In [1]:
```python
# 1. Classification Using Hand-Crafted Features
# (a)
# Load VizWiz dataset
import os
import json
import requests
from pprint import PrettyPrinter

base_url = 'https://ivc.ischool.utexas.edu/VizWiz/data'
img_dir = '%s/Images/' %base_url
print(img_dir)

train_split = 'train'
train_file = '%s/Annotations/%s.json' %(base_url, train_split)
train_data = requests.get(train_file, allow_redirects=True)
print(train_file)

test_split = 'test'
test_file = '%s/Annotations/%s.json' %(base_url, test_split)
test_data = requests.get(test_file, allow_redirects=True)
print(test_file)

val_split = 'val'
val_file = '%s/Annotations/%s.json' %(base_url, val_split)
val_data = requests.get(val_file, allow_redirects=True)
print(val_file)
```

```
https://ivc.ischool.utexas.edu/VizWiz/data/Images/
https://ivc.ischool.utexas.edu/VizWiz/data/Annotations/train.json
https://ivc.ischool.utexas.edu/VizWiz/data/Annotations/test.json
https://ivc.ischool.utexas.edu/VizWiz/data/Annotations/val.json
```

In [2]:
```python
# Read the local file
training_data = train_data.json()
testing_data = test_data.json()
validation_data = val_data.json()
print("Length of training data:", len(training_data))
print("Length of test data:", len(testing_data))
print("Length of validation data:", len(validation_data))

image_name_train = []
question_train = []
label_train = []

image_name_val = []
question_val = []
label_val = []

image_name_test = []
question_test = []
label_test = []

num_train_VQs = 20000
for vq in training_data[0:num_train_VQs]:
    image_name_train.append(vq['image'])
    question_train.append(vq['question'])
    label_train.append(vq['answerable'])

num_val_VQs = 8000
for vq in validation_data[0:num_val_VQs]:
    image_name_val.append(vq['image'])
    question_val.append(vq['question'])
    label_val.append(vq['answerable'])

num_test_VQs = 3173
for vq in testing_data[0:num_test_VQs]:
    image_name_test.append(vq['image'])
    question_test.append(vq['question'])
#     label_test.append(vq['answerable'])

import pandas as pd
image_name_train = pd.DataFrame(image_name_train, columns=['image'])
image_name_val = pd.DataFrame(image_name_val, columns=['image'])
image_name_test = pd.DataFrame(image_name_test, columns=['image'])
question_train = pd.DataFrame(question_train, columns=['question'])
question_val = pd.DataFrame(question_val, columns=['question'])
question_test = pd.DataFrame(question_test, columns=['question'])

X_train = pd.concat([image_name_train, question_train], axis=1)
y_train = pd.DataFrame(label_train, columns=['label'])
X_val = pd.concat([image_name_val, question_val], axis=1)
y_val = pd.DataFrame(label_val, columns=['label'])
X_test = pd.concat([image_name_test, question_test], axis=1)
# y_test = pd.DataFrame(label_test, columns='label')
```

```
Length of training data: 20000
Length of test data: 8000
Length of validation data: 3173
```

