Artificial Intelligence Project Report: Gene Data Mining

Weichen Li
5120309662
Department of Computer Science and Engineering
Shanghai Jiao Tong University
eizo.lee@sjtu.edu.cn

Introduction

This is the project report of the course Artificial Intelligence. In this project, students are free to choose their topics and data to study areas relevant to machine learning. I choose the data 'GeneChip', whose data is larger than 2.0 G. In the directory of 'GeneChip', there is a file called 'E-TABM-185.rawdata.txt', from which I can obtain gene data from 5986 samples and 22283 genes. And there is another file called 'E-TABM-185_sdrf.txt', which contains information for each sample, including sample source, material type, characteristics, etc. I choose the 'Characteristics[DiseaseState]' to research.

Then I use PCA to decompose the dimension of data from 22283 to 10, 15, 30, 40, 50 and 100. Then I divide the data of new dimension into training sets and testing sets. In each dimension, I use k-NN algorithm to train the model and predict the label ('Characteristics[DiseaseState]'). The result is listed below.

PCA

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. The number of principal components is less than or equal to the number of original variables. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. The resulting vectors are an uncorrelated orthogonal basis set. The principal components are orthogonal because they are the eigenvectors of the covariance matrix, which is symmetric. PCA is sensitive to the relative scaling of the original variables.

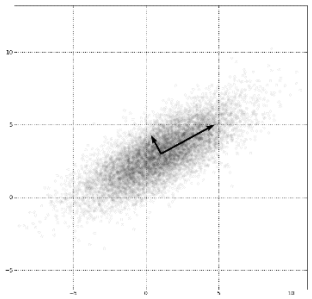


Figure 1 PCA

k-NN

In pattern recognition, the k-Nearest Neighbors algorithm (or k-NN for short) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression:

1. In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.
2. In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning algorithms.

The Nearest Neighbors algorithm is based on the idea that samples with similar-valued features tend to have the same label. To predict a label, the algorithm looks at the k training samples closest to the test sample, and predicts the most frequent label of the k samples. Distance is determined by the samples position in the feature space, an n-dimensional space with each dimension corresponding to a feature. When k is specified greater than or equal to the number of training samples, Nearest Neighbors is identical to Majority, and so useless.

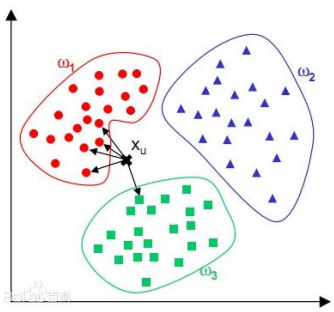


Figure 2 k-NN algorithm

Running Environment

Operating System: Ubuntu 14.04 LTS
Programming Language: Python 2.7.9
Library: SKlearn

Process of modeling and predicting

Here I will only talk about the idea and abstraction of this project, instead of going to details on the code level.

Preprocess

First of all, after the preprocess of the data, I obtain a large matrix S whose size is 5986*22283. We can call K the feature matrix, for it has 5986 samples, each of which has 22283 features. (I have to transpose the original matrix in the raw data file, so that I can obtain the matrix S)

Then I get the labels of these samples from the file mentioned above. There are 5986 labels totally, each label corresponds to a sample. Thus, I store the labels in a vector L. In the raw file, the label is a string, I transfer it into a number. Since there are totally 194 labels (one is unknown, one is normal, the other 192 are symptoms), there are 194 corresponding numbers, from 0 to 193.

Decomposition

Next, we must decompose the dimension of the features, since 22283 is too large for us to calculate and model.

Through the python library called ‘sklearn’, we can conveniently call the function of PCA to decompose the dimension.

I call PCA for 6 times, and each time I decompose the dimension into 10, 15, 30, 40, 50 or 100 dimensions, respectively.

Thus, at the end of the decomposition, I got 6 versions of matrix S.

Data Division

Then, for each of the 6 versions of matrix S, I split S and L into training set and testing set, using a random process to select 3/5 of the data into training set, and 2/5 into testing set.

