دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

# درس یادگیری ماشین

## تمرین دوم

| | |
|---|---|
| نام و نام خانوادگی | **میلاد محمدی** |
| شماره دانشجویی | **810100462** |
| تاریخ ارسال گزارش | **1401/08/25** |

# سوال اول

$$R = \frac{1}{2N}\left||y - X\theta|\right|_2^2 + \theta^T H\theta + \theta^T\theta + a^T\theta$$

ابتدا رابطه بالا را ساده‌سازی می‌کنیم:

$$\left||y - X\theta|\right|_2^2 = (y - X\theta)^T(y - X\theta) = (y^T - \theta^T X^T)(y - X\theta) =$$

$$(y^T y - y^T X\theta - \theta^T X^T y + \theta^T X^T X\theta)$$

با توجه به اینکه $\theta^T X^T y$ اسکالر می‌باشد، به جای آن می‌توان Transpose عبارت را قرار داد:

$$\left||y - X\theta|\right|_2^2 = y^T y - 2y^T X\theta + \theta^T X^T X\theta$$

$$R = \frac{1}{2N}(y^T y - 2y^T X\theta + \theta^T X^T X\theta) + \theta^T H\theta + \theta^T\theta + a^T\theta$$

حال گرادیان رابطه بالا نسبت به $\theta$ را حساب می‌کنیم:

به طور کلی در مشتق‌گیری رابطه $X^T AX = (A + A^T)X$ برقرار است:

$y^T y$ scalar $\rightarrow$ 0

$$-2y^T X\theta \rightarrow -2y^T X$$

$$\theta^T X^T X\theta = \left((X^T X) + (X^T X)^T\right)\theta = 2X^T X\theta$$

$$\theta^T H\theta = (H + H^T)\theta = 2H\theta$$

$$\theta^T\theta = 2\theta$$

$$a^T\theta = a$$

حال کل رابطه به صورت زیر است:

$$\frac{1}{2N}(2X^T X\theta - 2\mathrm{X}^{\mathrm{T}} y) + 2H\theta + 2\theta + a = 0$$

$$\frac{1}{N}(X^T X\theta) - \frac{1}{N}(\mathrm{X}^{\mathrm{T}} y) + 2H\theta + 2\theta + a = 0$$

$$\theta = \left(\frac{1}{N}(\mathrm{X}^{\mathrm{T}} y) - a\right)\left(\frac{1}{N}(\mathrm{X}^{\mathrm{T}} X) + 2H + 2\right)^{-1}$$

# سوال دوم

قسمت الف)

**L1 regularization (Lasso):** L1 regularization adds a penalty ($\alpha \sum_{i=1}^{n} |w_i|$ )to the loss function. Since each non-zero coefficient adds to the penalty, it forces weak features to have zero as coefficients. Thus, L1 regularization produces sparse solutions, inherently performing feature selection.

**L2 regularization (Ridge regression):** L2 regularization (called ridge regression for linear regression) adds the L2 norm penalty ($\alpha \sum_{i=1}^{n} w_i^2$) to the loss function. Since the coefficients are squared in the penalty expression, it has a different effect from L1-norm, namely it forces the coefficient values to be spread out more equally. For correlated features, it means that they tend to get similar coefficients.

## Differences:

**1. L1 regularization** penalizes the sum of absolute values of the weights, whereas **L2 regularization** penalizes the sum of squares of the weights.

**2. L1 regularization** tries to estimate the median of the data while the **L2 regularization** tries to estimate the mean of the data to avoid overfitting.

**3. L1 regularization** helps in feature selection by eliminating the features that are not important. This is helpful when the number of feature points are large in number.

**4. L1 regularization** solution is sparse. The **L2 regularization** solution is non-sparse.

**5. L2 regularization** doesn't perform feature selection, since weights are only reduced to values near 0 instead of 0**. L1 regularization** has built-in feature selection.

**6. L1 regularization** is robust to outliers, **L2 regularization** is not.

## قسمت ب)

روش نیوتن برای بهینه‌سازی به صورت زیر است (با فرض پارمتر β):

$$\beta\hat{} \;=\; \beta_0 \;-\; \left[H\big(J(\beta)\big)\right]^{-1} \nabla_{\beta}\, J(\beta)$$

$$\nabla_w\, J(\beta) \;=\; 2X^TX\beta \;-\; 2X^TY \;+\; 2\lambda\beta \;=\; 2[(X^TX \;+\; \lambda I)\beta_0 - \;X^TY] \qquad (1)$$

$$H(J(\beta)) \;=\; \nabla_{\beta}^2\, J(\beta) \;=\; 2X^TX \;+\; 2\lambda I \;=\; 2(X^TX \;+\; \lambda I) \qquad (2)$$

حال با جایگذاری روابط 1 و 2 در رابطه اولیه:

$$\beta\hat{} \;=\; \beta_0 - \;(2(X^TX \;+\; \lambda I))^{-1}\, 2[(X^TX \;+\; \lambda I)\beta_0 - \;X^TY] \;\; =$$

$$\beta_0 - \tfrac{1}{2}\,(X^TX \;+\; \lambda I)^{-1} * 2(X^TX \;+\; \lambda I)\beta_0 + \tfrac{1}{2}\,(X^TX \;+\; \lambda I)^{-1} * 2\; X^TY \;\; =$$

$$\beta_0 - \;\beta_0 + \;(X^TX \;+\; \lambda I)^{-1} * X^TY$$

با جایگزین کردن A و X به رابطه زیر می‌رسیم :

$$\beta\hat{} = (A^TA \;+\; \lambda I)^{-1} * A^TY$$

## سوال سوم

قسمت الف)

با توجه به اصل جمع احتمال:

$$P(Y = y_k|X) = 1 - \sum_{k=1}^{k-1} P(Y = y_k|X)$$

در طبقه‌بندی باینری داریم:

$$P(Y = y_k|X) = \frac{1}{1 + \sum_{k=1}^{k-1} \exp\left(w_{k_0} + \sum_{i=1}^{d} w_{k_i} X_i\right)}$$

برای حالت چندکلاسه نیز می‌توانیم:

$$P(Y = y_k|X) = \frac{\exp\left(w_{k_0} + \sum_{i=1}^{d} w_{k_i} X_i\right)}{1 + \sum_{k=1}^{k-1} \exp\left(w_{k_0} + \sum_{i=1}^{d} w_{k_i} X_i\right)}$$

قسمت ب) قانون طبقه‌بندی به این صورت است که برچسب با بیشترین احتمال انتخاب می‌شود:

$$k^* = arg\ max\ P(Y = y_k|X) \qquad k\ \epsilon\ 1, 2, \dots.k$$

## سوال چهارم

قسمت الف)

$$\beta_1 = \frac{\sum_{i=1}^{n} X_i Y_i}{\sum_{i=1}^{n} X_i^2} = \frac{4*31+9*58+65*10+14*73+4*37+7*44+12*60+22*91+21*1+17*84}{4^2+9^2+10^2+14^2+4^2+7^2+12^2+22^2+1+17^2} = 5.04$$

$$\bar{x} = \frac{4+9+10+14+4+7+12+22+1+17}{10} = 10$$

$$\bar{y} = \frac{31+58+65+73+37+44+60+91+21+84}{10} = 56.4$$

$$\beta_0 = \bar{y} - \beta_1 \bar{x} = 56.4 - 10 * 5.04 = 6$$

$SSE = \sum_{i=1}^{n}(y_i - \hat{y})^2 = \sum(23.42) + (44.08) + (73.96) + (12.67) + (117.50) + (7.39) + (41.99) + (699.74) + (99.20) + (58.98) = 1149$

$$\sigma^2 = \frac{SSE}{n-2} = 143.62$$

قسمت ب)

