CS464 Introduction to Machine Learning
Homework Assignment 1

Ilker Demirel

21502856

Q1

Q1.1

Probability of any two students having same birthday in a classroom of size n is (call this event, event A),

$$\begin{aligned} P(A) &= 1 - P(A') \\ &= 1 - \frac{365 \cdot 364 \cdot \cdots \cdot (365 - (n-1))}{365^n} \end{aligned} \tag{1}$$

In equation (1), the denominator of the subtracted term is the total number of birthday combinations, and the nominator is the total number of birthday combinations when everyone student has a different birthday. See figure (1) for n vs $P(A)$.

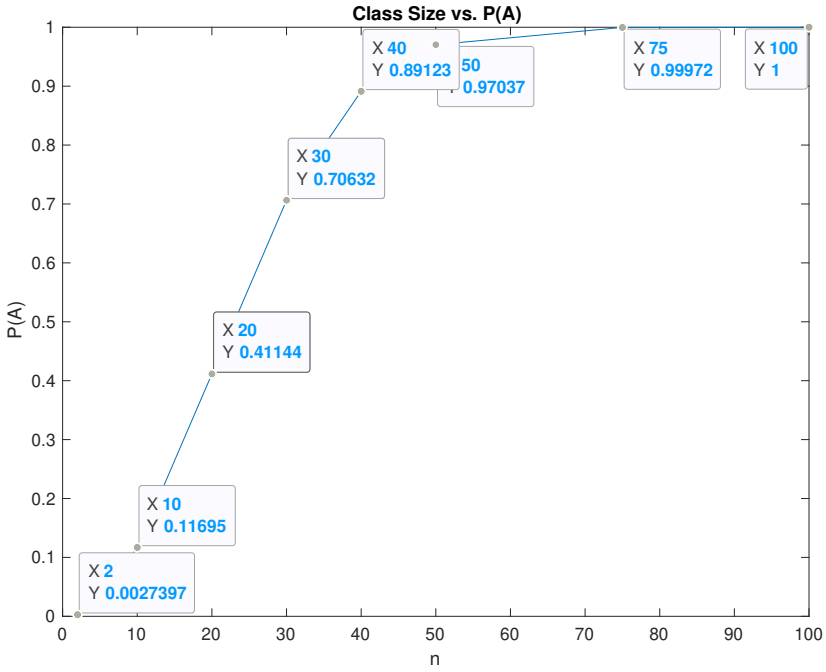

Figure 1: Class Size (n) vs. $P(A)$

Q1.2

Minimum number of students to make sure that any two students have the same birthday is 366, since there are 365 days in a year.

Q2

Q2.1

$P(S = disease) = 0.011$
$P(S = healthy) = 0.989$
$P(T = positive|S = disease) = 0.94$
$P(T = negative|S = disease) = 0.06$
$P(T = positive|S = healthy) = 0.02$
$P(T = negative|S = healthy) = 0.98$

Q2.2

$P(S = disease|T = positive) = \frac{P(T=positive|S=disease)P(S=disease)}{P(T=positive)} = \frac{0.94 \cdot 0.011}{0.94 \cdot 0.011 + 0.02 \cdot 0.989} \approx 0.35$

Given the test result for a patient is positive, the probability that patient has the disease is 0.35. Therefore it is not reasonable to diagnose a patient with the disease when the test result is positive.

Q2.3

Since $P(S = disease|T = positive) \neq 1$, one can **never** definitely diagnose a patient as sick. That being said, the criterion for **confidentally** diagnosing a patient as sick is not defined in the question. Given there are n positive test results, the probability that the patient is sick is,

$$P(S = disease|n\ positive\ tests) = 1 - 0.65^n \tag{2}$$

According to equation (2), if there are seven positive tests, the patient is sick with a probability of $1 - 0.65^7 \approx 0.95$.

