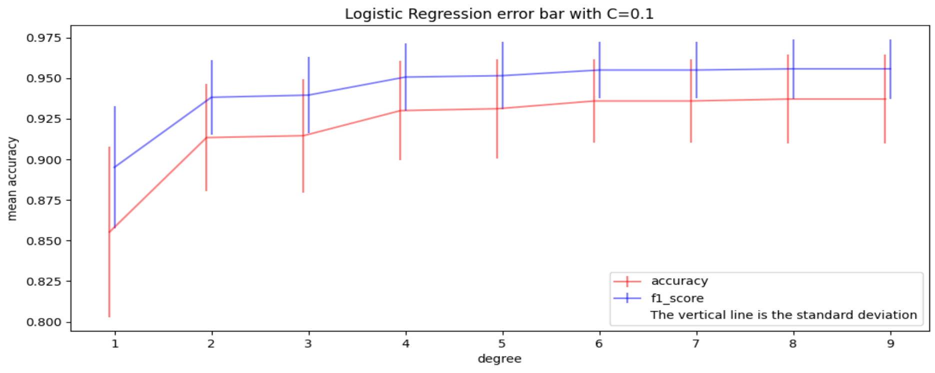
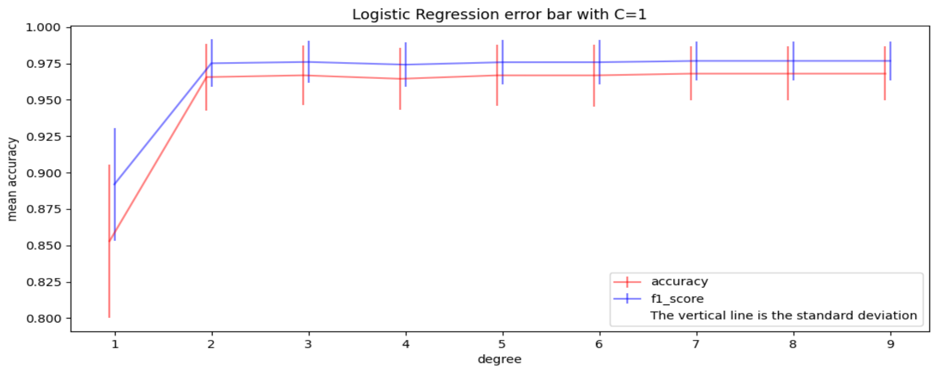
Xiangyu Zheng 21331025 CS7CS4 Machine Learning week4 assignment

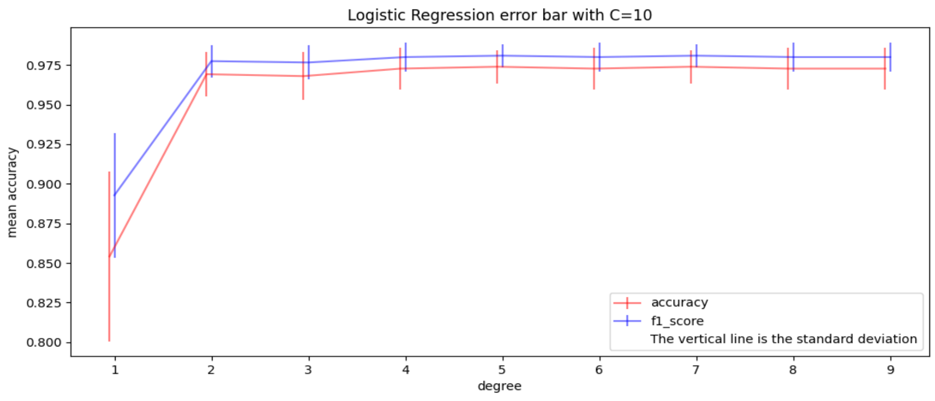
(i)(a)

I first determine the value of the maximum order of the polynomial: set C=0.1, C=1, C=10 and use ten-fold cross-validation to draw error bars of degree=1 to 9, you can get:

```
for i in range(1, 10):
    poly_train = PolynomialFeatures(degree=i).fit_transform(train)
    LR = LogisticRegression(penalty='l2', C=C, max_iter=10000,
tol=0.0001)
    scores = cross_val_score(LR, poly_train, target, cv=10)
    f1_score = cross_val_score(LR, poly_train, target, scoring='f1',
cv=10)
```

