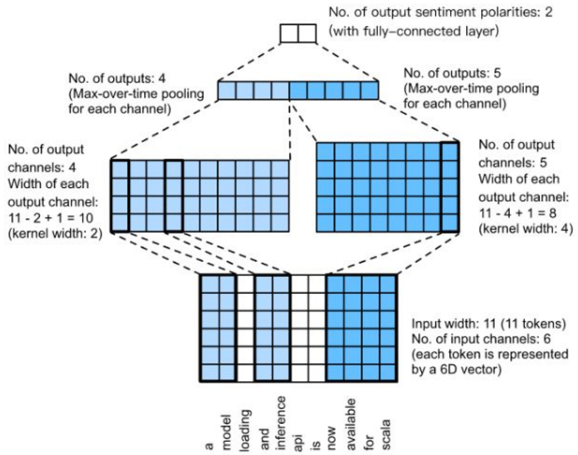


Original Model for Sentence Sentiment Classification: **Text-CNN**

Motivation: ML classification is more convenient for large dataset than human eyes.

What I Did

1. Changed the activation function from 'relu' to 'sigmoid'

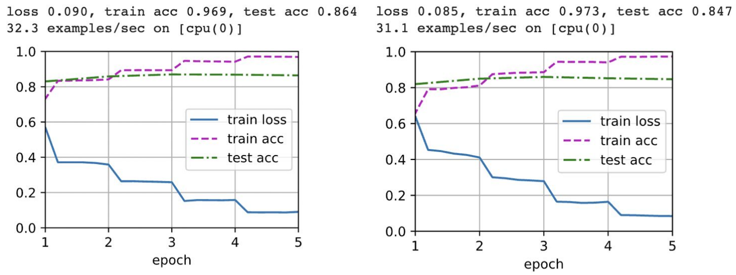


Fig 2. Text-cnn model performance results with 'relu', 'sigmoid' function (from left to right)

2. Transform the dataset into dataframe
3. Used tfidf to vectorize the dataset
4. Logistic Regression Model

	sentence	label
0	Anna Christie (1930) Anna Christie ...	1
1	In Spain, the former sailor Ramón Sampedro (Ja...	1
2	The Closer She Gets... is an artful documentar...	1
3	I consider myself lucky that I got to view a w...	1
4	Evil warlord puts a town through pain and suff...	1

```
train acc = 0.927
test acc = 0.87976
```

5. Used same vectorized dataset
6. Unsupervised k-means clustering

Since previous models work great, wonder if clustering can still achieve similar results.

```
train acc = 0.66204
test acc = 0.66248
```

Reference:

Yoon Kim. Convolutional Neural Networks for Sentence Classification. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP '14).

Sentence Sentiment Classification Report

Danrong Li - dl4111

1. Motivation

Although it is not a very complicated task to understand if a text is positive sentiment or negative for human-beings, machine learning can make it more convenient. In the case where we need to process a large amount of dataset regarding the opinions for a certain movie, the machine learning algorithm can learn and label these opinions in seconds, whereas human must go through the data piece by piece one-at-a-time. In addition, the human errors are inevitable. Thus, it is crucial to have a machine learning algorithm for sentence sentiment classification.

2. Original Paper

The author in the original paper utilized Convolutional Neural Network model. Figure 1 shown below illustrates the model. Suppose we have a sentence with 11 words "a model loading and inference api is now available for scala", and we represent each word as a 6-dimensional vector. Then we perform 2 different convolution operations with different kernel width to the same input. Then we max-over-time pool them and concatenate the results together. Then we add the fully connected layer with dropout to obtain the final label, whether it is 1 for positive or 0 for negative.

