(i)

(a)First I plot the dataset 1 , use function scattersth predefined in funcfile.py to see what the raw data likes :

And the original data scatters and there seems a quadric curve between y=1 and y=-1

```
polynomial_degrees=[1,2,3,4,5,10,15,20,30,40,50,60]
```

Figure a.1 the original dataset 1

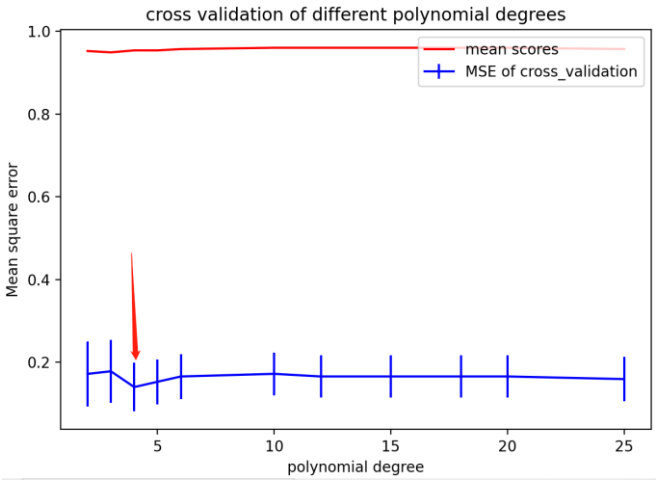

Figure a.2 dataset 1: the mean scores and MSE of cross validation

Then I use a iteration for each value of degree , I use K-fold(K=5) to cross validate the model , the "degree_mean_mse" array store the mean MSE of each degree , the "degree_std_mse" array to store the standard deviation of MSE of each degree , "mean_score_degree_range" array to store the mean scores of the model of each degree of polynomial.

And I use linear model of logisticregression of which the penalty is L2 , and I set the C=1 to see the impact of the degree.

```
    for inx1,degree in enumerate(polynomial_degrees):
    poly = PolynomialFeatures(degree=degree, interaction_only=False,include_bias=False)
    x_poly = poly.fit_transform(x)
    kf = KFold(n_splits=5)
    mse = []
    for train, test in kf.split(x_poly):
        mul_lr = linear_model.LogisticRegression(penalty='l2',C=1)
        mul_lr.fit(x_poly, y)
        y_pre = mul_lr.predict(x_poly[test])
        from sklearn.metrics import mean_squared_error
        mse.append(mean_squared_error(y_pre, y[test]))
    degree_mean_mse.append(np.mean(mse))
    degree_std_mse.append(np.std(mse))
    mean_score_degree_range.append(cross_val_score(mul_lr, x_poly, y, cv=5).mean())
```

And I plot the MSE and scores , as we can see from figure a.2 , the MSE is decreasing and the score is increasing slightly by the degree , and as degree gets bigger the accuracy of the model is better , and when degree is up to a certain value they don't change much , but to avoid over-fitting , I'd like to choose the simplest model , so the degree=4 is enough , as the red arrow I draw , the MSE is the smallest with the smaller standard deviation than other values of degree.

After picking degree=4 , I choose a wide range of C too :

```
c_range=[0.1,0.5,1,2,3,5,10,15,20,30,40]
```

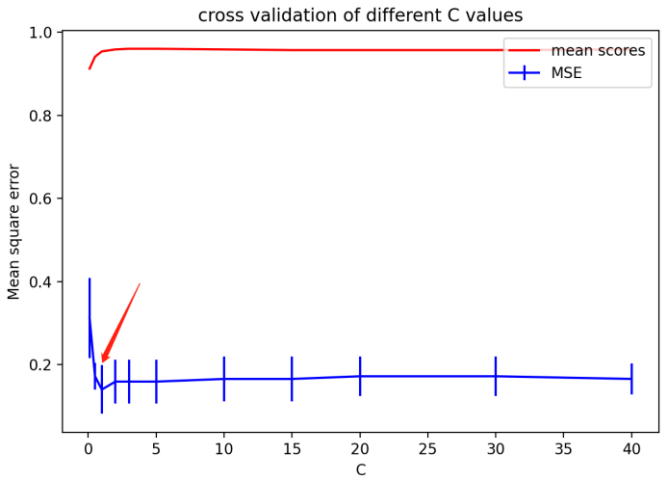

Figure a.3 dataset 1: the mean scores and MSE of different C values

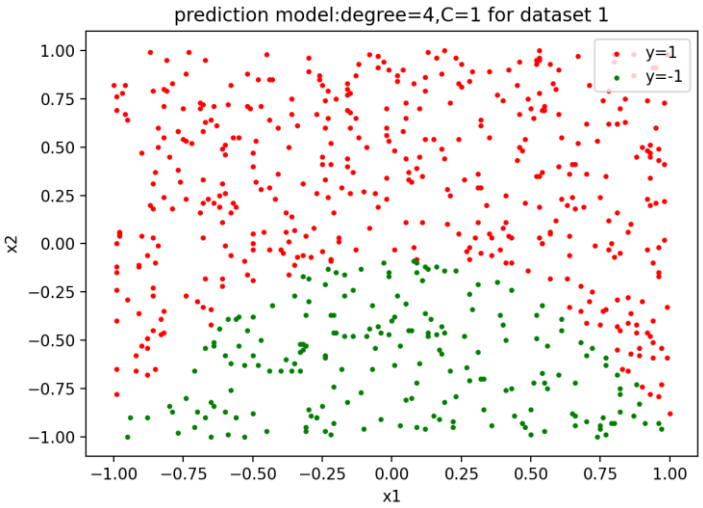

Figure a.4 dataset 1:the final prediction model

As we can see from Figure a.3 , the mean scores is increasing as the C goes by , the MSE drops first but when C > 1 MSE is increasing slightly again , the standard deviation seems not change so much when C is in 0~5 , so I would pick the simplest model to avoid over-fitting : C=1 , what's more , when C=1 the MSE is the smallest.Above all ,I pick degree=4 and C=1 , then define the new logistic regression model and plot it.

```
fig4=plt.figure()
poly_final = PolynomialFeatures(degree=4, interaction_only=False, include_bias=False)
x_poly_final = poly_final.fit_transform(x)
kf = KFold(n_splits=5)
mul_lr_final = linear_model.LogisticRegression(penalty='l2', C=1)
mul_lr_final.fit(x_poly_final, y)
y_pre_final=mul_lr_final.predict(x_poly_final)
print("The score of the model is {0}".format(mul_lr_final.score(x_poly_final,y)))
scattersth(x1,x2,y_pre_final,"prediction model:degree=4,C=1 for dataset 1")
plt.legend(loc='upper right', fontsize=10)
```

The score/accuracy of the model is 96.5% which is really high , as we can see from Figure a.4 and a.1 , we can see the prediction model is so nice that they are almost the same , like original data scatters and there seems a quadric curve between y=1 and y=-1 .