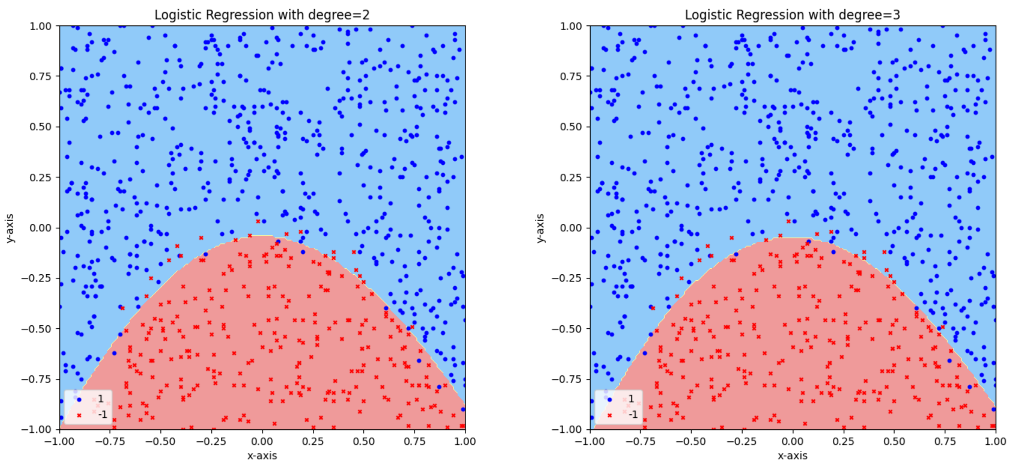






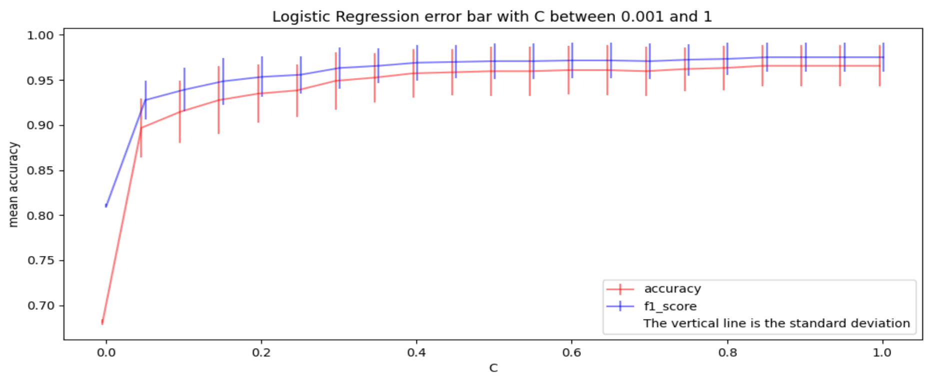
As can be seen from the above figure, through the accuracy and the value of f1_score, I initially chose degree=2 or 3, because starting from C=1 and starting from degree=2, the accuracy of the following polynomials and the value of f1 are not much different .

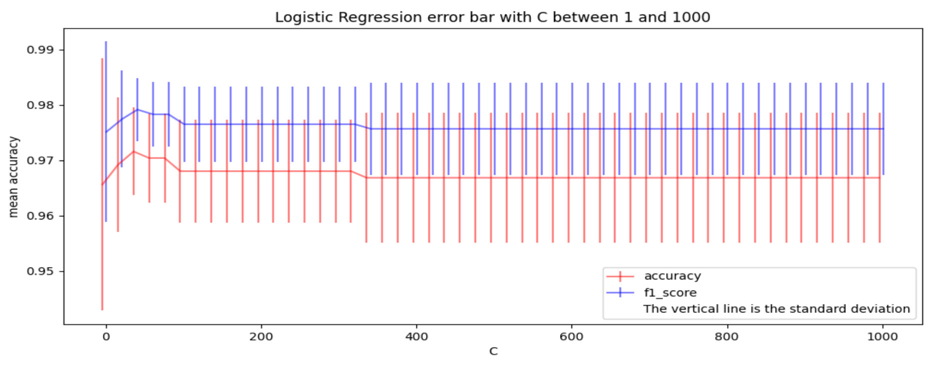
As you can see from the above figure, starting from C=1, degree=2, 3 have similar performance, so draw their scatter plot and decision boundary under C=1:



We can see that when degree=2, the performance of the classifier is good , and the decision boundary does not change when degree=3. So I chose 2 for degree.