K-NN Algorithm Modeling

After the division of training set and testing set, I use k-NN algorithm to train the training set. k is ranged from 1 to 50. And then we use the model to predict the testing set. At the same time, record the predicting result and calculate the accuracy rate (A) for each k.

$$\mathrm{A} = \frac{\mathrm{b}}{\mathrm{n}}$$

b is the number of correct predictions, n is the total number of labels of testing data.

Correct Rate

I introduce a measurement called 'Correct Rate' (C), to record the relative rate of correct prediction for each label.

$$C_x = \frac{b_x}{n_x}$$

b_x is the number of correct predictions on label x, n is the total number of testing samples whose label is x.

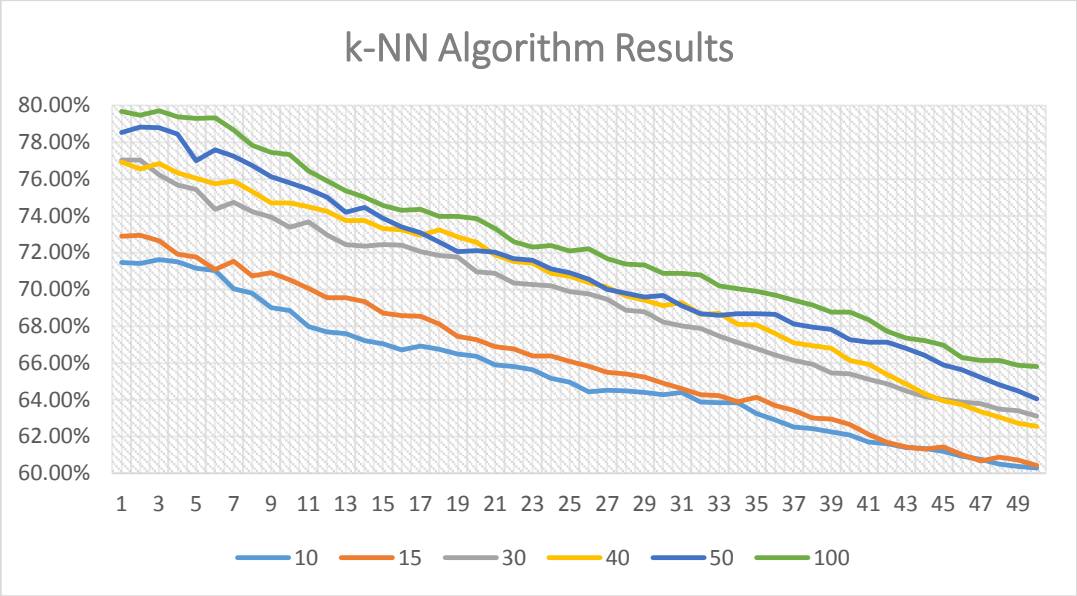
Result

Illness Index

The illness index is in the appendix, it links labels and numbers.

k-NN Results

Here are the results of k-NN algorithm with different dimensions and k's. Each line represents a dimension. And x axis is the range of k, y axis is the accuracy rate.