In [12]:

```
# (b)
# Use Microsoft Azure API to extract image-based features
subscription_key_vision = '412bc41b5b5844febf4d7cd63510fb4f'
vision_base_url = 'https://westcentralus.api.cognitive.microsoft.com/vis
ion/v1.0'
vision_analyze_url = vision_base_url + '/analyze?'
from time import sleep

def analyze_image(image_url):
    # Microsoft API headers, params, etc
    headers = {'Ocp-Apim-Subscription-key': subscription_key_vision}
    params = {'visualfeatures': 'Description, Tags'}
    data = {'url': image_url}
    # send request, get API response
    try:
        response = requests.post(vision_analyze_url,headers = headers,pa
rams=params,json=data)
    except:
        sleep(10)
        response = requests.post(vision_analyze_url,headers = headers,pa
rams=params,json=data)
#     response = requests.post(vision_analyze_url, headers=headers, para
ms=params, json=data)
    if (response.status_code == 200):
        analysis = response.json()
    else:
        print("get image {} failed".format(image_url))
        analysis = {"description":{"tags":[]}}
    return analysis

def extract_features(data):
    return {
        'tags': data['description']['tags'],
#         'confidence': data['tags'][0]['confidence']
    }

image_feature = {}
def get_image_feature(X):

    for i in range(20000):
        image_url = img_dir + '%s' %(X['image'][i])
        data = extract_features(analyze_image(image_url))
        tag_i = []
        for item in data['tags']:
            tag_i.append(item)
        tag_i_join = ' '.join(tag_i)
#         image_feature.append(tag_i_join)
        image_feature[str(i)] = tag_i_join
        if (i%500==0):
            print('get number',str(i))

    return image_feature
image_feature = get_image_feature(X_train)
```

```
get number 12500
get number 13000
get number 13500
get number 14000
get image https://ivc.ischool.utexas.edu/VizWiz/data/Images/VizWiz_trai
n_000000014307.jpg failed
get number 14500
get number 15000
get number 15500
get image https://ivc.ischool.utexas.edu/VizWiz/data/Images/VizWiz_trai
n_000000015541.jpg failed
get number 16000
get number 16500
get number 17000
get image https://ivc.ischool.utexas.edu/VizWiz/data/Images/VizWiz_trai
n_000000017089.jpg failed
get image https://ivc.ischool.utexas.edu/VizWiz/data/Images/VizWiz_trai
n_000000017311.jpg failed
get number 17500
get image https://ivc.ischool.utexas.edu/VizWiz/data/Images/VizWiz_trai
n_000000017821.jpg failed
get number 18000
get number 18500
get image https://ivc.ischool.utexas.edu/VizWiz/data/Images/VizWiz_trai
n_000000018603.jpg failed
get image https://ivc.ischool.utexas.edu/VizWiz/data/Images/VizWiz_trai
n_000000018777.jpg failed
get image https://ivc.ischool.utexas.edu/VizWiz/data/Images/VizWiz_trai
n_000000018938.jpg failed
get number 19000
get number 19500
get image https://ivc.ischool.utexas.edu/VizWiz/data/Images/VizWiz_trai
n_000000019757.jpg failed
```

In [13]:

```python
# Write image feature to csv file

import csv

data = pd.DataFrame()
indexlist = []
featurelist = []
for index,feature in image_feature.items():
    indexlist.append(index)
    featurelist.append(feature)
data["id"] = indexlist
data["image_feature"] = featurelist
data.columns = ["id", "image_feature"]
data.head()
data.to_csv('image_feature_train.csv', index=False)
```

In [ ]:
```
# Extract text features using Microsoft Azure
from time import sleep
subscription_key_text = 'e25225c679e74f61a2ab61924b41a866'
text_analytics_base_url = 'https://centralus.api.cognitive.microsoft.co
m/text/analytics/v2.0/'
key_phrase_api_url = text_analytics_base_url + 'keyPhrases'
question_feature = {}
def get_question_feature(question_train):

    for i in range(20000):

        question_json = question_train['question'][i]
        documents = {'documents': [{'id': i, 'text': question_json}]}
        headers = {"Ocp-Apim-Subscription-Key": subscription_key_text}
        maxiter = 10

        try:
            response = requests.post(key_phrase_api_url,headers = header
s,json=documents)
        except:
            sleep(10)
            response = requests.post(key_phrase_api_url,headers = header
s,json=documents)
        if(response.status_code == 200):
            question_json = response.json()['documents']
            question = pd.DataFrame(question_json)['keyPhrases']
            question = question.tolist()[0]
            tag_i=[]
            for item in question:
                tag_i.append(item)
            question = ' '.join(tag_i)
            question_feature[str(i)] = question
        else:
            print("not get",str(i))
            question_feature[str(i)] = ""
        if (i%500==0):
            print('get number',str(i))

    return question_feature
question_feature = get_question_feature(X_train)
#print(question_feature)
```

In [ ]:
```
# Write key phrase to csv file
data = pd.DataFrame()
indexlist = []
keywordlist = []
for index,keyword in question_feature.items():
    indexlist.append(index)
    keywordlist.append(keyword)
data["id"] = indexlist
data["question_keyword"] = keywordlist
data.columns = ["id", "question_keyword"]
data.head()
data.to_csv('question_feature_train.csv', index=False)
```