Then I looked for the best C based on the maximum order of polynomial being 2. I set two value ranges from C=0.001 to 1, increasing by 0.05 each time and C=1 to 1000 each time increasing by 20, and you can get error bar:

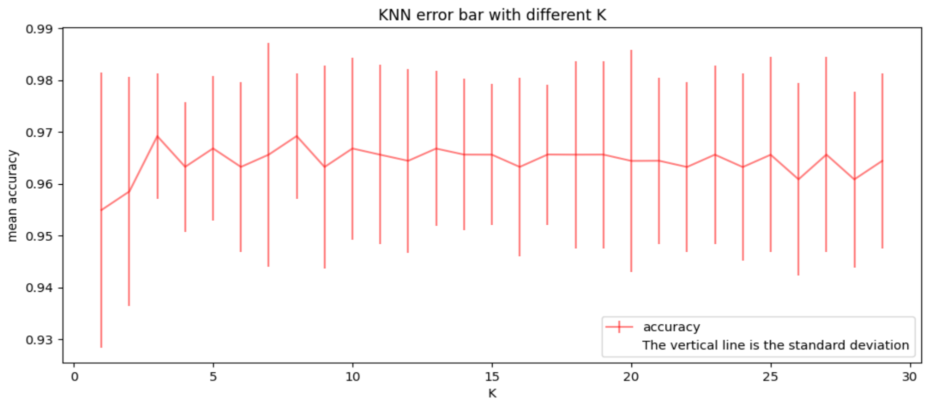




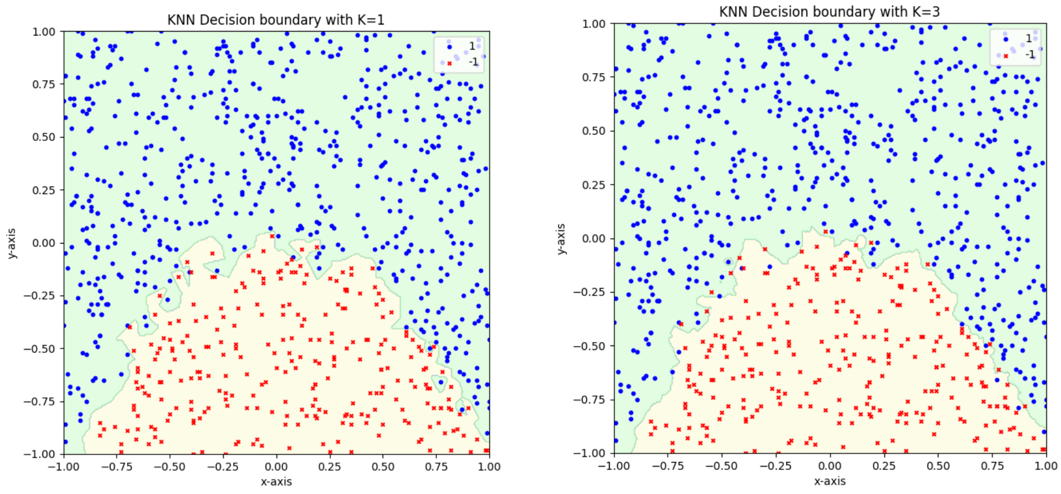
Through the comparison of the above two figures, we can know that when C=40, the accuracy and f1 values are the highest, and the standard deviation value is the smallest. So I choose C=40 and degree=2 as the best values for my model.

(b)

I first determine the range of K from 1-30, and draw cross-validation error bars. It can be determined that when k=3, the accuracy is the highest and the error is the smallest. Although there are better K values in the future, the overall trend is declining, and the larger the K, the more data needs to be calculated, and the more memory it consumes. So initially choose K=3:



We draw a scatter plot and the decision boundary of the knn model according to K=1 and k=3:



We can see that when k=1, the decision boundary fits each point -1 and 1, and there is an over-fitting phenomenon. The decision boundary of k=3 is better. According to the data of the error bar, when k=3, it is the best choice.

(c)

Three models for calculating confusion matrix:

1.Logistic regression, the heighest order of the polynomial is 2, using L2 regularization, C=40

2.KNN, K=3

3.DummyClassifier, using random prediction strategy

```
poly_features = PolynomialFeatures(degree=2)
poly_train = poly_features.fit_transform(X_train)
LR = LogisticRegression(penalty='l2', C=40)
```

```
knn = KNeighborsClassifier(n_neighbors=3)
```

```
dummy = DummyClassifier(strategy="uniform")
```

According to the confusion matrix:

True positives	False positives
False negatives	True negatives

I divided the training set into 20% of the test set, and then used the confusion_matrix function in sklearn to calculate the confusion matrix, and I could get:

```
confusion_matrix(y_test, poly_pred)
```

1.Logistic regression

True positives:70	False positives:3
False negatives:5	True negatives:133

2.KNN

True positives:70	False positives:3
False negatives:5	True negatives:133

3.DummyClassifier

True positives:33	False positives:40
False negatives:68	True negatives:70

(d)

