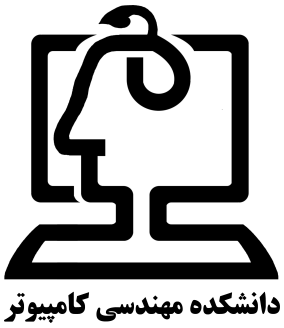
به نام خدا



دانشگاه صنعتی امیرکبیر
دانشکده مهندسی کامپیوتر

استاد درس: دکتر امین ناظرفرد

پاییز ۱۳۹۹

درس یادگیری ماشین

سروش مهدی
شماره دانشجویی: ۹۹۱۳۱۰۵۰

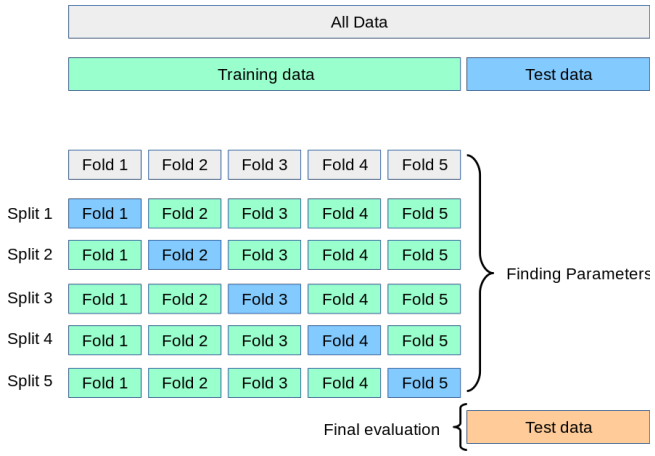
سری ۲ تمرین‌ها

فهرست مطالب

۱ سوالات تشریحی

۱.۱

در حالت تئوری اگر بی نهایت داده داشته باشیم هر چه K بزرگ تر باشد بهتر است اما در عمل تعداد داده ها محدود است. یکی از راه های پیدا کردن بهترین K استفاده از تکنیک Cross-validation میباشد. نرم افزار weka نیز از این روش استفاده میکند.



شکل ۱: cross k-fold validation (kevinzakka.github.io)

در این روش ما داده ها را به k بخش تقریبا هم اندازه تقسیم میکنیم که این k از پارامتری که به دنبال یافتن ان هستیم متفاوت است. فرض کنیم در اینجا k را برابر ۱۰ در نظر میگیریم. سپس ده مرحله این کار را تکرار میکنیم: در هر مرحله یک گروه از داده ها را کنار میگذاریم و الگوریتم را روی بقیه داده ها برازش میکنیم و میزان خطا را بدست می اوریم و در اخر میانگین این خطا ها را حساب میکنیم.
این کار را به ازای K های مختلف در الگوریتم KNN انجام میدهیم و پارامتر با کمترین خطا را انتخاب میکنیم. باید دقت کنیم در روش Cross-validation الگوریتم تا مرحله تست نباید داده های تست را ببیند.

۲.۱

۱.۲.۱

بهترین مقدار K در این حالت ۵ میباشد. که دقت الگوریتم به ازای ان برابر $\frac{10}{14}$ میباشد.

۲.۲.۱

اگر این پارامتر بیش از حد کوچک باشد الگوریتم به داده های نویز حساس میشود و دچار واریانس بالا و بیش برازش میشود . مثلا اگر این پارامتر را یک انتخاب کنیم برای داده های مثبت پایین صفحه در شکل الگوریتم سه داده وسط که متعلق به کلاس مثبت هستند را درست دسته بندی نمیکند. اما اگر این پارامتر بیش از حد

بزرگ باشد مدل دچار بایاس بالا و کم برازش میشود و مثلا اگر این پارامتر را ۹ انتخاب کنیم داده مثبت وسط در داده های پایین را به اشتباه متعلق به کلاس منفی پیش بینی میکند.

۳.۲.۱

کلاس مثبت

۳.۱

هر دو الگوریتم غیر پارامتریک هستند. یعنی پارامتری برای یادگیری از روی دیتا ندارند. الگوریتم های پارامتریک یک فرض کلی در مورد فرم داده ها دارند و میخواهند پارامتر های این فرض را از روی داده ها یادگیری کنند که تعداد این پارامتر ها از تعداد داده ها مستقل است اما الگوریتم های غیر پارامتریک بدون داشتن این فرض ازاد هستند هر گونه فرمی را از داده ها یاد بگیرند.