Q3

Q3.1

The test accuracy is $\approx$ **0.9489**

Q3.2

Cleavage occurs between the amino acids at the indices below:

(5, 6), (40, 41), (42, 43), (60, 61), (79, 80), (131, 132), (168, 169), (183, 184), (196, 197), (215, 216), (295, 296), (315, 316), (320, 321), (341, 342), (342, 343), (362, 363), (366, 367), (376, 377), (447, 448), (465, 466), (467, 468), (482, 483)

Q3.3

Normally, these two values should be the same. But I assume the question means the sum of the logarithms by saying **"probabilities"**. Thinking that way, one expects that these two

values will probably be different since we do not normalize the values while computing them via Bayes rule, and these values actually turn out to be different.

Starting index of the 8-mer assigned to class 1 with highest probability: **359**
Starting index of the 8-mer assigned to class 0 with lowest probability: **5**

Q3.4

In figure (2), test accuracy versus alpha is plotted using all the training samples. We see that smoothing does not really improve the performance because we already have an accuracy of 0.9489 without smoothing. The discussion here is, could we have a low test accuracy without smoothing? Yes, because we are using -inf values when we encounter 0 values at theta's. Imagine for some of the test samples, both $P(Y = 1|X)$ and $P(Y = 0|X)$ is equal to -inf. In this case (equality), we will assign all these samples to class 0, which means we will classify all the samples belonging to class 1 in this set incorrectly. However, in our case, I have checked the test samples and observed that all the test samples that have $P(Y = 1|X)$ = -inf belong to class 0, so we do not make incorrect decisions on this test set even though we do not use smoothing. But smoothing may significantly increase the performance with other test sets.

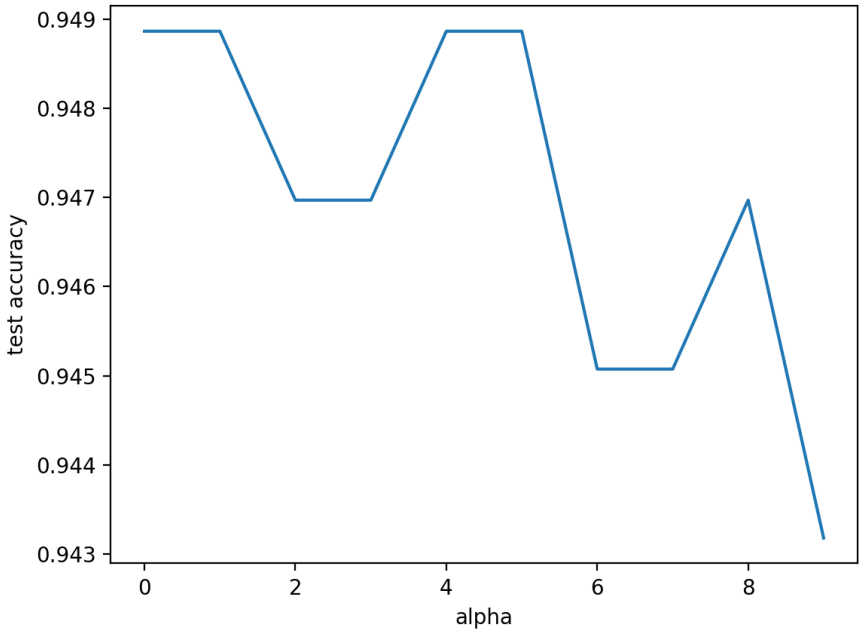

Figure 2: alpha vs. test accuracy using all training samples

In figure (3), test accuracy versus alpha is plotted using only first 75 rows of the training samples. We see that smoothing helps in this case increase test accuracy for $alpha = 1$. This is probably due to the reason discussed in the first part of this question. Smoothing helps us correctly classify the samples belong to class 1 and have $P(Y = 0|X) = P(Y = 1|X) = -inf$.

