Machine Learning 2019 Spring
Homework #2 Solution

Total Points: 100 points
Maximum Available Points (include bonus): 135.5
Maximum Available Points (All using high level libraries): 92.5

Dealing with Missing Values

All the answers in homework #2 with cancer dataset should be using the in-class average from the training set to deal with the missing values. Remember to do this for each cross validation since you will have different training set every time.

Marks Allocation: Total 9 points [not counted in bonus for each question]

- Evenly distributed for Question 3, 4, 5

Question 1

Marks Allocation: Total 14 points
(a): Appropriate explanation for orthogonally diagonalizable (7 points)
(b): Appropriate explanation for semi-definite (7 points)

Suggested Solution:

The definition of orthogonally diagonalizable is that, a matrix D, an orthogonal matrix C, and a diagonal matrix A, if D is orthogonally diagonalizable, that means we can have

$$D = CAC^T$$

D must be a symmetric matrix since A is a diagonal matrix ($A = A^T$)

Assume we have removed the mean values for the dataset. Now the dataset is a matrix X_{remove_mean}. ${X_{remove\_mean}}^T X_{remove\_mean}$ is actually computing a covariance matrix. We all know that this covariance matrix must be a symmetric matrix, so it is orthogonal diagonalizable.

A symmetric matrix M is semi-definite if and on if $u^T Mu \geq 0$ for every column vector u. The simple way to view that the covariance matrix ${X_{remove\_mean}}^T X_{remove\_mean}$ must be semi-definite is that all the values in this covariance matrix must be greater than 0.

Question 2

Marks Allocation: Total: 7 points

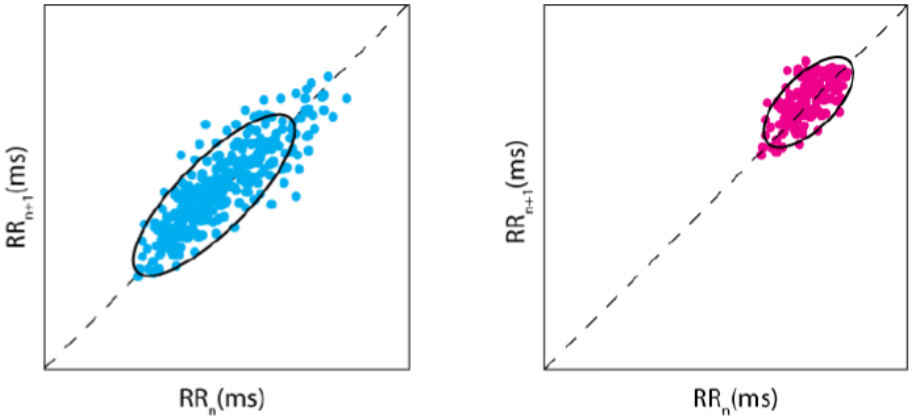
- Give explanations or clues for the question. (4 points)
- Correct answer (3 points)

Suggested Solution:

According to the IMOTIONS's blog about heart rate variability, https://imotions.com/blog/heart-rate-variability/, it says that a greater spread of values would mean an increased HRV, while the close they bunch together, the less HRV there is. It also claims that a higher HRV is associated with good health – the more your heart jumps around, the readier you are for action. On the other hand, a low HRV is associated with ill health.

The related research is from https://www.ahajournals.org/doi/full/10.1161/01.CIR.102.11.1239. This research was published in AHA Journal which claims that a low HRV can be used to predict heart disease and mortality from several causes.

From the lecture notes, the two Lorenz plots for R-R intervals shows that the left one spreads largely while the right one is more dense. It means that the left one is of higher HRV while the left one is of lower HRV. Therefore, the left one is more likely to be a healthy subject.



Question 3

Marks Allocation: Total 30 points, Max available points (include bonus): 50 points
(a): 25 points, Max available points (include bonus): 45 points

- Complete and correct or reasonable coding flow for Naïve Bayes Classifier (15 points)
- Correct or reasonable work flow for the entire use of model including the order of shuffling, splitting, cross validation… etc (10 points)
- Good explanation with mathematic expression for continuous Naïve Bayes Classifier (5 points bonus)
- Using high level libraries to perform Naïve Bayes Classifier (continuous) directly ((a) 20% discount)
- Do this from scratch: ((a) + 50% bonus)