First, according to the ROC accuracy formula:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Use the function in skleanr to get the accuracy:

```
knn_roc_auc = roc_auc_score(y_test, knn.predict(X_test))
```

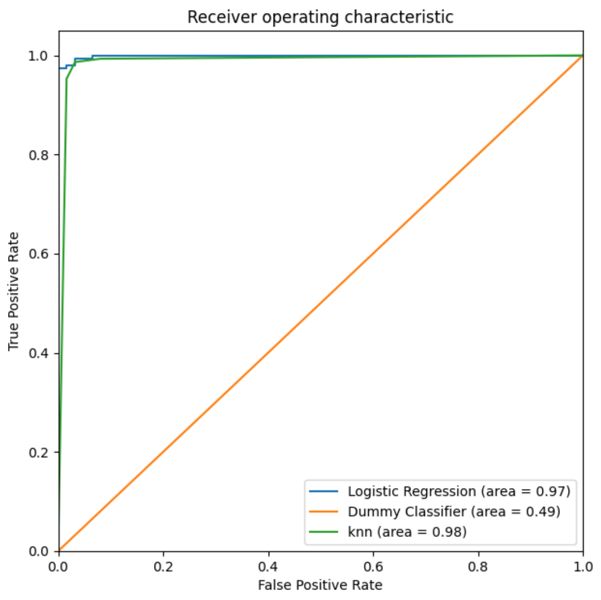
Then use the predict_proba function to get the probability of predicting the target of each value in the test set

```
y_prob_knn = knn.predict_proba(X_test)[:, 1]
```

Then use the roc_curve function and the target and prediction probability matrix of the parameter test set to get the false positive rate and true positive rate arrays

```
fpr_lr_knn, tpr_lr_knn, threshold_lr_knn = roc_curve(y_test,
y_prob_knn)
```

Finally, draw the roc curves of the three models together, and get the following figure:

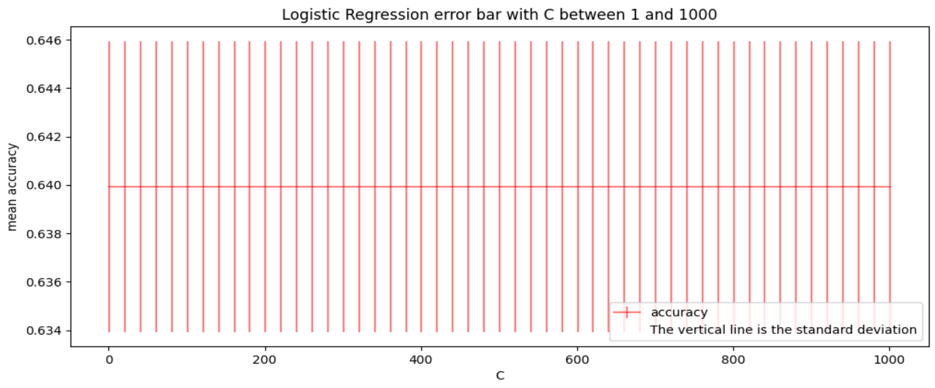


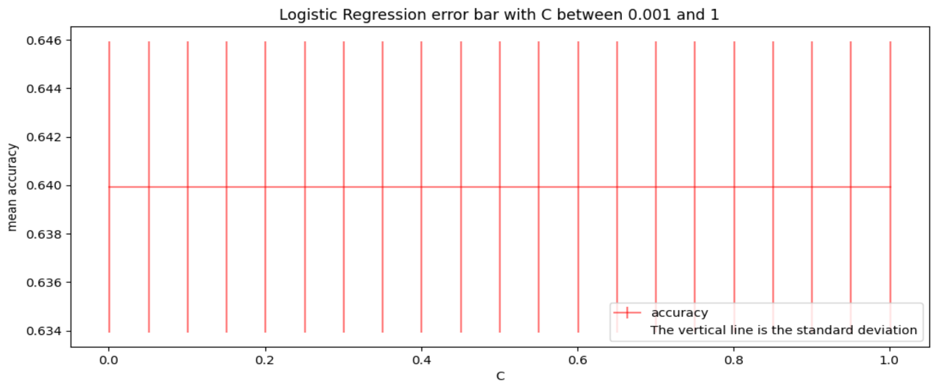
(e)

We can clearly see from the data in c and d that the performance of knn classifier and logistic regression is significantly better than that of Dummy Classifier based on random classification, but the performance of knn and logistic regression is equivalent. I would recommend the knn classifier under the data of this two classification, because the parameters of knn are relatively simple, it is only necessary to determine the value of k through cross-validation, while it is more troublesome to find the best hyperparameters through cross-validation in logistic regression.