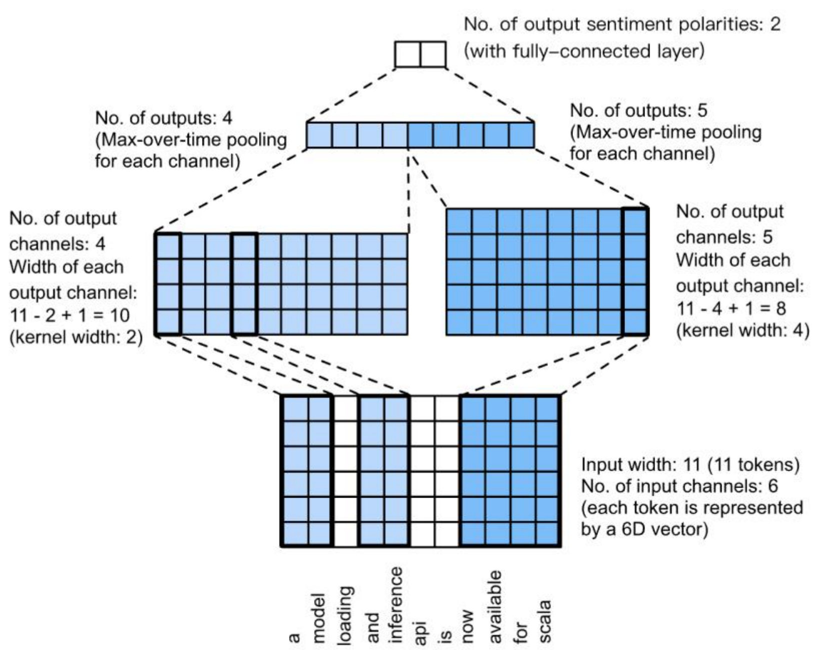


Fig 1. Text-CNN Model

3. What I Did

Firstly, I changed the activation function of convolution network from ‘relu’ to ‘sigmoid’. Since from class, prof Liu mentioned ‘relu’ is largely used in deeper networks to solve vanishing gradients problem, and the text-cnn model is not very deep, I changed the function to ‘sigmoid’. In addition, ‘sigmoid’ function contains the calculation for every z, whereas ‘relu’ function just report the maximum value of z and 0, so I made the change from ‘relu’ to ‘sigmoid’. Figure 2 shown below indicates the performance results of the original ‘relu’ function and the edited ‘sigmoid’ function. According to the figure, the change in the activation function did not make a big impact to the performance. However, the training accuracy for ‘sigmoid’ is higher and the test accuracy gets lower, it might be an indication that ‘sigmoid’ function results in a slight overfit.

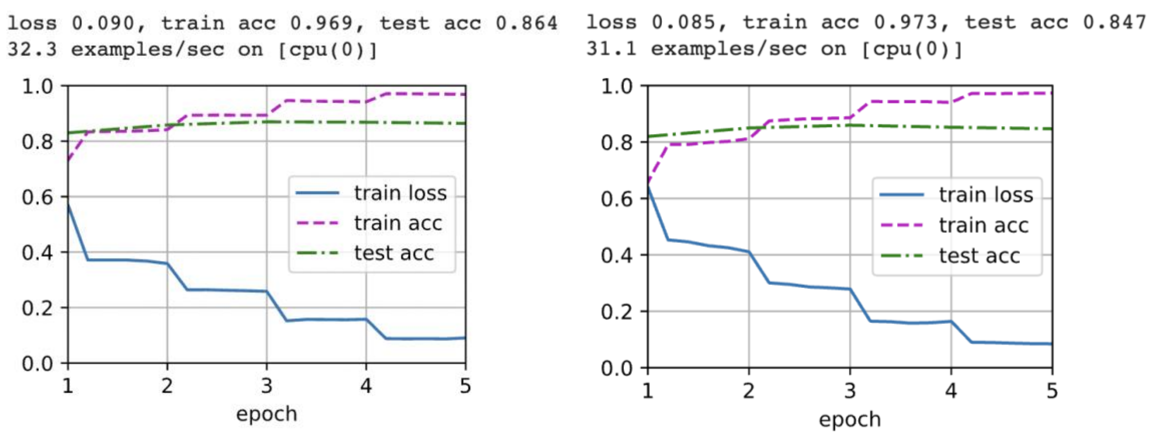


Fig 2. Text-cnn model performance results with ‘relu’, ‘sigmoid’ function (from left to right)

Second, I tfidf-vectorized the same dataset, imdb dataset, and then used logistic regression model. Since every entry in the dataframe is not very long, I initially set max_feature = 100, however, the prediction accuracy is around 0.74. Then I set max_feature = 10,000, and the training accuracy = 0.927 and test accuracy = 0.879.

Third, I used the same tfidf-vectorized dataset, and then used kmeans model. Since the previous models both worked very well, I wonder if the unsupervised cluster can also achieve the good results. I set n_clusters = 2 and fit the training model, then use the model to predict the test data. The test accuracy is derived from prediction results, and the training accuracy is derived from the model labels after fitting, training accuracy = 0.66, test accuracy = 0.66. From these results, we can see although clustering is better than guessing randomly, it is not as good as logistic regression or the convolution neural network.