۴.۱

هر دو مدل `Discriminative` هستند. به این معنی که سعی میکنند که تفاوت بین کلاس ها را یاد بگیرند و نه مشخصات هر کلاس. این مدل ها به دنبال پیدا کردن مرز بین کلاس ها هستند اما مدل های `Generative` به دنبال یاد گرفتن توزیع هر کلاس میباشند.

۵.۱

به الگوریتم هایی که در مرحله اموزش کار زیادی نمیکنند و بیشتر کار را در مرحله تست انجام میدهند الگوریتم های تنبل میگویند . الگوریتم `KNN` نیز یک الگوریتم تنبل است زیر در مرحله اموزش تقریبا کاری انجام نمیدهد و کار اصلی را در بخش تست انجام میدهد.

۶.۱

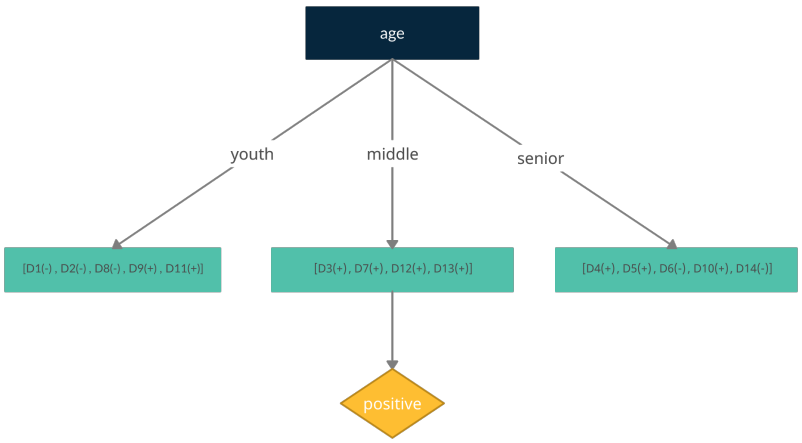
درخت تصمیم در صورت وجود داده های نویزی یا داده های ناکافی دچار بیش برازش میشود. با استفاده از هرس میتوانیم جلوی بیش برازش درخت تصمیم را بگیریم. این هرس را میتوانیم هم در حین اجرای الگوریتم انجام دهیم و با گذاشتن شرط هایی مانع از پیچیده شدن بیش از حد درخت شویم و هم میتوانیم ابتدا الگوریتم را کامل اجرا کرده و سپس درخت به دست امده را هرس کنیم به این ترتیب که برخی از زیر درخت ها را با یک برگ جایگزین کنیم. تمامی این مراحل در مرحله اموزش انجام میشود و برای هرس میتوانیم از مجموعه `Validation` استفاده کنیم.

۷.۱

۱.۷.۱

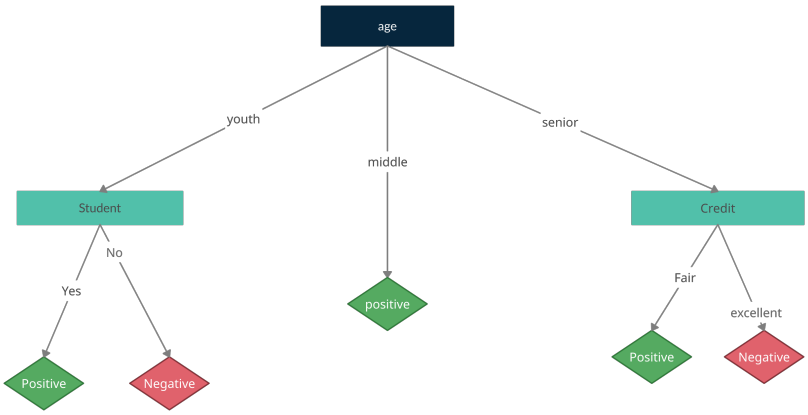
$\text{Gain(S , age)} = 0.246$
$\text{Gain(S , student)} = 0.151$
$\text{Gain(S , credit)} = 0.047$
$\text{Gain(S , income)} = 0.028$

همانطور که مشاهده میشود Gain برای ویژگی سن از همه بیشتر است پس این ویژگی را به عنوان ریشه درخت انتخاب میکنیم که درخت به صورت زیر میشود: سپس در زیردرخت های جدید Gain را محاسبه



شکل ۲: ویژگی سن در ریشه قرار میگیرد

میکنیم در زیر درخت سمت چپ ویژگی دانشجو بودن بیشترین مقدار Gain را دارد و در زیردرخت سمت راست نیز ویژگی Credit بیشترین مقدار Gain را دارد که بعد از در نظر گرفتن این ویژگی ها درخت به صورت زیر میشود. میبینیم که در عمق دو داده ها به خوبی از یکدیگر جدا میشوند.