(ii) (a)

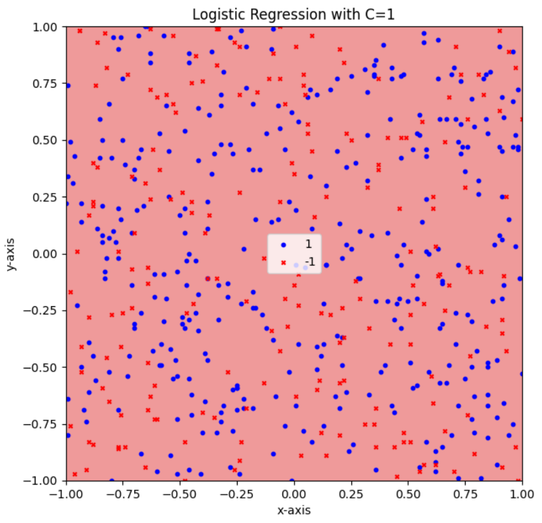
We can use the same steps as (i)(a). The best polynomial is not to use the polynomial, and to train with the original data, we can use cross-validation to get the error bar:





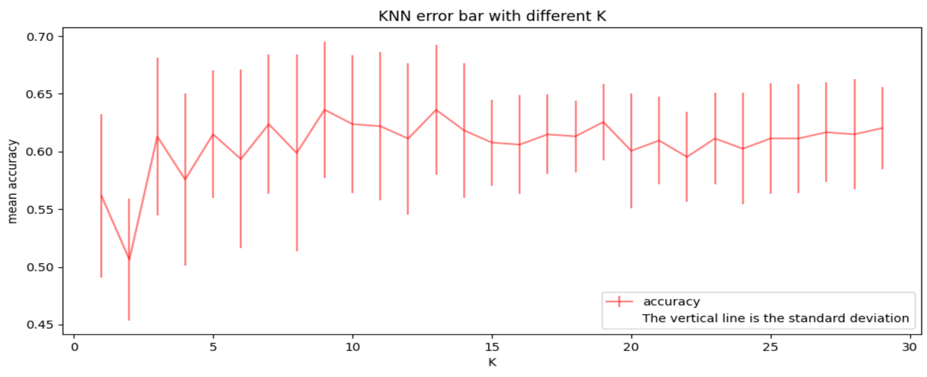
We can see that because there are too few features, C using L2 regularization seems to have lost its effect, regardless of the accuracy and error of C.

Finally, by drawing a scatter plot, we can see that the logistic regression model predicts the data to -1:

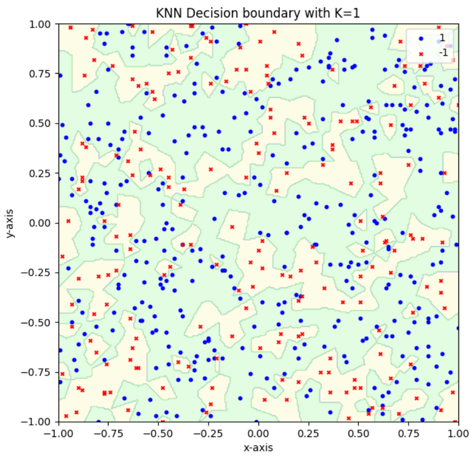


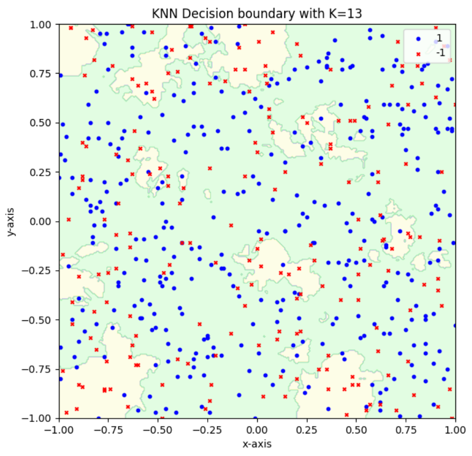
(b)

According to the same steps as (i)(b), we can get the error bar graph of KNN after cross-validation



We can see that the best value is K=13, so draw a scatter plot with k=13 and k=1:





We can see that when k=1, it still fits every training point, resulting in overfitting and decreased prediction accuracy.