$S_{xx} = \sum_{i=1}^{n}(x_i - \bar{x})^2 = \sum(36) + (1) + (0) + (16) + (36) + (9) + (4) + (144) + (81) + (49) = 376$

$$var(\beta_1) = \frac{\sigma^2}{S_{xx}} = 0.38$$

$$var(\beta_0) = \sigma^2 \left(\frac{1}{n} + \frac{\bar{x}^2}{S_{xx}}\right) = 51.7$$

قسمت ج)

$$\frac{(x_i-\bar{x})(y_i-\bar{y})}{n} = 130.5$$

$$\sigma_x = \sqrt{\frac{(x_i-\bar{x})^2}{n-1}} = 6.13 \ , \ \sigma_y = \sqrt{\frac{(y_i-\bar{y})^2}{n-1}} = 21.8 \ , \ \sigma_x \, \sigma_y = 133.67$$

$$Cor(\beta_0, \beta_1) = \frac{Cov(x,y)}{\sigma_x \, \sigma_y} = \frac{\frac{(x_i-\bar{x})(y_i-\bar{y})}{n}}{\sigma_x \, \sigma_y} = \frac{130.5}{133.67} = 0.97$$

# 1 Question 5

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
data = pd.read_csv('D:\ML\ML_HW2\Data\penguins.csv')
count_nan = data.isnull().sum()
print(count_nan)
```

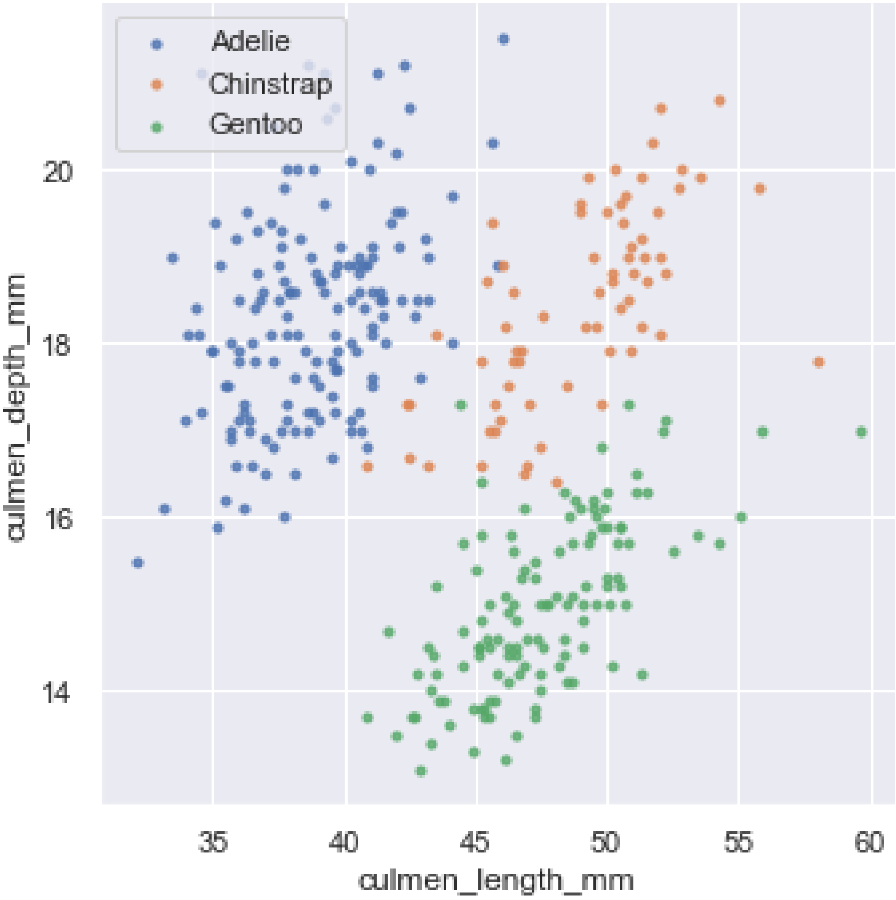
```
species              0
culmen_length_mm     2
culmen_depth_mm      2
flipper_length_mm    2
body_mass_g          2
dtype: int64
```

[3]:
```
data = data.dropna()
```

## 2 Part A

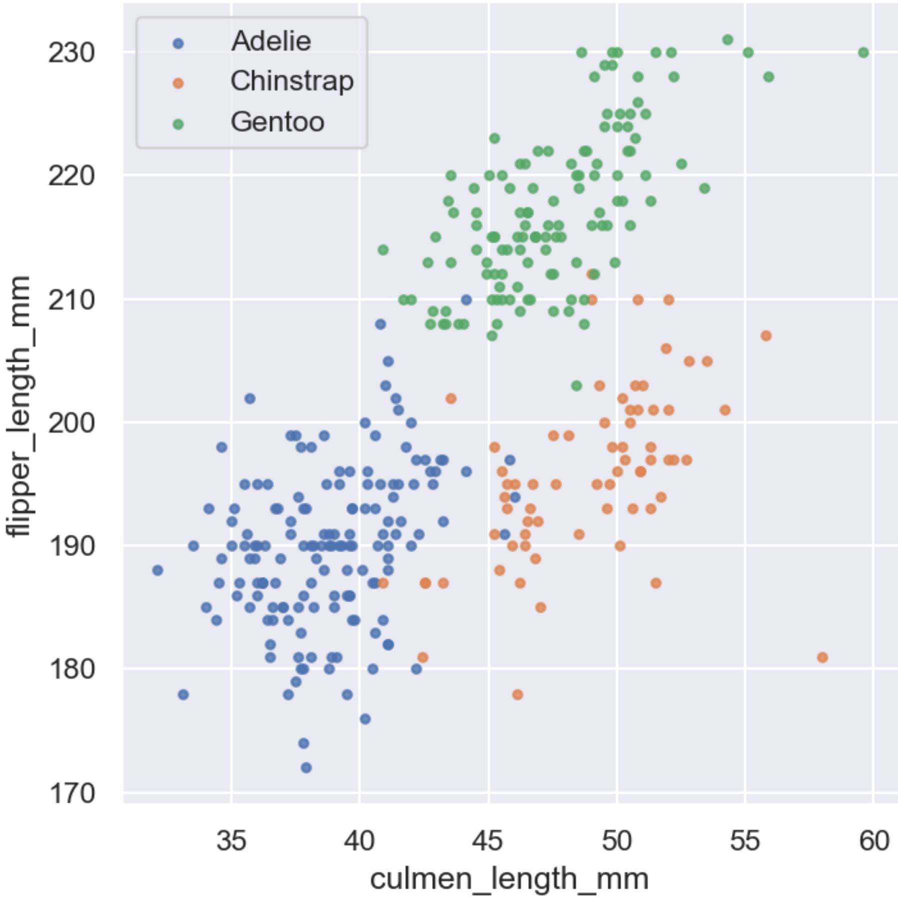
[4]:
```
sns.set(rc={"figure.dpi":150, 'savefig.dpi':150})
sns.lmplot( x="culmen_length_mm", y="culmen_depth_mm", data=data, fit_reg=False,␣
↪hue='species', legend=False, scatter_kws={"s": 10})
plt.legend(loc='upper left')