(b): 5 points

- Appropriate explanation for the solution of imbalance class problem for Naïve Bayes Classifier (5 points)

Suggested Solution:

The original Naïve Bayes Classifier for discrete data is,

$$P(C = c|x_1, \dots, x_n) = \frac{P((C = c) \cap x_1, \dots, x_n)}{P(x_1, \dots, x_n)} = \frac{P(C = c)\prod_{i=1}^{n} P(x_i|C = c)}{\prod_{i=1}^{n} P(x_i)}$$

For continuous data, we transform the original formula to,

$$P(C = c|x_1, \dots, x_n) = \frac{P(C = c)\prod_{i=1}^{n} pdf\left(x_i|\mu_{i,C=c}, \sigma^2_{i,C=c}\right)}{\prod_{i=1}^{n} P(x_i)}$$

where,

$$pdf\left(x_i|\mu_{i,C=c}, \sigma^2_{i,C=c}\right) = \frac{1}{\sigma_{i,C=c}\sqrt{2\pi}} e^{-\frac{(x_i-\mu_{i,C=c})^2}{2\sigma^2_{i,C=c}}}$$

Although it is not "probability" but "probability density", we can still use it since probability density can somehow reflect the likelihood of the behaviour. Also, we don't have an interval for us to compute the probability. So we use probability density instead. However, we can simplify the formula by omitting the denominator since it is a constant. The resulting formula would be,

$$P(C = c|x_1, \dots, x_n) = P(C = c)\prod_{i=1}^{n} pdf\left(x_i|\mu_{i,C=c}, \sigma^2_{i,C=c}\right)$$

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

```
"""
Load data
"""
def load_data(data_location):
    data = pd.read_csv(data_location, header=None, index_col=False).values
    features = data[:, 1:-1]
    labels = data[:, -1].reshape(-1, 1)
    return features, labels
```

```
"""
Perform data shuffling
"""
def data_shuffling(features, labels):
    num_data = features.shape[0]
    shuf_arr = np.random.permutation(num_data)
    features_shuf = features[shuf_arr]
    labels_shuf = labels[shuf_arr]
    return features_shuf, labels_shuf
```

```
"""
Split data into training, validation set
"""
```

```
def split_data(features_shuf, labels_shuf):
    num_data = features_shuf.shape[0]
    train_end = int(num_data * 0.7)
    train_features = features_shuf[:train_end]
    train_labels = labels_shuf[:train_end]
    val_features = features_shuf[train_end:]
    val_labels = labels_shuf[train_end:]
    return train_features, train_labels, val_features, val_labels
```

```
"""
Deal with the missing values with in-class average
Hardcoded for Breast Cancer Wisconsin (Original) Data Set
"""
def deal_with_missing_values(train_features, train_labels, val_features, val_labels):
    train_labels = train_labels.astype(np.float32)
    b_train_features = train_features[train_labels.flatten() == 2.]
    m_train_features = train_features[train_labels.flatten() == 4.]
    b_train_table = b_train_features != '?'
    m_train_table = m_train_features != '?'
    no_miss_b_train_features = b_train_features[np.all(b_train_table, axis=1)].astype(np.float32)
    no_miss_m_train_features = m_train_features[np.all(m_train_table, axis=1)].astype(np.float32)
    b_means = np.mean(no_miss_b_train_features, axis=0).reshape(1, -1)
    m_means = np.mean(no_miss_m_train_features, axis=0).reshape(1, -1)
    b_train_features[b_train_features == '?'] = 0.
    m_train_features[m_train_features == '?'] = 0.
    b_train_features = b_train_features.astype(np.float32)
    m_train_features = m_train_features.astype(np.float32)
    val_table = val_features != '?'
    val_features[val_features == '?'] = 0.