(c)

I chose 3 models:

1 Logistic regression, C=1, no polynomial is used.

2. KNN, K=13

3.DummyClassifier

Following the same steps as (i)(c), we can get three confusion matrices:

1.Logistic regression

True positives:0	False positives:54
False negatives:0	True negatives:87

2.KNN

True positives:10	False positives:44
False negatives:8	True negatives:79

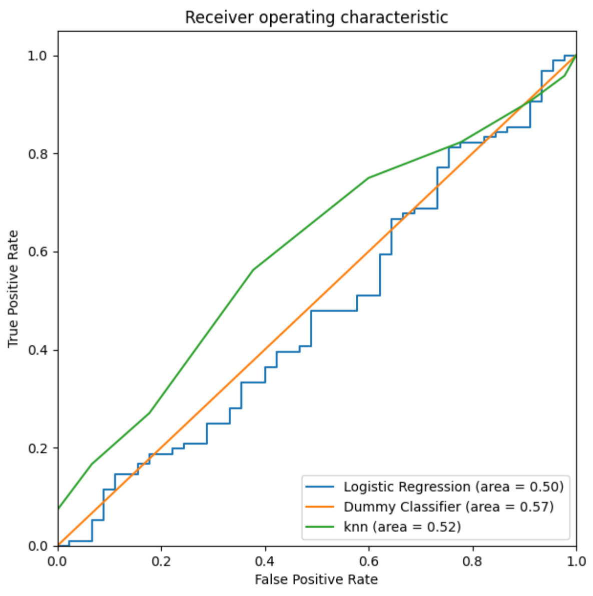
3.DummyClassifier

True positives:27	False positives:27
False negatives:36	True negatives:51

We can see that logistic regression because all predictions have become one value, resulting in both TP and FN being 0

(d)

In the same steps as (i)(d), we get the ROC curves of the three models:



(e)

We can see from the data in (c) and (d) that in this training set, compared to the training set in i, the performance of KNN and logistic regression has significantly decreased. Maybe these classifiers are not suitable for doing this For predicting the distribution of data, if you still want to choose one, you still choose KNN. Its performance is slightly better than logistic regression and baseline classifiers.

Appendix

code

ia iia identify C:

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import PolynomialFeatures
from matplotlib.transforms import Affine2D
import matplotlib.pyplot as plt

```

```python
# Get data1 from file
data = str(open("data2").read()).split('\n')
dataset = []
for d in data[1:]:
    nums = d.split(',')
    dxy = []
    dxy.append(float(nums[0]))
    dxy.append(float(nums[1]))
    dxy.append(float(nums[2]))
    dataset.append(dxy)
dataset = np.array(dataset)

train = dataset.take([0, 1], axis=1)
target = dataset[:, 2]

C = 0.001
# C = 0.1
# C = 1
# C = 10
# C = 1000
Cs, acc_means, acc_stds, f1_means, f1_stds = [], [], [], [], []
while C < 1.05:
    poly = PolynomialFeatures(degree=1)
    poly_train = poly.fit_transform(train)
    print(poly.get_feature_names())
    LR = LogisticRegression(penalty='l2', C=C, max_iter=10000, tol=0.0001)
    scores = cross_val_score(LR, poly_train, target, cv=10)
    # f1_score = cross_val_score(LR, poly_train, target, scoring='f1', cv=10)
    # print('C=' + str(C) + ' acc=' + str(scores.mean()) + ' std=' + str(scores.std()))
    Cs.append(C)
    acc_means.append(scores.mean())
    acc_stds.append(scores.std())
    # f1_means.append(f1_score.mean())
    # f1_stds.append(f1_score.std())
    C = C + 0.05

plt.figure()
fig, ax = plt.subplots(figsize=(12, 5))
plt.title('Logistic Regression error bar with C between 0.001 and 1')
plt.xlabel('C')
plt.ylabel('mean accuracy')
# trans1 = Affine2D().translate(-5, 0.0) + ax.transData
# trans2 = Affine2D().translate(0.0, 0.0) + ax.transData
plt.errorbar(Cs, acc_means, acc_stds, mfc='red', color="r",  label='accuracy', alpha=0.5)
# plt.errorbar(Cs, f1_means, f1_stds, mfc='blue', color="b", transform=trans2, label='f1_score', alpha=0.5)
plt.errorbar(Cs, acc_means, acc_stds, mfc='blue', color="b",
             label='The vertical line is the standard deviation', alpha=0.0)
plt.legend(loc=4)
plt.show()
```

ia iia identify degree:

```python
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from matplotlib.colors import ListedColormap
```