```

[4]: <matplotlib.legend.Legend at 0x1bc0a69f970>



[5]:
```
sns.set(rc={"figure.dpi":150, 'savefig.dpi':150})
sns.lmplot( x="culmen_length_mm", y="flipper_length_mm", data=data, fit_reg=False,␣
 ↪hue='species', legend=False, scatter_kws={"s": 10})
plt.legend(loc='upper left')
```

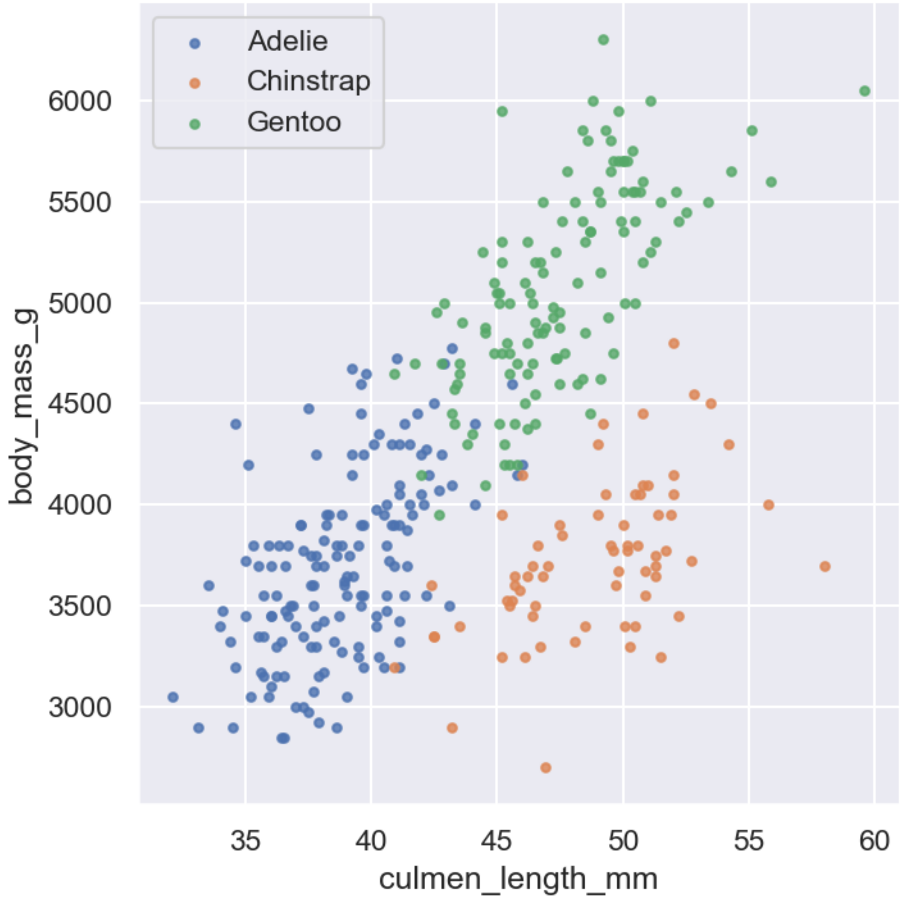
[5]: <matplotlib.legend.Legend at 0x1bc0f758670>



[6]:

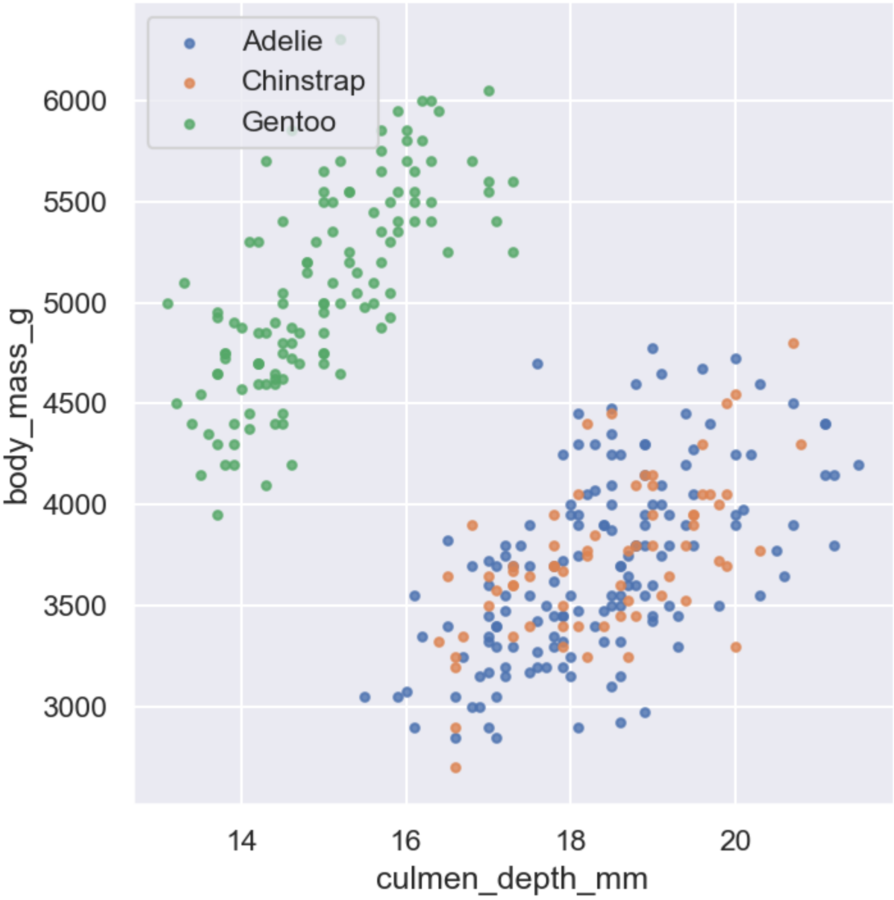
```
sns.set(rc={"figure.dpi":150, 'savefig.dpi':150})
sns.lmplot( x="culmen_length_mm", y="body_mass_g", data=data, fit_reg=False,␣
 ↪hue='species', legend=False, scatter_kws={"s": 10})
plt.legend(loc='upper left')
```

[6]: <matplotlib.legend.Legend at 0x1bc0f758160>



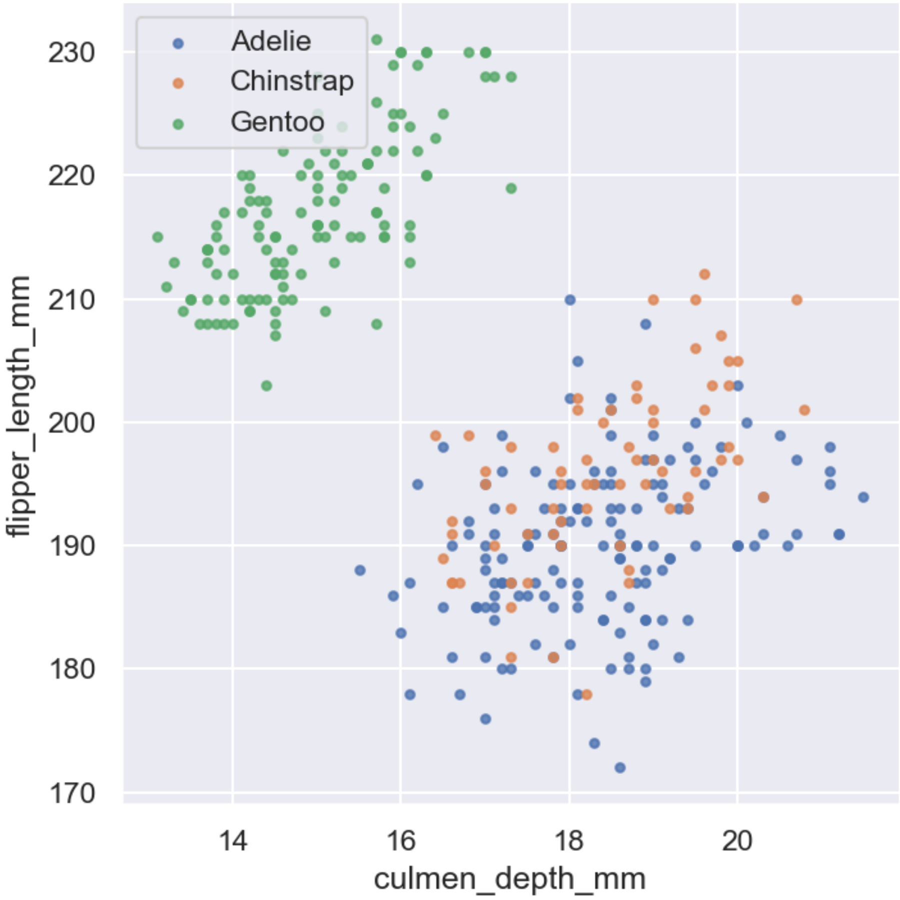
[7]:
```
sns.set(rc={"figure.dpi":150, 'savefig.dpi':150})
sns.lmplot( x="culmen_depth_mm", y="body_mass_g", data=data, fit_reg=False,␣
↪hue='species', legend=False, scatter_kws={"s": 10})
plt.legend(loc='upper left')
```

[7]: <matplotlib.legend.Legend at 0x1bc1023dd00>



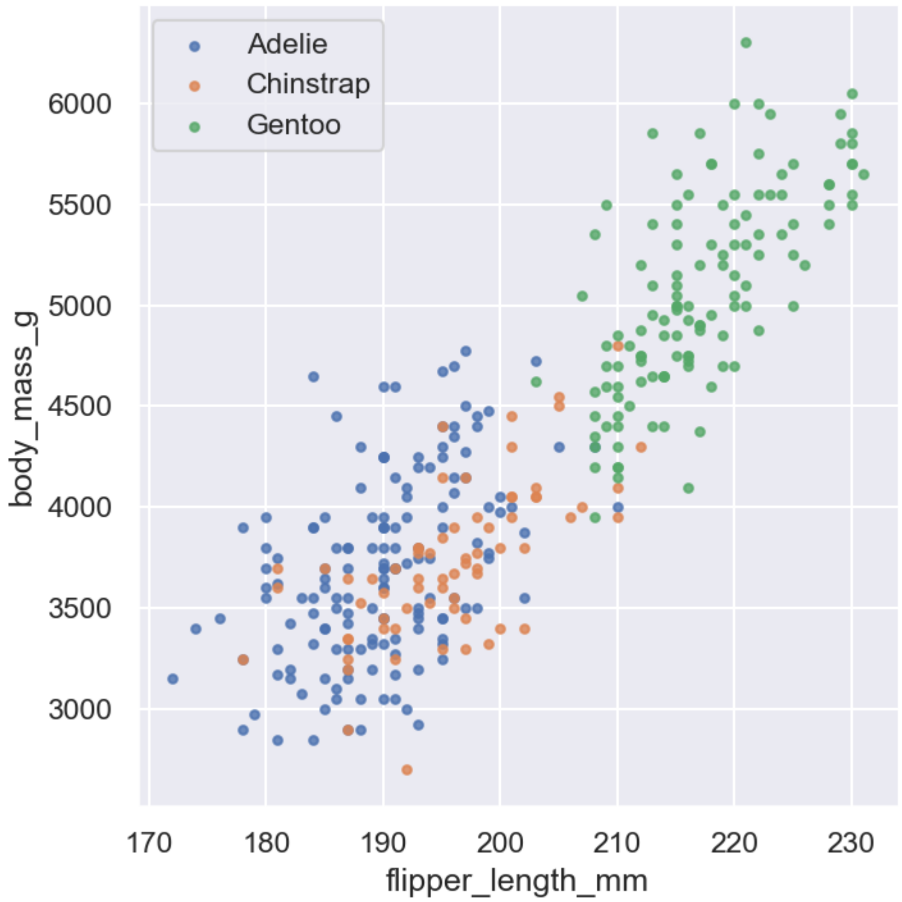
```
[8]: sns.set(rc={"figure.dpi":150, 'savefig.dpi':150})
     sns.lmplot( x="culmen_depth_mm", y="flipper_length_mm", data=data, fit_reg=False,␣
      ↪hue='species', legend=False, scatter_kws={"s": 10})
     plt.legend(loc='upper left')
```

