 if test[i]==K else -1
    test_Y = np.array(temp2, dtype=np.int)
    return train_Y, test_Y

def SVM(tfidf, perlist):
    typeclass =
['INTJ','INTP','INFJ','INFP','ISTJ','ISTP','ISFJ','ISFP','ENTJ','ENTP','ENFJ','E
NFP','ESTJ','ESTP','ESFJ','ESFP']
    X = tfidf
    Y = perlist
    randomseed = 42
    size = 0.3
    # split data into train set and test set with proportion 7:3(according to
experiment):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=size,
random_state=randomseed)
    kernels = ["linear","rbf","poly","sigmoid"]
    # predict the result in test set:
    for m in range(len(typeclass)):
        train_Y, test_Y = tobinary(Y_train, Y_test, m)
        for kernel in kernels:
            method = svm.SVC(decision_function_shape='ovr', kernel=kernel, C=1)
            model = method.fit(X_train, train_Y)
```

```
            output = model.predict(X_test)
            accuracy = accuracy_score(test_Y, output)
            print("{} for {} Accuracy:{}%".format(typeclass[m], kernel, accuracy
* 100.0))
```

3. Experimental Result

3.1 Cross Validation:

We use `cross_val_score` function to calculate the score of cross validation for different *C*. The score of cross validation are as follows(only when kernel function is `linear`):

```
scores = cross_val_score(method, X_train, train_Y, cv=10, scoring='accuracy')
# K-Fold, where K=10
```

	C=0.5	C=1	C=2
INTJ	0.8715	0.8719	0.8699
INTP	0.8518	0.8570	0.8536
INFJ	0.8294	0.8328	0.8211
INFP	0.7932	0.8053	0.8034
ISTJ	0.9778	0.9778	0.9778
ISTP	0.9611	0.9611	0.9625
ISFJ	0.9827	0.9827	0.9827
ISFP	0.9692	0.9692	0.9692
ENTJ	0.9715	0.9715	0.9715
ENTP	0.9203	0.9203	0.9201
ENFJ	0.9779	0.9779	0.9779
ENFP	0.9211	0.9211	0.9211
ESTJ	0.9964	0.9964	0.9964
ESTP	0.9895	0.9895	0.9895
ESFJ	0.9947	0.9947	0.9947
ESFP	0.9936	0.9936	0.9936

From the table, we can get the conclusion that *C*=1 is the best through horizontal comparison. We also see that many type classes in different *C* have the same score, because in these classes, the positive samples are far away from the negative samples, so changing the parameter *C* will not affect the performance.

3.2 SVM:

After determining the value of C, we can use `SVM` to predict the output on the test set:

```
method = svm.SVC(decision_function_shape='ovr', kernel=kernel , C=1)
model = method.fit(X_train, train_Y)
output = model.predict(X_test)
accuracy = accuracy_score(test_Y, output)
```

	linear	**rbf/poly/sigmoid**
INTJ	88.1291%	88.0522%
INTP	84.7484%	84.4794%
INFJ	83.3653%	83.3269%
INFP	78.4864%	78.2558%
ISTJ	97.3108%	97.3108%
ISTP	96.1199%	96.1199%
ISFJ	97.6565%	97.6565%
ISFP	96.7729%	96.7729%
ENTJ	97.7718%	97.7718%
ENTP	92.2781%	92.2781%
ENFJ	97.8486%	97.8486%
ENFP	92.4702%	92.4702%
ESTJ	99.3469%	99.3469%
ESTP	99.0395%	99.0395%
ESFJ	99.6158%	99.6158%
ESFP	99.6542%	99.6542%

From the table, we can get the conclusion that the top four types have difference when we use different kernel function. We get the best accuracy when we use `linear` as kernel function, and even get the same result in many classes, although we use different kernel functions. From the above, we can say that the given data is linearly separable.