In [ ]:

```

```

In [1]:

```
# Load dataset
import pandas as pd

image_feature_train = pd.read_csv('dataset/image_feature_train.csv', hea
der=None)[1].fillna(value='')
question_feature_train = pd.read_csv('dataset/question_feature_train.cs
v', header=None)[1].fillna(value='')
y_train = pd.read_csv('dataset/y_train.csv', header=None)[1].fillna(valu
e='')

image_feature_val = pd.read_csv('dataset/image_feature_val.csv', header=
None)[1].fillna(value='')
question_feature_val = pd.read_csv('dataset/question_feature_val.csv', h
eader=None)[1].fillna(value='')
y_val = pd.read_csv('dataset/y_val.csv', header=None)[1].fillna(value=''
)

image_feature_test = pd.read_csv('dataset/image_feature_test.csv', heade
r=None)[1].fillna(value='')
question_feature_test = pd.read_csv('dataset/question_feature_test.csv',
header=None)[1].fillna(value='')


print(y_train.shape)
print(y_val.shape)
```

```
(20000,)
(2000,)
```

In [2]:
```
# One hot encoding
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer

def one_hot_transform(text):
    count = CountVectorizer()
    bag = count.fit_transform(text).toarray()
    return bag


image_feature = pd.concat([image_feature_train, image_feature_val, image
_feature_test], axis=0)
question_feature = pd.concat([question_feature_train, question_feature_v
al, question_feature_test], axis=0)

image_feature_transformed = pd.DataFrame(one_hot_transform(image_feature
))
question_feature_transformed = pd.DataFrame(one_hot_transform(question_f
eature))

X = pd.concat([image_feature_transformed, question_feature_transformed],
axis=1)

X_train = X[:20000]
X_val = X[20000:22000]
X_test = X[22000:22100]

print(X_train.shape)
print(X_val.shape)
print(X_test.shape)
```

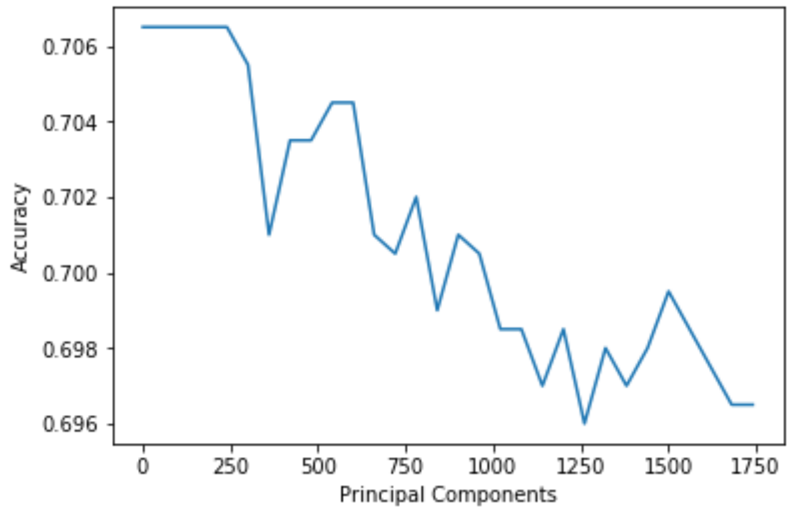
```
(20000, 3572)
(2000, 3572)
(100, 3572)
```

In [3]:
```
# Dimension reduction
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
%matplotlib inline

# X_train = X_train[:4000]
# y_train = y_train[:4000]

principal_com = []
accuracy = []
for i in range(1, 1800, 60):
    pca = PCA(n_components=i)
    pca.fit(X_train)
    X_train_reduced = pca.transform(X_train)
    X_val_reduced = pca.transform(X_val)
    model = LogisticRegression()
    model.fit(X_train_reduced, y_train)
    y_val_predicted = model.predict(X_val_reduced)
    accuracy.append(accuracy_score(y_val, y_val_predicted))
    principal_com.append(i)

plt.plot(principal_com, accuracy)
plt.xlabel("Principal Components")
plt.ylabel("Accuracy")
plt.show()
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
```

```
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
```

```
2. Specify a solver to silence this warning.
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
```