شکل ۳: درخت تصمیم

۲.۷.۱

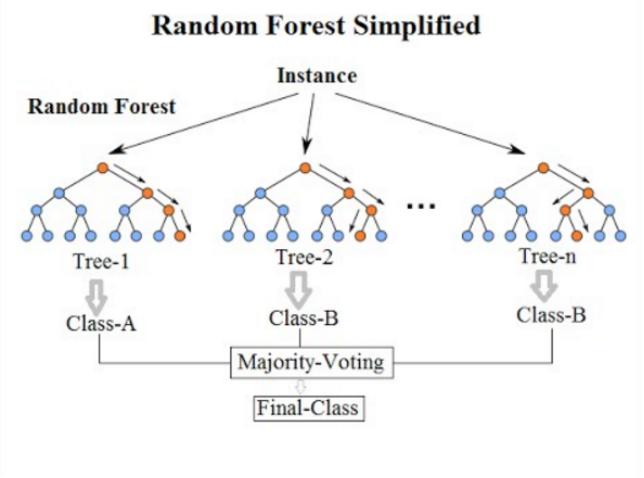
در داده اول چون ویژگی سن مقدار جوان را دارد در درخت تصمیم به زیردرخت چپ میرویم سپس چون ویژگی دانشجو بودن مقدار بله را دارد داده به عنوان کلاس مثبت دسته بندی میشود.
در داده دوم چون سن مقدار سالمند را دارد به زیردرخت سمت راست میرویم و چون ویژگی اعتبار مقدار عالی را دارد داده به عنوان کلاس منفی دسته بندی میشود.
در داده سوم چون ویژگی سن برابر میانسال است داده به عنوان کلاس مثبت دسته بندی میشود.

۸.۱

در این مورد اگر از درخت تصمیم استفاده کنیم هر چه عمق بیشتر باشد دقت بیشتر میشود اما اگر عمق محدود باشد درخت تصمیم چون مرز هایش موازی محور های اصلی هستند نمیتواند داده ها را در این مورد به خوبی جدا کند. مگر اینکه مرز جدا کننده دو کلاس موازی یکی ار محور های اصلی باشد .

۹.۱

این الگوریتم از تعدادی درخت تصمیم استفاده میکند به این صورت که به هر درخت به صورت تصادفی تعدادی داده از داده های اموزش و تعدادی از ویژگی ها را میدهد و هر درخت تصمیم روی این داده ها و ویژگی ها اموزش داده میشود و سپس با استفاده از نتایج این درخت های تصمیم تصمیم نهایی را به این صورت میگیرد که کلاسی را که تعداد درخت بیشتری ان را پیش بینی کرده باشند به عنوان کلاس نهایی انتخاب میکند. همچنین



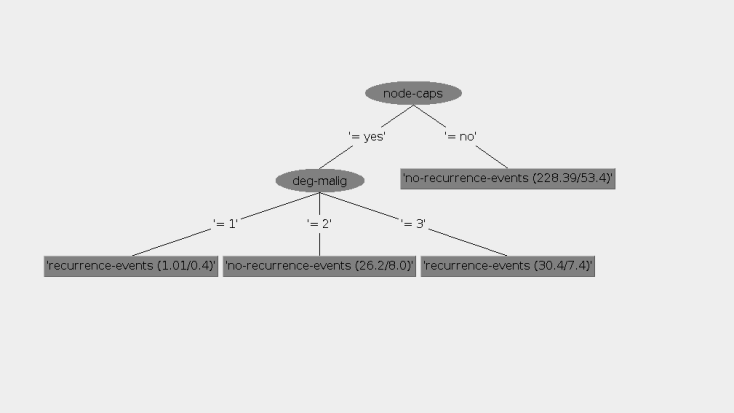
شکل ۴: جنگل تصادفی (medium.com)

این الگوریتم با محدود نگه داشتن عمق کمی بایاس را افزایش میدهد اما در کل عملکرد مدل را افزایش میدهد و از بیش برازش جلوگیری میکند.