```
[8]: <matplotlib.legend.Legend at 0x1bc0ff11f40>
```



[9]:
```
sns.set(rc={"figure.dpi":150, 'savefig.dpi':150})
sns.lmplot( x="flipper_length_mm", y="body_mass_g", data=data, fit_reg=False,␣
↪hue='species', legend=False, scatter_kws={"s": 10})
plt.legend(loc='upper left')
```

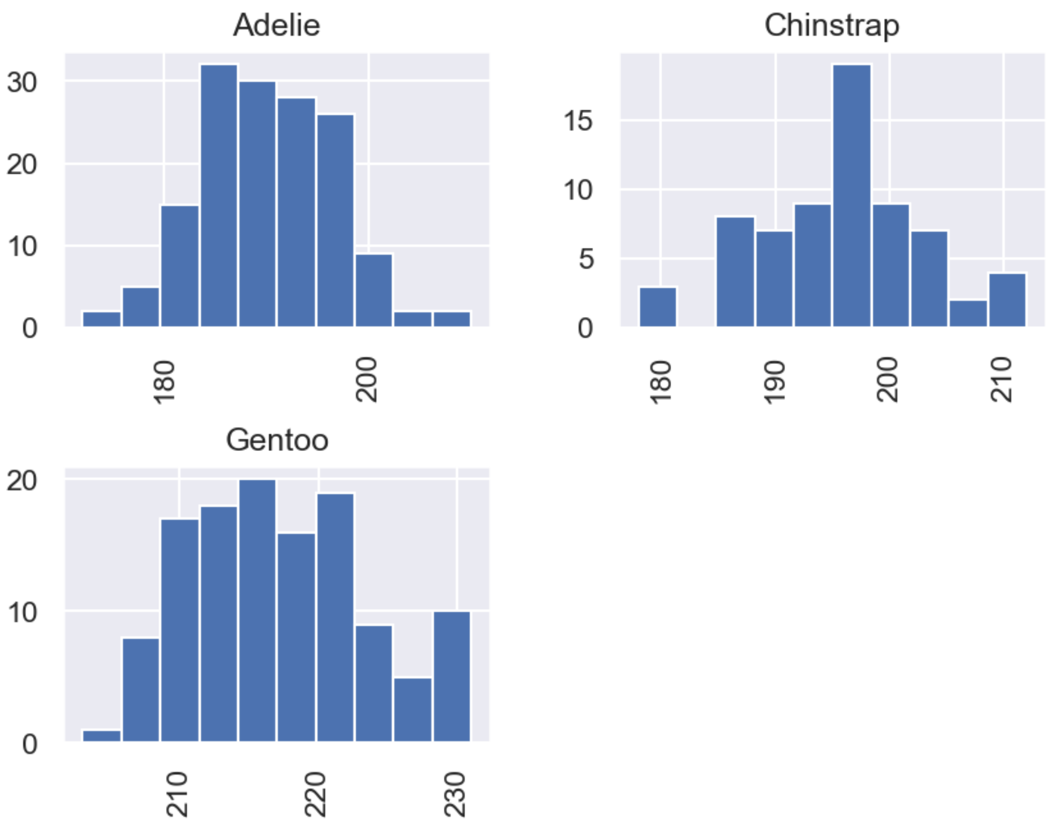
[9]: <matplotlib.legend.Legend at 0x1bc11063910>



**We will use culmen_length_mm and culmen_depth_mm**

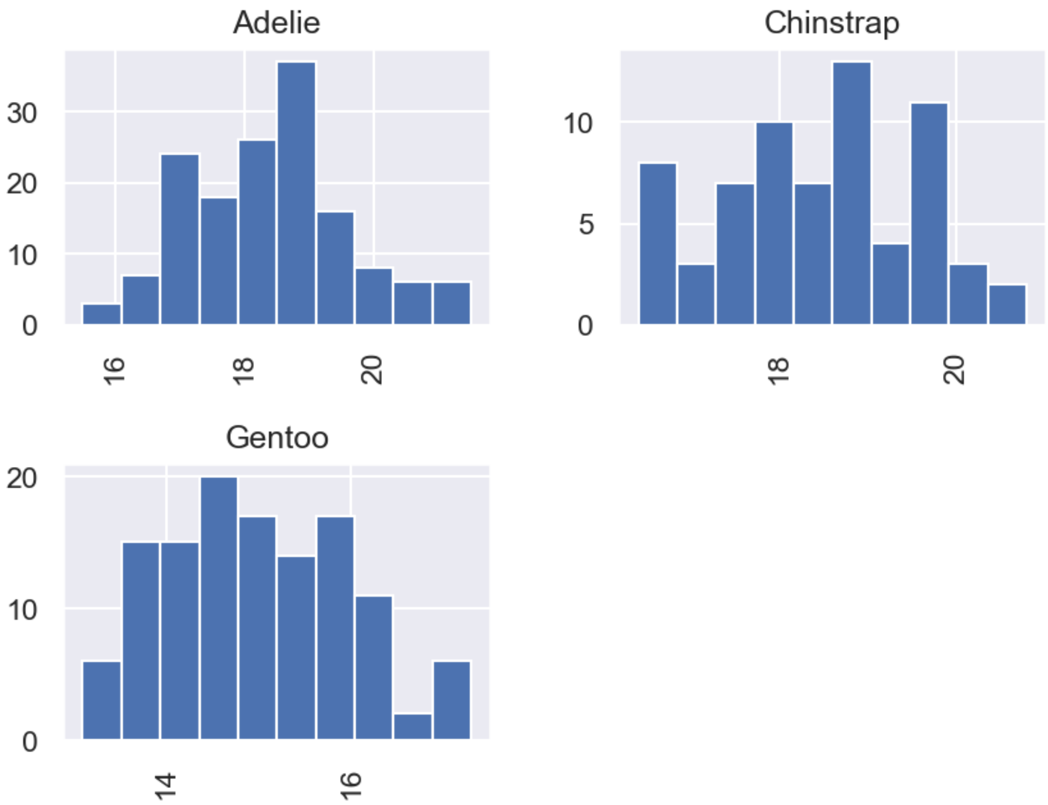
```
[10]: data.hist(column='flipper_length_mm', by='species')
```