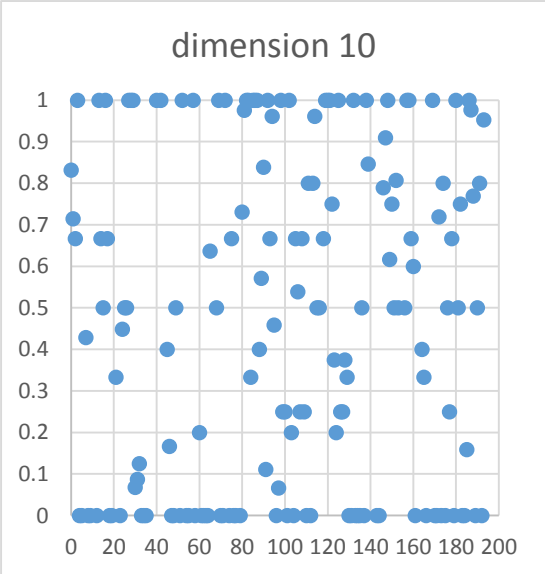


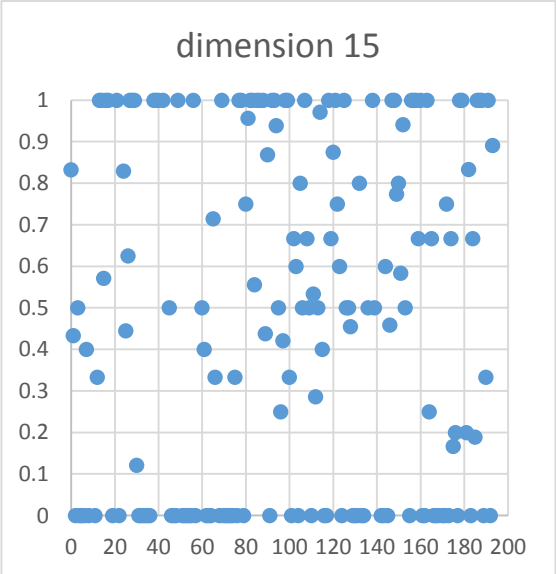
As we can see, with the increasing of the dimension, the accuracy rate grows. On the other hand, when k=1, we can see that the accuracy rate is relatively high. The highest accuracy rate I get is 79.72%, the corresponding dimension is 100, and k is 3.

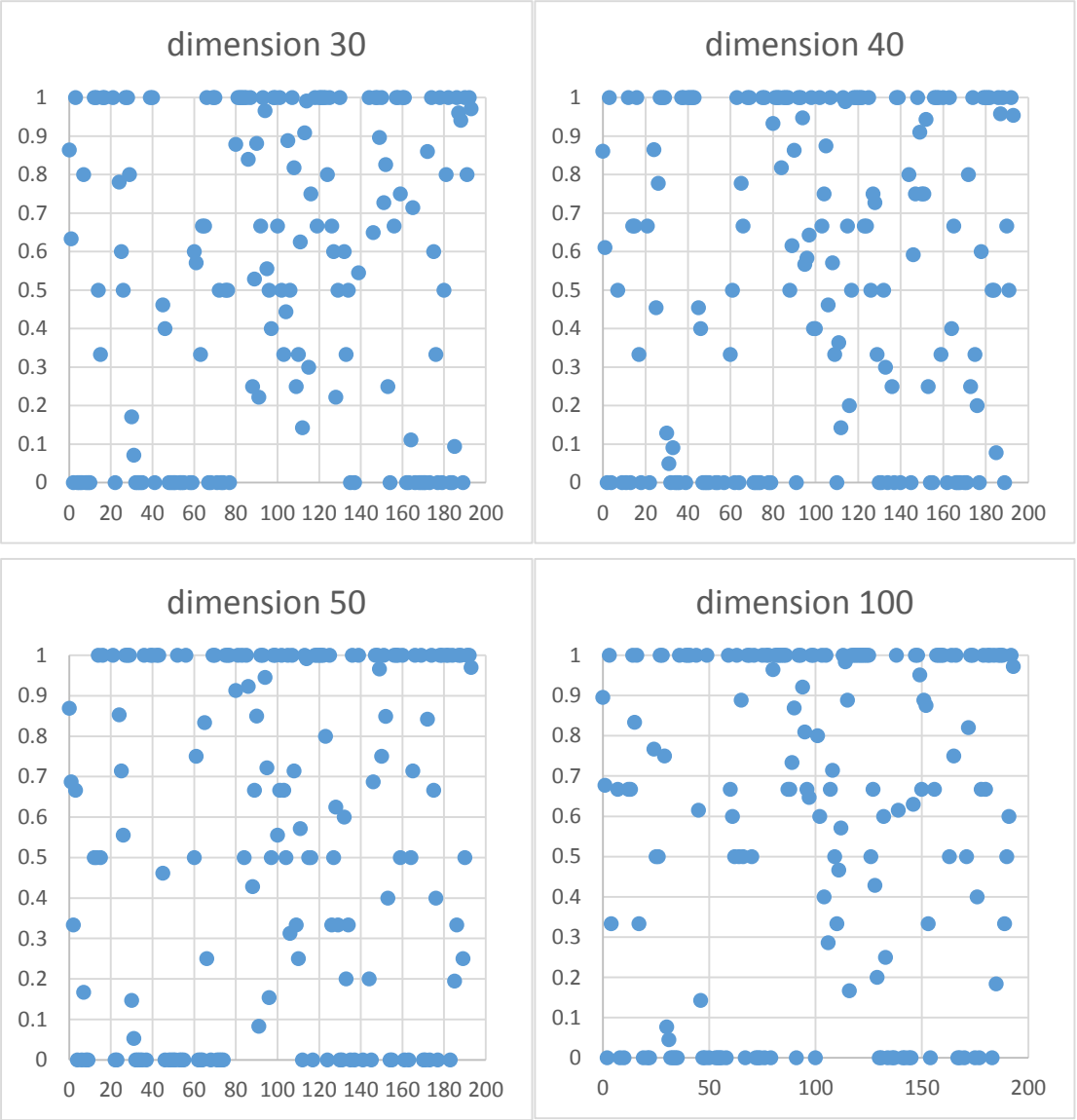
Due to the limited time and resources, I do not get enough time to calculate the accuracy rate in higher dimension, such as 200, 300, etc. I believe there will be a better dimension X beyond 100, X should have the climax value of accuracy rate. From 100 to X, the average accuracy rate should be increasing. However, once beyond X, the average accuracy rate should drop, since there may be too many dimensions, which will enlarge the error rate of prediction.