Weka ۲

۱.۲

شکل درخت تصمیم و ماتریس در هم ریختگی و مقادیر خواسته شده در زیر امده اند:



شکل ۵: درخت تصمیم

actual\predicted	no-recurrence-events	recurrence-events
no-recurrence-events	46	8
recurrence-events	23	9

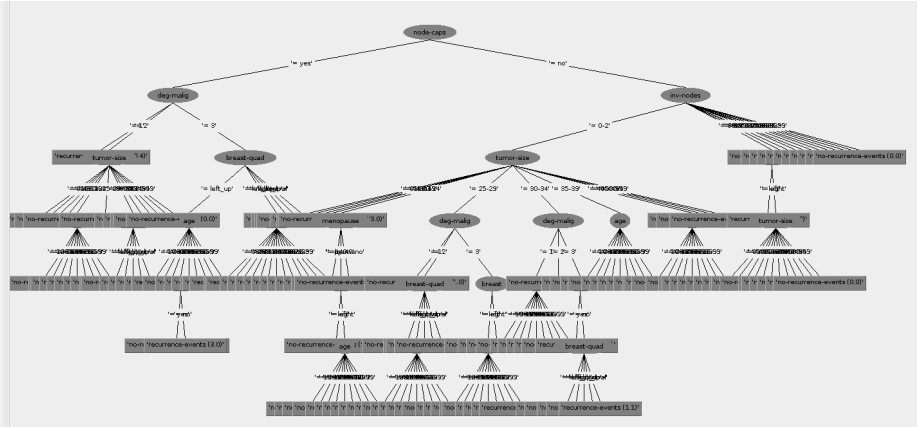
Confusion Matrix

Class	TP	FP	FN	Precision	Recall	F1measure
no-recurrence-events	46	23	8	0.667	0.852	0.748
recurrence-events	9	8	23	0.529	0.281	0.367
weighted Avg	-	-	-	0.616	0.64	0.606

میبینیم که دسته بندی کننده ۵۵ داده را درست و ۳۱ داده را غلط دسته بندی کرده است. همچنین این دسته بندی کننده فقط از دو ویژگی برای دسته بندی استفاده کرده. دسته بندی کننده در مورد کلاس اول با توجه به پارامتر `F1measure` عملکرد بهتری دارد.

۲.۲

این متغیر ویژگی هرس کردن درخت را کنترل میکند. اگر این متغیر را فعال کنیم درخت هرس نمیشود که این کار موجب بیش برازش مدل میشود. شکل درخت تصمیم و ماتریس در هم ریختگی و مقادیر خواسته شده در زیر امده اند:



شکل ۶: درخت تصمیم هرس نشده

actual\predicted	no-recurrence-events	recurrence-events
no-recurrence-events	41	13
recurrence-events	20	12

Confusion Matrix

Class	TP	FP	FN	Precision	Recall	F1measure
no-recurrence-events	41	20	13	0.672	0.759	0.713
recurrence-events	12	13	20	0.48	0.375	0.421
weighted Avg	-	-	-	0.601	0.616	0.604

میبینیم که مدل بسیار پیچیده شده و میتوان گفت دچار بیش برازش شده است و همچنین میبینیم با توجه به معیار `F1measure` و میانگین ان عملکرد مدل روی داده های تست ضعیف تر شده. در این حالت دسته بندی کننده ۵۳ داده را درست و ۳۳ داده را غلط پیشبینی کرده است.