In [5]:
```
pca = PCA(n_components=61)
pca.fit(X_train)
X_train_reduced = pca.transform(X_train)
X_val_reduced = pca.transform(X_val)

model = LogisticRegression()
model.fit(X_train_reduced, y_train)
y_val_predicted = model.predict(X_val_reduced)

print("Accuracy on test set: {:.2f}".format(accuracy_score(y_val_predict
ed, y_val)))
```

```
Accuracy on test set: 0.71

/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
y:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
```

In [ ]:

```python
# Ensemble learning
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
pca = PCA(n_components=0.99)
pca.fit(X_train)
X_train_reduced = pca.transform(X_train)
X_val_reduced = pca.transform(X_val)

#logistic = LogisticRegression()
mlp = MLPClassifier(hidden_layer_sizes=(2048,4096,4096),max_iter=1000,ra
ndom_state=42,activation='relu',solver='adam')
bagging = BaggingClassifier(max_samples=50)
boost = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_dep
th=10), n_estimators=500,learning_rate=0.1)
svc = SVC(random_state=42)

for model in [mlp, bagging, boost, svc]:
    model.fit(X_train_reduced, y_train)
    y_val_predicted = model.predict(X_val_reduced)
    print(accuracy_score(y_val_predicted, y_val))
```

In [ ]:

In [33]:

```python
# Predict test data

model = SVC()
model.fit(X_train_reduced, y_train)
X_test_reduced = pca.transform(X_test)
y_test_predicted = model.predict(X_test_reduced)
print(y_test_predicted)
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:196: FutureW
arning: The default value of gamma will change from 'auto' to 'scale' i
n version 0.22 to account better for unscaled features. Set gamma expli
citly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)

[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

In [34]:

```python
data = pd.DataFrame(y_test_predicted)
data.to_csv('prediction.csv', index=False, header=False)
```

In [ ]:

(e)
I use the image-based features and question-based features as the input data. To extract these features, I use Microsoft Azure API (computer vision and text analytics) to process all the VizWiz training dataset, 2000 rows of validation dataset, and 100 rows of test dataset. I use all the image tags from Azure Vision API, connect them word by word and store it as image feature. Then I use Text Analytics API to extract key phrases of each question and store it as question-based feature.
After getting all necessary data, I create the following dataset for training and testing. There are three image feature datasets of training, validation and testing. Three question feature datasets of training, validation and testing. Two output datasets of training and validation. Then I combine image and question feature datasets together and use one-hot to encode. That’s how I prepare all data following the following model training.
First I use PCA to reduce the dimension of input data. Then I train the neural network, logistic regression, bagging, boost, and SVM models using the training data. During this process, I tuned the model parameters to make sure each model works well and has a relatively accuracy on the validation data. After that, I choose the one with the highest accuracy (some of them has similar accuracy, I just randomly pick one among them) and make prediction of the test dataset using that model.

(f)
I tried neural network, logistic regression, bagging, AdaBoost, SVM models. I trained the model with 20000 rows of training data, then tested the model on 2000 rows of validation data. The accuracy of different models are as follows. MLPClassifier: 0.64; Logistic Regression: 0.71; bagging: 0.64; boost: 0.71; SVM: 0.71. According to the accuracy on validation data, I choose SVM to make prediction on the 100 examples from VizWiz_test000000020100.jpg to VizWiz_test_000000020199.jpg.
Here are some hyperparameters of different models.
MLPClassifier: hidden_layer_sizes=(2048,4096,4096), max_iter=1000, random_state=42, activation='relu', solver='adam'
BaggingClassifier: max_samples=50
AdaBoostClassifier: base_estimator=DecisionTreeClassifier(max_depth=10), n_estimators=500,learning_rate=0.1
Logistic regression, boost, and SVM models have same accuracy on validation dataset, we can apply any of them to the 100 examples of the test split.

In [2]:
```
# Classification using neural networks
# (a)
import pandas as pd
from tensorflow.examples.tutorials.mnist import input_data
from sklearn.model_selection import train_test_split

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
X = mnist.train.images
y = mnist.train.labels.astype("int")
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25
, random_state=42)
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
```

In [3]:
```
# (b)
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier

# Standardization
stdsc = StandardScaler()
X_train_std = stdsc.fit_transform(X_train)
X_test_std = stdsc.fit_transform(X_test)

for num_layer in range(1, 9):

    for num_neuron in range(5, 13):
        mlp = MLPClassifier(hidden_layer_sizes = ((num_neuron,)*num_laye
r), max_iter=50, random_state=42, activation='relu', solver='adam', batc
h_size=200)
        mlp.fit(X_train_std, y_train)
        print("Number of hidden layers", num_layer)
        print("Number of Neurons per layer", num_neuron)
        print("Accuracy on test set: {:.5f}".format(mlp.score(X_test_std
, y_test)))
#         print(mlp.coefs_)
```

```
/home/nbuser/anaconda3_501/lib/python3.6/site-packages/sklearn/neural_n
etwork/multilayer_perceptron.py:564: ConvergenceWarning: Stochastic Opt
imizer: Maximum iterations (50) reached and the optimization hasn't con
verged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Number of hidden layers 1
Number of Neurons per layer 5
Accuracy on test set: 0.74713
Number of hidden layers 1
Number of Neurons per layer 6
Accuracy on test set: 0.77469
Number of hidden layers 1
Number of Neurons per layer 7
Accuracy on test set: 0.81978
Number of hidden layers 1
Number of Neurons per layer 8
Accuracy on test set: 0.84873
Number of hidden layers 1
Number of Neurons per layer 9
Accuracy on test set: 0.86204
Number of hidden layers 1
Number of Neurons per layer 10
Accuracy on test set: 0.87033
Number of hidden layers 1
Number of Neurons per layer 11
Accuracy on test set: 0.88080
Number of hidden layers 1
Number of Neurons per layer 12
Accuracy on test set: 0.88022
Number of hidden layers 2
Number of Neurons per layer 5
Accuracy on test set: 0.79040
Number of hidden layers 2
Number of Neurons per layer 6
Accuracy on test set: 0.82364
Number of hidden layers 2
Number of Neurons per layer 7
Accuracy on test set: 0.86407
Number of hidden layers 2
Number of Neurons per layer 8
Accuracy on test set: 0.87658
Number of hidden layers 2
Number of Neurons per layer 9
Accuracy on test set: 0.88640
Number of hidden layers 2
Number of Neurons per layer 10
Accuracy on test set: 0.89324
Number of hidden layers 2
Number of Neurons per layer 11
Accuracy on test set: 0.90153
Number of hidden layers 2
Number of Neurons per layer 12
Accuracy on test set: 0.90953
Number of hidden layers 3
Number of Neurons per layer 5
Accuracy on test set: 0.79556
Number of hidden layers 3
Number of Neurons per layer 6
Accuracy on test set: 0.84276
Number of hidden layers 3
Number of Neurons per layer 7
Accuracy on test set: 0.87607
```

```
Number of hidden layers 3
Number of Neurons per layer 8
Accuracy on test set: 0.89375
Number of hidden layers 3
Number of Neurons per layer 9
Accuracy on test set: 0.89716
Number of hidden layers 3
Number of Neurons per layer 10
Accuracy on test set: 0.90524
Number of hidden layers 3
Number of Neurons per layer 11
Accuracy on test set: 0.91135
Number of hidden layers 3
Number of Neurons per layer 12
Accuracy on test set: 0.91098
Number of hidden layers 4
Number of Neurons per layer 5
Accuracy on test set: 0.46305
Number of hidden layers 4
Number of Neurons per layer 6
Accuracy on test set: 0.83949
Number of hidden layers 4
Number of Neurons per layer 7
Accuracy on test set: 0.85687
Number of hidden layers 4
Number of Neurons per layer 8
Accuracy on test set: 0.88582
Number of hidden layers 4
Number of Neurons per layer 9
Accuracy on test set: 0.89113
Number of hidden layers 4
Number of Neurons per layer 10
Accuracy on test set: 0.90458
Number of hidden layers 4
Number of Neurons per layer 11
Accuracy on test set: 0.90873
Number of hidden layers 4
Number of Neurons per layer 12
Accuracy on test set: 0.91433
Number of hidden layers 5
Number of Neurons per layer 5
Accuracy on test set: 0.68393
Number of hidden layers 5
Number of Neurons per layer 6
Accuracy on test set: 0.83404
Number of hidden layers 5
Number of Neurons per layer 7
Accuracy on test set: 0.86175
Number of hidden layers 5
Number of Neurons per layer 8
Accuracy on test set: 0.87738
Number of hidden layers 5
Number of Neurons per layer 9
Accuracy on test set: 0.88902
Number of hidden layers 5
Number of Neurons per layer 10
Accuracy on test set: 0.90924
```