(b)First I choose a wide range of K to plot each of them the MSE and score of cross-validation:

```
k_range=[1,2,3,4,5,6,10,15,20,30]
```

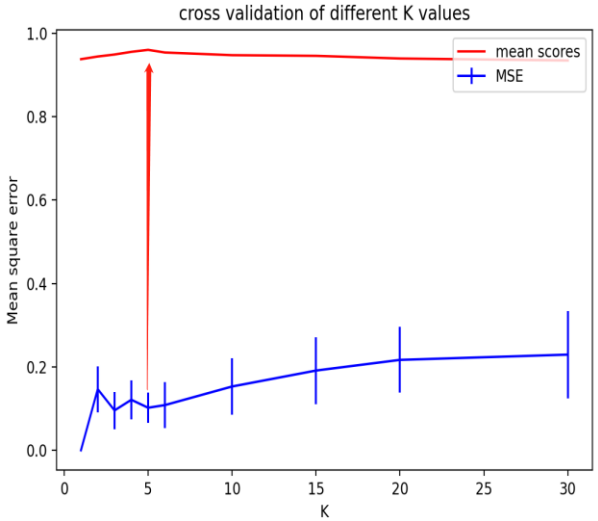

Figure b.1 dataset 1: the mean scores and MSE of different K values

As we can see from Figure b.1 , the MSE fluctuated as K goes up , when K>5 the MSE is increasing , so I'd like to choose the K<=5, and the score is increasing when K goes up , and when K=5 the score is the highest which I use red arrow to point out , and when K>5 the score drops , so I would choose K=5 because it has higher score with lower MSE as well as standard deviation.

Now it's time for picking the degree of polynomial features. First I choose a wide range of degree

```
polynomial_degrees=[2,3,4,5,6,10,12,15,20]
```

And I define "mean_score_degree_range" to store the mean score of each different degree's and in each iteration of degree get the mean score , and outside get the mean score of the knn model without polyfeatures then plot them in a plot.

```
    for degree in polynomial_degrees:
    poly = PolynomialFeatures(degree=degree, interaction_only=False,include_bias=False)
    x_poly = poly.fit_transform(x)
    knn_model = KNeighborsClassifier(n_neighbors=5, weights='uniform').fit(x_poly, y)
    mean_score_degree_range.append(cross_val_score(knn_model, x_poly, y, cv=5).mean())
#this is knn model without poly features mean score of cross validation
knn_model_no_poly = KNeighborsClassifier(n_neighbors=5, weights='uniform').fit(x,y)
knn_no_poly_mean_score= cross_val_score(knn_model, x, y, cv=5).mean()
```

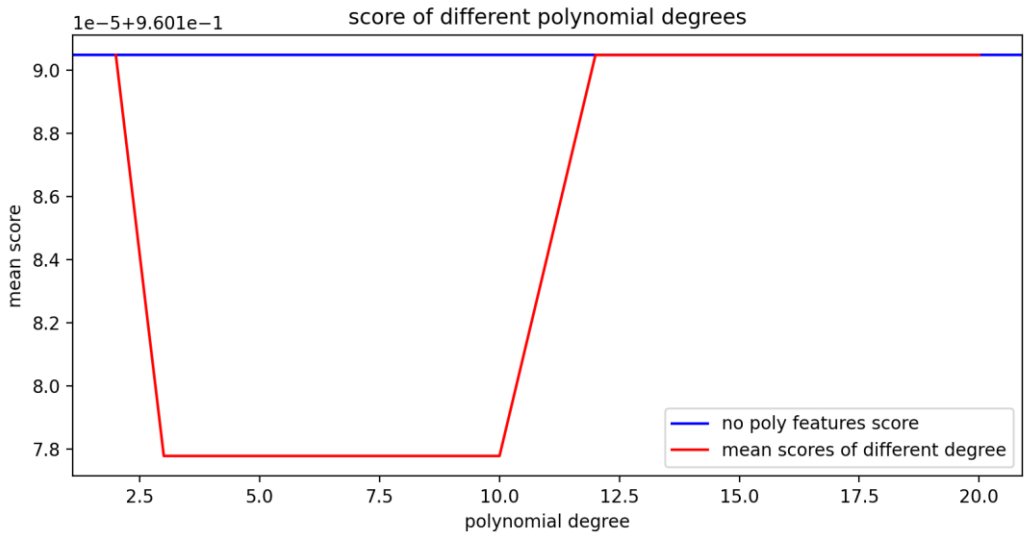

Figure b.2 dataset 1: the mean scores and MSE of different degree values with no polynomial features mean score

As we can see from Figure b.2 , the score with knn model of polynomial degree doesn't change (degree>12) or even lower(2<degree<12) than without polynomial features' model , so why do we need to add the polynomial feature just to lower the score of the model?

Obviously there's no meaning of add polynomial features in KNN model.

(C)

The confusion matrix is in the format bellow:

TP	FN
FP	TN

For logistic regression after adding the polynomial features of X I split the dataset into 2 parts of train and test of each x or y. Then train the model with dataset "train" , then use plot_confusion_matrix function to plot the confusion matrix and get the classification report. Also I get the KNN , dummy classifier confusion matrix plot and classification report.

```
X_train_logistic, X_test_logistic, y_train_logistic, y_test_logistic =
train_test_split(x_poly_final, y, random_state=0)
mul_lr_final = linear_model.LogisticRegression(penalty='l2', C=1)
mul_lr_final.fit(X_train_logistic, y_train_logistic)
y_pre_logistic=mul_lr_final.predict(X_test_logistic)
disp1=plot_confusion_matrix(mul_lr_final, X_test_logistic, y_test_logistic)
print("classification report")
print(classification_report(y_test_logistic, y_pre_logistic))
print("Logistic Regression:")
print(disp1.confusion_matrix)
plt.title("trained logistic regression")
```