4. References

Yoon Kim. Convolutional Neural Networks for Sentence Classification. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP '14).

Google Colab Notebook: https://colab.research.google.com/github/d2l-ai/d2l-en-colab/blob/master/chapter_natural-language-processing-applications/sentiment-analysis-cnn.ipynb

Jupyter Notebook:
https://github.com/pytorch/ignite/blob/master/examples/notebooks/TextCNN.ipynb

5. My Code

See next page

dl4111_project_code

December 20, 2021

1 Required Dependecies

I used google colab to run this file. The following code might cause some warnings, but the code will not be affected.

[ ]:
```
!pip install d2l==0.17.1
!pip install -U mxnet-cu101==1.7.0
```

[ ]:
```
!pip uninstall matplotlib
!pip install --upgrade matplotlib
```

2 Import Libraries

[2]:
```
from mxnet import gluon, init, np, npx, autograd
from mxnet.gluon import nn
from d2l import mxnet as d2l
import os
import pandas as pd

npx.set_np()
```

3 Process Data

Download imdb dataset from d2l.

[65]:
```
batch_size = 64
train_iter, test_iter, vocab = d2l.load_data_imdb(batch_size)
```

Write the function of convolution for 1 dimensional input with multiple input channels.

[ ]:
```

```

[66]:
```
def corr1d(X, K):
    w = K.shape[0]
    Y = np.zeros((X.shape[0] - w + 1))
    for i in range(Y.shape[0]):
        Y[i] = (X[i: i + w] * K).sum()
```

```
    return Y

def corr1d_multi_in(X, K):
    # First, iterate through the 0th dimension (channel dimension) of `X` and
    # `K`. Then, add them together
    return sum(corr1d(x, k) for x, k in zip(X, K))

X = np.array([[0, 1, 2, 3, 4, 5, 6],
              [1, 2, 3, 4, 5, 6, 7],
              [2, 3, 4, 5, 6, 7, 8]])
K = np.array([[1, 2], [3, 4], [-1, -3]])
corr1d_multi_in(X, K)
```

[66]:
```
array([ 2.,  8., 14., 20., 26., 32.])
```

4 TextCNN Model

[ ]:
```
```

[67]:
```
class TextCNN(nn.Block):
    def __init__(self, vocab_size, embed_size, kernel_sizes, num_channels,
                 **kwargs):
        super(TextCNN, self).__init__(**kwargs)
        self.embedding = nn.Embedding(vocab_size, embed_size)
        # The embedding layer not to be trained
        self.constant_embedding = nn.Embedding(vocab_size, embed_size)
        self.dropout = nn.Dropout(0.5)
        self.decoder = nn.Dense(2)
        # The max-over-time pooling layer has no parameters, so this instance
        # can be shared
        self.pool = nn.GlobalMaxPool1D()
        # Create multiple one-dimensional convolutional layers
        self.convs = nn.Sequential()
        for c, k in zip(num_channels, kernel_sizes):
            self.convs.add(nn.Conv1D(c, k, activation='relu'))

    def forward(self, inputs):
        # Concatenate two embedding layer outputs with shape (batch size, no.
        # of tokens, token vector dimension) along vectors
        embeddings = np.concatenate((
            self.embedding(inputs), self.constant_embedding(inputs)), axis=2)
        # Per the input format of one-dimensional convolutional layers,
        # rearrange the tensor so that the second dimension stores channels
        embeddings = embeddings.transpose(0, 2, 1)
        # For each one-dimensional convolutional layer, after max-over-time
        # pooling, a tensor of shape (batch size, no. of channels, 1) is
        # obtained. Remove the last dimension and concatenate along channels
```

```
        encoding = np.concatenate([
            np.squeeze(self.pool(conv(embeddings)), axis=-1)
            for conv in self.convs], axis=1)
        outputs = self.decoder(self.dropout(encoding))
        return outputs
```

Let us create a textCNN instance. It has 3 convolutional layers with kernel widths of 3, 4, and 5, all with 100 output channels.

[68]:
```
embed_size, kernel_sizes, nums_channels = 100, [3, 4, 5], [100, 100, 100]
devices = d2l.try_all_gpus()
net = TextCNN(len(vocab), embed_size, kernel_sizes, nums_channels)
net.initialize(init.Xavier(), ctx=devices)
```

5 Load pre-trained GloVe word vectors

Load pretrained 100-dimensional GloVe embeddings as the initialized token representations. These token representations (embedding weights) will be trained in `embedding` and fixed in `constant_embedding`.

