Homework 4

Due May 3, 2019 by 11:59pm

Instructions: Upload your answers to the questions below to Canvas. Submit the answers to the questions in a PDF file and your code in a (single) separate file. Be sure to comment your code to indicate which lines of your code correspond to which question part. There is 1 reading assignment and 4 exercises in this homework, including 1 exercise to get you started with the data competition and 1 optional exercise.

Reading Assignment

Read Ch. 10, Sec. 2 in *An Introduction to Statistical Learning* and Ch. 14, Sec. 5.1 in *The Elements of Statistical Learning*.

1 Exercise 1

In this exercise, you will implement by hand, i.e., **with pen and paper** and a five-function calculator, PCA on a simple dataset. There is no coding involved in this exercise. Please report all your derivations. Below by plotting x_1 we mean drawing by hand of x_1 in the Cartesian plane
Consider the dataset with points $x_1 = (-10, 10), x_2 = (12, -8), x_3 = (6, -14), x_4 = (-4, 4)$.

(a) Plot the data.

(b) Recall that PCA projects the data onto a subspace that is "closest" to the observations. Before calculating anything, draw a line on the plot where you think the line (i.e., shifted 1-D subspace) is that is closest to the points. Estimate the slope and intercept of the line.

(c) Center the data. For the centered data report the mean of each column (i.e., feature). Here we will assume that all variables are on the same scale, so we will not standardize them.

(d) Compute the empirical covariance matrix.

(e) Compute the eigenvalues λ_1, λ_2 of the empirical covariance matrix.

(f) Compute the eigenvectors v_1, v_2 of the empirical covariance matrix.

(g) Compute the PCA projections of the points onto a 1-dimensional space (a line). Denote the projected points by x'_1, x'_2, x'_3, x'_4.

(h) Plot x'_1, x'_2, x'_3, x'_4.

(i) Transform the projected points back to the original space using the top principal component and the mean vector from part (b). Denote these points by $\hat{x}_1$, $\hat{x}_2$, $\hat{x}_3$, $\hat{x}_4$.

(j) Plot the following quantities: The original points x_1, x_2, x_3, x_4, the principal components v_1, v_2 shifted by the mean μ of the original data, and the reconstructed points $\hat{x}_1$, $\hat{x}_2$, $\hat{x}_3$, $\hat{x}_4$. Does your plot look reasonable?

See the next two pages.

2 Exercise 2

In this problem you will generate simulated data and then perform PCA on the data. For this purpose, you will write and then use *your own power iteration algorithm*. The Power Iteration algorithm returns the top pair of eigenvalue λ and eigenvector v of a matrix A.

Algorithm 1 Power Iteration Algorithm

initialization v_0 random vector, and large number N .
repeat for $k = 1, 2, 3, \cdots, N$

- Perform update $z_k = Av_{k-1}$,
- Perform update $v_k = \frac{z_k}{\|z_k\|_2}$, $\lambda_k = v_k^T A v_k$.

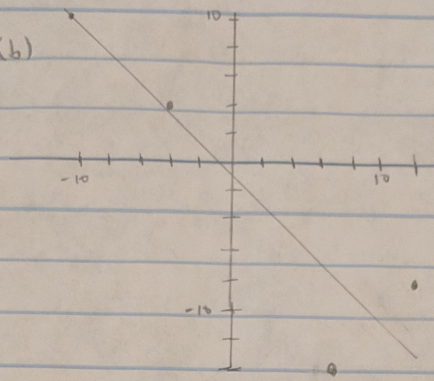
until the stopping criterion is satisfied.

Note that "first two principal component score vectors" refers to the results from projecting the original data to a two-dimensional space with PCA.