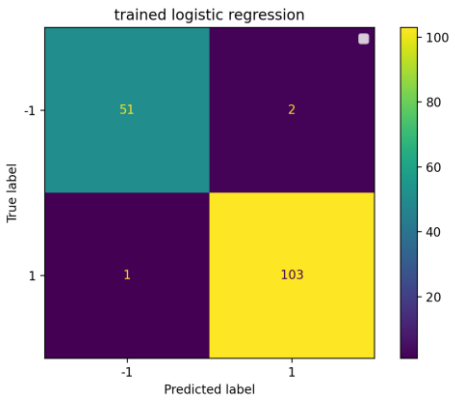

Figure c.1 logistic model confusion matrix

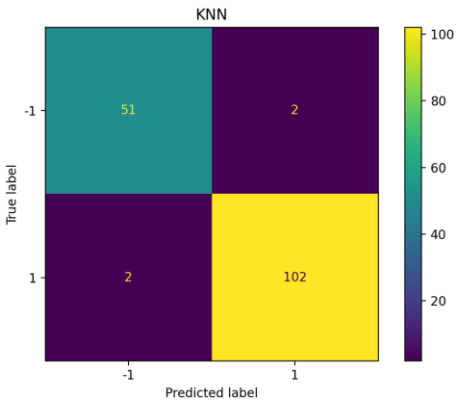

Figure c.2 KNN confusion matrix

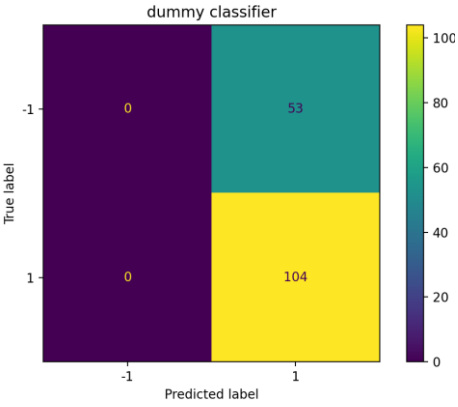

Figure c.3 dummy classifier confusion matrix

For logistic regression :

```
Logistic Regression confusion matrix:
[[ 51   2]
 [  1 103]]
classification report
              precision    recall  f1-score   support

          -1       0.98      0.96      0.97        53
           1       0.98      0.99      0.99       104

    accuracy                           0.98       157
   macro avg       0.98      0.98      0.98       157
weighted avg       0.98      0.98      0.98       157
```

For KNN :

```
KNN confusion matrix:
[[ 51   2]
 [  2 102]]
classification report
              precision    recall  f1-score   support

          -1       0.96      0.96      0.96        53
           1       0.98      0.98      0.98       104

    accuracy                           0.97       157
   macro avg       0.97      0.97      0.97       157
weighted avg       0.97      0.97      0.97       157
```

For baseline model : dummy classifier

```
dummy classifier confusion matrix:
[[  0  53]
 [  0 104]]
classification report
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00        53
           1       0.66      1.00      0.80       104

    accuracy                           0.66       157
   macro avg       0.33      0.50      0.40       157
weighted avg       0.44      0.66      0.53       157
```

As we can see from above , the logistic regression model and KNN model perform very well and the logistic regression(accuracy=0.98) is the highest, both of them have high accuracy , while the baseline of dummy model score is 0.66 , not so good. (Dummy model is the one makes random predictions.)

(d)

First I use roc_curve function to get the false positive rate and true positive rate , then plot them , because the curve of logistic regression model and KNN overlaps , so I make the line of KNN's thicker by linewidth=6 .

The codes are as follows:

```
from sklearn.metrics import roc_curve
#logistic regression
fpr1, tp1, _ =
roc_curve(y_test_logistic,mul_lr_final.decision_function(X_test_logistic))
plt.plot(fpr1,tp1,label='logistic regression',c='b',linewidth=6)
#KNN
y_scores_knn = knn_model.predict_proba(X_test_knn)
fpr2, tp2, threshold1 = roc_curve(y_test_knn, y_scores_knn[:, 1])
plt.plot(fpr2,tp2,label='KNN',c='y')
#baseline
y_scores_dummy = dummy.predict_proba(X_test_dummy)
fpr3, tp3, threshold2 = roc_curve(y_test_dummy, y_scores_dummy[:, 1])
plt.plot(fpr3,tp3,label='baseline model:dummy classifier',c='r',linewidth=6)
```

AUC - ROC curve is a performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability. It tells how much model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s.

As we can see from Figure d.1 , the AUC of logistic regression and KNN are all very high nearly 1 , so they perform really well , while the baseline model dummy model performs the same as the random prediction , really bad , its AUC is approximately 0.5, model has no discrimination capacity to distinguish between positive class and negative class.

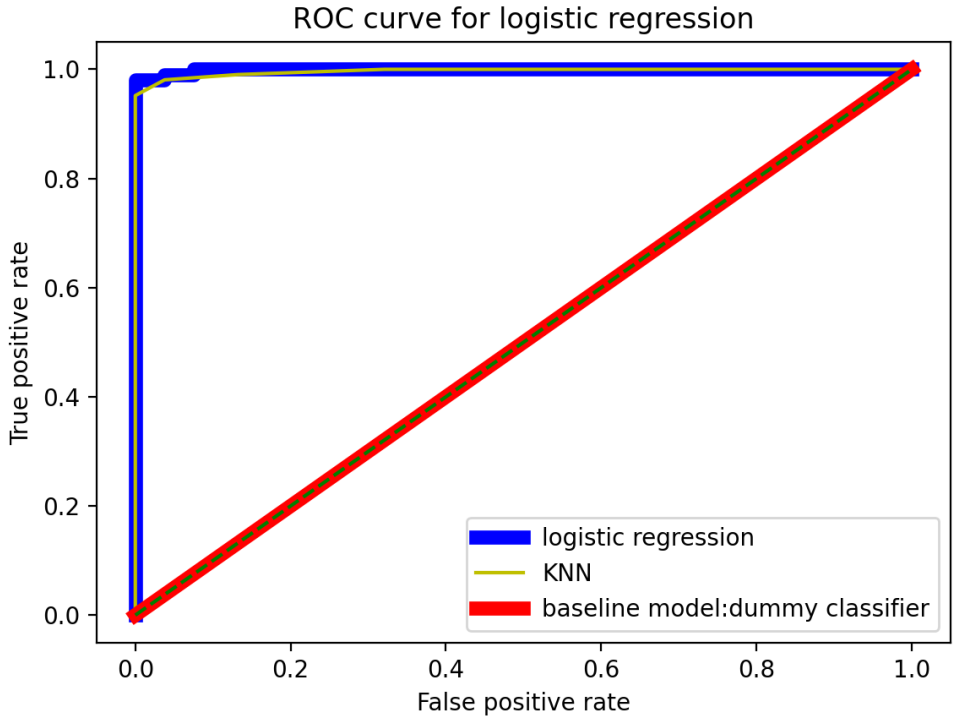

Figure d.1 the ROC curve for 3 models

(e)

```
Logistic Regression confusion matrix:
[[ 51   2]
 [  1 103]]
classification report
              precision    recall  f1-score   support

          -1       0.98      0.96      0.97        53
           1       0.98      0.99      0.99       104

    accuracy                           0.98       157
   macro avg       0.98      0.98      0.98       157
weighted avg       0.98      0.98      0.98       157
```

Figure e.1

```
KNN confusion matrix:
[[ 51   2]
 [  2 102]]
classification report
              precision    recall  f1-score   support

          -1       0.96      0.96      0.96        53
           1       0.98      0.98      0.98       104

    accuracy                           0.97       157
   macro avg       0.97      0.97      0.97       157
weighted avg       0.97      0.97      0.97       157
```

Figure e.2

```
dummy classifier confusion matrix:
[[  0  53]
 [  0 104]]
classification report
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00        53
           1       0.66      1.00      0.80       104

    accuracy                           0.66       157
   macro avg       0.33      0.50      0.40       157
weighted avg       0.44      0.66      0.53       157
```

Figure e.3

I copy the confusion matrix and classification report here , as we can see from above , and the format of the matrix is :

TP	FN
FP	TN

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \qquad F1 - score = \frac{2}{\frac{1}{Recall} + \frac{1}{Precision}}$$

F1-score , is a measure of a test's accuracy. It is calculated from the precision and recall of the test, where the precision is the number of correctly identified positive results divided by the number of all positive results, including those not identified correctly, and the recall is the number of correctly identified positive results divided by the number of all samples that should have been identified as positive.