Q3.5

In figure (4), you can see the test accuracy versus k (# of features with highest mutual information used). As expected, the test accuracy increases with k, but after some point, it increases

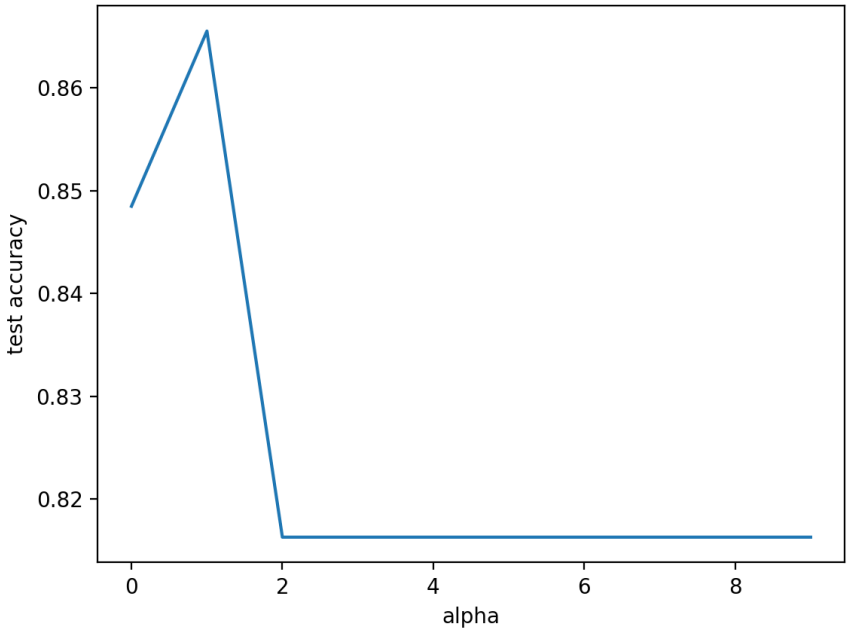

Figure 3: alpha vs. test accuracy using first 75 rows of the training samples

very slowly and we can actually see that using only 30-40 features, we can obtain a pretty good test accuracy.

Also, the highest test accuracy is **0.9527**, which is higher than the one in Q3.1 (**0.9489**). This is a very small improvement and it may be due to just stochastic processes, but also one should keep that in my that using all the features may lead to overfitting and poor performance in the test set. So this improvement may be due to avoiding overfitting to the training set by using less features.

Q3.6

In figure (5), you can see proportion of variance explained (PVE) vs. k (# of principal components used). Also, PVE(3) = **0.0759**. We can tell by looking at figure (5) and PVE(3) being very small that applying PCA is not really a good idea with this dataset since we can not explain a big portion of variance in the data (like %90) using a small percentage of features.

