20152410 배형준 머신러닝 과제11

In [1]:

```
# library import

#nltk.download('stopwords')
#nltk.download('wordnet')

import numpy as np
import re # 정규 표현식 : regular expression
import nltk # 자연어 처리 : natural language toolkit
from sklearn.datasets import load_files # Load text files with categories as subfolder names.
import pickle # 텍스트가 아닌 자료형 불러오기 (ex. list, class etc)
from nltk.corpus import stopwords # 불용어를 추출하는 함수
from nltk.stem import WordNetLemmatizer # 표제어를 추출하는 함수, 기본 사전형 단어
from sklearn.feature_extraction.text import CountVectorizer
# 문서 집합에서 단어 토큰을 생성하고 각 단어의 수를 세어 BOW 인코딩한 벡터를 만든다.
from sklearn.feature_extraction.text import TfidfTransformer
# CountVectorizer와 비슷하지만 TF-IDF 방식으로 단어의 가중치를 조정한 BOW 벡터를 만든다. (단어 가
from sklearn.model_selection import train_test_split # train, test 나눠주는 함수

import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
```

executed in 36.9s, finished 03:15:13 2020-06-17

문자열 전처리

In [2]:

```
# load files

review_data = load_files(r"movie_review")
review_data

# 문자열 앞에 r을 붙히면 raw 문자열로 인식해서 ₩n, ₩t같은 escape가 안통함
```

executed in 36.3s, finished 03:15:49 2020-06-17

Out[2]:

```
{'data': [b'bad . bad . ₩nbad . ₩nthat one word seems to pretty much sums up beyon
d the valley of the dolls . ₩nif that summary isn₩'t enough for you , how about t&
a , t&a , t&a ? ₩nstill haven₩'t got the point ? ₩nother than director russ meyer
₩'s predilection for casting attractive large breasted women who ultimately expose
the afore-mentioned anatomical areas , there is really only one other reason to re
commend even taking a look at this movie . ₩nthat is the fact that it was co-writt
en by famed film critic roger ebert , who also was responsible for the screenplay
. ₩nafter watching this movie you will never be able to sit through another one of
his reviews where he gives a movie a thumbs down for bad writing with a straight f
ace . ₩nthis movie stinks out loud . ₩nquite frankly , this movie deserves a . ₩nb
ut there are parts of it that are so bad they are almost funny . ₩nso i₩'m giving
it a . ₩nand maybe that is too generous . ₩nright from the opening credits , i kne
w that i had a class-a bomb on my hands . ₩nnot only are the way the credits actua
lly shot distracting , but the first scene you see includes a big breasted young w
oman being chased by a guy in a nazi uniform . ₩ni had absolutely no idea why the
hell that was happening ( it does get explained later ) and as soon as the first s
cene is over , we cut to a completely unrelated scene . ₩nto be honest , as i sat
through this movie mesmerized by just how incredibly awful it was . i actually for
```

In [3]:

```
# split data(review) and target(pos: 1, neg: 0)

X, y = review_data.data, review_data.target
```

executed in 10ms, finished 03:15:49 2020-06-17

In [4]:

```
# neg, pos 순으로 나와야 하니 확인

print(y[0], y[2])
print(review_data['filenames'][0], review_data['filenames'][2])
```

executed in 141ms, finished 03:15:50 2020-06-17

```
0 1
movie_review₩neg₩cv676_22202.txt movie_review₩pos₩cv238_12931.txt
```

re.sub 사용법

re.sub(pattern, repl, string) : string에서 pattern과 매치하는 텍스트를 repl로 치환한다

In [5]:

```
# 표제어 추출하기

documents = []

stemmer = WordNetLemmatizer()

for sen in range(0, len(X)):
    # Remove all the special characters
    document = re.sub(r'\W', ' ', str(X[sen])) # \W를 빈칸으로 치환

    # remove all single characters
    document = re.sub(r'\s+[a-zA-Z]\s+', ' ', document)

    # Remove single characters from the start
    document = re.sub(r'\^[a-zA-Z]\s+', ' ', document) # 문장 맨 앞에 A 같은거 제거

    # Substituting multiple spaces with single space
    document = re.sub(r'\s+', ' ', document, flags=re.I)

    # Removing prefixed 'b'
    document = re.sub(r'^b\s+', '', document)

    # Converting to Lowercase
    document = document.lower() # 소문자로 바꾸기

    # Lemmatization : 표제어 추출
    document = document.split()
    document = [stemmer.lemmatize(word) for word in document]
    document = ' '.join(document)

    documents.append(document)
```