Correct Rate

Here are the results of correct rates shown below. The y axis is the correct rate, the x axis is the label (each label corresponds to a symptom, 0 is unknown, 1 is normal).







As we can see, the distribution of the correct rate is relatively even. There are labels have a correct rate with 1, which means that the corresponding symptoms are easy to diagnose from the features. On the other hand, there are many labels have a correct rate with 0, which means that the corresponding symptoms are hard to diagnose, maybe we should use other techniques to predict these symptoms. Finally, a number of labels have a correct rate between 0 and 1, this corresponds to reality, for we are not always 100% sure that some symptoms are predicted correctly.

With the increasing of the dimension, we can see that average correct rate is increasing as well. It is reasonable to believe that there is a best dimension Y beyond 100, Y should have the climax value of average correct rate. (Just similar to the dimension X in accuracy rate)

Conclusion

From the results of accuracy rates and correct rates, we can see that the performance of the PCA and k-NN is relatively good. I believe there is still space for

improvements and enhancements.

Although PCA is good in dimension reduction, it blurs the link between true features and the label. For example, if we want to research which gene (or genes) is responsible for a certain illness, we cannot simply use PCA to perform a dimension reduction, we should use other learning methods.

As for k-NN, it has some strong consistency results. As the amount of data approaches infinity, the algorithm is guaranteed to yield an error rate no worse than twice the Bayes error rate (the minimum achievable error rate given the distribution of the data). k-NN is guaranteed to approach the Bayes error rate for some value of k (where k increases as a function of the number of data points). Various improvements to k-NN are possible by using proximity graphs.

Reference

1. k-nearest neighbors algorithm https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
2. Principal component analysis https://en.wikipedia.org/wiki/Principal_component_analysis
3. 邻近算法 http://baike.baidu.com/link?url=G8_9IKjaEWfVzFtPk4pnnJFt4fH3gnKnukiraGe6p8wC43P0rGM1lmIg5qkxMMKMftsiewQAr2PDqwTuXIOnbooxKFqQ8SeJj9huh_fCHYpXCi1DsUdlVdMp__qfsCzL9nVXHIbMhywiIXfisQrPHa
4. Machine learning applications in genetics and genomics *Maxwell W. Libbrecht, William Stafford Noble* *Nature Reviews Genetics* 16, 321–332 (2015)
5. The Elements of Statistical Learning: Data Mining, Inference, and Prediction *Trevor Hastie, Robert Tibshirani, Jerome Friedman* Springer
6. Pattern Recognition and Machine Learning *Christopher M. Bishop* Springer
7. Using machine learning to design and interpret gene-expression microarrays *Michael Molla, Michael Waddell, David Page, Jude Shavlik* AI Magazine
8. Machine learning algorithms for cancer diagnosis *Abraham Karplus* Santa Cruz County Science Fair 2012

Appendix

Illness index

0	unknown	71	well-differentiated liposarcoma	142	pituitary adenoma, GH-secreting
1	normal	72	schwannoma	143	pituitary adenoma, PRL-secreting