```
[10]: array([[<AxesSubplot:title={'center':'Adelie'}>,
              <AxesSubplot:title={'center':'Chinstrap'}>],
             [<AxesSubplot:title={'center':'Gentoo'}>, <AxesSubplot:>]],
            dtype=object)
```



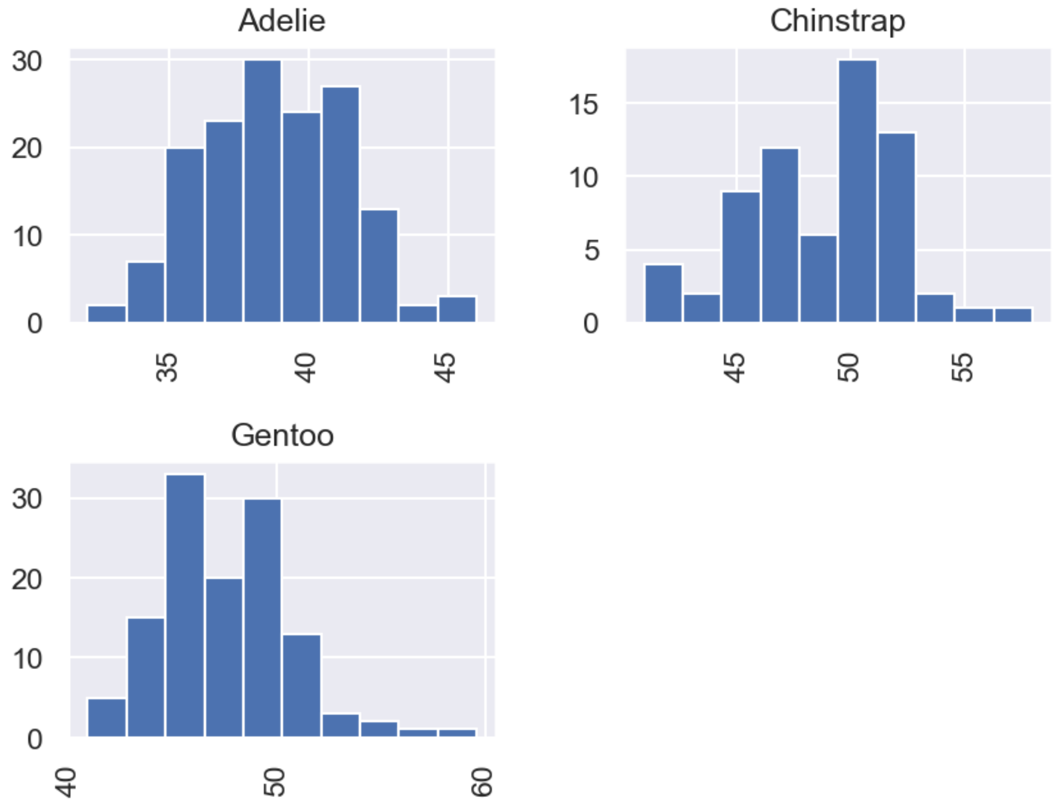
```
[11]: data.hist(column='culmen_depth_mm', by='species')
```