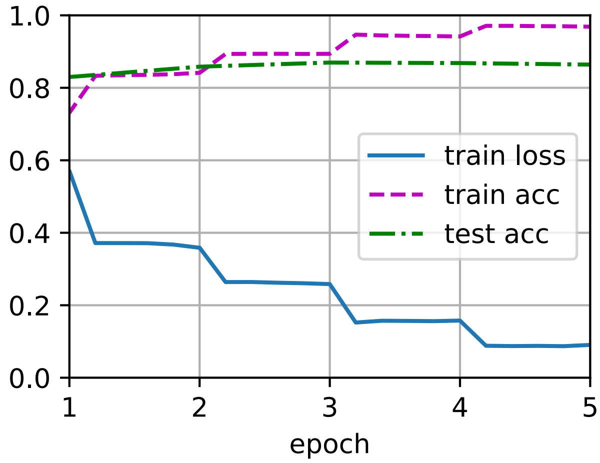
[69]:
```
glove_embedding = d2l.TokenEmbedding('glove.6b.100d')
embeds = glove_embedding[vocab.idx_to_token]
net.embedding.weight.set_data(embeds)
net.constant_embedding.weight.set_data(embeds)
net.constant_embedding.collect_params().setattr('grad_req', 'null')
```

6 Train & Evaluate Model

The following code to generate the graph would take >1h to run.

[70]:
```
lr, num_epochs = 0.001, 5
trainer = gluon.Trainer(net.collect_params(), 'adam', {'learning_rate': lr})
loss = gluon.loss.SoftmaxCrossEntropyLoss()
d2l.train_ch13(net, train_iter, test_iter, loss, trainer, num_epochs, devices)
```

```
loss 0.090, train acc 0.969, test acc 0.864
32.3 examples/sec on [cpu(0)]
```



[44]:
```
d2l.predict_sentiment(net, vocab, 'I cannot believe how amazing she is!')
```

[44]: 'positive'

[45]:
```
d2l.predict_sentiment(net, vocab, 'She is just a very terrible person')
```

[45]: 'negative'

[ ]:

7 Something Different - use sigmoid activation

[52]:
```
class TextCNN2(nn.Block):
    def __init__(self, vocab_size, embed_size, kernel_sizes, num_channels,
                 **kwargs):
        super(TextCNN2, self).__init__(**kwargs)
        self.embedding = nn.Embedding(vocab_size, embed_size)
        # The embedding layer not to be trained
        self.constant_embedding = nn.Embedding(vocab_size, embed_size)
        self.dropout = nn.Dropout(0.5)
        self.decoder = nn.Dense(2)
        # The max-over-time pooling layer has no parameters, so this instance
        # can be shared
        self.pool = nn.GlobalMaxPool1D()
        # Create multiple one-dimensional convolutional layers
        self.convs = nn.Sequential()
        for c, k in zip(num_channels, kernel_sizes):
            self.convs.add(nn.Conv1D(c, k, activation='sigmoid'))

    def forward(self, inputs):
```

```
        # Concatenate two embedding layer outputs with shape (batch size, no.
        # of tokens, token vector dimension) along vectors
        embeddings = np.concatenate((
            self.embedding(inputs), self.constant_embedding(inputs)), axis=2)
        # Per the input format of one-dimensional convolutional layers,
        # rearrange the tensor so that the second dimension stores channels
        embeddings = embeddings.transpose(0, 2, 1)
        # For each one-dimensional convolutional layer, after max-over-time
        # pooling, a tensor of shape (batch size, no. of channels, 1) is
        # obtained. Remove the last dimension and concatenate along channels
        encoding = np.concatenate([
            np.squeeze(self.pool(conv(embeddings)), axis=-1)
            for conv in self.convs], axis=1)
        outputs = self.decoder(self.dropout(encoding))
        return outputs
```

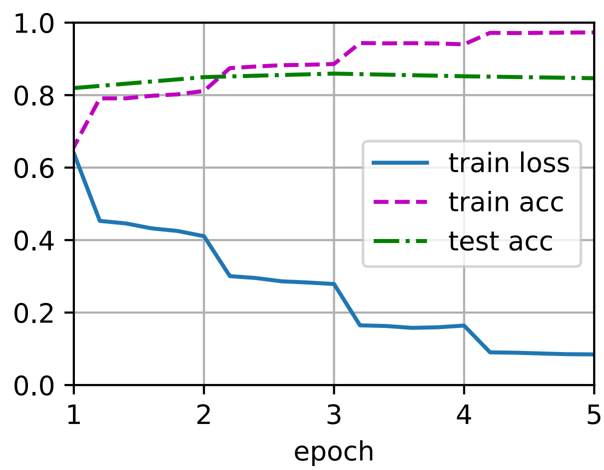
The following code to generate the graph takes >1h to run.