    val_features = val_features.astype(np.float32)
    b_train_features = b_train_features + np.multiply(1 - b_train_table.astype(np.float32), b_means)
    m_train_features = m_train_features + np.multiply(1 - m_train_table.astype(np.float32), m_means)
    average_means = (b_means * b_train_features.shape[0] + m_means * m_train_features.shape[0]) / train_features.shape[0]
    val_features = val_features + np.multiply(1 - val_table.astype(np.float32), average_means)
    train_features = np.append(b_train_features, m_train_features, axis=0)
    train_labels = np.append(np.ones((b_train_features.shape[0], 1), dtype=np.float32) * 2., np.ones((m_train_features.shape[0],
1), dtype=np.float32) * 4., axis=0)
    val_labels = val_labels.astype(np.float32)
    return train_features, train_labels, val_features, val_labels
```

```
"""
Compute the probability density using gaussian distribution
"""
def get_prob_den(x, variance, mean):
    prob_den = (1 / np.sqrt(2 * np.pi * variance)) * np.exp(-np.square(x - mean)/(2 * variance))
    return prob_den
```

```
"""
Naive Bayes Classifier (Continuous)
"""
def naive_bayes_classifier_cont(train_features, train_labels, val_features):
    num_features = train_features.shape[1]
    num_train_data = train_features.shape[0]
    prob_class = dict()
    classnames, counts = np.unique(train_labels, return_counts=True)
    classnames = classnames.astype(np.str).tolist()
    counts = counts.tolist()
    for classname, count in zip(classnames, counts):
        prob_class[classname] = count / num_train_data
    means = dict()
    variances = dict()
    for classname in classnames:
        classname_float = float(classname)
        index = (train_labels == classname_float).flatten()
        temp = train_features[index]
        current_means = np.mean(temp, axis=0)
        current_variances = np.var(temp, axis=0)
        for i in range(num_features):
            name = str(i) + '_of_' + classname
```

```
            means[name] = current_means[i]
            variances[name] = current_variances[i]
    predicts = list()
    for line in val_features:
        line_list = line.tolist()
        each_class_out = list()
        for classname in classnames:
            accumulate = 1.
            for i in range(num_features):
                accumulate *= get_prob_den(line_list[i], variances[str(i) + '_of_' + classname], means[str(i) + '_of_' + classname])
            accumulate *= prob_class[classname]
            each_class_out.append((classname, accumulate))
        each_class_out_np = np.array(each_class_out).astype(np.float32)
        predicts.append(each_class_out_np[:, 0][np.argmax(each_class_out_np[:, 1])])
    predicts = np.array(predicts).reshape(-1, 1)
    return predicts
```

```
avg_acc = 0.
for i in range(10):
    features, labels = load_data('breast-cancer-wisconsin.data')
    features_shuf, labels_shuf = data_shuffling(features, labels)
    train_features, train_labels, val_features, val_labels = split_data(features_shuf, labels_shuf)
    train_features, train_labels, val_features, val_labels = deal_with_missing_values(train_features, train_labels, val_features,
val_labels)
    predicts = naive_bayes_classifier_cont(train_features, train_labels, val_features)
    avg_acc += np.mean((predicts == val_labels).flatten()) / 10
print(avg_acc)
```