```python
import matplotlib.pyplot as plt




# Get data1 from file
data = str(open("data2").read()).split('\n')
dataset = []
px, py, nx, ny = [], [], [], []
for d in data[1:]:
    nums = d.split(',')
    dxy = []
    dxy.append(float(nums[0]))
    dxy.append(float(nums[1]))
    dxy.append(float(nums[2]))
    dataset.append(dxy)
    if nums[2] == '1':
        px.append(float(nums[0]))
        py.append(float(nums[1]))
    else:
        nx.append(float(nums[0]))
        ny.append(float(nums[1]))

dataset = np.array(dataset)

train = dataset.take([0, 1], axis=1)
target = dataset[:, 2]

degree = 1
poly_train = PolynomialFeatures(degree=degree).fit_transform(train)
LR = LogisticRegression(penalty='l2', C=1)
LR.fit(poly_train,target)
fig, ax = plt.subplots(figsize=(7, 7))


axis = [-1, 1, -1, 1]
x0, x1 = np.meshgrid(
    np.linspace(axis[0], axis[1], int((axis[1] - axis[0]) * 100)).reshape(-1, 1),
    np.linspace(axis[2], axis[3], int((axis[3] - axis[2]) * 100)).reshape(-1, 1),
)
X_new = np.c_[x0.ravel(), x1.ravel()]
X_new = PolynomialFeatures(degree=degree).fit_transform(X_new)
y_predict = LR.predict(X_new)
zz = y_predict.reshape(x0.shape)

custom_cmap = ListedColormap(['#EF9A9A', '#FFF59D', '#90CAF9'])
ax.contourf(x0, x1, zz, linewidth=1, cmap=custom_cmap)
# plt.contour(x0, x1, zz, [0])
ax.scatter(px, py, s=10, c='b', marker='o', label='1')
ax.scatter(nx, ny, s=10, c='r', marker='x', label='-1')
ax.legend()
plt.title('Logistic Regression with C=' + str(degree))
ax.set_xlabel("x-axis")
ax.set_ylabel("y-axis")
plt.show()
```

ib iib cross validation for k:

```python
import numpy as np
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
```

```python

data = str(open("data2").read()).split('\n')
dataset = []
for d in data[1:]:
    nums = d.split(',')
    dxy = []
    dxy.append(float(nums[0]))
    dxy.append(float(nums[1]))
    dxy.append(float(nums[2]))
    dataset.append(dxy)
dataset = np.array(dataset)

train = dataset.take([0, 1], axis=1)
target = dataset[:, 2]

Ks, acc_means, acc_stds, f1_means, f1_stds = [], [], [], [], []
for K in range(1,30):
    knn = KNeighborsClassifier(n_neighbors=K)
    scores = cross_val_score(knn, train, target, cv=10)
    Ks.append(K)
    acc_means.append(scores.mean())
    acc_stds.append(scores.std())

plt.figure()
fig, ax = plt.subplots(figsize=(12, 5))
ax.errorbar(Ks, acc_means, acc_stds, mfc='red', color="r", label='acc
uracy', alpha=0.5)
plt.errorbar(Ks, acc_means, acc_stds, mfc='blue', color="b",
             label='The vertical line is the standard deviation', alp
ha=0.0)
plt.title('KNN error bar with different K' )
plt.xlabel('K')
plt.ylabel('mean accuracy')
plt.legend(loc=4)
plt.show()
```

ib iib draw scatter plot:

```python
import numpy as np
from matplotlib.colors import ListedColormap
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt

data = str(open("data2").read()).split('\n')
dataset = []
px, py, nx, ny = [], [], [], []
for d in data[1:]:
    nums = d.split(',')
    dxy = []
    dxy.append(float(nums[0]))
    dxy.append(float(nums[1]))
    dxy.append(float(nums[2]))
    dataset.append(dxy)
    if nums[2] == '1':
        px.append(float(nums[0]))
        py.append(float(nums[1]))
    else:
        nx.append(float(nums[0]))
        ny.append(float(nums[1]))
dataset = np.array(dataset)

train = dataset.take([0, 1], axis=1)
target = dataset[:, 2]

```

```python

K=13
knn = KNeighborsClassifier(n_neighbors=K)
knn.fit(train,target)

fig, ax = plt.subplots(figsize=(7, 7))
xp = np.linspace(-1, 1, 300)
yp = np.linspace(-1, 1, 300)
x1, y1 = np.meshgrid(xp, yp)
xy = np.c_[x1.ravel(), y1.ravel()]
y_pred = knn.predict(xy).reshape(x1.shape)
custom_cmap = ListedColormap(['#fafab0', '#9898ff', '#a0faa0'])
ax.contourf(x1, y1, y_pred, alpha=0.3, cmap=custom_cmap)
ax.scatter(px, py, s=10, c='b', marker='o', label='1')
ax.scatter(nx, ny, s=10, c='r', marker='x', label='-1')
plt.title('KNN Decision boundary with K=' +str(K))
ax.legend(loc=1)
ax.set_xlabel("x-axis")
ax.set_ylabel("y-axis")
plt.show()
```

ic-iic confusion matrix