[53]:
```
embed_size, kernel_sizes, nums_channels = 100, [3, 4, 5], [100, 100, 100]
devices = d2l.try_all_gpus()
net2 = TextCNN2(len(vocab), embed_size, kernel_sizes, nums_channels)
net2.initialize(init.Xavier(), ctx=devices)

net2.embedding.weight.set_data(embeds)
net2.constant_embedding.weight.set_data(embeds)
net2.constant_embedding.collect_params().setattr('grad_req', 'null')

lr, num_epochs = 0.001, 5
trainer = gluon.Trainer(net2.collect_params(), 'adam', {'learning_rate': lr})
loss = gluon.loss.SoftmaxCrossEntropyLoss()
d2l.train_ch13(net2, train_iter, test_iter, loss, trainer, num_epochs, devices)
```

```
loss 0.085, train acc 0.973, test acc 0.847
31.1 examples/sec on [cpu(0)]
```



8 Something Different - use tfidf + logisitc regression model

Still use imdb dataset, but changed the data format to dataframe here.

[48]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import TfidfTransformer

import matplotlib
import matplotlib.pyplot as plt
from sklearn import linear_model
```

[3]:
```
d2l.DATA_HUB['aclImdb'] = (
    'http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz',
    '01ada507287d82875905620988597833ad4e0903')

data_dir = d2l.download_extract('aclImdb', 'aclImdb')
```

[14]:
```
def read_imdb(data_dir, is_train):
    # Read the IMDb review dataset text sequences and labels.
    data, labels = [], []
    for label in ('pos', 'neg'):
        folder_name = os.path.join(data_dir, 'train' if is_train else 'test',
                                   label)
        for file in os.listdir(folder_name):
            with open(os.path.join(folder_name, file), 'rb') as f:
                review = f.read().decode('utf-8').replace('\n', '')
                data.append(review)
                labels.append(1 if label == 'pos' else 0)
    return data, labels

train_data = read_imdb(data_dir, is_train=True)
test_data = read_imdb(data_dir, is_train=False)
print('# trainings:', len(train_data[0]))
for x, y in zip(train_data[0][:3], train_data[1][:3]):
    print('label:', y, 'review:', x[0:60])
```

```
# trainings: 25000
label: 1 review: Strangely, this version of OPEN YOUR EYES is more mature and
label: 1 review: Steve Biko was a black activist who tried to resist the whit
label: 1 review: I loved October Sky. The thing I loved most had to be the mu
```

[13]:
```
from google.colab import data_table
data_table.enable_dataframe_formatter()

train = pd.DataFrame(train_data).T
```

```
train.rename({0: 'sentence', 1: 'label'}, axis=1, inplace=True)
train.head()
```

[13]:

```
                                            sentence label
0  Strangely, this version of OPEN YOUR EYES is m…     1
1  Steve Biko was a black activist who tried to r…     1
2  I loved October Sky. The thing I loved most ha…     1
3  Tintin and I recently aired as an episode of P…     1
4  Once in the Life means that once a hoodlum, al…     1
```

[17]:

```
test = pd.DataFrame(test_data).T
test.rename({0: 'sentence', 1: 'label'}, axis=1, inplace=True)
test.head()
```

[17]:

```
                                            sentence label
0  Anna Christie (1930)<br /><br />Anna Christie …     1
1  In Spain, the former sailor Ramón Sampedro (Ja…     1
2  The Closer She Gets… is an artful documentar…     1
3  I consider myself lucky that I got to view a w…     1
4  Evil warlord puts a town through pain and suff…     1
```

[32]:

```
np.shape(train)
```

[32]: (25000, 2)

Set max_feature = 100, since every input text is not very long.

[113]:

```
vectorizer = TfidfVectorizer(
                max_df=0.5, # max doc freq (as a fraction) of any word to␣
 ↪include in the vocabulary
                min_df=2,   # min doc freq (as doc counts) of any word to␣
 ↪include in the vocabulary
                max_features=100,          # max number of words in the␣
 ↪vocabulary
                stop_words='english',         # remove English stopwords
                use_idf=True )        # use IDF scores

X_train = vectorizer.fit_transform(train['sentence'].values)
X_test = vectorizer.transform(test['sentence'].values)
y_train = train['label'].values.astype('int64')
y_test = test['label'].values.astype('int64')
logreg = linear_model.LogisticRegression(tol=1e-8,max_iter=50)
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
y_pred_train = logreg.predict(X_train)
test_acc = sum(1*(y_test==y_pred))/len(y_test)
train_acc = sum(1*(y_train==y_pred_train))/len(y_train)
print('train acc = '+str(train_acc))
```

```
print('test acc = '+str(test_acc))
```

```
train acc = 0.74488
test acc = 0.74172
```

Since the resultant acc is not promising, changed max_feature = 10000.

[106]:
```
vectorizer = TfidfVectorizer(
                max_df=0.5, # max doc freq (as a fraction) of any word to␣
 ↪include in the vocabulary
                min_df=2,   # min doc freq (as doc counts) of any word to␣
 ↪include in the vocabulary
                max_features=10000,         # max number of words in the␣
 ↪vocabulary
                stop_words='english',       # remove English stopwords
                use_idf=True )        # use IDF scores
```

[107]:
```
X_train = vectorizer.fit_transform(train['sentence'].values)
X_test = vectorizer.transform(test['sentence'].values)
```

[108]:
```
vectorizer.get_feature_names_out()
```

[108]:
```
array(['00', '000', '10', …, 'zoom', 'zorro', 'zu'], dtype=object)
```

[109]:
```
print("n_samples: %d, n_features: %d" % X_train.shape)
```

```
n_samples: 25000, n_features: 10000
```

[110]:
```
y_train = train['label'].values.astype('int64')
y_test = test['label'].values.astype('int64')
```

[111]:
```
logreg = linear_model.LogisticRegression(tol=1e-8,max_iter=50)
logreg.fit(X_train, y_train)
```

[111]:
```
LogisticRegression(max_iter=50, tol=1e-08)
```

[112]:
```
y_pred = logreg.predict(X_test)
y_pred_train = logreg.predict(X_train)
test_acc = sum(1*(y_test==y_pred))/len(y_test)
train_acc = sum(1*(y_train==y_pred_train))/len(y_train)
print('train acc = '+str(train_acc))
print('test acc = '+str(test_acc))
```

```
train acc = 0.927
test acc = 0.87976
```

Since the result is not showing overfitting, the parameters are okay.

9 Something Different - used tfidf + cluster

[77]:
```
from sklearn.cluster import KMeans
```

[79]:
```
kmeans = KMeans(n_clusters=2, random_state=0).fit(X_train)
k_labels = kmeans.labels_
k_labels
```

[79]:
```
array([0, 0, 0, …, 0, 1, 0], dtype=int32)
```

[80]:
```
sum(1*(y_train==k_labels))/len(y_train)
```

[80]:
```
0.33796
```

[83]:
```
changed_k_labels = []
for i in range(len(k_labels)):
  if k_labels[i] == 0:
    changed_k_labels.append(1)
  else:
    changed_k_labels.append(0)
```

[85]:
```
sum(1*(y_train==changed_k_labels))/len(y_train)
```

[85]:
```
0.66204
```

[87]:
```
y_pred = kmeans.predict(X_test)
```

[88]:
```
sum(1*(y_test==y_pred))/len(y_test)
```

[88]:
```
0.33752
```

[89]:
```
changed_y_pred = []
for i in range(len(y_pred)):
  if y_pred[i] == 0:
    changed_y_pred.append(1)
  else:
    changed_y_pred.append(0)
```

[90]:
```
sum(1*(y_test==changed_y_pred))/len(y_test)
```

[90]:
```
0.66248
```

[91]:
```
train_acc = sum(1*(y_train==changed_k_labels))/len(y_train)
test_acc = sum(1*(y_test==changed_y_pred))/len(y_test)

print('train acc = '+str(train_acc))
print('test acc = '+str(test_acc))
```

```
train acc = 0.66204
test acc = 0.66248
```

```
[ ]:
```