Output:
0.9647619047619048

Since the formula that we are calculating is,

$$P(C = c|x_1, \dots, x_n) = P(C = c)\prod_{i=1}^{n} pdf\left(x_i|\mu_{i,C=c}, \sigma^2_{i,C=c}\right)$$

$P(C = c)$ counts a lot. Imbalance classes will lead to disastrous predictions.

We can do modified stratified random sampling from the dataset by class and with cross validation to get a more robust result for the data with imbalance classes. That is, we sample a certain amount of data from the training set which is with balance classes. Then, we perform cross validation so that we can get an average accuracy.

For example, assume that we have n number of data for class A and m number of data for class B. n is far smaller than m. We can then select k samples from class A and class B where k < n. Such that our training set would be the size of 2k which is with balance classes.

Question 4

Marks Allocation: Total 25 points, Max available points (include bonus): 37.5 points

- Complete and correct or reasonable coding flow for wrapper-type feature selection (15 points)
- Correct or reasonable work flow for the question including the order of shuffling, splitting, cross validation… etc (10 points)
- Using high level libraries to perform wrapper-type feature selection directly (20% discount)
- Do this from scratch (+ 50% bonus)

Suggested Solution:

```
import numpy as np
import pandas as pd
```

```
"""
Load data
"""
def load_data(data_location):
    data = pd.read_csv(data_location, header=None, index_col=False).values
    features = data[:, 1:-1]
    labels = data[:, -1].reshape(-1, 1)
    return features, labels
```

```
"""
Perform data shuffling
"""
def data_shuffling(features, labels):
    num_data = features.shape[0]
    shuf_arr = np.random.permutation(num_data)
    features_shuf = features[shuf_arr]
    labels_shuf = labels[shuf_arr]
    return features_shuf, labels_shuf
```

```
"""
Split data into training, validation, and testing set according to classes
Hardcoded for Breast Cancer Wisconsin (Original) Data Set
"""
def split_data(features_shuf, labels_shuf):
    b_features_shuf = features_shuf[labels_shuf.flatten().astype(np.float32) == 2.]
    b_labels_shuf = labels_shuf[labels_shuf.flatten().astype(np.float32) == 2.]
    m_features_shuf = features_shuf[labels_shuf.flatten().astype(np.float32) == 4.]
    m_labels_shuf = labels_shuf[labels_shuf.flatten().astype(np.float32) == 4.]
    b_num_train = int(b_features_shuf.shape[0] * 0.5)
    m_num_train = int(m_features_shuf.shape[0] * 0.5)
    b_num_val = int(b_features_shuf.shape[0] * 0.2)
    m_num_val = int(m_features_shuf.shape[0] * 0.2)
    train_features = np.append(b_features_shuf[:b_num_train], m_features_shuf[:m_num_train], axis=0)
    train_labels = np.append(b_labels_shuf[:b_num_train], m_labels_shuf[:m_num_train], axis=0)
    val_features = np.append(b_features_shuf[b_num_train:b_num_train + b_num_val],
m_features_shuf[m_num_train:m_num_train + m_num_val], axis=0)
    val_labels = np.append(b_labels_shuf[b_num_train:b_num_train + b_num_val], m_labels_shuf[m_num_train:m_num_train
+ m_num_val], axis=0)
    test_features = np.append(b_features_shuf[b_num_train + b_num_val:], m_features_shuf[m_num_train + m_num_val:],
axis=0)
    test_labels = np.append(b_labels_shuf[b_num_train + b_num_val:], m_labels_shuf[m_num_train + m_num_val:], axis=0)
    return train_features, train_labels, val_features, val_labels, test_features, test_labels
```

```
"""
Deal with the missing values with in-class average
Hardcoded for Breast Cancer Wisconsin (Original) Data Set
"""
def deal_with_missing_values(train_features, train_labels, val_features, val_labels):
    train_labels = train_labels.astype(np.float32)
    b_train_features = train_features[train_labels.flatten() == 2.]
    m_train_features = train_features[train_labels.flatten() == 4.]
    b_train_table = b_train_features != '?'
    m_train_table = m_train_features != '?'
    no_miss_b_train_features = b_train_features[np.all(b_train_table, axis=1)].astype(np.float32)
    no_miss_m_train_features = m_train_features[np.all(m_train_table, axis=1)].astype(np.float32)
```

```
    b_means = np.mean(no_miss_b_train_features, axis=0).reshape(1, -1)
    m_means = np.mean(no_miss_m_train_features, axis=0).reshape(1, -1)
    b_train_features[b_train_features == '?'] = 0.
    m_train_features[m_train_features == '?'] = 0.