(a) Generate a simulated data set with 30 observations in each of three classes (i.e. 90 observations total), and 50 features. Hint: There are a number of functions in numpy that you can use to generate data. One example is the `numpy.random.normal()` function; `numpy.random.uniform()` is another option. Be sure to add a mean shift to the observations in each class so that there are three distinct classes.

```
import copy
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sklearn.cluster
from sklearn.decomposition import PCA
import sklearn.preprocessing
```

1. (a) and (b)



Slope ≈ -1

Intercept ≈ -1

(c)

Original data		
x_1	-10	10
x_2	12	-8
x_3	6	-14
x_4	-4	4
Mean	1	-2

Centered data		
$\tilde{x}_1$	-11	12
$\tilde{x}_2$	11	-6
$\tilde{x}_3$	5	-12
$\tilde{x}_4$	-5	6
Mean	0	0

(d) $\hat{C} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i-\bar{x})(x_i-\bar{x})^T$

$= \frac{1}{3}\begin{bmatrix}-11 & 11 & 5 & -5\\ 12 & -6 & -12 & 6\end{bmatrix}\begin{bmatrix}-11 & 12\\ 11 & -6\\ 5 & -12\\ -5 & 6\end{bmatrix}$

$= \frac{1}{3}\begin{bmatrix}292 & -288\\ -288 & 360\end{bmatrix}$

(e) $\det(\hat{C}-\lambda I) = \begin{vmatrix}292/3-\lambda & -288/3\\ -288/3 & 360/3-\lambda\end{vmatrix} = 0$

$\Leftrightarrow \left(\frac{292}{3}-\lambda\right)\left(\frac{360}{3}-\lambda\right) - \left(\frac{288}{3}\right)^2 = 0$

$\Leftrightarrow \lambda^2 - \frac{652}{3}\lambda + 2464 = 0$

$\Leftrightarrow \lambda = \frac{\frac{652}{3} \pm \sqrt{\left(\frac{652}{3}\right)^2 - 4(1)(2464)}}{2}$

$= \frac{652/3 \pm 580/3}{2}$

$\Leftrightarrow \lambda_1 = \frac{616}{3}, \lambda_2 = 12$

(f) $\hat{C} - \lambda_1 I = \frac{1}{3}\begin{bmatrix} -324 & -288 \\ -288 & -256 \end{bmatrix}$

$(\hat{C} - \lambda_1 I)v_1 = \frac{1}{3}\begin{bmatrix} -324 & -288 \\ -288 & -256 \end{bmatrix}\begin{bmatrix} v_{11} \\ v_{12} \end{bmatrix} \equiv \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$\Rightarrow 324 v_{11} + 288 v_{12} = 0$

$v_1 \propto \begin{bmatrix} -288 \\ 324 \end{bmatrix} = 36\begin{bmatrix} -8 \\ 9 \end{bmatrix}$

$v_1 = \frac{1}{\sqrt{145}}\begin{bmatrix} -8 \\ 9 \end{bmatrix}$

$\hat{C} - \lambda_2 I = \frac{1}{3}\begin{bmatrix} 256 & -288 \\ -288 & 324 \end{bmatrix}$

$(\hat{C} - \lambda_2 I)v_2 = \frac{1}{3}\begin{bmatrix} 256 & -288 \\ -288 & 324 \end{bmatrix}\begin{bmatrix} v_{21} \\ v_{22} \end{bmatrix} \equiv \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$\Rightarrow 256 v_{21} - 288 v_{22} = 0$

$v_2 \propto \begin{bmatrix} 288 \\ 256 \end{bmatrix} = 32\begin{bmatrix} 9 \\ 8 \end{bmatrix}$

$v_2 = \frac{1}{\sqrt{145}}\begin{bmatrix} 9 \\ 8 \end{bmatrix}$

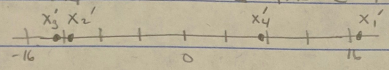
(g) $x_1' = \begin{bmatrix} -11 & 12 \end{bmatrix}\frac{1}{\sqrt{145}}\begin{bmatrix} -8 \\ 9 \end{bmatrix} = \frac{196}{\sqrt{145}} \approx 16.28$

$x_2' = \begin{bmatrix} 11 & -6 \end{bmatrix}\frac{1}{\sqrt{145}}\begin{bmatrix} -8 \\ 9 \end{bmatrix} = \frac{-142}{\sqrt{145}} \approx -11.79$

$x_3' = \begin{bmatrix} 5 & -12 \end{bmatrix}\frac{1}{\sqrt{145}}\begin{bmatrix} -8 \\ 9 \end{bmatrix} = \frac{-148}{\sqrt{145}} \approx -12.29$

