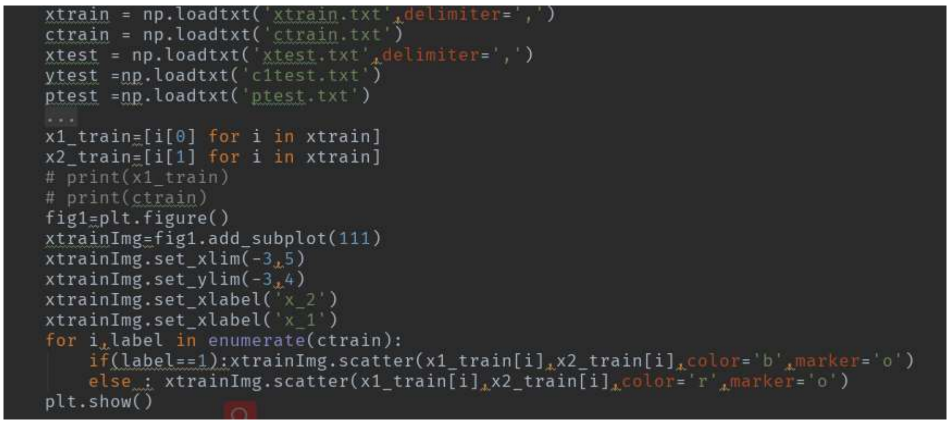
Binary Classification

1. 实验要求:

1. Use linear regression with basis functions, you can choose any basis functions as you like
   a) Calculate the least squares solution
   b) Calculate the ridge regression solution with different choices of λ
2. Use logistic regression with basis functions, you can choose any basis functions as you like
   a) Calculate the maximum likelihood solution
   b) Use cross validation (within training data) to find out an optimal set of basis functions; for example, you can select from polynomials of different orders, and find out which order performs the best using cross validation; verify whether your choice is correct using the test data
3. If you fulfill the basic requirements, you can get at most 7/10
4. Try to do the following to get additional score (at most 3/10)
   a) Use lasso regression (+0.5)
   b) For linear regression: Try different sets of basis functions, and find which one is better (+1)
   c) Use Fisher’ s LDA (+0.5)
   d) For logistic regression: Implement Newton-Raphson method by yourself, and use it in your experiments (+0.5)
   e) For logistic regression: Use the Bayesian approach (+1)
   f) Try to divide the training data into multiple classes (e.g. using k-means), and train a multi-class classifier accordingly, use it on the test data, and convert the multi-class results into binary classes; show the performance (+3)

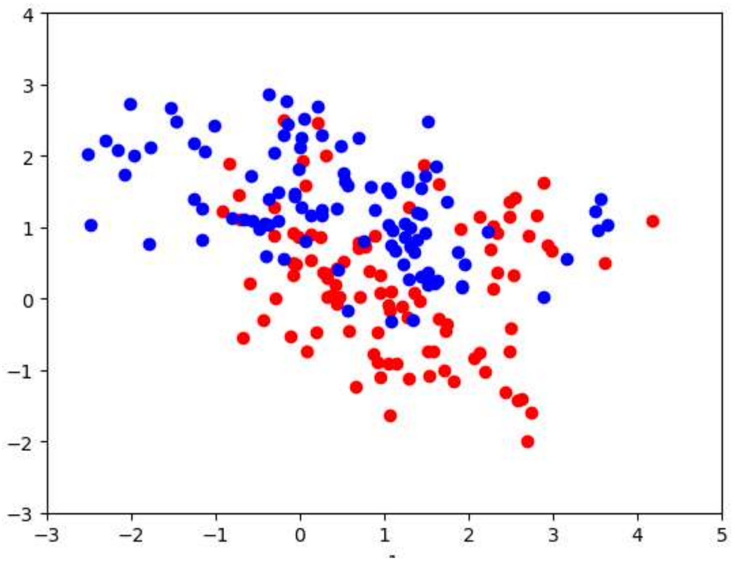
具体实验步骤:

首先我们先把数据加载进去，并且可视化数据：



```
xtrain = np.loadtxt('xtrain.txt',delimiter=',')
ctrain = np.loadtxt('ctrain.txt')
xtest = np.loadtxt('xtest.txt',delimiter=',')
ytest =np.loadtxt('c1test.txt')
ptest =np.loadtxt('ptest.txt')
...
x1_train=[i[0] for i in xtrain]
x2_train=[i[1] for i in xtrain]
# print(x1_train)
# print(ctrain)
fig1=plt.figure()
xtrainImg=fig1.add_subplot(111)
xtrainImg.set_xlim(-3,5)
xtrainImg.set_ylim(-3,4)
xtrainImg.set_xlabel('x_2')
xtrainImg.set_xlabel('x_1')
for i,label in enumerate(ctrain):
    if(label==1):xtrainImg.scatter(x1_train[i],x2_train[i],color='b',marker='o')
    else : xtrainImg.scatter(x1_train[i],x2_train[i],color='r',marker='o')
plt.show()
```