From the classification report we could know the F1-score of logistic regression is 0.97(y=-1) and 0.99(y=1) , KNN is 0.96(y=-1) and 0.98(y=1) , dummy classifier is 0(y=-1) and 0.8(y=1) , so the logistic regression model performs the best and KNN is also good while the baseline model dummy classifier performs significantly worse than others.

From Figure d.1 we could know logistic regression model and KNN model are excellent models have AUC near to the 1 which means it has good measure of separability. A poor model has AUC near to the 0

which means it has worst measure of separability. In fact it means it is reciprocating the result. It is predicting 0s as 1s and 1s as 0s. And when AUC is 0.5 like the dummy classifier , has no discrimination capacity to distinguish between positive class and negative class.

Above all , I'd like to choose logistic regression model because it not only has the highest F1-score(accuracy) but also the highest AUC near to 1 , so the model is really good.

(ii)

(a) First I plot the dataset 2 , use function scattersth predefined in funcfile.py to see what the raw data likes : it's really noisy and in a mess , scatters randomly , and I can't see the decision boundary clearly ,so I decide to choose wider range of polynomial degrees :

```
polynomial_degrees=[2,3,4,5,6,10,12,15,40,50,60,80,100]
```

Figure 1 the original dataset 2

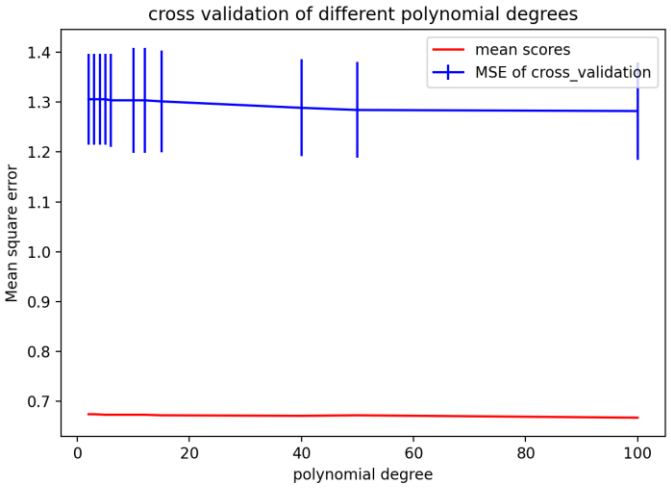

Figure 2 dataset 2:the mean scores and MSE of cross validation

Then I use a iteration for each value of degree , I use K-fold(K=5) to cross validate the model , the "degree_mean_mse" array store the mean MSE of each degree , the "degree_std_mse" array to store the standard deviation of MSE of each degree , "mean_score_degree_range"array to store the mean scores of the model of each degree of polynomial.

I use errorbar to draw the MSE , and the x axis is the value of degree , the y axis is the MSE.

```
plt.plot(c_range,mean_score_c_range,label='mean scores',color='red')
plt.errorbar(c_range, crange_mean_mse, label="MSE", color='blue', yerr=crange_std_mse)
plt.ylabel('Mean square error')
plt.xlabel('C')
plt.title("cross validation of different C values")
plt.legend(loc='upper right', fontsize=10)
plt.show()
```

And I plot the MSE and scores , as we can see from Figure 2 , I find the curve is really smooth , so I decide to plot the score and MSE separately to see clearly.

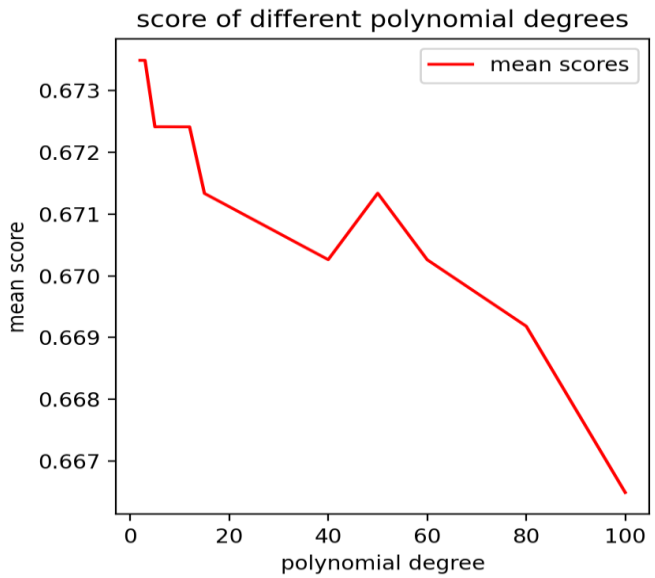

Figure 3 dataset 2: the mean scores of different degree

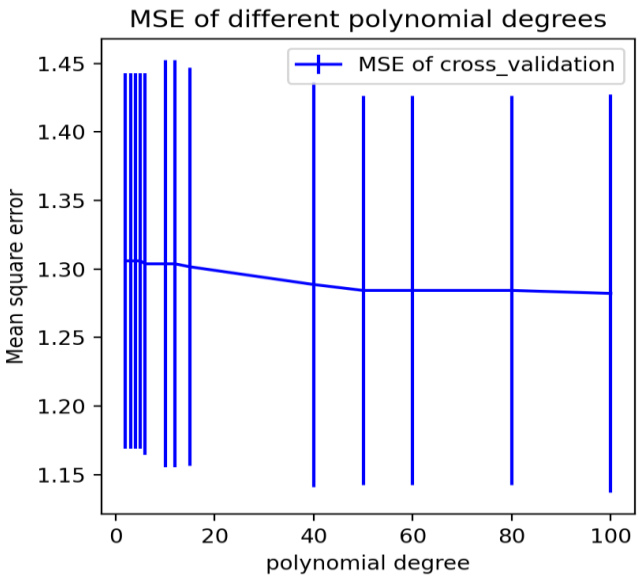

Figure 4 dataset 2: MSE of different degree values

As we can see from Figure 3 the score is decreasing but when degree near 50 the score is the higher and then drops as degree goes by , and from Figure 4 we can see the MSE is decreasing of which degree=50 is very low , and then the MSE seems no change at all as degree increases , so I would choose degree=50 as my final choice.