$x_4' = \begin{bmatrix} -5 & 6 \end{bmatrix}\frac{1}{\sqrt{145}}\begin{bmatrix} -8 \\ 9 \end{bmatrix} = \frac{94}{\sqrt{145}} \approx 7.81$

(i)



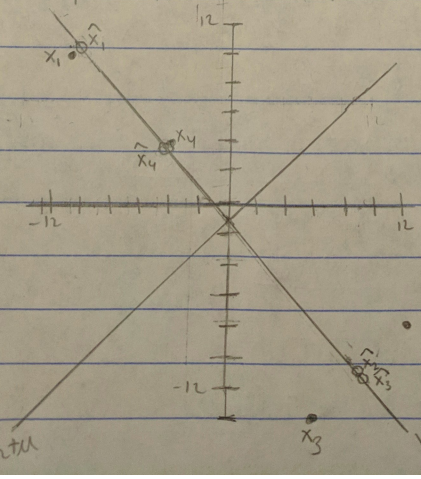
(h) $\hat{x}_1 = x_1' v_1 + \mu = \frac{196}{\sqrt{145}}\frac{1}{\sqrt{145}}\begin{bmatrix} -8 \\ 9 \end{bmatrix} + \begin{bmatrix} 1 \\ -2 \end{bmatrix} \approx \begin{bmatrix} -9.81 \\ 10.17 \end{bmatrix}$

$\hat{x}_2 = x_2' v_1 + \mu = \frac{-142}{\sqrt{145}}\frac{1}{\sqrt{145}}\begin{bmatrix} -8 \\ 9 \end{bmatrix} + \begin{bmatrix} 1 \\ -2 \end{bmatrix} \approx \begin{bmatrix} 8.83 \\ -10.81 \end{bmatrix}$

$\hat{x}_3 = x_3' v_1 + \mu = \frac{-148}{\sqrt{145}}\frac{1}{\sqrt{145}}\begin{bmatrix} -8 \\ 9 \end{bmatrix} + \begin{bmatrix} 1 \\ -2 \end{bmatrix} \approx \begin{bmatrix} 9.17 \\ -11.19 \end{bmatrix}$

$\hat{x}_4 = x_4' v_1 + \mu = \frac{94}{\sqrt{145}}\frac{1}{\sqrt{145}}\begin{bmatrix} -8 \\ 9 \end{bmatrix} + \begin{bmatrix} 1 \\ -2 \end{bmatrix} \approx \begin{bmatrix} -4.19 \\ 3.83 \end{bmatrix}$

(j)



The plot looks reasonable:

- $V_1 + \mu$ is close to the line from part (a)
- Each $\hat{x}_i$ looks to be the orthogonal projection of x_i onto V_1.
- $V_1 + \mu$ is perpendicular to $V_2 + \mu$.

```
# Simulated data set with 30 obs in each of 3 classes and 50 variables
np.random.seed(0)
data = np.zeros((90, 50))
data[0:30, :] = np.random.normal(scale=1, size=(30, 50))
data[30:60, :] = np.random.normal(loc=10, scale=5, size=(30, 50))
data[60:90, :] = np.random.normal(loc=30, scale=5, size=(30, 50))
classes = [0]*30 + [1]*30 + [2]*30
```

(b) Run *your own power iteration algorithm* on the 90 observations. This algorithm was discussed in the week 4 lecture. Plot the first two principal component score vectors. Compare your results to the ones obtained with scikit-learn's PCA algorithm. Use a different color to indicate the observations in each of the three classes. If the three classes appear separated in this plot, then you're done. If not, then return to part (a) and modify the simulation so that there is greater separation between the three classes.