结果：



要求一，线性回归

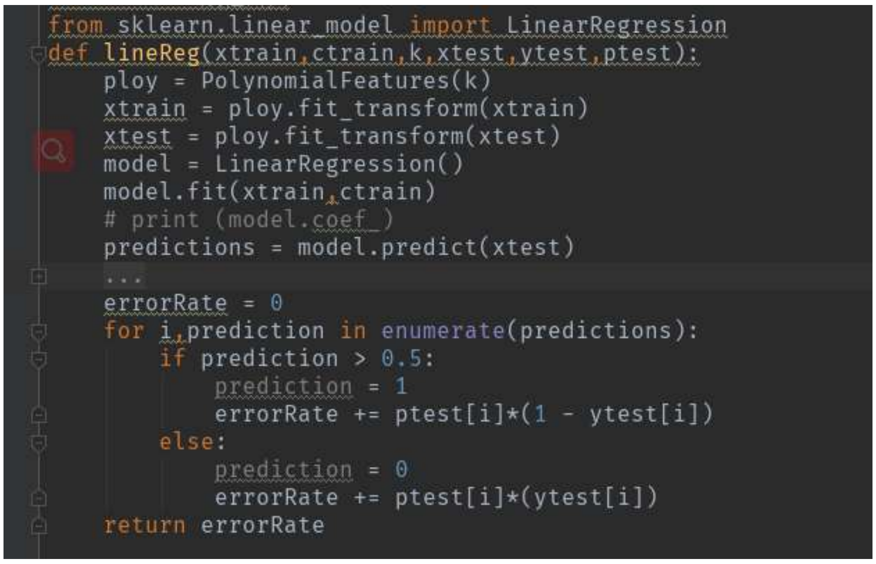
通过基带函数训练一个最小误差的线性回归器，并且选取多项式函数为基函数，并且通过简单的交叉验证，来验证选取最优的阶数。通过计算错误率，来判断不同的阶数错误率为多少。

错误率计算公式如下：

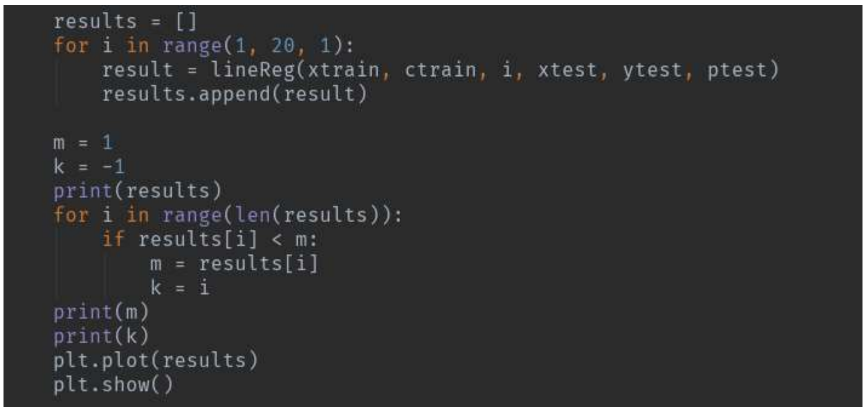
$$ErrorRate = \sum_{x \in \{x \text{ is classified as } FALSE\}} p(x)p(T|x) + \sum_{x \in \{x \text{ is classified as } TRUE\}} p(x)(1 - p(T|x))$$

where $p(x)$ is the probability of appearance (ptest.txt) and $p(T|x)$ is the probability of belonging to class 1 (c1test.txt)