    b_train_features = b_train_features.astype(np.float32)
    m_train_features = m_train_features.astype(np.float32)
    val_table = val_features != '?'
    val_features[val_features == '?'] = 0.
    val_features = val_features.astype(np.float32)
    b_train_features = b_train_features + np.multiply(1 - b_train_table.astype(np.float32), b_means)
    m_train_features = m_train_features + np.multiply(1 - m_train_table.astype(np.float32), m_means)
    average_means = (b_means * b_train_features.shape[0] + m_means * m_train_features.shape[0]) / train_features.shape[0]
    val_features = val_features + np.multiply(1 - val_table.astype(np.float32), average_means)
    train_features = np.append(b_train_features, m_train_features, axis=0)
    train_labels = np.append(np.ones((b_train_features.shape[0], 1), dtype=np.float32) * 2., np.ones((m_train_features.shape[0],
1), dtype=np.float32) * 4., axis=0)
    val_labels = val_labels.astype(np.float32)
    return train_features, train_labels, val_features, val_labels
```

```
"""
k-NN inferring
"""
def kNN_infer(train_features, train_labels, inf_features, k):
    train_data_size = train_features.shape[0]
    train_features_size = train_features.shape[1]
    inf_data_size = inf_features.shape[0]
    predictions = np.empty((inf_features.shape[0], 1), dtype=np.float32)
    for i in range(inf_data_size):
        current_inf = np.tile(inf_features[i].reshape(1, -1), (train_data_size, 1))
        euclidean = np.linalg.norm(np.subtract(train_features, current_inf), axis=1)
        sort_index = np.argsort(euclidean)
        k_labels = train_labels.reshape(-1)[sort_index][:k]
        pred_class, counts = np.unique(k_labels, return_counts=True)
        predict = pred_class[np.argmax(counts)]
        predictions[i, 0] = predict
    return predictions
```

```
"""
Wrapper-type feature selection
"""
all_records = list()
for k in range(10):
    features, labels = load_data('breast-cancer-wisconsin.data')
    features_shuf, labels_shuf = data_shuffling(features, labels)
    a, b, c, d, e, f = split_data(features_shuf, labels_shuf)
    train_features, train_labels, val_features, val_labels = deal_with_missing_values(a, b, c, d)
    train_features, train_labels, test_features, test_labels = deal_with_missing_values(a, b, e, f)
    l = list(range(0, 9))
    best_features = list()
    wrapper_train_features = np.empty((train_features.shape[0], 0), dtype=np.float32)
    wrapper_val_features = np.empty((val_features.shape[0], 0), dtype=np.float32)
    records = list()
    for i in range(3):
        wrapper_train_features = np.append(wrapper_train_features, np.zeros((wrapper_train_features.shape[0], 1),
dtype=np.float32), axis=1)
        wrapper_val_features = np.append(wrapper_val_features, np.zeros((wrapper_val_features.shape[0], 1), dtype=np.float32),
axis=1)
        best_feature = -1.
        best_acc = -1.
        for j in l:
            wrapper_train_features[:, i] = train_features[:, j]
            wrapper_val_features[:, i] = val_features[:, j]
            predictions = kNN_infer(wrapper_train_features, train_labels, wrapper_val_features, 3)
            accuracy = np.mean((predictions == val_labels).flatten().astype(np.float32))
            if best_acc < accuracy:
                best_feature = j
                best_acc = accuracy
        records.append(best_feature)
```

```
        l.remove(best_feature)
    all_records.append(records)
all_records_np = np.array(all_records).flatten()
temp, cnt = np.unique(all_records_np, return_counts=True)
sort_idx = np.argsort(cnt)
final = temp[sort_idx][-3:]
print(final)
```

```
"""
Average accuracy for test set
"""
accs = 0.
for k in range(10):
    features, labels = load_data('breast-cancer-wisconsin.data')
    features_shuf, labels_shuf = data_shuffling(features, labels)
    a, b, c, d, e, f = split_data(features_shuf, labels_shuf)
    train_features, train_labels, val_features, val_labels = deal_with_missing_values(a, b, c, d)
    train_features, train_labels, test_features, test_labels = deal_with_missing_values(a, b, e, f)
    w_train_features = train_features[:, final]
    w_test_features = test_features[:, final]
    predictions = kNN_infer(w_train_features, train_labels, w_test_features, 3)
    accs += np.mean((predictions == test_labels).flatten().astype(np.float32)) / 10
print(accs)
```