2	primary hyperparathyroidism	73	malignant peripheral nerve sheath tumor	144	atopic mild asthmatics
3	Down syndrome, transient myleoproliferative disorder	74	sarcoma	145	non-atopic mild asthma
4	Down syndrome, acute megakaryoblastic leukaemia	75	chondroblastoma	146	atopic severe asthma
5	Down syndrome, non-leukaemic	76	chordoma	147	cardiovascular disease, obesity
6	ganglioneuroma	77	chondromyxoid fibroma	148	lung cancer
7	neuroblastoma-poorly differentiated	78	no tendon xanthomas	149	breast cancer
8	ganglioneuroblastoma	79	tendon xanthomas	150	atrial fibrillation
9	neuroblastoma-differentiating	80	prostate cancer	151	squamous cell carcinoma
10	neuroblastoma-undifferentiated	81	precursor T lymphoblastic leukemia	152	acute lymphoblastic leukemia, chemotherapy response
11	ganglioneuroblastoma intermixed	82	Burkitt's lymphoma	153	pulmonary disease, cystic fibrosis
12	CVID	83	breast carcinoma	154	uterated prostate cancer
13	XLA	84	chronic myelogenous leukemia	155	ulcerative colitis
14	follicular thyroid adenoma	85	T cell acute lymphoblastic leukemia	156	irritable bowel syndrome
15	follicular thyroid carcinoma	86	chronic myeloid leukemia	157	glioblastoma
16	colon carcinoma	87	low-stage neuroblastoma	158	colon cancer
17	AIDS-KS, HIV+, nodular (late) stage	88	high-stage neuroblastoma	159	pterygium
18	Classic-KS, HIV-, nodular (late) stage	89	B-cell lymphoma	160	periodontitis
19	AIDS-KS, KSHV-	90	B-cell lymphoma, dlbcl	161	brain tumor, glioblastoma
20	iatrogenic-KS, KSHV-	91	B-cell lymphoma, nhl	162	hlrcc
21	KSHV infection, 2 days	92	cystic fibrosis	163	adenovirus expressing GFP
22	KSHV infection, 7 days	93	meningitis infected	164	carcinoma in situ, bladder tumor
23	KSHV infection, 14 days	94	breast tumor	165	bladder tumor

24	acute lymphoblastic leukemia	95	breast tumor, luminal	166	t3b, bladder tumor
25	embryonal rhabdomyosarcoma	96	breast tumor, basal	167	t4b
26	alveolar rhabdomyosarcoma	97	breast tumor, normal like	168	t4b, bladder tumor
27	acute promyelocytic leukemia	98	response	169	t3a, bladder tumor
28	healthy	99	promyelocytic leukemia	170	t4a, bladder tumor
29	colon adenocarcinoma	100	pcos	171	carcinoma in situ
30	control brain	101	renal clear cell carcinoma	172	germ cell tumor
31	Huntington's Disease (HD) Pathological Grade 2	102	dermatomyositis	173	becker muscular dystrophy
32	Huntington's Disease (HD) Pathological Grade 3	103	duchenne muscular dystrophy	174	amyotrophic lateral sclerosis
33	Huntington's Disease (HD) Pathological Grade 1	104	facioscapulohumeral muscular dystrophy	175	dysferlinopathy
34	Huntington's Disease (HD) Pathological Grade 4	105	juvenile dermatomyositis	176	calpainopathy
35	Huntington's Disease (HD) Pathological Grade 0	106	bipolar disorder	177	Emery-Dreifuss muscular dystrophy
36	promyelocytic leukemia HL-60	107	ebv infection	178	limb-girdle muscular dystrophy type 2I
37	chronic myelogenous leukemia K562	108	trauma	179	acute quadriplegic myopathy
38	lymphoblastic leukemia MOLT-4	109	Ischemia	180	hereditary spastic paraplegia
39	Daudi Burkitt's lymphoma	110	myocardial infarction	181	cockayne syndrome
40	colorectal adenocarcinoma	111	current smoker	182	oral squamous cell carcinoma
41	Raji Burkitt's lymphoma	112	former smoker	183	cardiomyopathy, calcifications
42	congestive heart failure	113	metabolic syndrome	184	cardiomyopathy
43	sudden death	114	acute myeloid leukemia	185	Huntington's disease
44	intracranial hemorrhage NOS	115	alzheimer's disease	186	progeria syndrome
45	lung adenocarcinoma (NCI_Thesaurus C0152013) has DiseaseStaging Stage I	116	breast cancer cells, adenovirus expressing GFP	187	bone marrow relapse