具体代码如下：

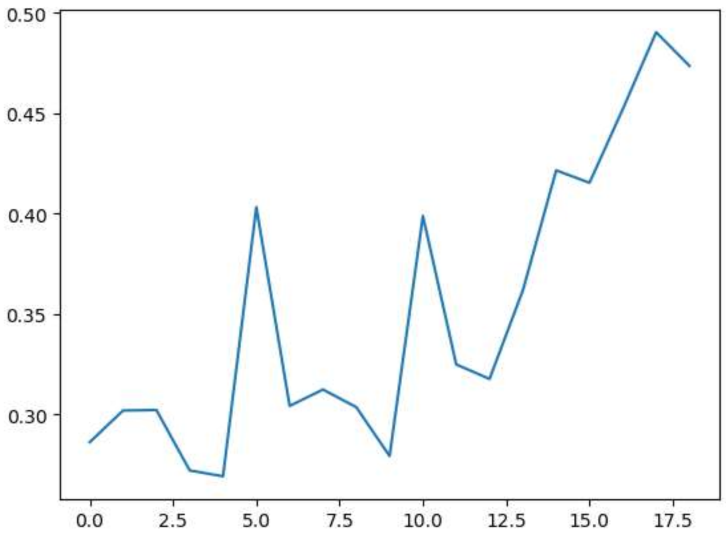


```python
from sklearn.linear_model import LinearRegression
def lineReg(xtrain,ctrain,k,xtest,ytest,ptest):
    ploy = PolynomialFeatures(k)
    xtrain = ploy.fit_transform(xtrain)
    xtest = ploy.fit_transform(xtest)
    model = LinearRegression()
    model.fit(xtrain,ctrain)
    # print (model.coef_)
    predictions = model.predict(xtest)
    ...
    errorRate = 0
    for i,prediction in enumerate(predictions):
        if prediction > 0.5:
            prediction = 1
            errorRate += ptest[i]*(1 - ytest[i])
        else:
            prediction = 0
            errorRate += ptest[i]*(ytest[i])
    return errorRate
```



```
results = []
for i in range(1, 20, 1):
    result = lineReg(xtrain, ctrain, i, xtest, ytest, ptest)
    results.append(result)

m = 1
k = -1
print(results)
for i in range(len(results)):
    if results[i] < m:
        m = results[i]
        k = i
print(m)
print(k)
plt.plot(results)
plt.show()
```

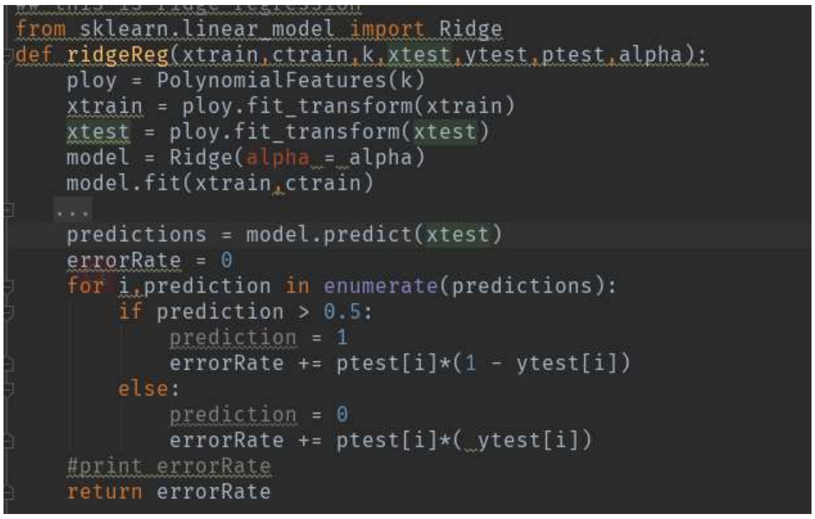
运行结果：从结果种我们可以看出。在 4 得到的误差结果最小。所以我们选取 4 阶多项式基函数。



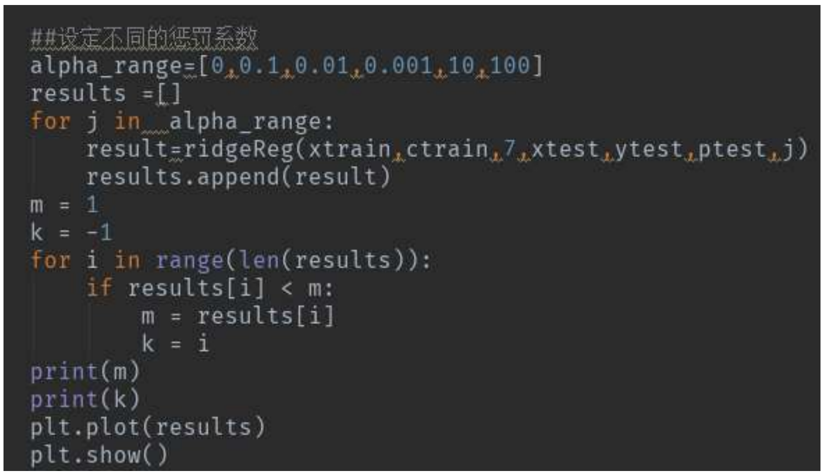
要求二。对于不同的惩罚系数选择使用岭回归

通过上图我们知道在 7 阶的时候发生过拟合，在这里我们主要考虑不同的惩罚系数对于结果的影响，所以我们设定惩罚系数有以下几个 alpha_range=[0, 0. 1, 0. 01, 0. 001, 10, 100]；然后在通过计算错误率来选取最适合的惩罚系数。