executed in 12.8s, finished 03:16:02 2020-06-17

0. Optimization

In [6]:

```
class penalized_neural_network:

    def __init__(self, learning_rate, error_bound, iteration, cutoff, random_state,
                 hidden_layer, number_node, fit_intercept, alpha):
        self.learning_rate = learning_rate
        self.error_bound = error_bound
        self.iteration = iteration
        self.cutoff = cutoff
        self.random_state = random_state
        self.alpha = alpha # penalized hyper parameter
        self.number_parameter = 0

        self.hidden_layer = hidden_layer # int
        self.number_node = number_node # list of int
        self.fit_intercept = fit_intercept # True or False

        self.record_train_cost = []
        self.record_test_cost = []
        self.record_train_accuracy = []
        self.record_test_accuracy = []

        self.coef_list = []
        self.train_predict = []
        self.test_predict = []
        self.last_gradient = []

    def sigmoid(self, X, coef):
        z = np.dot(X, coef)
        sigmoid_value = 1 / (1 + np.exp(-z))

        return sigmoid_value

    def cost(self, X, coef_list, onehot_label):
        delta = 10**(-8)
        m = X.shape[0]
        temp = X
        sigmoid_list = []

        # forward propagation
        for coef in coef_list:
            sig = self.sigmoid(temp, coef)
            sigmoid_list.append(sig)

            if self.fit_intercept == True:
                temp = np.column_stack((np.ones((sig.shape[0], 1)), sig))
            else:
                temp = sig

        error_term = -np.mean(np.sum(onehot_label * np.log(sig + delta) + (1 - onehot_label) * np

        temp = 0
        for coef in coef_list:
            temp = temp + np.sum(coef**2)

        l2_term = (self.alpha * temp) / (2 * self.number_parameter)

        cost_value = error_term + l2_term

        return cost_value, sigmoid_list
    def gradient(self, X, coef_list, onehot_label, sigmoid_list):
        m = X.shape[0]
        delta_list = []
        gradient_list = []

        add_constant_sigmoid = []

        for i in range(len(sigmoid_list)):
            temp = np.column_stack((np.ones((sigmoid_list[i].shape[0], 1)), sigmoid_list[i]))
            add_constant_sigmoid.append(temp)

        sigmoid_list.insert(0, X)
        add_constant_sigmoid.insert(0, X)

        # backward propagation
        for i in range(self.hidden_layer+1):
            if i == 0:
                delta_value = sigmoid_list[-1] - onehot_label
                penarlized_term = self.alpha * coef_list[-1] / self.number_parameter
                gradient_value = np.dot(add_constant_sigmoid[-2].T, delta_value) / m + penarlized

                delta_list.insert(0, delta_value)
                gradient_list.insert(0, gradient_value)

            else:
                delta_value = np.dot(delta_list[0], coef_list[-i][1:, :].T) * sigmoid_list[-i-1]
                penarlized_term = self.alpha * coef_list[-i-1] / self.number_parameter
                gradient_value = np.dot(add_constant_sigmoid[-i-2].T, delta_value) / m + penarliz

                delta_list.insert(0, delta_value)
                gradient_list.insert(0, gradient_value)

        return gradient_list

    def predict(self, sigmoid_list, predict_type='class'):
        output_layer = sigmoid_list[-1]

        if predict_type == 'class':
            predict_value = np.where(output_layer.reshape(-1, 1) >= self.cutoff, 1, 0)
            # softmax가 아니고 binary response니까 그거에 맞춰 수정해줌

        elif predict_type == 'response':
            predict_value = output_layer

        return predict_value

    def fit(self, X_train, Y_train, X_test, Y_test): # Y_train, Y_test는 onehotencoding이 완료된
        X_train = np.array(X_train)
        Y_train = np.array(Y_train)
        X_test = np.array(X_test)
        Y_test = np.array(Y_test)
        m = X_train.shape[0]
        n = X_train.shape[1]
        q = X_test.shape[0]
        p = Y_train.shape[1]

        self.number_node.insert(0, n)
        self.number_node.append(p)
        coef_list = []
```

```python
        # fit_intercept
        if self.fit_intercept == True:
            number_node_with_intercept = []

            X_train = np.column_stack((np.ones((m, 1)), X_train))
            X_test = np.column_stack((np.ones((q, 1)), X_test))

            for number in self.number_node:
                number_node_with_intercept.append(number+1)

        else:
            number_node_with_intercept = self.number_node

        # calculate number of parameters
        number_parameter = 0
        for i in range(len(number_node_with_intercept)-1):
            temp = number_node_with_intercept[i]*number_node_with_intercept[i+1]
            number_parameter = number_parameter + temp

        self.number_parameter = number_parameter

        # set initial parameters
        np.random.seed(self.random_state) # for reproducibility

        for layer in range(self.hidden_layer+1):
            temp_theta = np.random.randn(number_node_with_intercept[layer], self.number_node[layer
            coef_list.append(temp_theta)

        # check model fitting progress
        import time
        start = time.time()

        # model fitting
        while True:
            # calculate train and test cost
            train_cost, train_sigmoid = self.cost(X_train, coef_list, Y_train)
            test_cost, test_sigmoid = self.cost(X_test, coef_list, Y_test)

            self.record_train_cost.append(train_cost)
            self.record_test_cost.append(test_cost)

            # calculate train and test accuracy
            train_predict = self.predict(train_sigmoid, predict_type='class').reshape(-1, 1)
            test_predict = self.predict(test_sigmoid, predict_type='class').reshape(-1, 1)

            train_accuarcy = np.mean(train_predict == Y_train)
            test_accuarcy = np.mean(test_predict == Y_test)

            self.record_train_accuracy.append(train_accuarcy)
            self.record_test_accuracy.append(test_accuarcy)

            # calculate gradient using back propagation and renew the parameters
            gradient_list = self.gradient(X_train, coef_list, Y_train, train_sigmoid)

            for i in range(len(coef_list)):
                coef_list[i] = coef_list[i] - self.learning_rate * gradient_list[i]

            # stopping rules
            length = len(self.record_train_accuracy)

            if length > self.iteration:
                if self.record_train_accuracy[-2] - self.record_train_accuracy[-1] < self.error_b
                    break

            # print model fitting progress
            running_time = time.time() - start
            minute = int(running_time // 60)
            second = round(running_time % 60, 1)

            if length % 500 == 0:
                print('Iter : {}, Running time : {}m {}s'.format(length, minute, second), end=', '
                print('Train accuracy : {}%, Test accuracy : {}%'.format(round(100*train_accuarcy,
                                                                     round(100*test_accuarcy,
                print('Train Cost : {}, Test Cost : {}₩n'.format(train_cost, test_cost))

            # error situation : too much iteration
            if length > 10000:
                print('반복 횟수가 너무 많습니다. Train Cost가 수렴하지 못했습니다. 학습률을 조정ㅎ
                break

        self.coef_list = coef_list
        self.train_predict = train_predict
        self.test_predict = test_predict
        self.last_gradient = gradient_list

        return self
```

executed in 72ms, finished 03:16:02 2020-06-17

모델링을 위한 데이터 준비

max_df : float in range [0.0, 1.0] or int, default=1.0

When building the vocabulary ignore terms that have a document frequency strictly higher than the given threshold (corpus-specific stop words). If float, the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None.

어떤 단어가 너무 많이 나온다 => 의미 없을 가능성이 있음 => 너무 많이 출몰하는 단어 제거

min_df : float in range [0.0, 1.0] or int, default=1

When building the vocabulary ignore terms that have a document frequency strictly lower than the given threshold. This value is also called cut-off in the literature. If float, the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None.