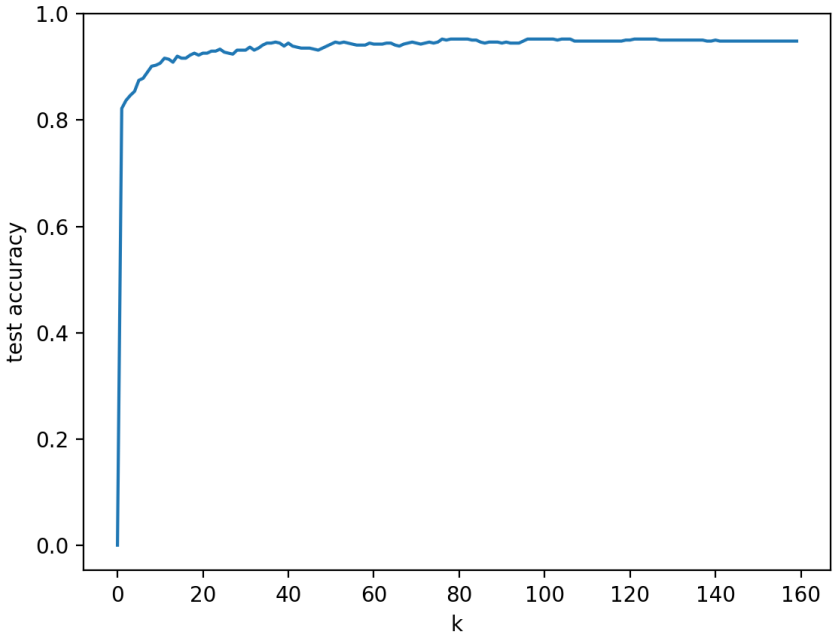

Figure 4: k vs. test accuracy

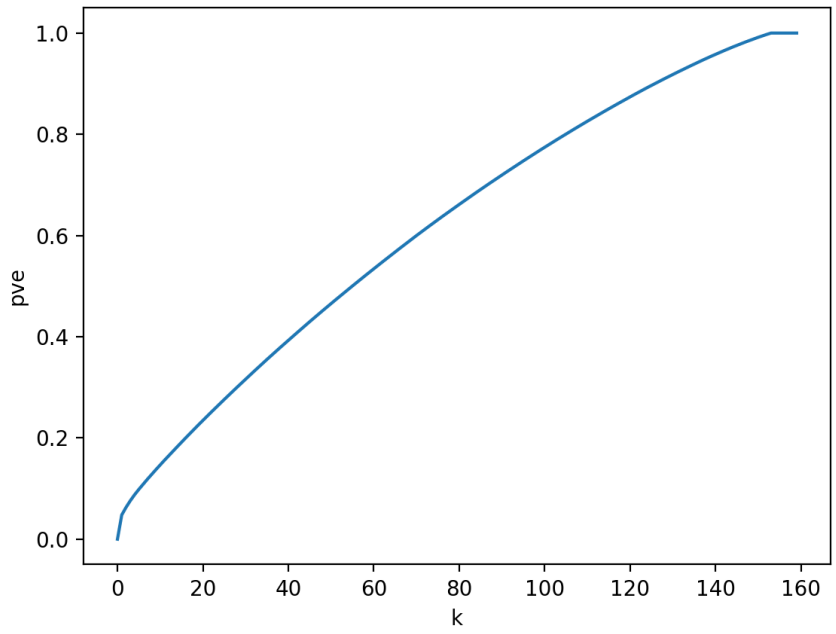

Figure 5: k vs. pve

Appendix

Listing 1: q3main.py

```
import numpy as np
import csv
import matplotlib.pyplot as plt


# DEFINITIONS

AA_DICT = {  # amino-acid indices
    'g': np.array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]),
    'p': np.array([0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]),
    'a': np.array([0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]),
    'v': np.array([0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]),
    'l': np.array([0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]),
    'i': np.array([0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]),
    'm': np.array([0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]),
    'c': np.array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]),
    'f': np.array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]),
    'y': np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]),
    'w': np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]),
    'h': np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]),
    'k': np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]),
    'r': np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]),
    'q': np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]),
    'n': np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]),
    'e': np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]),
    'd': np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]),
    's': np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]),
    't': np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1])
}

AA_COUNT = 20  # amino-acid count
SEQ_LENGTH = 8  # 8-mers
FEATURE_COUNT = AA_COUNT*SEQ_LENGTH
TRAINING_SAMPLE_COUNT = 6062  # Change to 75 for Q4
TEST_SAMPLE_COUNT = 528


# PRE-PROCESSING

train_file = open('q2_train_set.txt', 'r')
reader = csv.reader(train_file)
train_list = [row for row in reader]
train_samples = np.asarray(train_list, dtype=int)
del train_list

test_file = open('q2_test_set.txt', 'r')
reader = csv.reader(test_file)
test_list = [row for row in reader]
test_samples = np.asarray(test_list, dtype=int)
del test_list

test_set = test_samples[0:TEST_SAMPLE_COUNT, 0:FEATURE_COUNT]
test_labels = test_samples[0:TEST_SAMPLE_COUNT, FEATURE_COUNT]
```

```
train_set = train_samples[0:TRAINING_SAMPLE_COUNT, 0:FEATURE_COUNT]
train_labels = train_samples[0:TRAINING_SAMPLE_COUNT, FEATURE_COUNT]


def naive_bayes(alpha, indices):  # alpha for smoothing, indices for MI

    # Q1
    # TRAINING NAIVE-BAYES CLASSIFIER

    train_z = np.where(train_labels == 0)[0]  # '0' labeled indices
    train_nz = np.where(train_labels == 1)[0]  # '1' labeled indices

    n = TRAINING_SAMPLE_COUNT  # total count
    n_0 = train_z.shape[0]  # samples with label '1'
    n_1 = train_nz.shape[0]  # samples with label '0'

    pi_0 = n_0/n
    pi_1 = n_1/n