具体代码如下：

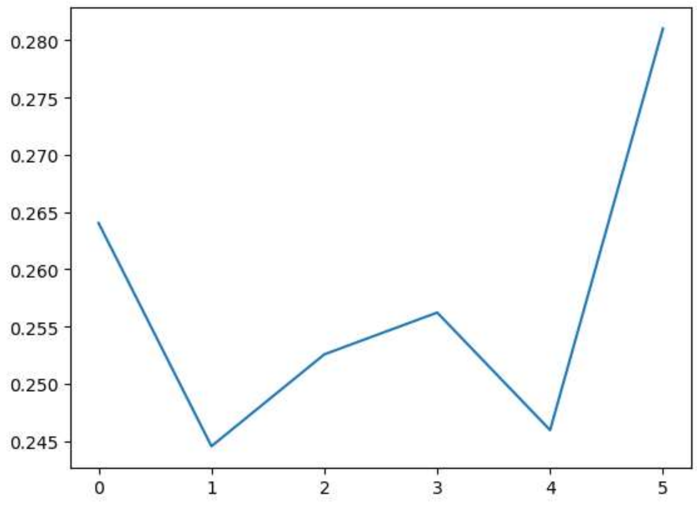


```
from sklearn.linear_model import Ridge
def ridgeReg(xtrain,ctrain,k,xtest,ytest,ptest,alpha):
    ploy = PolynomialFeatures(k)
    xtrain = ploy.fit_transform(xtrain)
    xtest = ploy.fit_transform(xtest)
    model = Ridge(alpha = alpha)
    model.fit(xtrain,ctrain)
  ...
    predictions = model.predict(xtest)
    errorRate = 0
    for i,prediction in enumerate(predictions):
        if prediction > 0.5:
            prediction = 1
            errorRate += ptest[i]*(1 - ytest[i])
        else:
            prediction = 0
            errorRate += ptest[i]*( ytest[i])
    #print errorRate
    return errorRate
```



```python
##设定不同的惩罚系数
alpha_range=[0,0.1,0.01,0.001,10,100]
results =[]
for j in  alpha_range:
    result=ridgeReg(xtrain,ctrain,7,xtest,ytest,ptest,j)
    results.append(result)
m = 1
k = -1
for i in range(len(results)):
    if results[i] < m:
        m = results[i]
        k = i
print(m)
print(k)
plt.plot(results)
plt.show()
```

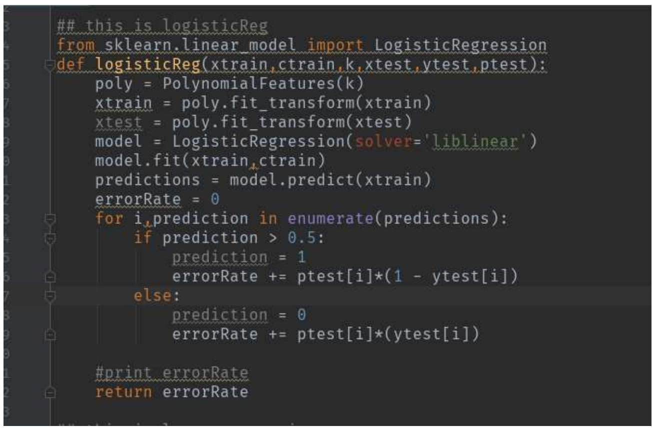
实验结果：从实验结果种我们可以看到，在惩罚系数为 0.1 的时候错误率最低，错误率是：0.24453



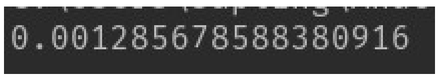
要求三。实现 logistics 回归：

 对于 logitcis 回归，选用多项式基函数，选用之前的 5 阶基函数，利用训练数据集当作测试数据集。

具体代码如下：



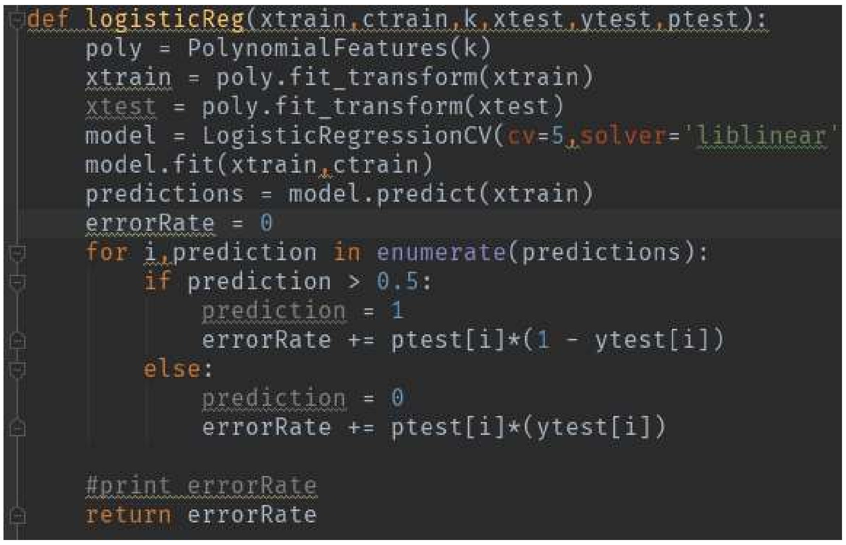
```python
## this is logisticReg
from sklearn.linear_model import LogisticRegression
def logisticReg(xtrain,ctrain,k,xtest,ytest,ptest):
    poly = PolynomialFeatures(k)
    xtrain = poly.fit_transform(xtrain)
    xtest = poly.fit_transform(xtest)
    model = LogisticRegression(solver='liblinear')
    model.fit(xtrain,ctrain)
    predictions = model.predict(xtrain)
    errorRate = 0
    for i,prediction in enumerate(predictions):
        if prediction > 0.5:
            prediction = 1
            errorRate += ptest[i]*(1 - ytest[i])
        else:
            prediction = 0
            errorRate += ptest[i]*(ytest[i])

    #print errorRate
    return errorRate
```