46	lung adenocarcinoma (NCI_Thesaurus C0152013) has DiseaseStaging Stage III	117	UV treated	188	cololrectal tumor
47	lung adenocarcinoma (NCI_Thesaurus C0152013) has DiseaseStaging Stage IV	118	colorectal cancer	189	lung cancer, cytotoxicity
48	lung adenocarcinoma (NCI_Thesaurus C0152013) has DiseaseStaging Stage II	119	barrett's esophagus	190	squamous cell cancer
49	seasonal allergy	120	mitochondrial disorder	191	small cell cancer
50	2h after infection with fasX-mutant Streptococcus pyogenes	121	breast cancer, inflammatory	192	lymphoma
51	control sample without infection after 2h	122	T-cell lymphoblastic lymphoma, T-LL	193	brain tumor
52	2h after infection with wildtype Streptococcus pyogenes	123	T-cell acute lymphoblastic leukemia, T-ALL		
53	4h after infection with fasX-mutant Streptococcus pyogenes	124	B-cell acute lymphoblastic leukemia, B-ALL		
54	4h after infection with wildtype Streptococcus pyogenes	125	emphysema		
55	6h after infection with fasX-mutant Streptococcus pyogenes	126	grade 2, primary hnscc		
56	6h after infection with wildtype Streptococcus pyogenes	127	acute promyelocytic leukemia, apl		
57	8h after infection with fasX-mutant Streptococcus pyogenes	128	aml		
58	control sample without infection after 8h	129	acute monoblastic/monocytic leukemia		
59	8h after infection with wildtype Streptococcus pyogenes	130	acute myelomonocytic leukemia		
60	adenocarcinoma	131	acute erythroid leukemia		
61	monophasic synovial sarcoma	132	ischemic cardiomyopathy		
62	leiomyosarcoma	133	nonischemic		

			cardiomyopathy		
63	Ewing's Sarcoma	134	acute rejection		
64	chondrosarcoma	135	inflammatory myopathy		
65	osteosarcoma	136	myositis		
66	myxoid liposarcoma	137	polymyositis		
67	lipoma	138	response, dominant negative		
68	neurofibroma	139	uterine fibroid		
69	fibromatosis	140	pituitary adenoma, non-functioning		
70	dedifferentiated chondrosarcoma	141	pituitary adenoma, ACTH-secreting		

Source code (AI.py)