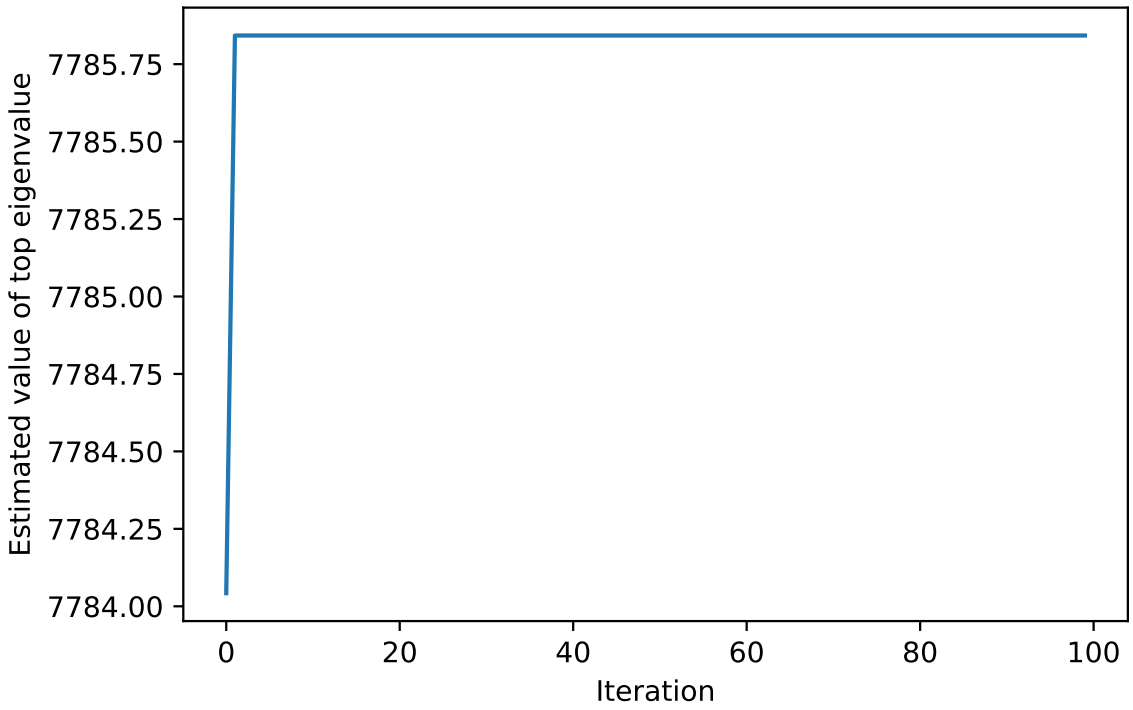
```
# Find the first and second eigenvectors with the power iteration
# algorithm and compare to sklearn
def power_iteration(Z, a_0, eta_0, t_0, num_iters):
    """
    :param Z: Centered data matrix
    :param a_0: Initial starting point
    :param eta_0: Parameter in the numerator of the step size
    :param t_0: Parameter in the denominator of the step size
    :param num_iters: Number of iterations before we stop
    :return: a: The estimated eigenvector corresponding to the maximum
        eigenvalue of 1/N*Z^TZ
    :return lambdas: The estimated value of the maximum eigenvalue at
        each iteration
    """
    t = 0
    a = a_0
    n = np.size(Z, 0)
    lambdas = np.zeros(num_iters)
    for i in range(num_iters):
        # Note it's faster not to compute the matrix Z^TZ
        a = a + eta_0/(t+t_0)*np.dot(Z.T, np.dot(Z, a))
        a = a/np.linalg.norm(a)
        t += 1
        lambdas[i] = a.dot(Z.T).dot(Z).dot(a)/n

    return a, lambdas

def deflate(Z, a):
    return Z - Z.dot(np.outer(a, a))

def display_results(v, lambdas):
```