```
[11]: array([[<AxesSubplot:title={'center':'Adelie'}>,
              <AxesSubplot:title={'center':'Chinstrap'}>],
             [<AxesSubplot:title={'center':'Gentoo'}>, <AxesSubplot:>]],
            dtype=object)
```



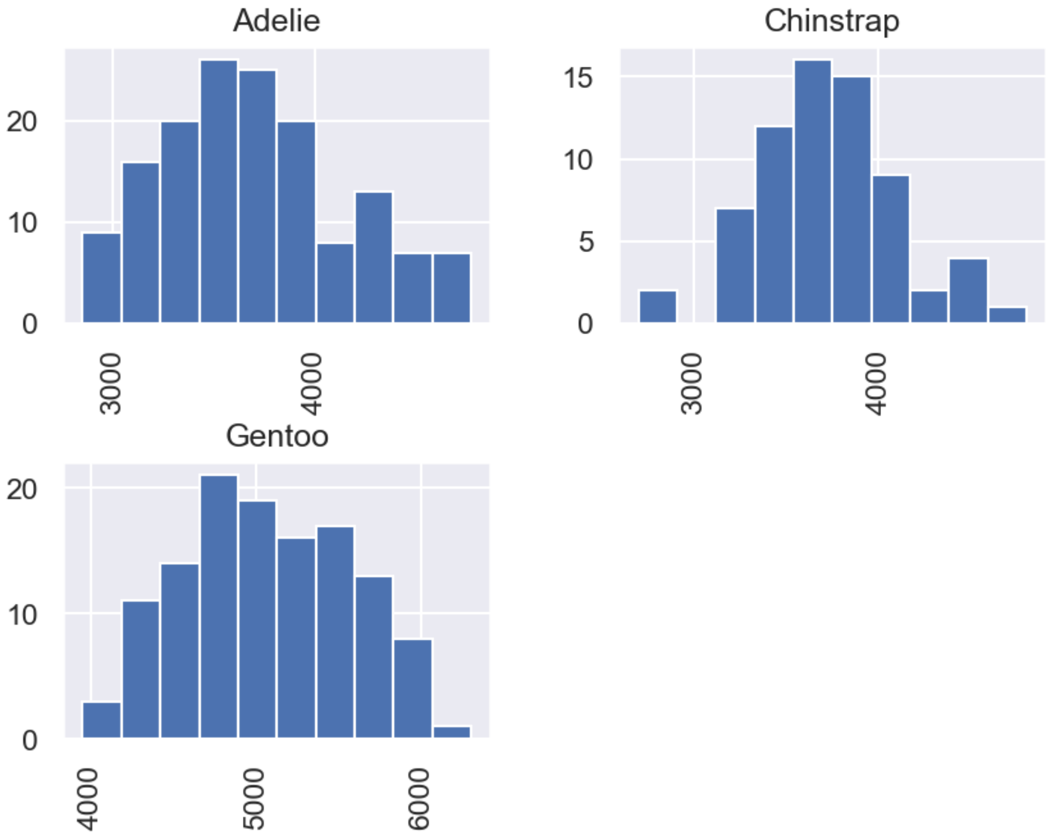
```
[12]: data.hist(column='culmen_length_mm', by='species')
```

```
[12]: array([[<AxesSubplot:title={'center':'Adelie'}>,
              <AxesSubplot:title={'center':'Chinstrap'}>],
             [<AxesSubplot:title={'center':'Gentoo'}>, <AxesSubplot:>]],
            dtype=object)
```



[13]:
```
data.hist(column='body_mass_g', by='species')
```

[13]:
```
array([[<AxesSubplot:title={'center':'Adelie'}>,
        <AxesSubplot:title={'center':'Chinstrap'}>],
       [<AxesSubplot:title={'center':'Gentoo'}>, <AxesSubplot:>]],
      dtype=object)
```



## 3 Part B

[14]:
```
def split_train_test(data):
    df = data.sample(frac=1).reset_index(drop=True)
    cut_point = int(data.shape[0]*0.8)
    return df.iloc[ 0:cut_point] , df.iloc[cut_point:]
```

[15]:
```
class LogisticRegression:
    w = np.array
    b = 0
    def sigmoid(self,z):
        return 1.0/(1 + np.exp(-z))

    def loss(self,y, y_hat):
        loss = -np.mean(y*(np.log(y_hat)) - (1-y)*np.log(1-y_hat))
        return loss
    def gradients(self,X, y, y_hat):

        m = X.shape[0]
        dw = (1/m)*np.dot(X.T, (y_hat - y))
        db = (1/m)*np.sum((y_hat - y))

        return dw, db

    def normalize(self,X):
        m, n = X.shape
        for i in range(n):
            X = (X - X.mean(axis=0))/X.std(axis=0)
        return X

    def train(self,X, y, bs = 20, epochs = 100, lr = 0.01):
        m, n = X.shape
        w = np.zeros((n,1))
        b = 0
        y = y.values.reshape(m,1)
        x = self.normalize(X)
        losses = []
        for epoch in range(epochs):
            for i in range((m-1)//bs + 1):
                start_i = i*bs
                end_i = start_i + bs
                xb = X[start_i:end_i]
                yb = y[start_i:end_i]
                y_hat = self.sigmoid(np.dot(xb, w) + b)
                dw, db = self.gradients(xb, yb, y_hat)
                w -= lr*dw
                b -= lr*db
            l = self.loss(y, self.sigmoid(np.dot(X, w) + b))
            losses.append(l)
        self.w = w
        self.b = b
        return w, b, losses

    def predict(self, X):
        x = self.normalize(X)
        preds = np.dot(X, self.w) + self.b
        pred_class = []
```