۳.۲

در حالت بدون هرس مقادیر به صورت زیر است:

actual\predicted	no-recurrence-events	recurrence-events
no-recurrence-events	39	15
recurrence-events	19	13

Confusion Matrix

Class	TP	FP	FN	Precision	Recall	F1measure
no-recurrence-events	39	19	15	0.672	0.722	0.696
recurrence-events	13	15	19	0.464	0.406	0.433
weighted Avg	-	-	-	0.595	0.605	0.599

در حالت با هرس مقادیر به صورت زیر است:

actual\predicted	no-recurrence-events	recurrence-events
no-recurrence-events	47	7
recurrence-events	23	9

Confusion Matrix

Class	TP	FP	FN	Precision	Recall	F1measure
no-recurrence-events	47	23	7	0.671	0.87	0.758
recurrence-events	9	7	23	0.563	0.281	0.375
weighted Avg	-	-	-	0.631	0.651	0.616

در حالت بدون هرس میبینیم که درخت ۵۲ داده را درست و ۳۴ داده را غلط دسته بندی میکند و نتایج به درخت بدون هرس و بدون نویز نزدیک است. در حالت هرس شده نیز درخت ۵۶ داده را درست و ۳۰ داده را غلط دسته بندی میکند. در برخورد با نویز میبینیم که درخت هرس شده بهتر عمل میکند چون در حالت بدون هرس ممکن است درخت دچار بیش برازش شده و نویز ها را نیز یاد بگیرد اما با هرس کردن درخت از بیش برازش ان جلوگیری میشود.

۳ سوالات پیاده سازی

کدی که در این بخش برای ماتریس در هم ریختگی و k-fold cross validation استفاده شده در زیر امده است که این کدها توسط خودم طراحی شده همچنین توضیحات کد ها در کامنت ها امده است.

```python
def confusion_matrix1(y_true , y_predicted):
    """calculate confusion matrix and print it
        class of samples should be 0 or 1
    Args:
        y_true ([numpy array n*1]): [true value of labels]
        y_predicted ([numpy array n*1]): [predicted value of labels]
    """
    # class 1 is posetive
    TP , TN , FP , FN = 0,0,0,0
    difference = y_true - y_predicted
    #true ->0 , predicted->1
    FP = np.count_nonzero(difference == -1)
    #true ->1 , predicted->0
    FN = np.count_nonzero(difference == 1)
    #smaples that has predicted correctly
    T = difference == 0
    #smaples that has predicted posetive
    P = y_predicted == 1
    #smaples that has predicted negative
    N = y_predicted == 0
    #smaples that has predicted correctly and posetive
    tp = T*P
    #smaples that has predicted correctly and negative
    tn = T*N
    TP = np.count_nonzero(tp)
    TN = np.count_nonzero(tn)
    print('class 1 is posetive')
    print('----------------------------')
    print('actual\predict|__0__1_')
    print('            0 |',TN,FP)
    print('            1 |',FN,TP)
    print('----------------------------')
    print()
```

Listing : ۱

```python
def K_fold(X , Y, K = 1 , dist_type = 'Euc' , folds_num = 10 ):
    """this function uses k-fold cross validation and KNN for predicting labes
    Args:
        X ([numpy array m*n]): [features]
        Y ([numpy array m*1]): [targets]
        K (int, optional): [num of neighbours for KNN]. Defaults to 1.
        dist_type (str, optional): [type of distance for KNN, it should be one of these: 'Euc','Mnhtn','Cosin'].
        Defaults to 'Euc'.
        folds_num (int, optional): [number of folds]. Defaults to 10.
    Returns:
        [numpy array m*1]: [predicted labels for X with k-fold cross validation and KNN]
    """
    #initializing predictions vector
    Y_pred = np.zeros((Y.shape[0] , 1))
    #picking folds ranges
    folds = np.linspace(0.0, 1.0, num=folds_num+1)
    #number of samples
    m = X.shape[0]
    #executing KNN for diffrent folds
    for i in range(folds_num):
        #calculating start and end of folds
        start = int(m * folds[i])
        end = int(m * folds[i+1])
        #calculating fold set and training set
        x_train = np.delete(X , np.arange(start,end) , axis = 0)
        x_test = X[start:end , :]
        y_train = np.delete(Y , np.arange(start,end))
        Y_pred[start:end] = KNN(x_train , y_train , x_test , K , dist_type).reshape(end - start,1)
    return Y_pred
```