计算的误差结果为: 

```
0.001285678588380916
```

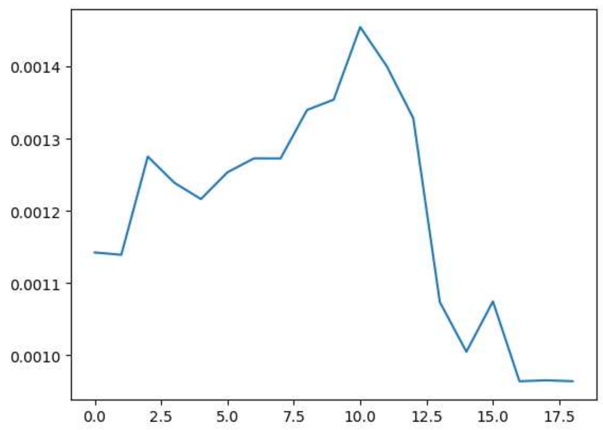
要求四: 利用交叉验证(在训练数据中)找出一组最优基函数;例如，您可以从不同阶数的多项式中进行选择，并使用交叉验证来找出哪个阶数执行得最好;使用测试数据验证您的选择是否正确。

通过训练数据集找出一组最有的基函数，我们这里采集多项式基函数作为基函数，并且设定不同的阶数，得到最有的结果，一开始我们先用我们的训练集作为我们的测试机来验证在多少阶的时候得到最优解。然后在用测试数据集来验证我们得到的模型是否为最好。这里我们采用 5 折交叉验证。具体代码如下：

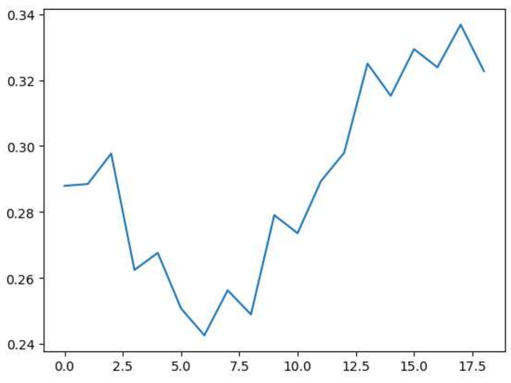


```
def logisticReg(xtrain,ctrain,k,xtest,ytest,ptest):
    poly = PolynomialFeatures(k)
    xtrain = poly.fit_transform(xtrain)
    xtest = poly.fit_transform(xtest)
    model = LogisticRegressionCV(cv=5,solver='liblinear'
    model.fit(xtrain,ctrain)
    predictions = model.predict(xtrain)
    errorRate = 0
    for i,prediction in enumerate(predictions):
        if prediction > 0.5:
            prediction = 1
            errorRate += ptest[i]*(1 - ytest[i])
        else:
            prediction = 0
            errorRate += ptest[i]*(ytest[i])

    #print errorRate
    return errorRate
```