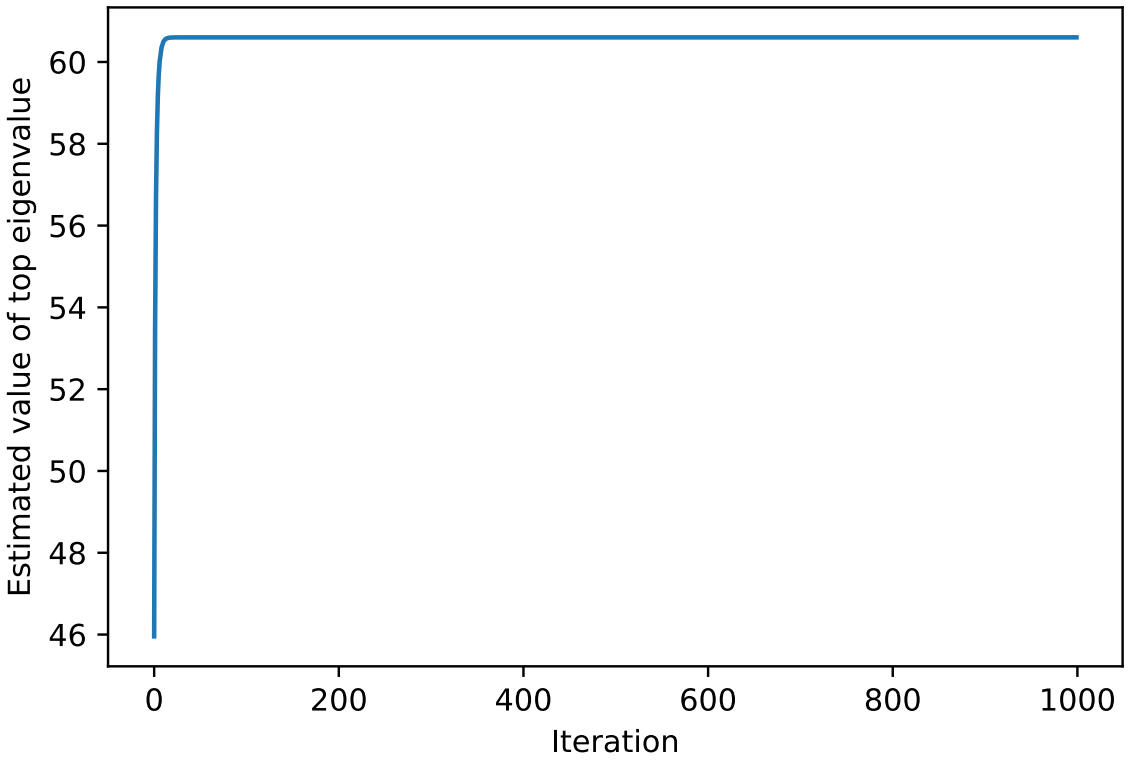
```
    plt.plot(lambdas)
    plt.xlabel('Iteration')
    # Note that the top eigenvalue is the negative of the value of the
    # objective function.
    plt.ylabel('Estimated value of top eigenvalue')
    plt.show()
    print('Estimated top eigenvalue:', lambdas[-1])
    print('Estimated corresponding eigenvector:', v)

Z = data - np.mean(data, axis=0)
a_0 = np.random.randn(np.size(Z, 1))  # Generate a starting point
a_0 /= np.linalg.norm(a_0, axis=0)
v1, lambdas1 = power_iteration(copy.deepcopy(Z), a_0, 0.001, 1, 100)
display_results(v1, lambdas1)

Z1 = deflate(Z, v1)
v2, lambdas2 = power_iteration(copy.deepcopy(Z1), a_0, 0.2, 1, 1000)
display_results(v2, lambdas2)

# Compare to sklearn
pca = PCA(5, svd_solver='full')
pca.fit(data)
print('Scikit-learn results:')
print('First principal component:', pca.components_[0])
print('Second principal component:', pca.components_[1])
print('Third principal component:', pca.components_[2])

V = np.vstack((v1, v2))
scores = np.dot(Z, V.T)
plt.scatter(scores[:, 0], scores[:, 1], c=classes)
plt.xlabel('First principal component score')
plt.ylabel('Second principal component score')
plt.show()
```