Listing :۲ k-fold cross validation

۱.۳

در این بخش برای ارزیابی از k-fold cross validation به ازای k=10 استفاده شده است.که کد ان در بخش قبلی نمایش داده شد. همچنین پیش پردازش داده ها نیز به صورت زیر میباشد.

```
#reading data
data = pd.read_csv('mammographic_masses.data', sep=",", header=None)
data.columns = ['BI-RADS' , 'Age' ,'Shape'  , 'Margin' ,'Density','Severity']
#preprocessing
#fill ? entries with NaN
data = data.replace(['?'] , np.nan)
#changeing type of data to numeric
data = data.apply(pd.to_numeric)
#drop raws with 2 or more null values
data = data.dropna(thresh=5)
#filling remaining null values with the mean of its columns
data = data.fillna((data.mean()+0.5).astype('int64'))

data = data.astype('float64')
#normalize Age Attribute to 0-5 range
data['Age'] = ((data['Age'] - data['Age'].min())/(data['Age'].max()-data['Age'].min()))*5

#splitting targets and features
X = data.drop('Severity' , axis = 1)
Y = data['Severity']
X = X.to_numpy()
Y = Y.to_numpy().reshape(931,1)
```

Listing :۳ preprocess

همانطور که مشاهده میشود بعد از خواندن اطلاعات ابتدا مواردی که مقدار مشخصی ندارند را با null مقداردهی میکنیم. سپس نمونه هایی که تعداد ۲ یا بیشتری مقدار نامشخص در ویژگی هایشان داشتند را حذف میکنیم. تعداد این نمونه ها ۳۰ عدد میباشد که در مقابل تعداد کل نمونه ها یعنی ۹۶۱ تاثیر کمی دارد و میتوان از ان ها چشم پوشی کرد. سپس مقادیر نامشخص باقیمانده را با میانگین ویژگی مربوط با هر مقدار پر میکنیم. همچنین ویژگی سن را نیز در بازه ۰ تا ۵ نرمال میکنیم.

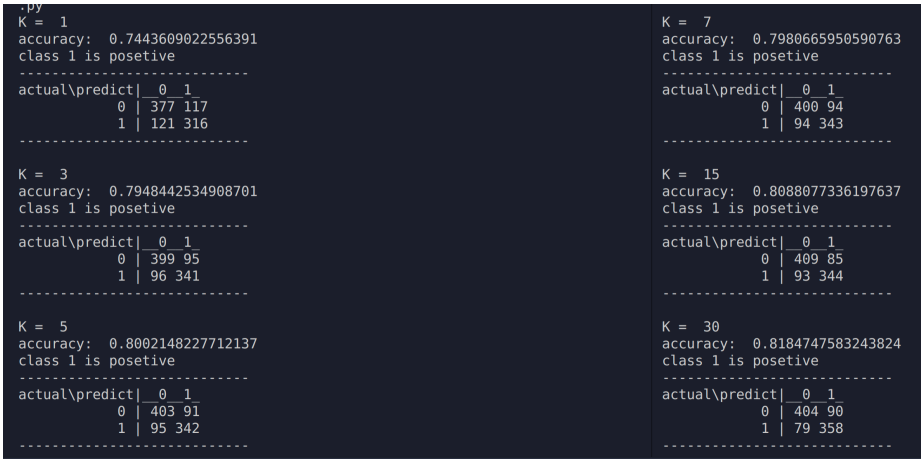
۱.۱.۳

کد این بخش با نام 1a.py ذخیره شده .همچنین پیاده سازی الگوریتم KNN به صورت زیر میباشد.

```python
def KNN( X_train , Y_train , X_test , K = 1 , dist_type = 'Euc' ):
    """this functions execute KNN algorithm
    Args:
        X_train ([numpy array m*n]): [features for training]
        Y_train ([numpy array m*1]): [targets for training]
        X_test ([numpy array z*n]): [features for predicting]
        K (int, optional): [num of neighbours]. Defaults to 1.
        dist_type (str, optional): [type of distance, it should be one of these: 'Euc','Mnhtn','Cosin']. Defaults
    Returns:
        [numpy array z*1]: [predicted values for X_test]
    """
    #for all types of distances the dists is a z*m matrix which z is num of test points and m is num of train poin
    #and in the ith row of dists we have distances for ith test point from all of train points
    if dist_type == 'Euc':
        #calculating L2 norm for each test point from each train point , L2 norm is the Euclidean distance
        dists = np.linalg.norm(X_test[:,np.newaxis]-X_train ,axis = 2)

    if dist_type == 'Mnhtn':
        #calculating manhatan distance for each test point from each train point
        dists = np.sum(np.absolute(X_test[:,np.newaxis]-X_train) , axis = 2)

    if dist_type == 'Cosin':
        #calculating cosin distance for each test point from each train point
        norm_xtrain = np.linalg.norm(X_train,axis = 1 ).reshape(X_train.shape[0],1)
        norm_xtest = np.linalg.norm(X_test,axis = 1 ).reshape(X_test.shape[0],1)
        norms = norm_xtest @ norm_xtrain.T
        dists = 1 - ((X_test @ X_train.T) / norms)

    #choosing min distances
    min_dists_indices = np.argpartition(dists,K,axis = 1)[:,:K]
    #calculating prediction labels
    Y_sum_min_distances = np.sum(Y_train[min_dists_indices] , axis = 1)
    y_pred = Y_sum_min_distances > (K-1)/2
    return y_pred.astype('int64')
```