어떤 단어가 너무 적게 나온다 => 의미 없을 가능성이 있음 => 너무 적게 출몰하는 단어 제거

하이퍼 파라미터 튜닝 순서

tfidf로 변환했을 때 0이 적을수록 설명력이 뛰어나고 잡음이 많이 제거된 데이터라고 가정하겠음

1. max_features
2. min_df
3. max_df

test accuracy가 높을수록 성능이 좋은 모델

test accuracy가 높을 수록 성능이 좋은 모델

4. number of hidden layer
5. number of node of each hidden layer
6. alpha : weight of penalized term

1. max_features 튜닝

In [7]:

```
candidate_max_features = np.arange(40000, 100, -200)
x_axis = np.arange(0, 200, 1)
nonzero_list = []
zero_list = []
```

executed in 128ms, finished 03:16:03 2020-06-17

In [8]:

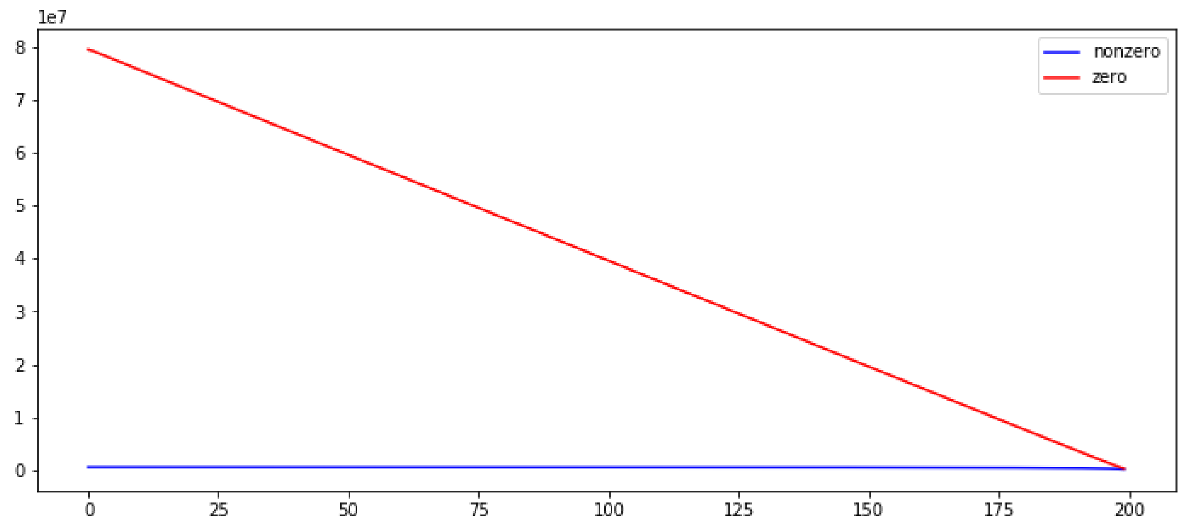
```
for i in candidate_max_features:
    vectorizer = CountVectorizer(max_features = i, min_df=0, max_df=1.0, stop_words=stopwords.wor
    X = vectorizer.fit_transform(documents).toarray()

    tfidfconverter = TfidfTransformer()
    X = tfidfconverter.fit_transform(X).toarray()

    shape = X.shape
    non_zero = np.count_nonzero(X)
    zero = shape[0] * shape[1] - np.count_nonzero(X)

    nonzero_list.append(non_zero)
    zero_list.append(zero)
```

executed in 9m 58s, finished 03:26:00 2020-06-17

In [9]:

```
plt.figure(figsize=(12, 5))
plt.plot(x_axis, nonzero_list, 'b-', label='nonzero')
plt.plot(x_axis, zero_list, 'r-', label='zero')
plt.legend()
plt.show()

# 이 그래프를 통해 어느 점에서 max_features를 정해야 할 지 알기 힘들다
```

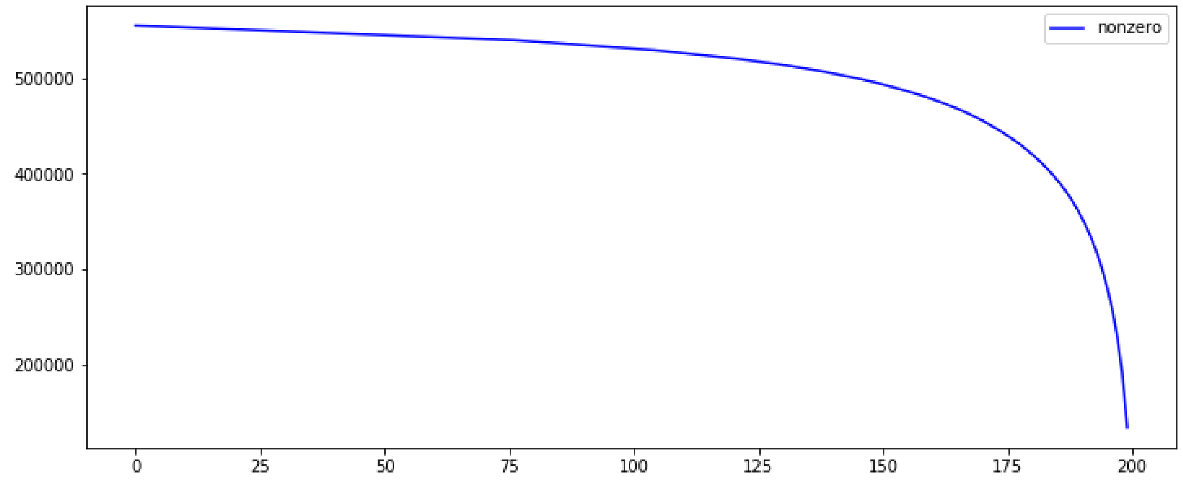
executed in 996ms, finished 03:26:01 2020-06-17



In [10]:

```
plt.figure(figsize=(12, 5))
plt.plot(x_axis, nonzero_list, 'b-', label='nonzero')
plt.legend()
plt.show()

# 서서히 감소하다가 급격히 감소하기 시작하는 점으로 max_features를 선정하겠다 : 약 175번째 max_fea
```

executed in 251ms, finished 03:26:02 2020-06-17



In [11]:

```
my_max_features = candidate_max_features[175]
```

executed in 100ms, finished 03:26:02 2020-06-17

2. min_df 튜닝

In [12]:

```
candidate_min_df = np.arange(200, -1, -1)
x_axis = np.arange(0, 201, 1)
min_df_nonzero_list = []
min_df_zero_list = []
```

executed in 531ms, finished 03:26:02 2020-06-17

In [13]:

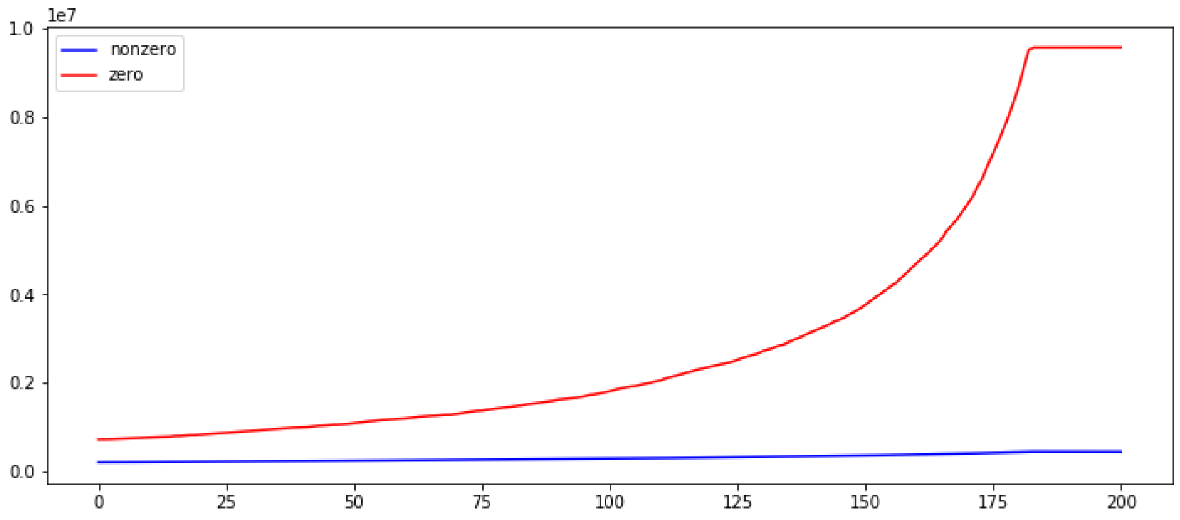
```
for i in candidate_min_df:
    vectorizer = CountVectorizer(max_features = my_max_features, min_df=i, max_df=1.0, stop_words
    X = vectorizer.fit_transform(documents).toarray()

    tfidfconverter = TfidfTransformer()
    X = tfidfconverter.fit_transform(X).toarray()

    shape = X.shape
    non_zero = np.count_nonzero(X)
    zero = shape[0] * shape[1] - np.count_nonzero(X)

    min_df_nonzero_list.append(non_zero)
    min_df_zero_list.append(zero)
```

executed in 5m 10s, finished 03:31:12 2020-06-17

In [14]:

```
plt.figure(figsize=(12, 5))
plt.plot(x_axis, min_df_nonzero_list, 'b-', label='nonzero')
plt.plot(x_axis, min_df_zero_list, 'r-', label='zero')
plt.legend()
plt.show()

# 이 그래프를 통해 어느 점에서 max_features를 정해야 할 지 알기 힘들다
```

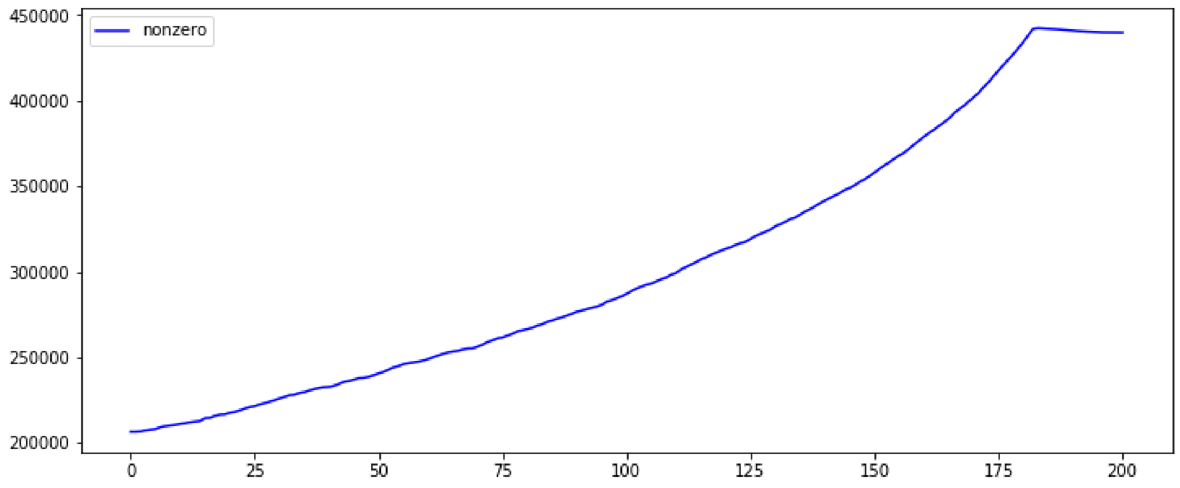
executed in 345ms, finished 03:31:13 2020-06-17



In [15]:

```
plt.figure(figsize=(12, 5))
plt.plot(x_axis, min_df_nonzero_list, 'b-', label='nonzero')
plt.legend()
plt.show()

# 160에서 180 사이가 적절할 것으로 추정됨
```

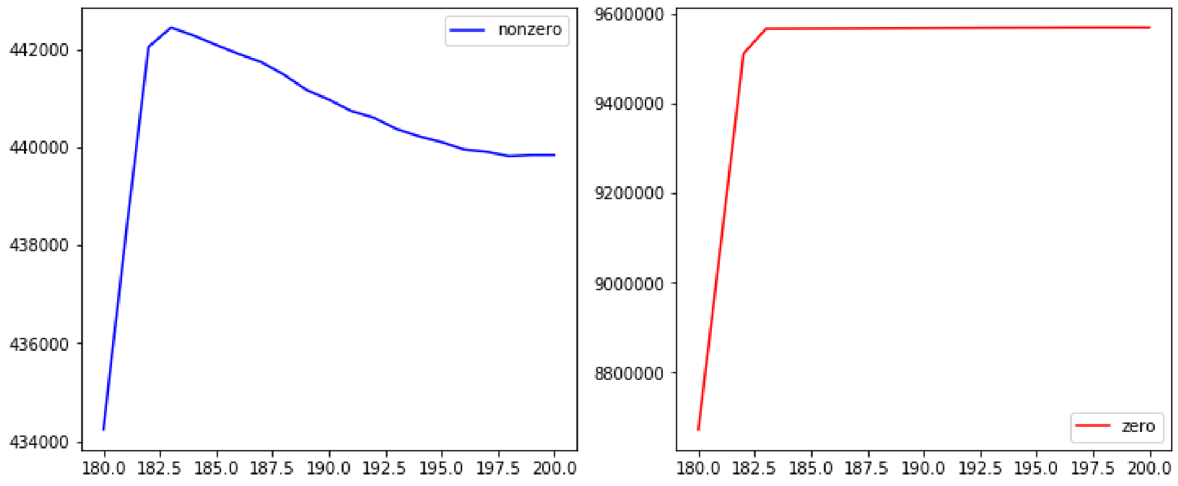
executed in 243ms, finished 03:31:13 2020-06-17



In [16]:

```
plt.figure(figsize=(12, 5))
plt.subplot(121)
plt.plot(x_axis[180:], min_df_nonzero_list[180:], 'b-', label='nonzero')
plt.legend()
plt.subplot(122)
plt.plot(x_axis[180:], min_df_zero_list[180:], 'r-', label='zero')
plt.legend()
plt.show()
```