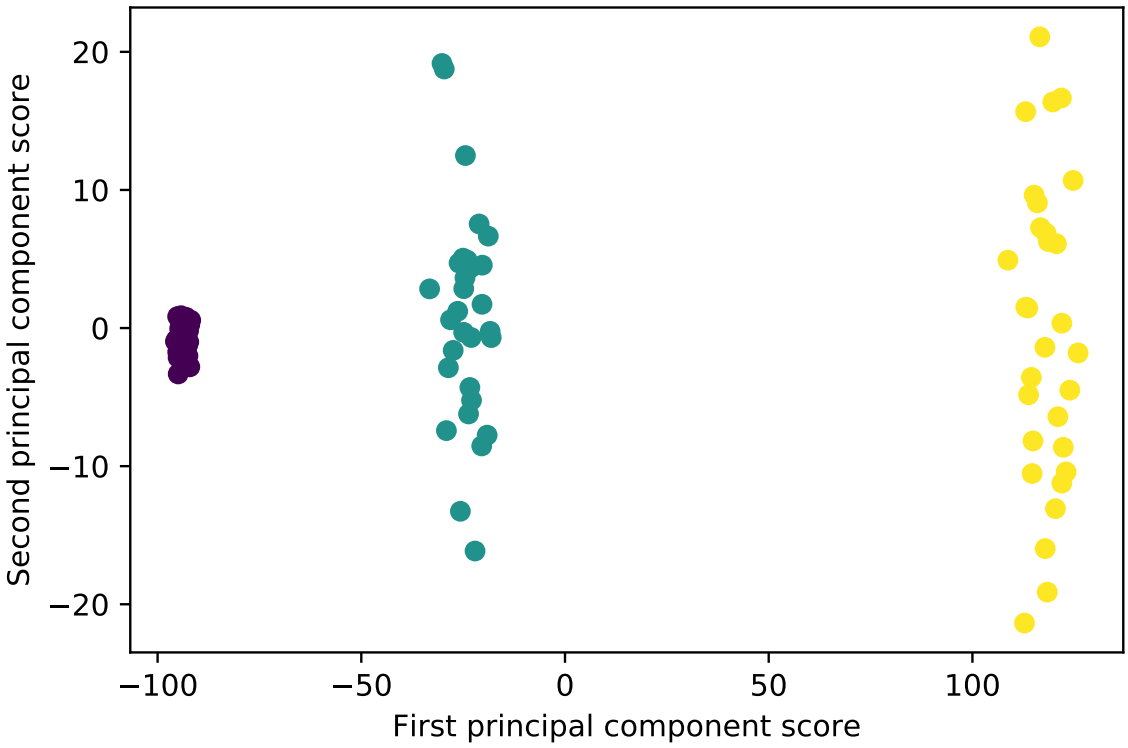
```
Estimated top eigenvalue: 7785.84241545
Estimated corresponding eigenvector: [ 0.14035547  0.15038507
0.13376681  0.13800718  0.13908105  0.1371873
  0.14537124  0.14872575  0.14575471  0.14434398  0.13733506
0.14239661
  0.14314632  0.13253003  0.13231277  0.13883501  0.14170806
0.13576799
  0.14599651  0.1408699   0.13242215  0.14503472  0.14356401
0.14049699
  0.13656458  0.14083578  0.14177369  0.14965884  0.14898204
0.13469159
  0.14395638  0.15186626  0.1393371   0.13928476  0.14915049
0.13374586
```

```
  0.13585994  0.13268708  0.13423011  0.14101681  0.14255016
0.14554164
  0.14217375  0.1490604   0.14321394  0.14180515  0.14492609  0.141148
  0.14146334  0.14548433]
Estimated top eigenvalue: 60.6017555193
Estimated corresponding eigenvector: [-0.2473439  -0.19081904
-0.23836734  0.1144155   0.04480123 -0.09678498
 -0.06737173 -0.10726473  0.12002154 -0.04762503 -0.02393628
-0.01010593
  0.07114522  0.22553404 -0.0188835  -0.11968267 -0.15332092
-0.29902853
 -0.00283003  0.03328903  0.11550796  0.04614586  0.18722805
0.07453675
  0.12878013  0.04175474  0.18950966 -0.14766283 -0.11467818
0.22527839
  0.01838497 -0.00275947 -0.31514919  0.10563546  0.16639145
0.31347822
  0.05076435 -0.21965286 -0.06326779 -0.07450586  0.03028121
0.00286265
 -0.0139902  -0.11722518  0.21938669  0.07387687  0.08825989
0.15357579
 -0.09692194 -0.02601255]
Scikit-learn results:
First principal component: [ 0.14035547  0.15038507  0.13376681
0.13800718  0.13908105  0.1371873
  0.14537124  0.14872575  0.14575471  0.14434398  0.13733506
0.14239661
  0.14314632  0.13253003  0.13231277  0.13883501  0.14170806
0.13576799
  0.14599651  0.1408699   0.13242215  0.14503472  0.14356401
0.14049699
  0.13656458  0.14083578  0.14177369  0.14965884  0.14898204
0.13469159
  0.14395638  0.15186626  0.1393371   0.13928476  0.14915049
0.13374586
  0.13585994  0.13268708  0.13423011  0.14101681  0.14255016
0.14554164
  0.14217375  0.1490604   0.14321394  0.14180515  0.14492609  0.141148
  0.14146334  0.14548433]
Second principal component: [ 0.2473439   0.19081904  0.23836734
-0.1144155  -0.04480123  0.09678498
  0.06737173  0.10726473 -0.12002154  0.04762503  0.02393628
0.01010593
 -0.07114522 -0.22553404  0.0188835   0.11968267  0.15332092
0.29902853
  0.00283003 -0.03328903 -0.11550796 -0.04614586 -0.18722805
-0.07453675
```

```
 -0.12878013 -0.04175474 -0.18950966  0.14766283  0.11467818
-0.22527839
 -0.01838497  0.00275947  0.31514919 -0.10563546 -0.16639145
-0.31347822
 -0.05076435  0.21965286  0.06326779  0.07450586 -0.03028121
-0.00286265
  0.0139902   0.11722518 -0.21938669 -0.07387687 -0.08825989
-0.15357579
  0.09692194  0.02601255]
Third principal component: [ 0.08874455 -0.0097513   0.17666035
0.07382761  0.01902618  0.02903808
 -0.17005649 -0.03075347  0.17999775  0.07579172 -0.17330568
-0.16593221
  0.38868904 -0.21850189 -0.01591537  0.20178466  0.18667182
-0.11769952
 -0.04270144 -0.14289581 -0.04108697  0.15203483  0.04417053
-0.03464859
 -0.19443127 -0.08965128  0.04325893 -0.02136635  0.22174956
0.11721561
  0.10141679 -0.02622193 -0.11151947  0.15363963 -0.07681837
-0.0476485
 -0.04914786 -0.36115514  0.09322139 -0.07113258 -0.09014252
-0.03953804
 -0.03715776 -0.10788383  0.03226577 -0.24681513  0.09623082
-0.0850562
  0.27418342  0.02634554]
```