(to run the source code, the file location should be modified)

```
import numpy as np
import numpy as num
import random
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier

def transpose_raw():

    temp_1 = []

    with open('/home/blade/Desktop/AI project/E-TABM-185.rawdata.txt', 'r') as gene_sample:
        for line in gene_sample.readlines():
            temp = line.strip('\n').split('\t')
            temp_1.append(temp)


    with open('/home/blade/Desktop/AI project/transpose.txt', 'w') as training:

        # 0-sign; 1~5896-tag
        # y is 5897
        y = len(temp_1)

        # x is 22284
        x = len(temp_1[0])

        countX = 0
        countY = 0

        for j in range(1, x):
            countX = 0
            for i in range(1, y):
                training.write(temp_1[i][j])
                countX += 1
                if i != y - 1:
                    training.write('\t')
            training.write('\n')
            countY += 1
            if j % 500 == 0:
                print('Transposing: %.2f%%' %(j*100/float(5900)))
```

```
        print('countX:%d countY:%d' %(countX,countY))

def filt_illness():

    temp_1 = []
    temp_2 = {}
    index = 2

    # 0-sign; 1~5896-tag
    max = 5897

    with open('/home/blade/Desktop/AI project/E-TABM-185_sdrf.txt', 'r') as gene_sample:
        i = 1
        for line in gene_sample.readlines():
            temp_1.append(line.strip('\n').split('\t'))
            i+=1
            if i > max:
                break


    with open('/home/blade/Desktop/AI project/filt_illness.txt', 'w') as filt:

        x = len(temp_1[0])

        illness = 0

        for j in range(0, x):
            if (cmp(temp_1[0][j], 'Characteristics[DiseaseState]') == 0):
                illness = j

        count = 0

        for i in range(1, max):
            k = temp_1[i][illness]
            if k == '':
                k = 'unknown'
            if not (k in temp_2):
                if cmp(k, 'unknown') == 0:
                    temp_2[k] = 0
                elif cmp(k, 'normal') == 0:
                    temp_2[k] = 1
                else:
                    temp_2[k] = index
                    index += 1
            filt.write('%d\n' %temp_2[k])
            count += 1

        print('count:%d' %(count))

    with open('/home/blade/Desktop/AI project/illness_index.txt', 'w') as illness_index:

        tmp_list = []

        for item in temp_2:
            tmp_list.append([temp_2[item], item])

        tmp_list.sort(key=lambda x:x[0])

        for item in tmp_list:
            illness_index.write('%d\t%s\n' %(item[0], item[1]))

def integrate_features():

    temp_1 = []

    count = 0
```

```
    with open('/home/blade/Desktop/AI project/transpose.txt', 'r') as all_feature:
        for feature in all_feature.readlines():
            count += 1
            temp_1.append(feature.strip('\n').split('\t'))

            if count % 200 == 0:
                print('Reading: %.2f%%' %(count*100/float(5896)))

    num = 0

    #temp_1: 5896 rows, 22283 columns

    with open('/home/blade/Desktop/AI project/training_testing_features.txt', 'w') as training_testing:

        len_1 = 5896
        len_2 = 22283

        while num < len_1:

            if num % 300 == 0:
                print('Writing: %.2f%%'%(num*100/float(len_1)))

            for index in range(0, len_2):
                training_testing.write(temp_1[num][index])
                if index != len_2 - 1:
                    training_testing.write('\t')
                else:
                    training_testing.write('\n')

            num += 1

def PCA_decomposition(dim):

    with open('/home/blade/Desktop/AI project/training_testing_features.txt', 'r') as training_testing:

        training_testing_features = []
        num = 0
        for line in training_testing.readlines():

            if num % 300 == 0:
                print('Reading: %.2f%%' %(num*100/float(5896)))

            num += 1

            training_testing_features.append(line.strip('\n').split('\t'))

    pca = PCA(n_components = dim)
    print 'Reading Over, len:', len(training_testing_features)
    new_training_testing_features = pca.fit_transform(training_testing_features)

    print new_training_testing_features

    str_temp = '/home/blade/Desktop/AI project/new_training_testing_features_%ddim.txt' %dim

    np.savetxt(str_temp, new_training_testing_features, delimiter="\t", fmt="%s")

def divide_data(dim_1):

    temp_1 = []
    temp_2 = []

    count = 0

    str_temp = '/home/blade/Desktop/AI
```

```
project/new_training_testing_features_%ddim.txt' %dim_1

    with open(str_temp, 'r') as all_feature:

        for feature in all_feature.readlines():
            count += 1
            temp_1.append(feature.strip('\n').split('\t'))

            #if count % 200 == 0:
                #print('Reading features: %.2f%%' %(count*100/float(5896)))


    count = 0

    with open('/home/blade/Desktop/AI project/filt_illness.txt', 'r') as all_label:

        for label in all_label.readlines():
            count += 1
            temp_2.append(label.strip('\n'))

            #if count % 200 == 0:
                #print('Reading labels: %.2f%%' %(count*100/float(5896)))

        print 'count is:', count


    num = 0


    #temp_1: 5896 rows, 15 columns
    #temp_2: 5896 rows, 1 columns