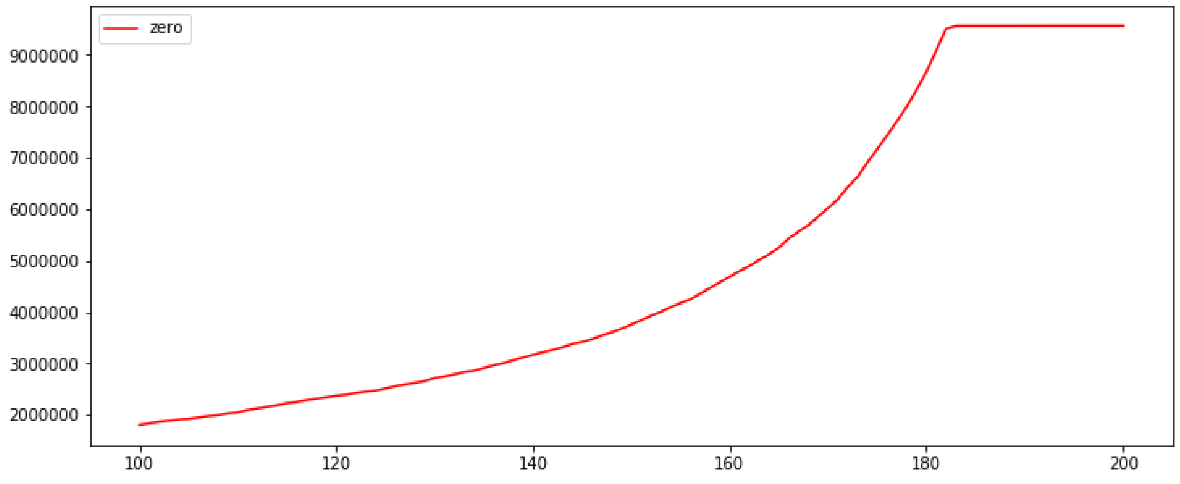
executed in 417ms, finished 03:31:13 2020-06-17



In [17]:

```
plt.figure(figsize=(12, 5))
plt.plot(x_axis[100:], min_df_zero_list[100:], 'r-', label='zero')
plt.legend()
plt.show()

# 그래프의 변곡점인 170번째 min_df를 선택하도록 하겠다.
# min_df가 더 커지면 0이 사라지는거보다 0이 아닌 친구들이 비교적 더 많이 사라져버림
```

executed in 244ms, finished 03:31:14 2020-06-17



In [18]:

```
my_min_df = candidate_min_df[170]
```

executed in 6ms, finished 03:31:14 2020-06-17

3. max_df 튜닝

In [19]:

```
candidate_max_df = np.arange(0.1, 1.0, 0.005)
x_axis = np.arange(0, 180, 1)
max_df_nonzero_list = []
max_df_zero_list = []
```

executed in 79ms, finished 03:31:14 2020-06-17

In [20]:

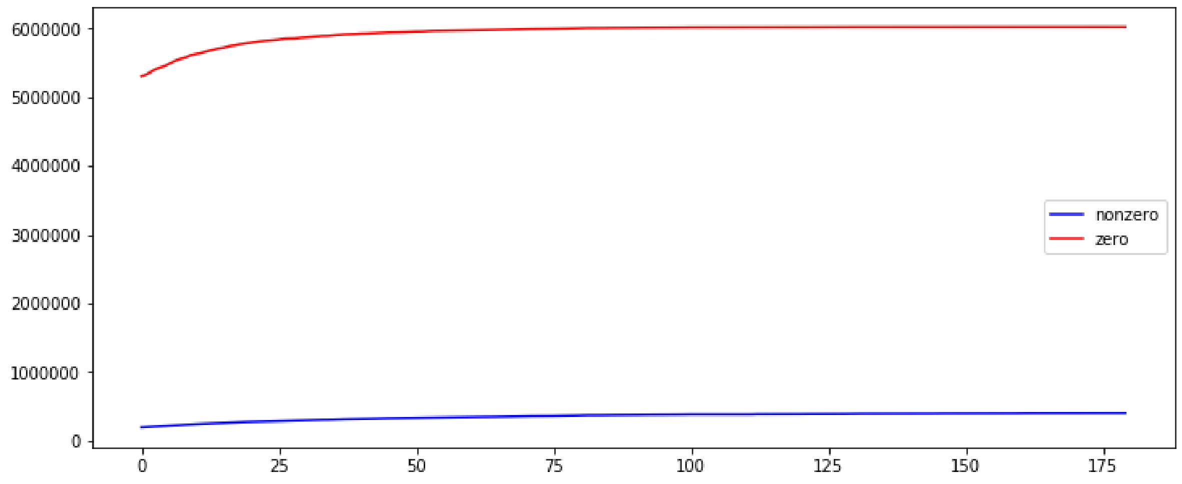
```
for i in candidate_max_df:
    vectorizer = CountVectorizer(max_features = my_max_features, min_df=my_min_df, max_df=i, stop
    X = vectorizer.fit_transform(documents).toarray()

    tfidfconverter = TfidfTransformer()
    X = tfidfconverter.fit_transform(X).toarray()

    shape = X.shape
    non_zero = np.count_nonzero(X)
    zero = shape[0] * shape[1] - np.count_nonzero(X)

    max_df_nonzero_list.append(non_zero)
    max_df_zero_list.append(zero)
```

executed in 4m 52s, finished 03:36:06 2020-06-17

In [21]:

```
plt.figure(figsize=(12, 5))
plt.plot(x_axis, max_df_nonzero_list, 'b-', label='nonzero')
plt.plot(x_axis, max_df_zero_list, 'r-', label='zero')
plt.legend()
plt.show()
```