实验结果;我们在 16 阶的时候得到的结果最优，下面我们更改测试集。

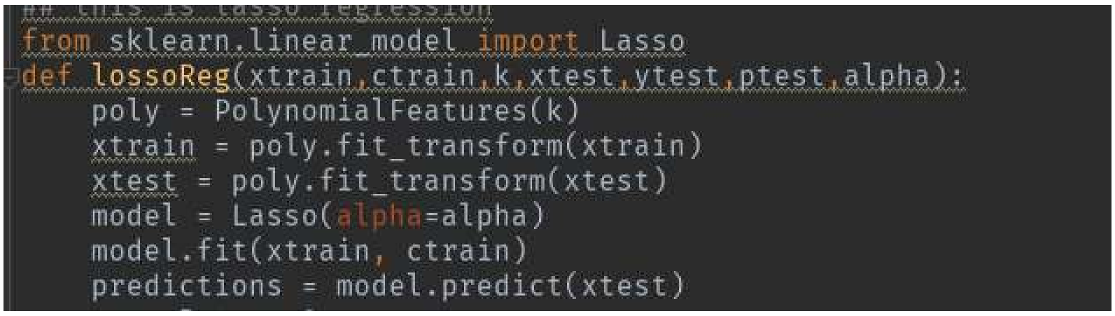


测试集结果：在 6 阶的时候为最大。可能的原因是，训练集数据量太少，导致训练出来的模型不是最优的。

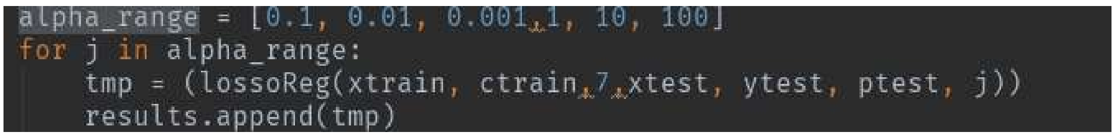


要求五。实现 lasso 回归：

对于不同的惩罚系数，选择最优的惩罚系数，基函数选择 7 阶多项式。
代码如下

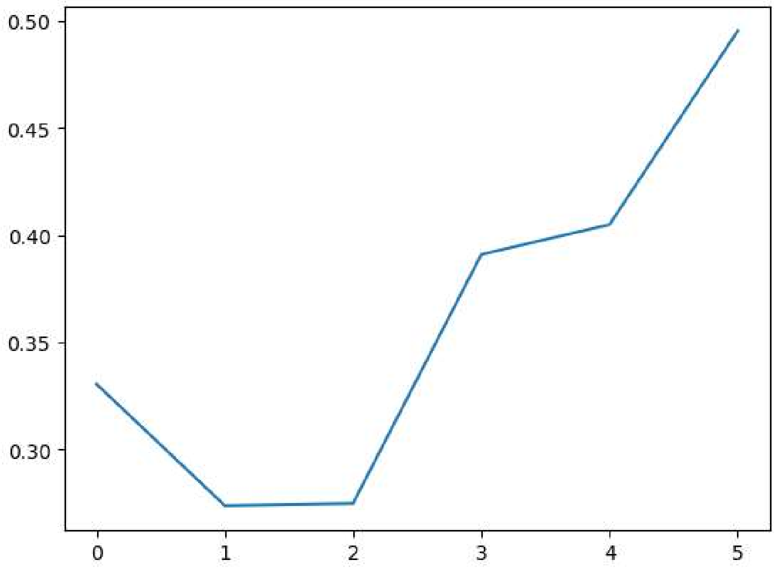


```
## this is lasso regression
from sklearn.linear_model import Lasso
def lossoReg(xtrain,ctrain,k,xtest,ytest,ptest,alpha):
    poly = PolynomialFeatures(k)
    xtrain = poly.fit_transform(xtrain)
    xtest = poly.fit_transform(xtest)
    model = Lasso(alpha=alpha)
    model.fit(xtrain, ctrain)
    predictions = model.predict(xtest)
```



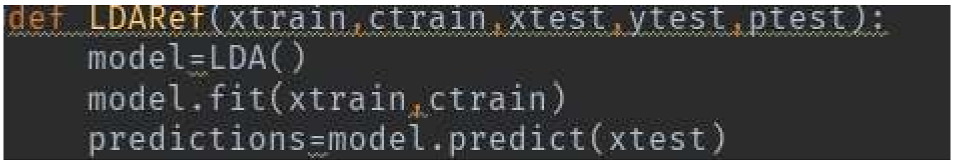
```
alpha_range = [0.1, 0.01, 0.001,1, 10, 100]
for j in alpha_range:
    tmp = (lossoReg(xtrain, ctrain,7,xtest, ytest, ptest, j))
    results.append(tmp)
```

实验结果：惩罚系数为 0.01 的是最优。



要求六。实现 LDA：

具体代码：



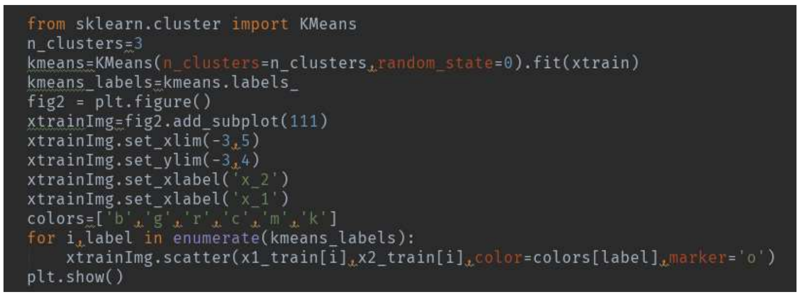
```
def LDARef(xtrain,ctrain,xtest,ytest,ptest):
    model=LDA()
    model.fit(xtrain,ctrain)
    predictions=model.predict(xtest)
```