    with open('/home/blade/Desktop/AI project/training_feature.txt', 'w') as training:
        with open('/home/blade/Desktop/AI project/testing_feature.txt', 'w') as testing:
            with open('/home/blade/Desktop/AI project/training_label.txt', 'w') as
training_label:
                with open('/home/blade/Desktop/AI project/testing_label.txt', 'w') as
testing_label:
                    len_1 = 5896
                    len_2 = dim_1

                    while num < len_1:

                        ran_result = random.randint(0,4)

                        #if num % 300 == 0:
                            #print('Writing: %.2f%%'%(num*100/float(len_1)))

                        if ran_result >= 0 and ran_result <= 2:
                            training_label.write(temp_2[num])
                            training_label.write('\n')

                            for index in range(0, len_2):
                                training.write(temp_1[num][index])
                                if index != len_2 - 1:
                                    training.write('\t')
                                else:
                                    training.write('\n')
                        else:
                            testing_label.write(temp_2[num])
                            testing_label.write('\n')

                            for index in range(0, len_2):
                                testing.write(temp_1[num][index])
                                if index != len_2 - 1:
                                    testing.write('\t')
                                else:
```

```
                                        testing.write('\n')

                                num += 1

def error_check(dim, est_y, testing_y):

    testing_y_count = {}

    for item in testing_y:
        if item in testing_y_count:
            testing_y_count[item] += 1
        else:
            testing_y_count[item] = 1

    est_right ={}

    for index in range(0, len(est_y)):

        if est_y[index] == testing_y[index]:
            if testing_y[index] in est_right:
                est_right[testing_y[index]] += 1
            else:
                est_right[testing_y[index]] = 1
        else:
            if not (testing_y[index] in est_right):
                est_right[testing_y[index]] = 0


    check_list = []

    for item in testing_y_count:
        if item in est_right:
            check_list.append([item,          est_right[item]/float(testing_y_count[item]),
testing_y_count[item]])
        else:
            check_list.append([item, 0.0, testing_y_count[item]])

    check_list.sort(key=lambda x:x[1])

    with open('/home/blade/Desktop/AI project/est_rate_%ddim.txt' %dim, 'w') as est_rate:
        for item in check_list:
            est_rate.write('%s\t%.4f\t%d\n' %(item[0], item[1], item[2]))

def kNN(neigh_num, dim):

    X = []
    y = []

    Testing_X = []
    Testing_est = []
    Testing_y = []

    count = 0
    with open('/home/blade/Desktop/AI project/training_feature.txt', 'r') as training_feature:

        for feature in training_feature.readlines():
            X.append(feature.strip('\n').split('\t'))
            count += 1

        #print 'Training Features:', count


    count = 0
    with open('/home/blade/Desktop/AI project/training_label.txt', 'r') as training_label:

        for label in training_label.readlines():
            y.append(label.strip('\n'))
```

```
                count += 1

        #print 'Training Labels:', count


    count = 0
    with open('/home/blade/Desktop/AI project/testing_feature.txt', 'r') as testing_feature:

        for feature in testing_feature.readlines():
            Testing_X.append(feature.strip('\n').split('\t'))
            count += 1

        #print 'Testing Features:', count


    count = 0
    with open('/home/blade/Desktop/AI project/testing_label.txt', 'r') as testing_label:

        for label in testing_label.readlines():
            Testing_y.append(label.strip('\n'))
            count += 1

        #print 'Testing Labels:', count

    knn = KNeighborsClassifier(n_neighbors = neigh_num)
    knn.fit(X, y)

    Testing_est = knn.predict(Testing_X)

    sum = 0

    for index in range(0, len(Testing_est)):
        if Testing_est[index] == Testing_y[index]:
            sum += 1

    if (neigh_num == 1):
        error_check(dim, Testing_est, Testing_y)

    #print('Neigh Num: %d \t Prediction Rate: %.3f%%' %(neigh_num,
sum*100/float(len(Testing_est))))

    return [neigh_num, sum/float(len(Testing_est))]

def kNN_combine(dim):

    divide_data(dim)

    result = []

    for num in range(1, 51):

        result.append(kNN(num, dim))

    return result

def main():
    transpose_raw()
    filt_illness()

    integrate_features()

    dimen = [10, 15, 30, 40, 50, 100]

    for num in dimen:

        PCA_decomposition(num)
```

```
        #dim = 10, 15, 30, 40, 50, 100

        for num in dimen:

                str_temp = '/home/blade/Desktop/AI project/knn_result(dim=%d).txt' %num

                result = kNN_combine(num)

                with open(str_temp, 'w') as output:
                        for index in result:
                                str_temp_2 = '%d\t%.5f\n' %(index[0], index[1])
                                output.write(str_temp_2)


if __name__ == '__main__':
        main()
```