    theta = np.zeros((2, indices.shape[0]))  # 2 because 2 labels, '0' and '1'
    for i in range(2):
        denominator = (i == 0)*n_0 + (i == 1)*n_1
        for j in range(indices.shape[0]):
            if i == 0:
                numerator = np.sum(train_set[train_z, indices[j]])
                theta[i, j] = (numerator + alpha)/(denominator + 2*alpha)
            else:
                numerator = np.sum(train_set[train_nz, indices[j]])
                theta[i, j] = (numerator + alpha)/(denominator + 2*alpha)

    return pi_0, pi_1, theta


# EVALUATION ON TEST SET

def nb_pred(arr, pi_0, pi_1, theta, indices):  # naive-bayes prediction
    pred_labels = np.zeros(arr.shape[0])  # first index is the number of samples
    pr_cleavage = np.zeros((arr.shape[0], 2))
    for i in range(arr.shape[0]):
        y_hat_0 = np.log(pi_0)
        y_hat_1 = np.log(pi_1)
        for j in range(indices.shape[0]):
            if arr[i, indices[j]] == 0:
                y_hat_0 = y_hat_0 + (-np.inf) if 1 - theta[0, j] == 0 else\
                    y_hat_0 + np.log(1 - theta[0, j])
                y_hat_1 = y_hat_1 + (-np.inf) if 1 - theta[1, j] == 0 else\
                    y_hat_1 + np.log(1 - theta[1, j])
            else:
                y_hat_0 = y_hat_0 + (-np.inf) if theta[0, j] == 0 else\
                    y_hat_0 + np.log(theta[0, j])
                y_hat_1 = y_hat_1 + (-np.inf) if theta[1, j] == 0 else\
                    y_hat_1 + np.log(theta[1, j])
        pr_cleavage[i, :] = np.array([y_hat_0, y_hat_1])
        if pr_cleavage[i, 1] > pr_cleavage[i, 0]:
            pred_labels[i] = 1
        else:
            pred_labels[i] = 0
    return pred_labels, pr_cleavage
```

```
def cleavage_indices(labels):
    indices = np.where(labels == 1)[0]
    indices = indices + 3
    indices = list(map(lambda x: (x, x+1), indices))
    return indices


pi_0, pi_1, theta = naive_bayes(0, np.arange(160))
pred_test_labels, pred_test_cleavage = nb_pred(test_set, pi_0, pi_1, theta, np.arange(16
test_acc = np.sum(pred_test_labels == test_labels)/TEST_SAMPLE_COUNT
print(test_acc)  # Q1 test accuracy


# Q2 & Q3

with open('q2_gag_sequence.txt', 'r') as gag_file:
    gag_str = gag_file.readlines()[0]

eightmer_count = len(gag_str) - SEQ_LENGTH + 1
gag_eightmers = np.zeros((eightmer_count, FEATURE_COUNT))

for i in range(eightmer_count):
    gag_eightmers[i, :] = np.append(AA_DICT[gag_str[i]],
                                    [AA_DICT[gag_str[i+1]],
                                     AA_DICT[gag_str[i+2]],
                                     AA_DICT[gag_str[i+3]],
                                     AA_DICT[gag_str[i+4]],
                                     AA_DICT[gag_str[i+5]],
                                     AA_DICT[gag_str[i+6]],
                                     AA_DICT[gag_str[i+7]]])

pred_gag_labels, pr_cleavage = nb_pred(gag_eightmers, pi_0, pi_1, theta, np.arange(160))
gag_cleavage_indices = cleavage_indices(pred_gag_labels)
print(gag_cleavage_indices)  # Q2: aa indices where cleavages occurs