Clearly for a given training set when the degree is low the hypothesis will underfit the data and there will be a high bias error, however when the degree of the polynomial is high then the fit will get better and better on the training set.

Then I choose a wide range of C to see which one is the best for the model:

```
c_range=[0.1,0.5,1,2,3,5,10,15,20,30,40,50,60]
```

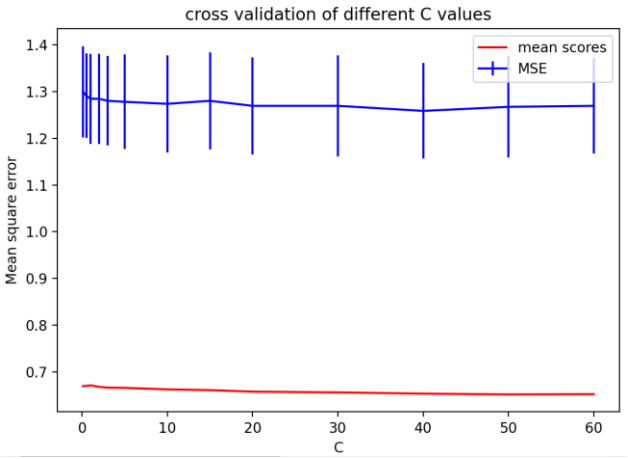

Figure 5 the mean scores of different C(large range)

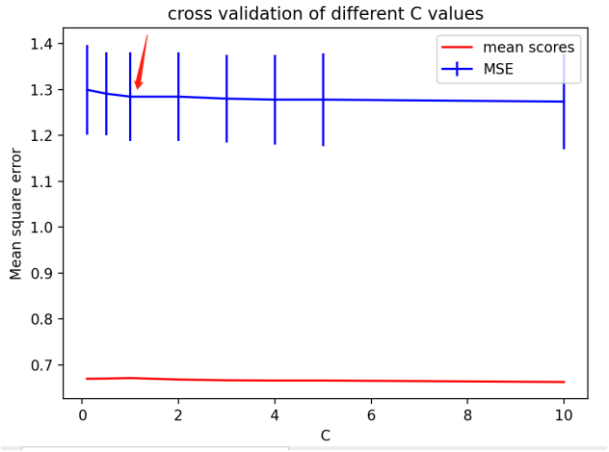

Figure 6 the mean scores of different C(small range)

As we can see from Figure 5 , the score of the model is decreasing as C goes by , while MSE doesn't change much , so I decide to choose smaller range of C to find the best C:

c_range=[0.1,0.5,1,2,3,4,5,10]

As we can see from Figure b.6 , when C=1 the score is high while the MSE is smaller , so I would choose C=1.

Above all , I choose degree=50 and C=1 , then define the new logistic regression model and plot it.

The score of the model is 0.6788793103448276 , but as we can see from Figure 7 , the prediction is almost all the data of y=1, only few green points of y=-1 , but the score is not so low I think the reason for that is the data of y=1 is more than y=1 in the original data , but we can tell by Figure 1 the original data is not like

this , and the model is bad , I think maybe degree=50 the prediction model is overfitting

Figure 1 the original dataset 2

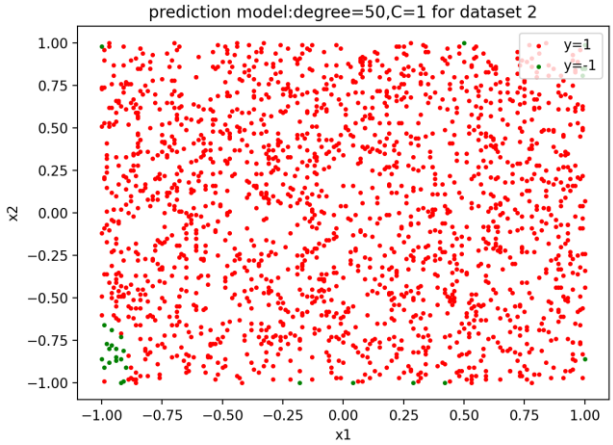

Figure 7 the final prediction model

(b)For KNN model the parameter of K :

First I choose a wide range of K to plot each of them the MSE and score of cross-validation:

```
k_range=[2,3,4,5,6,10,15,20,30,40,50,60,70,80,90,100,200]
```

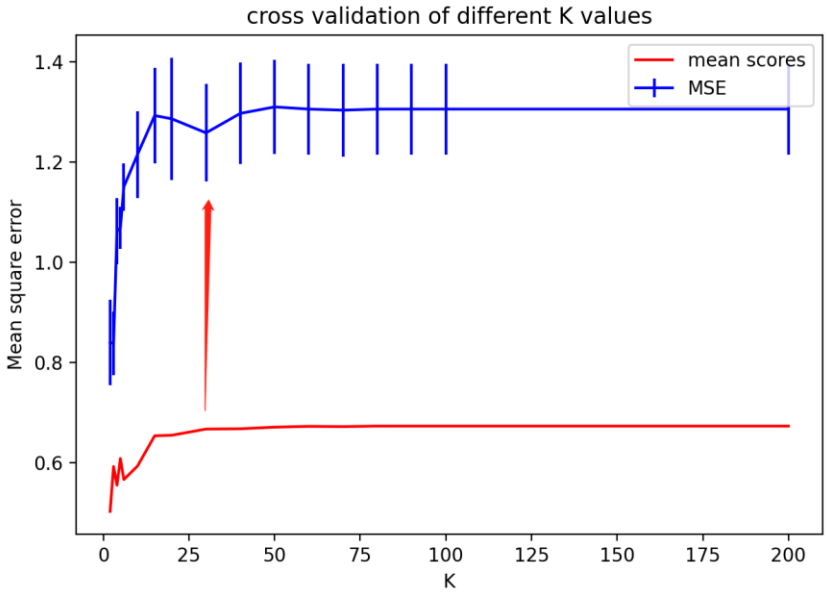

Figure 8 the MSE and score of different K

```
for k in k_range:
kf = KFold(n_splits=5)
mse = []
for train, test in kf.split(x):
    from sklearn.neighbors import KNeighborsClassifier
    knn_model = KNeighborsClassifier(n_neighbors=k, weights='uniform').fit(x, y)
    y_pre = knn_model.predict(x[test])
    from sklearn.metrics import mean_squared_error
    mse.append(mean_squared_error(y_pre, y[test]))
knn_mean_mse.append(np.mean(mse))
knn_std_mse.append(np.std(mse))
mean_score_k_range.append(cross_val_score(knn_model, x, y, cv=5).mean())
```

As we can see from Figure 8 , as K goes by the MSE and score are increasing , but when K up to a certain value the score doesn't change much , so to avoid overfitting I would choose K=30 and as the red arrow points , the MSE is the smallest nearby (with almost the same score).