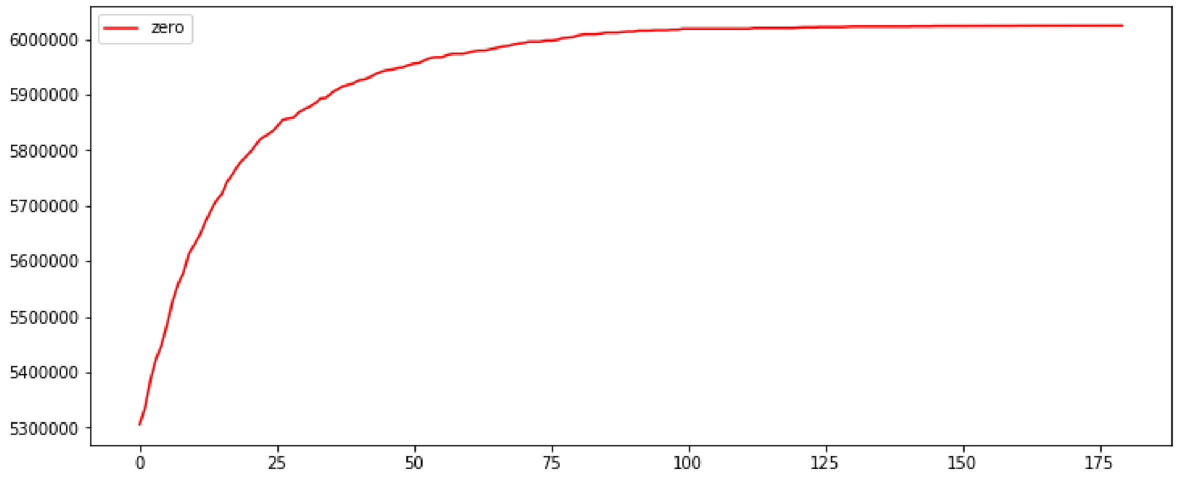
executed in 368ms, finished 03:36:07 2020-06-17



In [22]:

```
plt.figure(figsize=(12, 5))
plt.plot(x_axis, max_df_zero_list, 'r-', label='zero')
plt.legend()
plt.show()

# 이미 0은 많이 제거했으니 이 그래프로 max_df를 찾는 것은 의미가 없다고 생각함
```

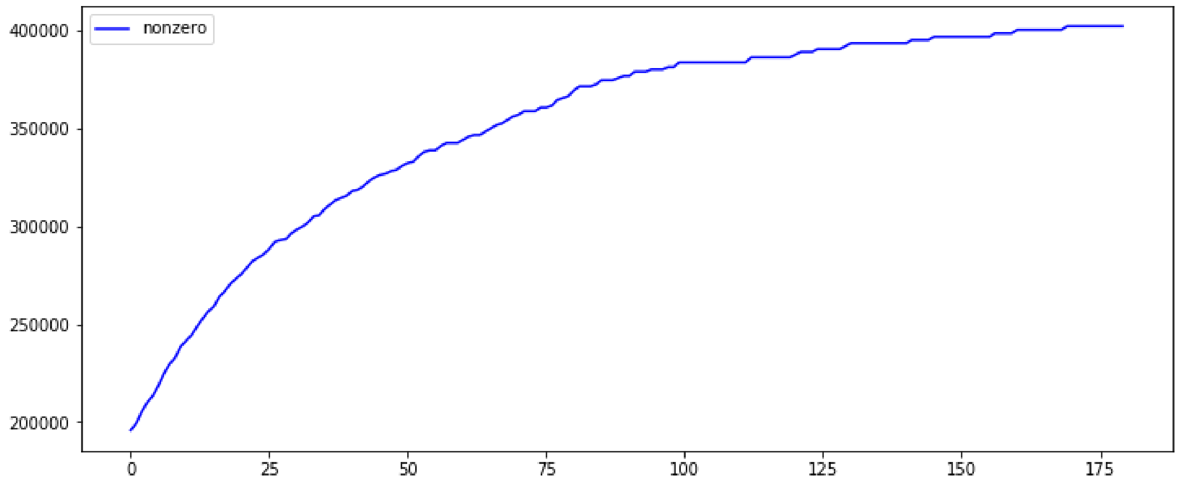
```
executed in 247ms, finished 03:36:07 2020-06-17
```



In [23]:

```
plt.figure(figsize=(12, 5))
plt.plot(x_axis, max_df_nonzero_list, 'b-', label='nonzero')
plt.legend()
plt.show()

# 너무 많이 나온 단어들만 제거해야되니까 150번째 정도에서 자르는 것이 좋아보인다
```

```
executed in 230ms, finished 03:36:07 2020-06-17
```



In [24]:

```
my_max_df = round(candidate_max_df[150], 3)
```

```
executed in 6ms, finished 03:36:07 2020-06-17
```

In [25]:

```
print('Output of tuning about CountVectorizer',
      '₩nmax_features: {}, min_df: {}, max_df: {}'.format(my_max_features, my_min_df, my_max_df))
```

executed in 90ms, finished 03:36:07 2020-06-17

```
Output of tuning about CountVectorizer
max_features: 5000, min_df: 30, max_df: 0.85
```

In [26]:

```
vectorizer = CountVectorizer(max_features=my_max_features, min_df=my_min_df, max_df=my_max_df, st
X = vectorizer.fit_transform(documents).toarray()

tfidfconverter = TfidfTransformer()
X = tfidfconverter.fit_transform(X).toarray()
```

executed in 2.12s, finished 03:36:09 2020-06-17

In [27]:

```
# 학습용 데이터와 테스트용 데이터 분리

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, shuffle=False)

y_train = y_train.reshape(-1, 1)
y_test = y_test.reshape(-1, 1)
```

executed in 54ms, finished 03:36:09 2020-06-17

In [28]:

```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

executed in 74ms, finished 03:36:09 2020-06-17

```
(1401, 3207) (601, 3207) (1401, 1) (601, 1)
```

In [37]:

```
model_neural_network = penalized_neural_network(error_bound=10**(-7),
                                                random_state=20152410,
                                                fit_intercept=True,
                                                iteration=6000,
                                                cutoff=0.5,
                                                learning_rate=20,
                                                hidden_layer=1,
                                                number_node=[350],
                                                alpha=5)
```

executed in 7ms, finished 03:58:05 2020-06-17

In [38]:

```
model_neural_network.fit(X_train=X_train,
                         Y_train=y_train,
                         X_test=X_test,
                         Y_test=y_test)
```

executed in 33m 14s, finished 04:31:19 2020-06-17

```
Train Cost : 1.6584153219540956, Test Cost : 2.5251105643524565

Iter : 3000, Running time : 16m 34.7s, Train accuracy : 100.0%, Test accuracy : 8
3.5275%
Train Cost : 1.5177646442043347, Test Cost : 2.3752942919583595

Iter : 3500, Running time : 19m 26.5s, Train accuracy : 100.0%, Test accuracy : 8
3.3611%
Train Cost : 1.389071726234615, Test Cost : 2.2338349860841307

Iter : 4000, Running time : 22m 9.2s, Train accuracy : 100.0%, Test accuracy : 83.
5275%
Train Cost : 1.2713181218338427, Test Cost : 2.1013500727717327