```python
import numpy as np
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import PolynomialFeatures

data = str(open("data2").read()).split('\n')
dataset = []
px, py, nx, ny = [], [], [], []
for d in data[1:]:
    nums = d.split(',')
    dxy = []
    dxy.append(float(nums[0]))
    dxy.append(float(nums[1]))
    dxy.append(float(nums[2]))
    dataset.append(dxy)
    if nums[2] == '1':
        px.append(float(nums[0]))
        py.append(float(nums[1]))
    else:
        nx.append(float(nums[0]))
        ny.append(float(nums[1]))
dataset = np.array(dataset)

train = dataset.take([0, 1], axis=1)
target = dataset[:, 2]

X_train, X_test, y_train, y_test = train_test_split(train, target)
knn = KNeighborsClassifier(n_neighbors=13)
knn.fit(X_train, y_train)
pred = knn.predict(X_test)
knn_m = confusion_matrix(y_test, pred)
print('confusion matrix: ', knn_m, sep='\n')

poly_features = PolynomialFeatures(degree=1)
poly_train = poly_features.fit_transform(X_train)
LR = LogisticRegression(penalty='l2', C=1)
LR.fit(poly_train, y_train)
poly_test = poly_features.fit_transform(X_test)
```

```
poly_pred = LR.predict(poly_test)
lr_m = confusion_matrix(y_test, poly_pred)
print('confusion matrix: ', lr_m, sep='\n')

dummy = DummyClassifier(strategy="uniform").fit(X_train, y_train)
dummy_pred = dummy.predict(X_test)
lr_m = confusion_matrix(y_test, dummy_pred)
print('confusion matrix: ', lr_m, sep='\n')
```

id iid draw roc curve:

```
import numpy as np
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, roc_auc_score, roc_curve
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt

data = str(open("data2").read()).split('\n')
dataset = []
px, py, nx, ny = [], [], [], []
for d in data[1:]:
    nums = d.split(',')
    dxy = []
    dxy.append(float(nums[0]))
    dxy.append(float(nums[1]))
    dxy.append(float(nums[2]))
    dataset.append(dxy)
    if nums[2] == '1':
        px.append(float(nums[0]))
        py.append(float(nums[1]))
    else:
        nx.append(float(nums[0]))
        ny.append(float(nums[1]))
dataset = np.array(dataset)

train = dataset.take([0, 1], axis=1)
target = dataset[:, 2]

X_train, X_test, y_train, y_test = train_test_split(train, target)

poly_features = PolynomialFeatures(degree=1)
poly_train = poly_features.fit_transform(X_train)
LR = LogisticRegression(penalty='l2', C=1)
LR.fit(poly_train, y_train)

knn = KNeighborsClassifier(n_neighbors=13)
knn.fit(X_train, y_train)

dummy = DummyClassifier(strategy="uniform").fit(X_train, y_train)
# dummy_pred = dummy.predict(X_test)


logit_roc_auc = roc_auc_score(y_test, LR.predict(poly_features.fit_transform(X_test)))
y_prob = LR.predict_proba(poly_features.fit_transform(X_test))[:, 1]

fpr_lr, tpr_lr, threshold_lr = roc_curve(y_test, y_prob)


dummy_roc_auc = roc_auc_score(y_test, dummy.predict(X_test))
```

```python
y_prob_dummy = dummy.predict_proba(X_test)[:, 1]
fpr_lr_dummy, tpr_lr_dummy, threshold_lr_dummy = roc_curve(y_test, y_prob_dummy)


knn_roc_auc = roc_auc_score(y_test, knn.predict(X_test))
y_prob_knn = knn.predict_proba(X_test)[:, 1]
print(y_prob_knn)
fpr_lr_knn, tpr_lr_knn, threshold_lr_knn = roc_curve(y_test, y_prob_knn)

plt.figure()
fig,ax = plt.subplots(figsize=(7,7))
ax.plot(fpr_lr, tpr_lr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
ax.plot(fpr_lr_dummy, tpr_lr_dummy, label='Dummy Classifier (area = %0.2f)' % dummy_roc_auc)
ax.plot(fpr_lr_knn, tpr_lr_knn, label='knn (area = %0.2f)' % knn_roc_auc)
# ax.plot([0, 1], [0, 1], 'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```