```
Number of hidden layers 5
Number of Neurons per layer 11
Accuracy on test set: 0.91324
Number of hidden layers 5
Number of Neurons per layer 12
Accuracy on test set: 0.91745
Number of hidden layers 6
Number of Neurons per layer 5
Accuracy on test set: 0.81025
Number of hidden layers 6
Number of Neurons per layer 6
Accuracy on test set: 0.65251
Number of hidden layers 6
Number of Neurons per layer 7
Accuracy on test set: 0.84175
Number of hidden layers 6
Number of Neurons per layer 8
Accuracy on test set: 0.88015
Number of hidden layers 6
Number of Neurons per layer 9
Accuracy on test set: 0.89527
Number of hidden layers 6
Number of Neurons per layer 10
Accuracy on test set: 0.90611
Number of hidden layers 6
Number of Neurons per layer 11
Accuracy on test set: 0.91011
Number of hidden layers 6
Number of Neurons per layer 12
Accuracy on test set: 0.90691
Number of hidden layers 7
Number of Neurons per layer 5
Accuracy on test set: 0.54880
Number of hidden layers 7
Number of Neurons per layer 6
Accuracy on test set: 0.83615
Number of hidden layers 7
Number of Neurons per layer 7
Accuracy on test set: 0.84800
Number of hidden layers 7
Number of Neurons per layer 8
Accuracy on test set: 0.87382
Number of hidden layers 7
Number of Neurons per layer 9
Accuracy on test set: 0.85789
Number of hidden layers 7
Number of Neurons per layer 10
Accuracy on test set: 0.90022
Number of hidden layers 7
Number of Neurons per layer 11
Accuracy on test set: 0.90495
Number of hidden layers 7
Number of Neurons per layer 12
Accuracy on test set: 0.90727
Number of hidden layers 8
Number of Neurons per layer 5
Accuracy on test set: 0.58625
```

```
Number of hidden layers 8
Number of Neurons per layer 6
Accuracy on test set: 0.76356
Number of hidden layers 8
Number of Neurons per layer 7
Accuracy on test set: 0.86007
Number of hidden layers 8
Number of Neurons per layer 8
Accuracy on test set: 0.87753
Number of hidden layers 8
Number of Neurons per layer 9
Accuracy on test set: 0.89789
Number of hidden layers 8
Number of Neurons per layer 10
Accuracy on test set: 0.89069
Number of hidden layers 8
Number of Neurons per layer 11
Accuracy on test set: 0.89811
Number of hidden layers 8
Number of Neurons per layer 12
Accuracy on test set: 0.90422
```

In [9]:

```
mlp = MLPClassifier(hidden_layer_sizes = (12, 12, 12, 12, 12), max_iter=
50, random_state=42, activation='relu', solver='adam', batch_size=200)
mlp.fit(X_train_std, y_train)
weights = mlp.coefs_
print(len(weights))
```

```
6
```

In [10]:

```
for i in range(6):
    print(weights[i].shape)
```

```
(784, 12)
(12, 12)
(12, 12)
(12, 12)
(12, 12)
(12, 10)
```

In [11]:

```
print(784*12+12*12*4+12*10)
```

```
10104
```

(c)
The optimal hyperparameter: number of hidden layer = 5, number of neurons per layer = 12
The number of weights is 784*12+12*12*4+12*10 = 10104

(d)
The performance of the neural network largely depends on the number of hidden layers and number of neurons per layer. We can improve the performance of the neural network by increasing the number of hidden layers and number of neurons per layer. If the number is too small, the model could be underfitting.
When the number of hidden layer and neurons per layer reach certain amount, the performance of the neural network won't improve too much if we keep enlarging that number. The appropriate number depends on the complexity of our training data. If the number is too big, it might cause the accuracy to decrease since it could be overfitting.