Iter : 4500, Running time : 24m 55.4s, Train accuracy : 100.0%, Test accuracy : 8
3.3611%
Train Cost : 1.1635740973257416, Test Cost : 1.9778694220760757

Iter : 5000, Running time : 27m 38.7s, Train accuracy : 100.0%, Test accuracy : 8
3.8602%
```

Result record

CountVectorizer에 관련된 하이퍼 파라미터를 찾지 않고 예시 코드대로 사용했을 때 17번정도 시도했었는데 그 중 가장 좋은 결과를 제외하곤 결과 기록을 삭제

4. max_features=1500, min_df=0, max_df=1.0, iteration=2000, cutoff = 0.5, learning_rate=20, hidden_layer=1, number_node=[400], alpha=0

Iter : 2000, Running time : 8m 13.3s, Train accuracy : 100.0%, Test accuracy : 82.5291% Train Cost : 0.00043022399146022446, Test Cost : 1.157898729877365

max_features=5000, min_df=30, max_df=0.85로 고정 후 파라미터 탐색

1. iteration=1500, cutoff=0.5, learning_rate=10, hidden_layer=1, number_node=[500], alpha=0

Iter : 1500, Running time : 11m 41.8s, Train accuracy : 100.0%, Test accuracy : 79.7005% Train Cost : 0.0015612657136663637, Test Cost : 0.778852010503039

2. iteration=1500, cutoff=0.5, learning_rate=10, hidden_layer=1, number_node=[1000], alpha=0

Iter : 1000, Running time : 14m 43.3s, Train accuracy : 100.0%, Test accuracy : 77.8702% Train Cost : 0.004019393077712377, Test Cost : 0.6541943109506908

3. iteration=1500, cutoff=0.5, learning_rate=15, hidden_layer=1, number_node=[400], alpha=0

Iter : 1500, Running time : 10m 7.7s, Train accuracy : 100.0%, Test accuracy : 82.1963% Train Cost : 0.0009284473121591843, Test Cost : 0.8970229332735968

4. iteration=1500, cutoff=0.5, learning_rate=15, hidden_layer=1, number_node=[300], alpha=0

Iter : 1500, Running time : 8m 53.1s, Train accuracy : 100.0%, Test accuracy : 78.5358% Train Cost : 0.0008996072192878819, Test Cost : 1.0232509497588846

5. iteration=1500, cutoff=0.5, learning_rate=15, hidden_layer=1, number_node=[200], alpha=0

Iter : 1500, Running time : 7m 24.0s, Train accuracy : 100.0%, Test accuracy : 79.7005% Train Cost : 0.0010663689138798698, Test Cost : 0.9480445937527249

6. iteration=1500, cutoff=0.5, learning_rate=15, hidden_layer=1, number_node=[350], alpha=0

Iter : 1500, Running time : 10m 51.8s, Train accuracy : 100.0%, Test accuracy : 82.5291% Train Cost : 0.0009011038651082682, Test Cost : 0.7660667053950381

7. iteration=1500, cutoff=0.5, learning_rate=15, hidden_layer=1, number_node=[450], alpha=0

Iter : 1500, Running time : 13m 36.5s, Train accuracy : 100.0%, Test accuracy : 80.1997% Train Cost : 0.0009071227556834926, Test Cost : 0.9334214974224351

hidden_layer=1, number_node=[350]로 고정 후 alpha 탐색

8. iteration=2000, cutoff=0.5, learning_rate=20, hidden_layer=1, number_node=[350], alpha=30

Iter : 2000, Running time : 13m 10.6s, Train accuracy : 100.0%, Test accuracy : 84.3594% Train Cost : 1.909552907198544, Test Cost : 2.499531966477294

9. iteration=5000, cutoff=0.5, learning_rate=20, hidden_layer=1, number_node=[350], alpha=50

Iter : 4000, Running time : 22m 3.5s, Train accuracy : 52.6767%, Test accuracy : 51.5807% Train Cost : 2.9942257824866436, Test Cost : 3.630209000055648

Iter : 5000, Running time : 27m 35.2s, Train accuracy : 100.0%, Test accuracy : 85.1913% Train Cost : 0.19361394916440608, Test Cost : 0.7581774090129727

발산 (진동)

10. iteration=3000, cutoff=0.5, learning_rate=20, hidden_layer=1, number_node=[350], alpha=20

Iter : 3000, Running time : 16m 34.0s, Train accuracy : 100.0%, Test accuracy : 84.6922% Train Cost : 1.2449553011669234, Test Cost : 1.8010886497372876

11. iteration=3000, cutoff=0.5, learning_rate=20, hidden_layer=1, number_node=[350], alpha=10

Iter : 3000, Running time : 16m 49.8s, Train accuracy : 100.0%, Test accuracy : 84.3594% Train Cost : 1.7861736855238604, Test Cost : 2.5195472112989084

12. iteration=3000, cutoff=0.5, learning_rate=20, hidden_layer=1, number_node=[350], alpha=15

Iter : 3000, Running time : 16m 37.2s, Train accuracy : 100.0%, Test accuracy : 84.6922% Train Cost : 1.5785523504841517, Test Cost : 2.2087082380803804

13. iteration=3000, cutoff=0.5, learning_rate=20, hidden_layer=1, number_node=[350], alpha=5

Iter : 3000, Running time : 17m 3.7s, Train accuracy : 100.0%, Test accuracy : 83.5275% Train Cost : 1.5177646442043347, Test Cost : 2.3752942919583595

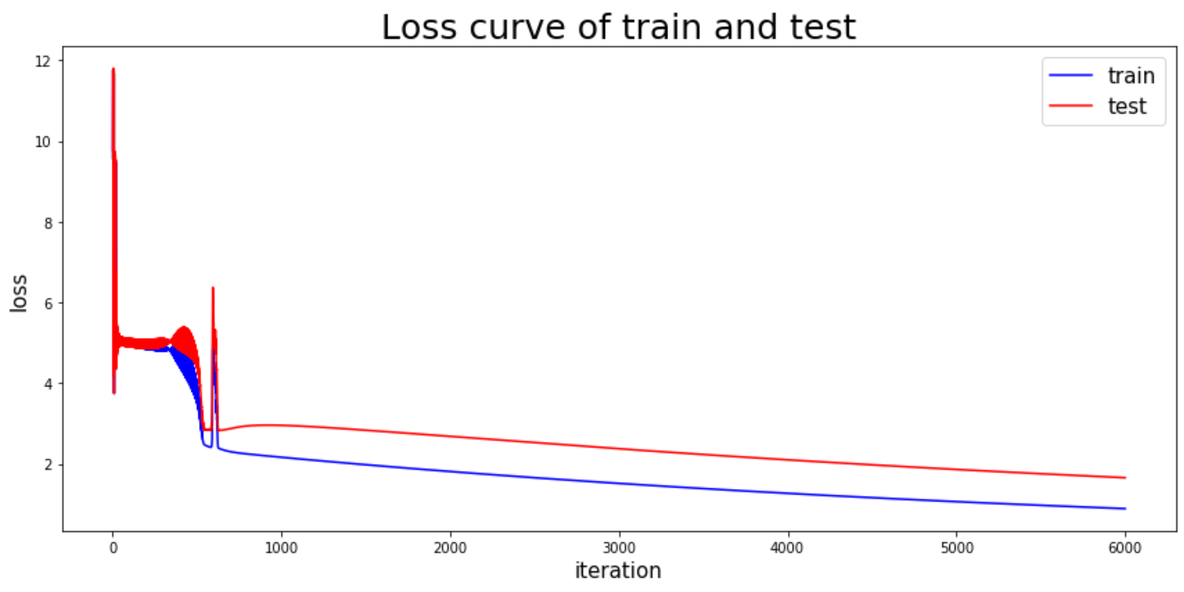
14. (최종) iteration=6000, cutoff=0.5, learning_rate=20, hidden_layer=1, number_node=[350], alpha=20

1. Plot the loss curve

In [39]:

```
traincost = model_neural_network.record_train_cost
testcost = model_neural_network.record_test_cost

plt.figure(figsize=(12, 6))
plt.plot(traincost, 'b', label='train')
plt.plot(testcost, 'r', label='test')
plt.title('Loss curve of train and test', fontsize=25)
plt.xlabel('iteration', fontsize=15)
plt.ylabel('loss', fontsize=15)
plt.legend(loc='best', fontsize=15)
plt.tight_layout()
plt.show()
```

executed in 296ms, finished 04:31:21 2020-06-17

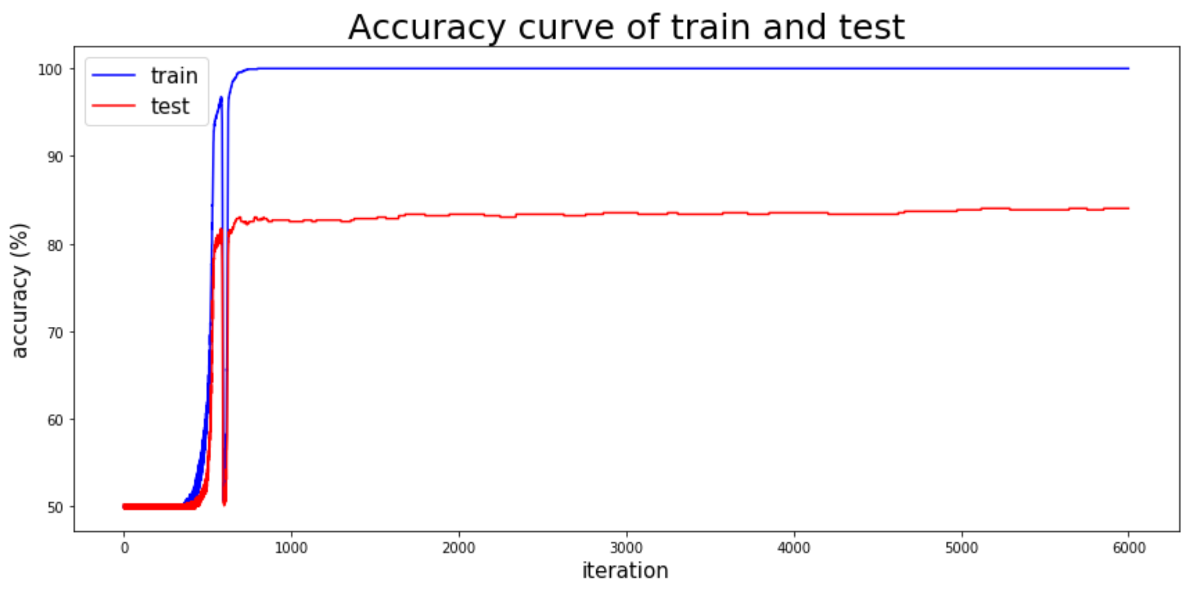


2. Plot the accuracy curve

In [40]:

```
trainacc100 = 100*np.array(model_neural_network.record_train_accuracy)
testacc100 = 100*np.array(model_neural_network.record_test_accuracy)

plt.figure(figsize=(12, 6))
plt.plot(trainacc100, 'b', label='train')
plt.plot(testacc100, 'r', label='test')
plt.title('Accuracy curve of train and test', fontsize=25)
plt.xlabel('iteration', fontsize=15)
plt.ylabel('accuracy (%)', fontsize=15)
plt.legend(loc='best', fontsize=15)
plt.tight_layout()
plt.show()
```

executed in 308ms, finished 04:31:22 2020-06-17



3. Plot the quantitative results

In [41]:

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

traina = trainacc100[-1]
trainb = traincost[-1]

print(confusion_matrix(y_train, model_neural_network.train_predict), '₩n')
print(classification_report(y_train, model_neural_network.train_predict), '₩n')
print('Final train accuracy : {}%, Final train loss : {}'.format(traina, trainb))
```

executed in 18ms, finished 04:31:23 2020-06-17

```
[[699   0]
 [  0 702]]

              precision    recall  f1-score   support

           0       1.00      1.00      1.00       699
           1       1.00      1.00      1.00       702

    accuracy                           1.00      1401
   macro avg       1.00      1.00      1.00      1401
weighted avg       1.00      1.00      1.00      1401


Final train accuracy : 100.0%, Final train loss : 0.8920975656504387
```

In [42]:

```
testa = testacc100[-1]
testb = testcost[-1]

print(confusion_matrix(y_test, model_neural_network.test_predict), '₩n')
print(classification_report(y_test, model_neural_network.test_predict), '₩n')
print('Final test accuracy : {}%, Final test loss : {}'.format(testa, testb))
```

executed in 17ms, finished 04:31:24 2020-06-17

```
[[245  57]
 [ 39 260]]

              precision    recall  f1-score   support

           0       0.86      0.81      0.84       302
           1       0.82      0.87      0.84       299

    accuracy                           0.84       601
   macro avg       0.84      0.84      0.84       601
weighted avg       0.84      0.84      0.84       601


Final test accuracy : 84.02662229617304%, Final test loss : 1.6581495008188467
```