Note that the eigenvectors from the power iteration algorithm are approximately the same as scikit-learn's up to a possible sign flip.

(c) (Bonus) Note that the Power Iteration algorithm solves the equation $Av = \lambda v$, that is $\min_v \ \|Av - \lambda v\|_2$. What other stopping criterion than the maximum number of iterations does it suggest for your own power iteration algorithm?
We can test whether $\|Av - \lambda v\| \leq \epsilon$ for some tolerance ϵ.

3 Data Competition Project

(a) Pick two classes of your choice from the dataset.

(b) Run PCA on this subset of the data using Scikit-Learn and project the data down to two dimensions.

(c) You will be training an ℓ_2-regularized logistic regression classifier. Find the value of the regularization parameter λ using Scikit-Learn with 5-fold cross-validation.

(d) Train a classifier using ℓ_2-regularized logistic regression on the training set using **your own fast gradient algorithm**.

(e) Repeat steps (b)-(d), trying all powers of two up to the number of features in the dataset.

(f) Plot, with different colors, the *misclassification error* on the training set and on the test set vs. the dimension of the projection. Note that to obtain the performance on the test set you will need to submit to Kaggle, and you can only submit three times per day. If you only have enough time for three submissions, submit your predictions for dimensions 2, 16, and 128.

4 Optional Exercise

In this exercise, you will compute the gradients of several loss functions. Notation: for an example-label pair (x, y), the prediction is $x^T\beta$.

- $\ell(y, x^T\beta) = (y - x^T\beta)^4$
  $\nabla\ell = -4x(y - x^T\beta)^3$
- $\ell(y, x^T\beta) = yx^T\beta - \exp(x^T\beta)$
  $\nabla\ell = yx - x\exp(x^T\beta)$
- $\ell(y, x^T\beta) = \big(\max(0,\ 1 - yx^T\beta)\big)^2$
  $\nabla\ell = -2yx\max(0, 1 - yx^T\beta)$