Output:

[1 5 2]

0.9497630298137665

That is, the 2nd, 6th, 3rd features are the result of wrapper-type feature selection

10 times validation for test set, accuracy = 94.976%

Question 5

Marks Allocation: Total 15 points, Max available points (include bonus): 18 points
(a): 10 points, Max available points (include bonus): 13 points

- Using numerical libraries for the computation of eigenvalues and eigenvectors will not suffer from mark deduction.
-
- Complete and correct or reasonable coding flow for PCA (10 points)
- Using libraries to perform PCA directly ((a) 15% discount)
- Do the part of PCA from scratch (except the computation of eigenvalues and eigenvectors) ((a) + 30% bonus)

(b): 5 points

- Reasonable explanation (5 points)

Suggested Solution:

```
import numpy as np
import pandas as pd
```

```
"""
Load data
"""
def load_data(data_location):
    data = pd.read_csv(data_location, header=None, index_col=False).values
    features = data[:, 1:-1]
    labels = data[:, -1].reshape(-1, 1)
    return features, labels
```

```
"""
Perform data shuffling
"""
def data_shuffling(features, labels):
    num_data = features.shape[0]
    shuf_arr = np.random.permutation(num_data)
    features_shuf = features[shuf_arr]
    labels_shuf = labels[shuf_arr]
    return features_shuf, labels_shuf
```

```
"""
Split data into training, validation set
"""
def split_data(features_shuf, labels_shuf):
    num_data = features_shuf.shape[0]
    train_end = int(num_data * 0.7)
    train_features = features_shuf[:train_end]
    train_labels = labels_shuf[:train_end]
    val_features = features_shuf[train_end:]
    val_labels = labels_shuf[train_end:]
    return train_features, train_labels, val_features, val_labels
```

```
"""
Deal with the missing values with in-class average
Hardcoded for Breast Cancer Wisconsin (Original) Data Set
"""
def deal_with_missing_values(train_features, train_labels, val_features, val_labels):
    train_labels = train_labels.astype(np.float32)
    b_train_features = train_features[train_labels.flatten() == 2.]
    m_train_features = train_features[train_labels.flatten() == 4.]
    b_train_table = b_train_features != '?'
    m_train_table = m_train_features != '?'
    no_miss_b_train_features = b_train_features[np.all(b_train_table, axis=1)].astype(np.float32)
    no_miss_m_train_features = m_train_features[np.all(m_train_table, axis=1)].astype(np.float32)
    b_means = np.mean(no_miss_b_train_features, axis=0).reshape(1, -1)
    m_means = np.mean(no_miss_m_train_features, axis=0).reshape(1, -1)
    b_train_features[b_train_features == '?'] = 0.
    m_train_features[m_train_features == '?'] = 0.
    b_train_features = b_train_features.astype(np.float32)
    m_train_features = m_train_features.astype(np.float32)
```

```
    val_table = val_features != '?'
    val_features[val_features == '?'] = 0.
    val_features = val_features.astype(np.float32)
    b_train_features = b_train_features + np.multiply(1 - b_train_table.astype(np.float32), b_means)
    m_train_features = m_train_features + np.multiply(1 - m_train_table.astype(np.float32), m_means)
    average_means = (b_means * b_train_features.shape[0] + m_means * m_train_features.shape[0]) / train_features.shape[0]
    val_features = val_features + np.multiply(1 - val_table.astype(np.float32), average_means)
    train_features = np.append(b_train_features, m_train_features, axis=0)
    train_labels = np.append(np.ones((b_train_features.shape[0], 1), dtype=np.float32) * 2., np.ones((m_train_features.shape[0],
1), dtype=np.float32) * 4., axis=0)
    val_labels = val_labels.astype(np.float32)
    return train_features, train_labels, val_features, val_labels
```

```
features, labels = load_data('breast-cancer-wisconsin.data')
features_shuf, labels_shuf = data_shuffling(features, labels)
train_features, train_labels, val_features, val_labels = split_data(features_shuf, labels_shuf)
train_features, train_labels, val_features, val_labels = deal_with_missing_values(train_features, train_labels, val_features,
val_labels)
X = np.append(train_features, val_features, axis=0)
mean = np.mean(X, axis=0).reshape(1, -1)
X_remove_mean = X - mean
A = np.dot(X_remove_mean.T, X_remove_mean) / (X_remove_mean.shape[0] - 1)
eigenvalues, eigenvectors = np.linalg.eig(A)
eigenvectors = eigenvectors.T
index = np.argsort(eigenvalues)
eigenvalues = eigenvalues[index]
eigenvectors = eigenvectors[index]
temp = eigenvalues / np.sum(eigenvalues)
print('eigenvalues: {}'.format(eigenvalues))
print('proportion: {}'.format(temp))
accu = 0.
for i in range(8, -1, -1):
    accu += temp[i]
    if accu > 0.9:
        print('top k = {}, PoV(k) = {}'.format(i + 1, accu))
        break
vecs = eigenvectors[4:]
X_reduced = np.dot(X_remove_mean, vecs.T)
print('X_reduced shape: {}'.format(X_reduced.shape))
```

Output:

eigenvalues: [0.802949 1.5944308 1.7781997 2.4215531 2.739235 3.127751 4.2812605 5.044362 48.629574]
proportion: [0.0114024 0.02264195 0.02525159 0.03438763 0.03889892 0.0444161 0.06079668 0.07163322 0.69057155]
top k = 5, PoV(k) = 0.9063164629042149
X_reduced shape: (699, 5)

Since we use all the data to perform PCA, it should be done when we are really performing prediction. We should only do PCA on the training set and use the computed and selected vectors to perform dimensionality reduction on training set and validation set, as well as testing set.

As usual, we perform 10 times cross validation with shuffling and splitting. Each time, the training set is different and the computed PCA may also be different. This is the reason why we would like to perform several times to investigate how the average performance is.