Listing :۴ KNN

نتایج به ازای مقادیر مختلف برای K در صفحه بعد امده است.



شکل ۷: ماتریس های پریشانی و دقت الگوریتم

میبینیم که با افزایش K تا مقدار ۵ عملکرد الگوریتم نیز بهتر میشود اما بعد از ان برای مقدار ۷ دقت الگوریتم کاهش میابد و سپس برای مقادیر بعدی دقت دوباره افزایش میابد و بهترین عملکرد برای K=30 میباشد. برای مقادیر کوچک این متغیر الگوریتم دچار بیش برازش میشود اما با افزایش مقدار این متغیر بایاس افزایش می یابد و در نتیجه از بیش برازش جلوگیری میشود اما برای مقادیر خیلی بزرگ نیز ممکن است دچار کم برازش شویم.

۲.۱.۳

کد این بخش با نام 1b.py ذخیره شده است. نتایج برای معیار های مختلف فاصله به ازای K=30 به صورت زیر است. میبینیم که بهترین نتیجه به ازای فاصله منهتنی بدست می اید.

```
dist_type:  Euc
accuracy:  0.8184747583243824
class 1 is posetive
----------------------------
actual\predict|__0__1_
            0 | 404 90
            1 | 79 358


dist_type:  Mnhtn
accuracy:  0.8216970998925887
class 1 is posetive
----------------------------
actual\predict|__0__1_
            0 | 402 92
            1 | 74 363


dist_type:  Cosin
accuracy:  0.7722878625134264
class 1 is posetive
----------------------------
actual\predict|__0__1_
            0 | 348 146
            1 | 66 371
```

شکل ۸: ماتریس های پریشانی و دقت الگوریتم

۳.۱.۳

کد این بخش با نام 1c.py ذخیره شده است. در این بخش از کتابخانه SKlearn استفاده شده است. مقایسه کامل دو الگوریتم در صفحه بعد امده است. مشاهده میشود که از لحاظ دقت دو الگوریتم بسیار به هم نزدیک هستند. همچنین چون در پیاده سازی الگوریتم ما سعی شده در همه بخش ها از محاسبات ماتریسی استفاده شود و حلقه نداریم سرعت ان نیز با کتابخانه اماده قابل مقایسه هست و سرعت خوبی دارد.

```
K =  1
time for our KNN:  0.04281139373779297
time for sklearn KNN:  0.0386199951171875
accuracy for our KNN:  0.7443609022556391
accuracy for sklearn KNN:  0.7561761546723953
confusion_matrix for our KNN
class 1 is posetive
----------------------------
actual\predict|__0__1_
           0 | 377 117
           1 | 121 316

confusion_matrix for our sklearn KNN
class 1 is posetive
----------------------------
actual\predict|__0__1_
           0 | 381 113
           1 | 114 323
```

```
K =  5
time for our KNN:  0.055954694747924805
time for sklearn KNN:  0.04577803611755371
accuracy for our KNN:  0.8002148227712137
accuracy for sklearn KNN:  0.7969924812030075
confusion_matrix for our KNN
class 1 is posetive
----------------------------
actual\predict|__0__1_
           0 | 403 91
           1 | 95 342

confusion_matrix for our sklearn KNN
class 1 is posetive
----------------------------
actual\predict|__0__1_
           0 | 399 95
           1 | 94 343
```

```
K =  3
time for our KNN:  0.053261518478393555
time for sklearn KNN:  0.039757490158081055
accuracy for our KNN:  0.7948442534908701
accuracy for sklearn KNN:  0.8023630504833512
confusion_matrix for our KNN
class 1 is posetive
----------------------------
actual\predict|__0__1_
           0 | 399 95
           1 | 96 341

confusion_matrix for our sklearn KNN
class 1 is posetive
----------------------------
actual\predict|__0__1_
           0 | 402 92
           1 | 92 345
```

```
K =  7
time for our KNN:  0.0626521110534668
time for sklearn KNN:  0.044721364974975586
accuracy for our KNN:  0.7980665950590763
accuracy for sklearn KNN:  0.8002148227712137
confusion_matrix for our KNN
class 1 is posetive
----------------------------
actual\predict|__0__1_
           0 | 400 94
           1 | 94 343

confusion_matrix for our sklearn KNN
class 1 is posetive
----------------------------
actual\predict|__0__1_
           0 | 402 92
           1 | 94 343
```

```
K =  15
time for our KNN:  0.057218313217163086
time for sklearn KNN:  0.05327153205871582
accuracy for our KNN:  0.8088077336197637
accuracy for sklearn KNN:  0.8066595059076263
confusion_matrix for our KNN
class 1 is posetive
----------------------------
actual\predict|__0__1_
           0 | 409 85
           1 | 93 344

confusion_matrix for our sklearn KNN
class 1 is posetive
----------------------------
actual\predict|__0__1_
           0 | 409 85
           1 | 95 342
```

```
K =  30
time for our KNN:  0.05705666542053223
time for sklearn KNN:  0.05849051475524902
accuracy for our KNN:  0.8184747583243824
accuracy for sklearn KNN:  0.8195488721804511
confusion_matrix for our KNN
class 1 is posetive
----------------------------
actual\predict|__0__1_
           0 | 404 90
           1 | 79 358

confusion_matrix for our sklearn KNN
class 1 is posetive
----------------------------
actual\predict|__0__1_
           0 | 415 79
           1 | 89 348
```