Now it's time for picking the degree of polynomial features. First I choose a wide range of degree

```
polynomial_degrees=[2,3,4,5,6,10,12,15,20]
```

And I define "mean_score_degree_range" to store the mean score of each different degree's and in each iteration of degree get the mean score , and outside get the mean score of the knn model without polyfeatures then plot them in a plot.

```
for degree in polynomial_degrees:
    poly = PolynomialFeatures(degree=degree, interaction_only=False,include_bias=False)
    x_poly = poly.fit_transform(x)
    knn_model = KNeighborsClassifier(n_neighbors=30, weights='uniform').fit(x_poly, y)
    mean_score_degree_range.append(cross_val_score(knn_model, x_poly, y, cv=5).mean())
#this is knn model without poly features mean score of cross validation
knn_model_no_poly = KNeighborsClassifier(n_neighbors=30, weights='uniform').fit(x,y)
knn_no_poly_mean_score= cross_val_score(knn_model, x, y, cv=5).mean()
```

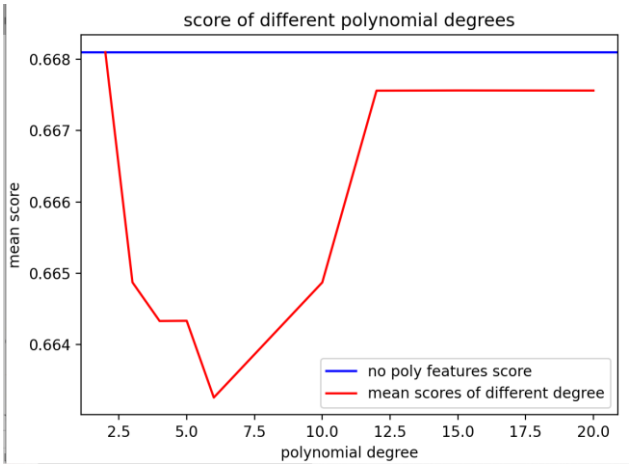

Figure 9 dataset 2: the mean scores and MSE of different degree values with no polynomial features mean score

As we can see from Figure 9 , the score with knn model of polynomial degree even lower than without polynomial features' model , so why do we need to add the polynomial feature just to lower the score of the model?

Obviously there's no meaning of add polynomial features in KNN model.

(c)

For logistic regression after adding the polynomial features of X I split the dataset into 2 parts of train and test of each x or y. Then train the model with dataset "train" , then use plot_confusion_matrix function to plot the confusion matrix and get the classification report. Also I get the KNN , dummy classifier confusion matrix plot and classification report.

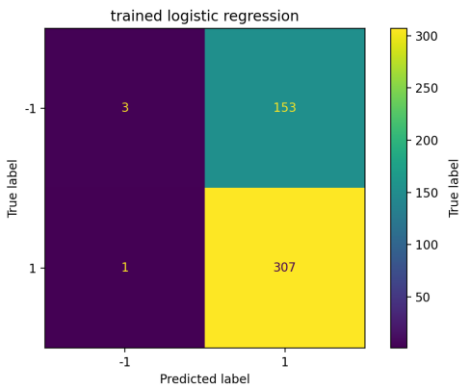

Figure 10 logistic model confusion matrix

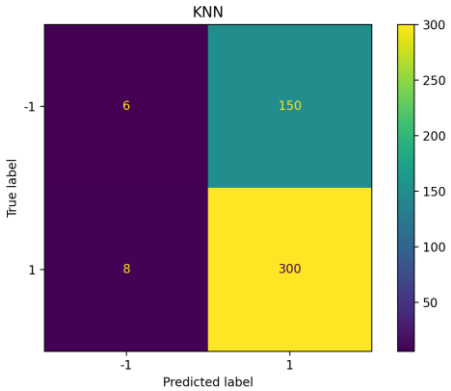

Figure 11 KNN confusion matrix

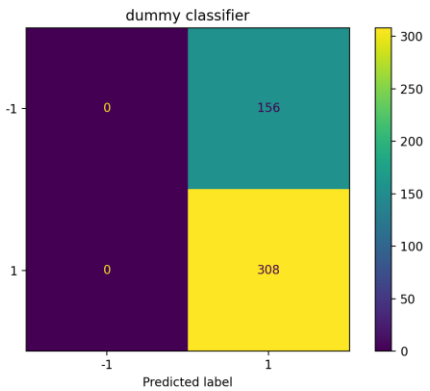

Figure 12 dummy classifier confusion matrix

For logistic regression : For KNN : For baseline model : dummy classifier:

Logistic Regression confusion matrix:

[[3 153]
 [1 307]]

classification report

	precision	recall	f1-score	support
-1	0.75	0.02	0.04	156
1	0.67	1.00	0.80	308
accuracy			0.67	464
macro avg	0.71	0.51	0.42	464
weighted avg	0.70	0.67	0.54	464

KNN confusion matrix:

[[6 150]
 [8 300]]

classification report

	precision	recall	f1-score	support
-1	0.43	0.04	0.07	156
1	0.67	0.97	0.79	308
accuracy			0.66	464
macro avg	0.55	0.51	0.43	464
weighted avg	0.59	0.66	0.55	464

dummy classifier confusion matrix:

[[0 156]
 [0 308]]

classification report

	precision	recall	f1-score	support
-1	0.00	0.00	0.00	156
1	0.66	1.00	0.80	308
accuracy			0.66	464
macro avg	0.33	0.50	0.40	464
weighted avg	0.44	0.66	0.53	464

As we can see from above , the accuracy of all the 3 models is almost the same 0.66 , with logistic regression is a little higher:0.67 , I think they are all not so good. Because even the best model : logistic regression has highest score , the prediction plot Figure 7 doesn't look like the original data , the F1-score only when y=1 is high(0.8) while when y=-1 the F1-score is 0.04 , which can't reflect the features of the data .

(d)