# Q3: aa index with highest calculated log prob-like value of cleavage
highest_pr1_index = np.argmax(pr_cleavage[:, 1])
# Q3: aa index with lowest calculated log prob-like value of cleavage
lowest_pr0_index = np.argmax(pr_cleavage[:, 0])

print(highest_pr1_index)
print(lowest_pr0_index)


# Q4
# Change ALPHA and TRAINING_SAMPLE_COUNT variables for this part.

q4_accuracies = np.zeros(10)
for alpha in range(10):
    pi_0, pi_1, theta = naive_bayes(alpha, np.arange(160))
    pred_labels, pred_cleavages = nb_pred(test_set, pi_0, pi_1, theta, np.arange(160))
    q4_accuracies[alpha] = np.sum(pred_labels == test_labels)/TEST_SAMPLE_COUNT

plt.plot(q4_accuracies)
plt.xlabel('alpha')
plt.ylabel('test accuracy')
```

```
plt.show()


# Q5

def calc_mi(arr, arr_labels, k):
    arr_z = np.where(arr_labels == 0)[0]
    arr_nz = np.where(arr_labels == 1)[0]
    n = np.sum(arr_labels)
    n_11 = np.sum(arr[arr_nz, :], axis=0)
    n_10 = np.sum(arr[arr_z, :], axis=0)
    n_01 = arr_nz.shape[0] - n_11
    n_00 = arr_z.shape[0] - n_10
    n_1_dot = n_11 + n_10
    n_dot_1 = n_01 + n_11
    n_0_dot = n_00 + n_01
    n_dot_0 = n_00 + n_10
    mi = 1/n*(n_11*np.log2((n*n_11 + 1e-6)/(n_1_dot*n_dot_1 + 1e-6)) +
              n_01*np.log2((n*n_01 + 1e-6)/(n_0_dot*n_dot_1 + 1e-6)) +
              n_10*np.log2((n*n_10 + 1e-6)/(n_1_dot*n_dot_0 + 1e-6)) +
              n_00*np.log2((n*n_00 + 1e-6)/(n_0_dot*n_dot_0 + 1e-6)))
    max_indices = mi.argsort()[-k:][::-1]
    return max_indices


q5_accuracies = np.zeros(160)
for mi in np.arange(1, 160):
    max_indices = calc_mi(train_set, train_labels, mi)
    pi_0, pi_1, theta = naive_bayes(0, max_indices)
    pred_mi_labels, pred_mi_cleavages = nb_pred(test_set, pi_0, pi_1, theta, max_indices
    q5_accuracies[mi] = np.sum(pred_mi_labels == test_labels)/TEST_SAMPLE_COUNT

print(np.amax(q5_accuracies))
print(np.argmax(q5_accuracies))
plt.plot(q5_accuracies)
plt.xlabel('k')
plt.ylabel('test_accuracy')
plt.show()


# Q6
# PCA

def pca(arr):
    n, m = arr.shape
    for i in range(m):
        arr[:, i] = (arr[:, i] - np.mean(arr[:, i])) / np.std(arr[:, i])
    covariance = np.dot(arr.T, arr)/(n - 1)
    eig_vals, eig_vecs = np.linalg.eig(covariance)
    max_indices = eig_vals.argsort()[-160:][::-1]
    return arr, eig_vals, eig_vecs, max_indices


def pve(arr, eig_vecs, max_indices, k):  # k is the # of principal components
    denominator = np.sum(np.multiply(arr, arr))
    numerator = 0
    for i in range(k):
        temp = np.dot(arr, eig_vecs[:, max_pca_indices[i]])
```

```
        temp = np.multiply(temp, temp)
        temp = np.sum(temp)
        numerator = numerator + temp
    pve = numerator/denominator
    return pve


cent_train_set, eig_vals, eig_vecs, max_pca_indices = pca(train_set)
pve_3 = pve(train_set, eig_vecs, max_pca_indices, 3)
print(pve_3)

pve_arr = np.zeros(160)
for k in range(160):
    pve_arr[k] = pve(train_set, eig_vecs, max_pca_indices, k)

plt.plot(pve_arr)
plt.xlabel('k')
plt.ylabel('pve')
plt.show()
```