```
        pred_class = [1 if i > 0.5 else 0 for i in preds]

        return np.array(preds)
```

[16]:
```
def F1_score(y,y_hat):
    tp,tn,fp,fn = 0,0,0,0
    for i in range(len(y)):
        if y[i] == 1 and y_hat[i] == 1:
            tp += 1
        elif y[i] == 1 and y_hat[i] == 0:
            fn += 1
        elif y[i] == 0 and y_hat[i] == 1:
            fp += 1
        elif y[i] == 0 and y_hat[i] == 0:
            tn += 1
    precision = tp/(tp+fp)
    recall = tp/(tp+fn)
    f1_score = 2*precision*recall/(precision+recall)
    return f1_score
```

[17]:
```
def one_vs_all(x_Train,y_train,x_test , epoch = 1000):
    reg = LogisticRegression()
    temp = pd.get_dummies(y_train)
    classes = pd.DataFrame()
    for i in temp.head():
        reg.train(x_Train,temp[i] , epochs=epoch)
        x = reg.predict(x_test)
        classes[i] = x[:,0]
    return classes
```

[18]:
```
train_set, test_set = split_train_test(data)

prediction =␣
↪one_vs_all(train_set[['culmen_length_mm','culmen_depth_mm']],train_set['species'],test_set[['culm
```

[19]:
```
class measurements():

    @staticmethod
    def confusion_matrix(results):
        confusion = pd.crosstab(results['gold'], results['pred'])
        return confusion

    @staticmethod
    def jaccard(results):
        eq = results.apply(lambda x: (x['gold']==x['pred']) , axis = 1)
        intersect = np.sum(eq)
        union = results.shape[0] * results.shape[1] - intersect
        return intersect / union

    @staticmethod
    def accuracy(results):
        x = measurements.confusion_matrix(results)
        trace = np.trace(x)
        sum_all = np.sum(x)
        return trace / np.sum(sum_all)
```

[20]:
```
test_res = pd.DataFrame()
test_res['gold'] = test_set['species']
test_res['pred'] = list(prediction.idxmax(axis=1))

print(measurements.confusion_matrix(test_res))
print("Jaccard Value is:")
print(measurements.jaccard(test_res))
print("Accuracy is:")
print(measurements.accuracy(test_res))
```

```
pred        Adelie  Chinstrap  Gentoo
gold
Adelie          26          0       0
Chinstrap        1          8       2
Gentoo           0          1      31
Jaccard Value is:
0.8904109589041096
Accuracy is:
0.9420289855072463
```

# 4 Part c

[21]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.multiclass import OneVsRestClassifier
import sklearn.metrics as skm
```

[22]:
```
X = data.drop('species', axis=1)
y = data['species']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
model = LogisticRegression()

ovr = OneVsRestClassifier(model)
ovr.fit(X_train, y_train)

y_pred = ovr.predict(X_test)
```

[23]:
```
print('confusion_matrix of Sk - learn is:')
print(skm.confusion_matrix(y_test,y_pred))
print('------------------------------------')
print('Accuracy of Sk - learn is:')
print(skm.accuracy_score(y_test,y_pred))
print('------------------------------------')

print('f1-score of Sk - learn is:')
print(skm.f1_score(y_test,y_pred , average='micro'))
print('------------------------------------')

print('jaccard-score of Sk - learn is:')
print(skm.jaccard_score(y_test,y_pred , average='micro'))
```

```
confusion_matrix of Sk - learn is:
[[27  0  0]
 [ 1 18  0]
 [ 0  0 23]]
------------------------------------
```

```
Accuracy of Sk - learn is:
0.9855072463768116
------------------------------------
f1-score of Sk - learn is:
0.9855072463768116
------------------------------------
jaccard-score of Sk - learn is:
0.9714285714285714
```

[24]:
```python
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import jaccard_score
from sklearn.metrics import f1_score
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
from itertools import cycle
```

[25]:
```python
label_encoder = LabelEncoder()
y_int_pred = label_encoder.fit_transform(y_pred)
y_one_pred = OneHotEncoder(sparse=False)
y_int_pred = y_int_pred.reshape(len(y_int_pred), 1)
y_onehot_pred = y_one_pred.fit_transform(y_int_pred)
```

[26]:
```python
label_encoder = LabelEncoder()
y_int_test = label_encoder.fit_transform(y_test)
y_one_test = OneHotEncoder(sparse=False)
y_int_test = y_int_test.reshape(len(y_int_test), 1)
y_onehot_test = y_one_test.fit_transform(y_int_test)
```

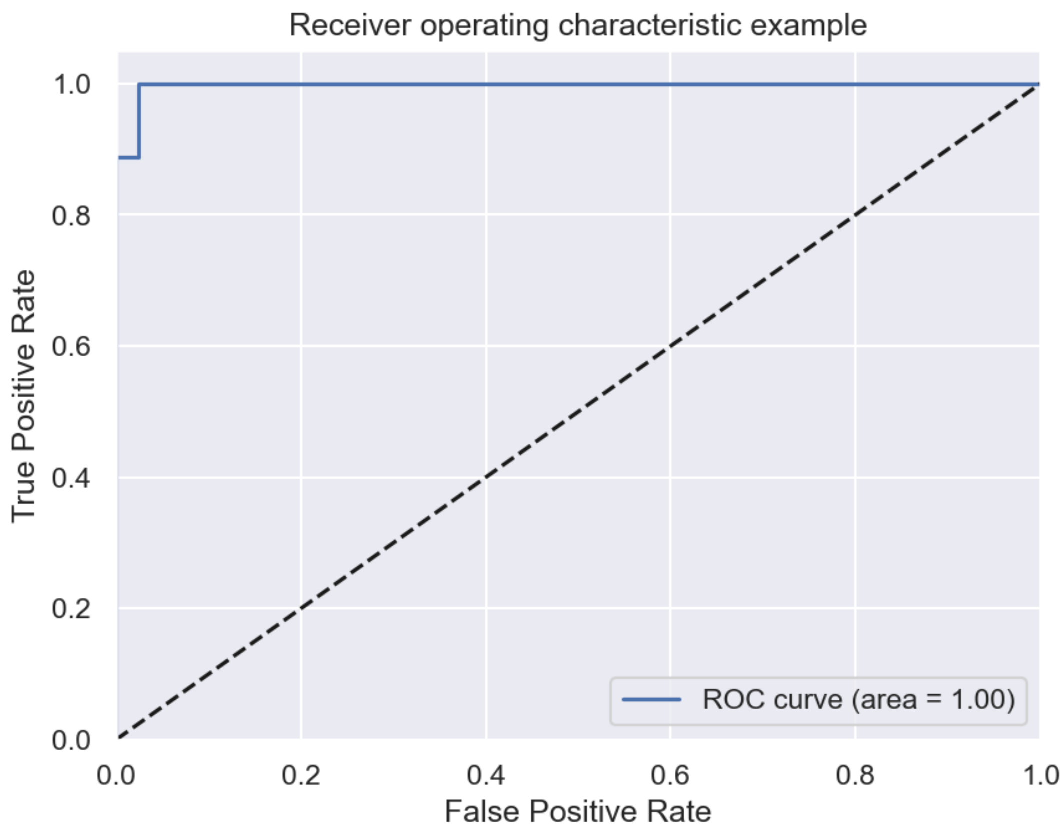
[27]:
```python
print('f1_score of Sk - learn is:')
print(skm.f1_score(y_onehot_test, y_onehot_pred, average='micro'))
print('------------------------------------')
print('precision_score of Sk - learn is:')
print(skm.precision_score(y_onehot_test, y_onehot_pred, average='micro'))
print('------------------------------------')
print('recall_score of Sk - learn is:')
print(skm.recall_score(y_onehot_test, y_onehot_pred, average='micro'))
```