实验结果:计算出来的误差为 0.2862

要求 7：k-means

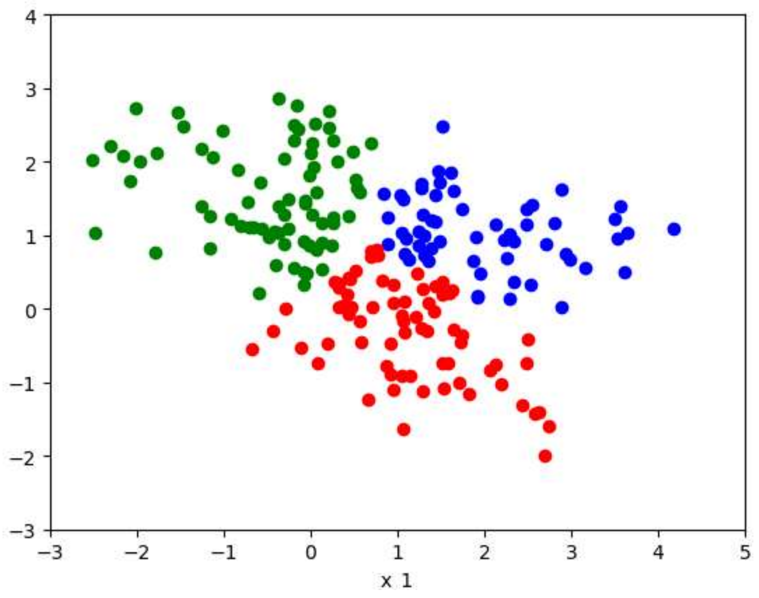
首先我们通过选取合适的 k 值对数据集进行聚类。并画出结果图。我们先选取 k 值位 3：

代码如下：



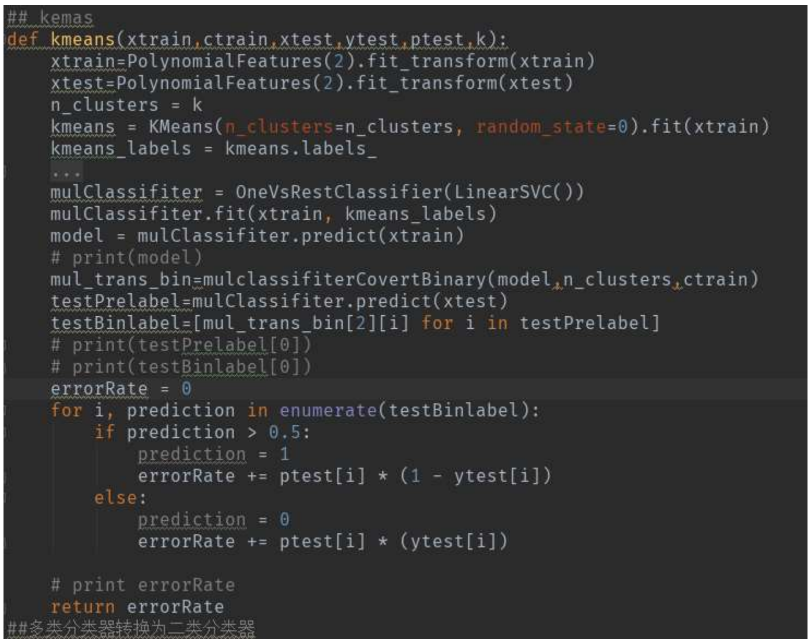
```
from sklearn.cluster import KMeans
n_clusters=3
kmeans=KMeans(n_clusters=n_clusters,random_state=0).fit(xtrain)
kmeans_labels=kmeans.labels_
fig2 = plt.figure()
xtrainImg=fig2.add_subplot(111)
xtrainImg.set_xlim(-3,5)
xtrainImg.set_ylim(-3,4)
xtrainImg.set_xlabel('x_2')
xtrainImg.set_xlabel('x_1')
colors=['b','g','r','c','m','k']
for i,label in enumerate(kmeans_labels):
    xtrainImg.scatter(x1_train[i],x2_train[i],color=colors[label],marker='o')
plt.show()
```