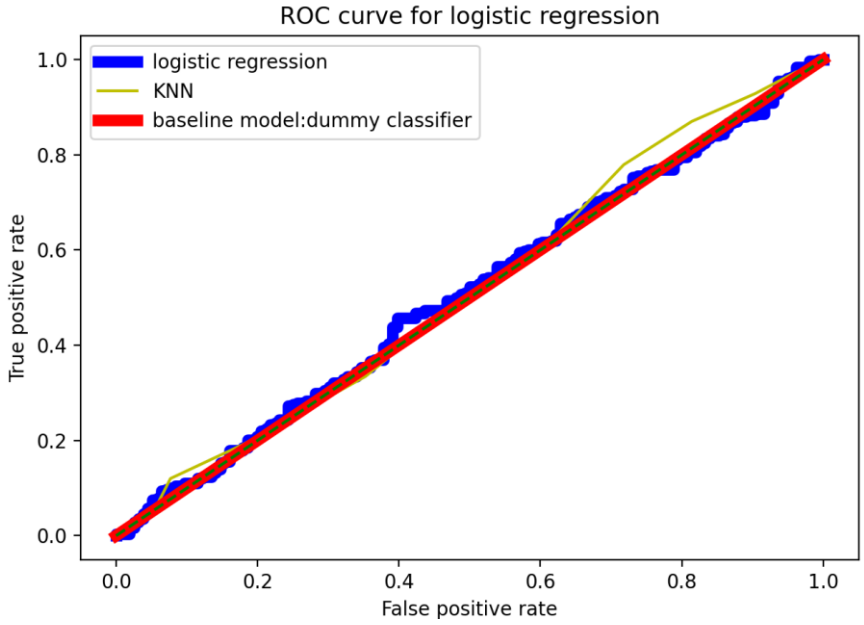

Figure 13 ROC curve

All the 3 models above AUC are almost 0.5 , which means has no discrimination capacity to distinguish between positive class and negative class. So all of the models are unable to extract the features of the data , which means the original dataset may be at random at first , so all the model are bad for it. They all overlaps with the green dotted line (fpr=tpr) , which means random classification , for a poor classifier, where the algorithm is randomly guessing, the ROC curve hugs the diagonal and the area under the curve is 1/2.

So all of the models are bad to predict.

(e) I think all of the 3 models are bad to predict because

1) even the best model with the highest accuracy——logistic regression has highest score , the prediction plot Figure 7 doesn't look like the original data , the F1-score only when y=1 is high(0.8) while when y=-1 the F1-score is 0.04 , which means only when the data of y=1 can predict well while data of y=-1 predict very poor , and this means the model can not extract the features of the data . And for other worse

models , when y=1 their F1-score is 0.79(KNN) and 0.80(dummy) , and when y=-1 , the F1-score of KNN is 0.07 , of dummy is 0 , they all can not predict data well when y=-1

2)all the 3 models above AUC are almost 0.5 , which means has no discrimination capacity to distinguish between positive class and negative class. So all of the models are unable to extract the features of the data , which means the original dataset may be at random at first , so all the model are bad for it. They all overlaps with the green dotted line (fpr=tpr) , which means random classification , for a poor classifier, where the algorithm is randomly guessing, the ROC curve hugs the diagonal and the area under the curve is 1/2.

3)for dataset 1 the logistic regression and KNN model is very good while for dataset 2 I cannot find any suitable model , the dataset 2 is randomly distributed and there's no model can predict random data well.