شکل ۹: مقایسه دو الگوریتم

۲.۳

کد این بخش با نام 2.py ذخیره شده است. بهترین مقدار پارامتر K برای این قسمت ۹ میباشد که خطای های مجموعه اموزش و ارزیابی برای این مقدار به صورت زیر است.

test error: 28.32302913580248
train error: 30.62149985048486

تفاوتی که الگوریتم KNN پیاده شده در این قسمت با قسمت قبل دارد این است که به جای انتخاب مقداری که بین نزدیک ترین همسایه ها بیشترین تکرار را دارد از این مقادیر میانگین میگیرد.

۳.۳

کد این بخش با نام 3.py خیره شده است. قسمت پیش پردازش داده ها به صورت زیر میباشد.

```
#reading data
train_data = pd.read_csv('breast-cancer-wisconsin-train.data', sep=",", header=None)
test_data = pd.read_csv('breast-cancer-wisconsin-test.data', sep=",", header=None)

#preprocessing
#add column names
train_data.columns = ['id' , 'x1','x2','x3','x4','x5','x6','x7','x8','x9','y']
test_data.columns = ['id' , 'x1','x2','x3','x4','x5','x6','x7','x8','x9','y']
#droping id column
train_data = train_data.drop('id' ,axis = 1)
test_data = test_data.drop('id' ,axis = 1)
#fill ? entries with NaN
train_data = train_data.replace(['?'] , np.nan)
#changeing type of data to numeric
train_data = train_data.apply(pd.to_numeric)
#fill null values with a new value
train_data = train_data.fillna(11)

#splitting targets and featurs
x_train = train_data.drop('y' ,axis = 1).to_numpy()
y_train = train_data['y'].to_numpy().reshape(499)
x_test = test_data.drop('y' , axis = 1).to_numpy()
y_test = test_data['y'].to_numpy().reshape(200,1)
```

Listing :۵ preprocess

در این مجموعه داده ها تمامی مقدار های نامشخص مربوط به ویژگی X6 میباشند. همچنین مقادیر این ویژگی اعداد صحیح ۱ تا ۱۰ هستند.برای حل مشکل مقادیر نامعلوم این مقادیر را یک مقدار جدید برای این داده در نظر میگیریم و ان ها را با عدد ۱۱ نمایش میدهیم.همچنین ستون ویژگی id را نیز از هر دو مجموعه اموزش و ارزیابی حذف میکنیم. نتایج این بخش برای مجموعه ازمون به صورت زیر است.

accuracy: 0.945

actual\predicted	2	4
2	135	5
4	6	54

Confusion Matrix