结果如下：

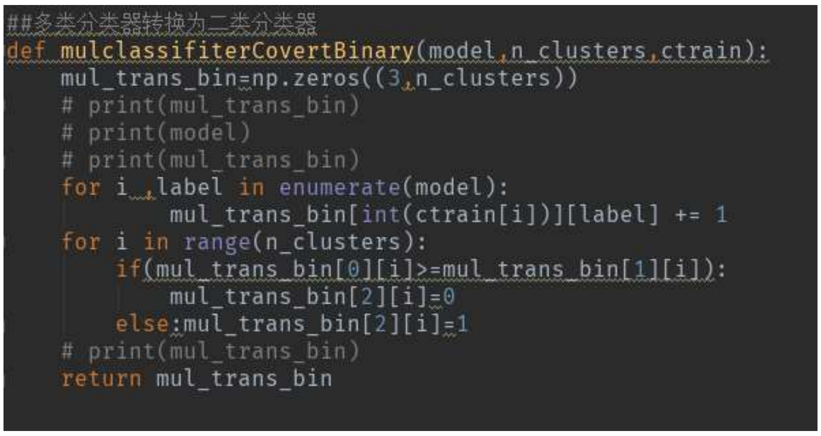


然后我们构建多类分类器， mulClassifier = OneVsRestClassifier(LinearSVC())。采用这个函数直接构建一个基于 SVM 的线性多类分类器。通过将多累分类器转换成二类分类器：假设多分类器已经将数据划分成了 3 类，即 D1={x|preLabel(x)=C1}， D2={x|preLabel(x)=C2}，D3={x|preLabel(x)=C3}，则在每一类数据 Di 中， 我们根据原始数据的 True-False 标签继续训练一个二分类器。则整个二分类任务变成了“分治任务”，即先做多分类，然后在每一类中做二分类。最后在二类分类器上做测试。对我们的测试数据集进行测试。更具不同的 k 均值做测试，得到最适合的 k 值

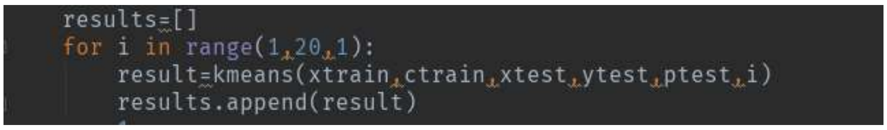
具体代码如下：



```
## kemas
def kmeans(xtrain,ctrain,xtest,ytest,ptest,k):
    xtrain=PolynomialFeatures(2).fit_transform(xtrain)
    xtest=PolynomialFeatures(2).fit_transform(xtest)
    n_clusters = k
    kmeans = KMeans(n_clusters=n_clusters, random_state=0).fit(xtrain)
    kmeans_labels = kmeans.labels_
    ...
    mulClassifiter = OneVsRestClassifier(LinearSVC())
    mulClassifiter.fit(xtrain, kmeans_labels)
    model = mulClassifiter.predict(xtrain)
    # print(model)
    mul_trans_bin=mulclassifiterCovertBinary(model,n_clusters,ctrain)
    testPrelabel=mulClassifiter.predict(xtest)
    testBinlabel=[mul_trans_bin[2][i] for i in testPrelabel]
    # print(testPrelabel[0])
    # print(testBinlabel[0])
    errorRate = 0
    for i, prediction in enumerate(testBinlabel):
        if prediction > 0.5:
            prediction = 1
            errorRate += ptest[i] * (1 - ytest[i])
        else:
            prediction = 0
            errorRate += ptest[i] * (ytest[i])

    # print errorRate
    return errorRate
##多类分类器转换为二类分类器
```



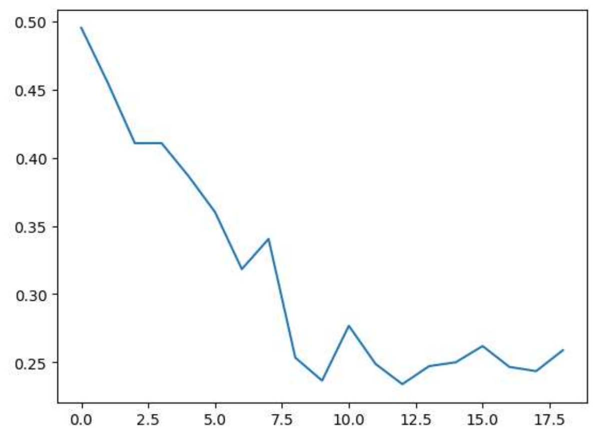
```
##多类分类器转换为二类分类器
def mulclassifiterCovertBinary(model,n_clusters,ctrain):
    mul_trans_bin=np.zeros((3,n_clusters))
    # print(mul_trans_bin)
    # print(model)
    # print(mul_trans_bin)
    for i,label in enumerate(model):
            mul_trans_bin[int(ctrain[i])][label] += 1
    for i in range(n_clusters):
        if(mul_trans_bin[0][i]>=mul_trans_bin[1][i]):
            mul_trans_bin[2][i]=0
        else:mul_trans_bin[2][i]=1
    # print(mul_trans_bin)
    return mul_trans_bin
```



```
    results=[]
    for i in range(1,20,1):
        result=kmeans(xtrain,ctrain,xtest,ytest,ptest,i)
        results.append(result)
```

。

实验结果：根据实验结果我们得到，当 k 值为 12 的时候，得到的误差率最小，为 0.233914331。



参考文献

[1]李航 统计学习方法