Appendix:

```
import numpy as np
import pandas as pd
from sklearn import linear_model
from sklearn.datasets import make_classification
from sklearn.linear_model import Lasso, Ridge
from sklearn.metrics import plot_confusion_matrix, confusion_matrix,
classification_report
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import PolynomialFeatures
from funcfile import *
df = pd.read_csv("week4_2.csv",comment='#')
#df = pd.read_csv("week4_2.csv")
print(df.head())
x = np.array(df.iloc[:, 0:2])#read the colunms as array
x1 = np.array(df.iloc[:, 0])#read the first colunm as array
x2 = np.array(df.iloc[:, 1])#read the second colunm as array
y = np.array(df.iloc[:, 2])#read the y value

fig1=plt.figure()
scattersth(x1,x2,y,"orignal data for dataset 1")
#to plot the impact of different values of degree
#model:linear_model.LogisticRegression,penalty:L2,C=1
#use K-Fold(K=5) cross-validation
#plot the mse and the mean scores of cross-validation
fig2=plt.figure()
polynomial_degrees=[2,3,4,5,6,10,12,15,40,50,60,80,100]
degree_mean_mse = []
degree_std_mse = []
mean_score_degree_range=[]
for inx1,degree in enumerate(polynomial_degrees):
    poly = PolynomialFeatures(degree=degree, interaction_only=False,include_bias=False)
    x_poly = poly.fit_transform(x)
    kf = KFold(n_splits=5)
```

```
    mse = []
    for train, test in kf.split(x_poly):
        mul_lr = linear_model.LogisticRegression(penalty='l2',C=1)
        mul_lr.fit(x_poly, y)
        y_pre = mul_lr.predict(x_poly[test])
        from sklearn.metrics import mean_squared_error
        mse.append(mean_squared_error(y_pre, y[test]))
    degree_mean_mse.append(np.mean(mse))
    degree_std_mse.append(np.std(mse))
    mean_score_degree_range.append(cross_val_score(mul_lr, x_poly, y, cv=5).mean())
plt.subplot(1,2,1)
plt.plot(polynomial_degrees,mean_score_degree_range,label='mean scores',color='red')
plt.ylabel('mean score')
plt.xlabel('polynomial degree')
plt.title("score of different polynomial degrees")
plt.legend(loc='upper right', fontsize=10)
plt.subplot(1,2,2)
plt.errorbar(polynomial_degrees, degree_mean_mse, label="MSE of cross_validation",
color='blue', yerr=degree_std_mse)
plt.ylabel('Mean square error')
plt.xlabel('polynomial degree')
plt.title("MSE of different polynomial degrees")
plt.legend(loc='upper right', fontsize=10)
#plt.show()
#to plot the impact of different values of C
#model:linear_model.LogisticRegression,penalty:L2,degree=50
#use K-Fold(K=5) cross-validation
#plot the mse and the mean scores of cross-validation
fig3=plt.figure()
crange_mean_mse = []
crange_std_mse = []
mean_score_c_range=[]
#c_range=[0.1,0.5,1,2,3,5,10,15,20,30,40,50,60]
c_range=[0.1,0.5,1,2,3,4,5,10]
for c in c_range:
    poly = PolynomialFeatures(degree=50, interaction_only=False, include_bias=False)
    x_poly = poly.fit_transform(x)
    kf = KFold(n_splits=5)
    mse = []
    for train, test in kf.split(x_poly):
        mul_lr = linear_model.LogisticRegression(penalty='l2', C=c)
        mul_lr.fit(x_poly, y)
        y_pre = mul_lr.predict(x_poly[test])
        from sklearn.metrics import mean_squared_error
        mse.append(mean_squared_error(y_pre, y[test]))
    crange_mean_mse.append(np.mean(mse))
    crange_std_mse.append(np.std(mse))
    mean_score_c_range.append(cross_val_score(mul_lr, x_poly, y, cv=5).mean())
plt.plot(c_range,mean_score_c_range,label='mean scores',color='red')
```

```
plt.errorbar(c_range, crange_mean_mse, label="MSE", color='blue', yerr=crange_std_mse)
plt.ylabel('Mean square error')
plt.xlabel('C')
plt.title("cross validation of different C values")
plt.legend(loc='upper right', fontsize=10)
#plot final choice
#(i)degree=4 C=1 penalty=L2
#(ii)degree=50 and C=1 ,penalty=L2
fig4=plt.figure()
poly_final = PolynomialFeatures(degree=50, interaction_only=False, include_bias=False)
x_poly_final = poly_final.fit_transform(x)
kf = KFold(n_splits=5)
mul_lr_final = linear_model.LogisticRegression(penalty='l2', C=1)
mul_lr_final.fit(x_poly_final, y)
y_pre_final=mul_lr_final.predict(x_poly_final)
print("The score of the model is {0}".format(mul_lr_final.score(x_poly_final,y)))
scattersth(x1,x2,y_pre_final,"prediction model:degree=50,C=1 for dataset 2")
plt.legend(loc='upper right', fontsize=10)
#(b) for KNN
#to plot the impact of different values of K for KNN model
#use K-Fold(K=5) cross-validation
#plot the mse and the mean scores of cross-validation
fig5=plt.figure()
knn_mean_mse = []
knn_std_mse = []
mean_score_k_range=[]
k_range=[2,3,4,5,6,10,15,20,30,40,50,60,70,80,90,100,200]
for k in k_range:
    kf = KFold(n_splits=5)
    mse = []
    for train, test in kf.split(x):
        from sklearn.neighbors import KNeighborsClassifier
        knn_model = KNeighborsClassifier(n_neighbors=k, weights='uniform').fit(x, y)
        y_pre = knn_model.predict(x[test])
        from sklearn.metrics import mean_squared_error
        mse.append(mean_squared_error(y_pre, y[test]))
    knn_mean_mse.append(np.mean(mse))
    knn_std_mse.append(np.std(mse))
    mean_score_k_range.append(cross_val_score(knn_model, x, y, cv=5).mean())
plt.plot(k_range,mean_score_k_range,label='mean scores',color='red')
plt.errorbar(k_range, knn_mean_mse, label="MSE", color='blue', yerr=knn_std_mse)
plt.ylabel('Mean square error')
plt.xlabel('K')
plt.title("cross validation of different K values")
plt.legend(loc='upper right', fontsize=10)
#to find the degree of polynomial
fig6=plt.figure()
polynomial_degrees=[2,3,4,5,6,10,12,15,20]
mean_score_degree_range=[]
```

```
for degree in polynomial_degrees:
    poly = PolynomialFeatures(degree=degree, interaction_only=False,include_bias=False)
    x_poly = poly.fit_transform(x)
    knn_model = KNeighborsClassifier(n_neighbors=5, weights='uniform').fit(x_poly, y)
    mean_score_degree_range.append(cross_val_score(knn_model, x_poly, y, cv=5).mean())
#this is knn model without poly features mean score of cross validation
knn_model_no_poly = KNeighborsClassifier(n_neighbors=5, weights='uniform').fit(x,y)
knn_no_poly_mean_score= cross_val_score(knn_model, x, y, cv=5).mean()
plt.axhline(y=knn_no_poly_mean_score, color='blue', label='no poly features
score',linestyle='-')
plt.plot(polynomial_degrees,mean_score_degree_range,label='mean scores of different
degree',color='red')
plt.ylabel('mean score')
plt.xlabel('polynomial degree')
plt.title("score of different polynomial degrees")
plt.legend(loc='lower right', fontsize=10)
#(c)confusion matrices
#for trained Logistic Regression and kNNclassifier.
#baseline:dummy classifier : random prediction
#final choice of logistic regression(i)degree=4 C=1 (ii)degree=50 and C=1
fig7=plt.figure()
poly_final = PolynomialFeatures(degree=50, interaction_only=False, include_bias=False)
x_poly_final = poly_final.fit_transform(x)
X_train_logistic, X_test_logistic, y_train_logistic, y_test_logistic =
train_test_split(x_poly_final, y, random_state=0)
mul_lr_final = linear_model.LogisticRegression(penalty='l2', C=1)
mul_lr_final.fit(X_train_logistic, y_train_logistic)
y_pre_logistic=mul_lr_final.predict(X_test_logistic)
disp1=plot_confusion_matrix(mul_lr_final, X_test_logistic, y_test_logistic)
print("Logistic Regression confusion matrix:")
print(disp1.confusion_matrix)
print("classification report")
print(classification_report(y_test_logistic, y_pre_logistic))
plt.title("trained logistic regression")
#KNN model (i)K=5 (i)K=30
fig8=plt.figure()
X_train_knn, X_test_knn, y_train_knn, y_test_knn = train_test_split(x, y,
random_state=0)
knn_model = KNeighborsClassifier(n_neighbors=30,
weights='uniform').fit(X_train_knn,y_train_knn)
y_pre_knn = knn_model.predict(X_test_knn)
disp2=plot_confusion_matrix(knn_model,X_test_knn,y_test_knn)
print("KNN confusion matrix:")
print(disp2.confusion_matrix)
print("classification report")
print(classification_report(y_test_knn, y_pre_knn))
plt.title("KNN")
#baseline model dummy classifier
fig9=plt.figure()
```

```
from sklearn.dummy import DummyClassifier
X_train_dummy, X_test_dummy, y_train_dummy, y_test_dummy = train_test_split(x, y,
random_state=0)
dummy = DummyClassifier(strategy="most_frequent").fit(X_train_dummy, y_train_dummy)
ydummy = dummy.predict(X_test_dummy)
disp3=plot_confusion_matrix(dummy,X_test_dummy,y_test_dummy)
print("dummy classifier confusion matrix: ")
print(disp3.confusion_matrix)
print("classification report")
print(classification_report(y_test_dummy, ydummy))
plt.title("dummy classifier")
#d ROC curve
fig10=plt.figure()
from sklearn.metrics import roc_curve
#logistic regression
fpr1, tp1, _ =
roc_curve(y_test_logistic,mul_lr_final.decision_function(X_test_logistic))
plt.plot(fpr1,tp1,label='logistic regression',c='b',linewidth=6)
#KNN
y_scores_knn = knn_model.predict_proba(X_test_knn)
fpr2, tp2, threshold1 = roc_curve(y_test_knn, y_scores_knn[:, 1])
plt.plot(fpr2,tp2,label='KNN',c='y')
#baseline
y_scores_dummy = dummy.predict_proba(X_test_dummy)
fpr3, tp3, threshold2 = roc_curve(y_test_dummy, y_scores_dummy[:, 1])
plt.plot(fpr3,tp3,label='baseline model:dummy classifier',c='r',linewidth=6)
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.plot([0,1], [0,1], color="green",linestyle="--")
plt.title("ROC curve for logistic regression")
plt.legend()
plt.show()
def scattersth(x1, x2, y, str):
    positivey_x1=[]
    positivey_x2 = []
    negaivey_x1=[]
    negaivey_x2 = []
    for i in range(len(x1)):
        if y[i] == 1:
            positivey_x1.append(x1[i])
            positivey_x2.append(x2[i])
        else:
            negaivey_x1.append(x1[i])
            negaivey_x2.append(x2[i])
    plt.scatter(positivey_x1,positivey_x2, label='y=1',color='red', s=5)
    plt.scatter(negaivey_x1, negaivey_x2, label='y=-1',color='green', s=5)
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.title(str)
```