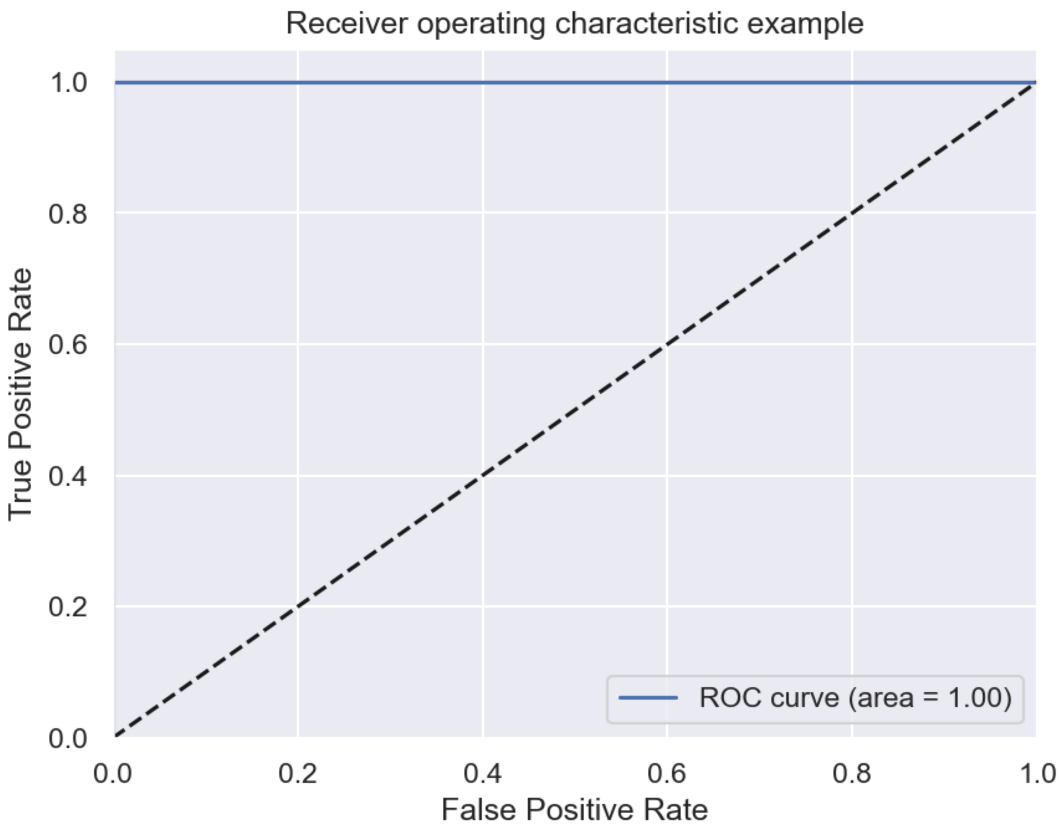
```
f1_score of Sk - learn is:
0.9855072463768116
------------------------------------
precision_score of Sk - learn is:
0.9855072463768116
------------------------------------
recall_score of Sk - learn is:
0.9855072463768116
```

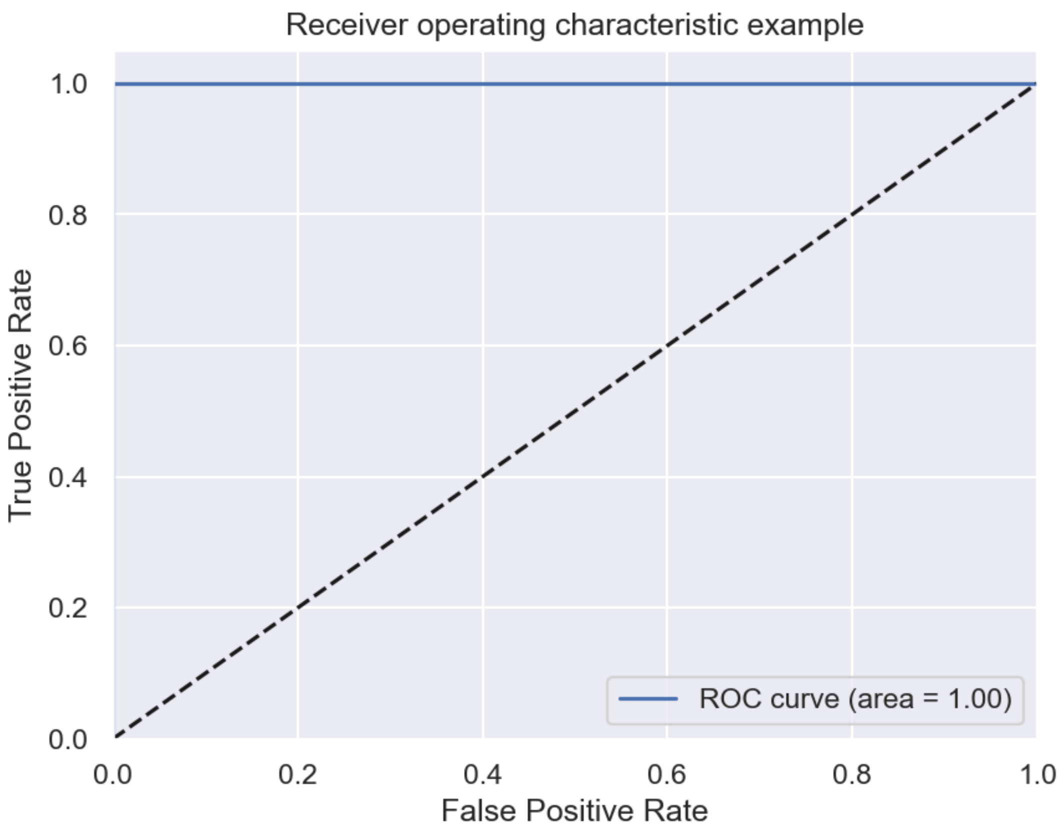
[28]:
```python
reg = LogisticRegression(max_iter = 1000)

y_score = reg.fit(X_train, y_train).decision_function(X_test)
fpr = dict()
tpr = dict()
roc_auc = dict()
y_test_dummy= y_test.str.get_dummies()
n_classes = y_test_dummy.shape[1]
for i in range(n_classes):
    fpr[i], tpr[i], _ = skm.roc_curve(y_test_dummy.iloc[:,i], y_score.T[i])
    roc_auc[i] = skm.auc(fpr[i], tpr[i])
```

```
for i in range(n_classes):
    plt.figure()
    plt.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], 'k--')
    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 example')
    plt.legend(loc="lower right")
    plt.show()
```




Receiver operating characteristic example
True Positive Rate
False Positive Rate
ROC curve (area = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0


Receiver operating characteristic example
True Positive Rate
False Positive Rate
ROC curve (area = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0

## 5 Question 6

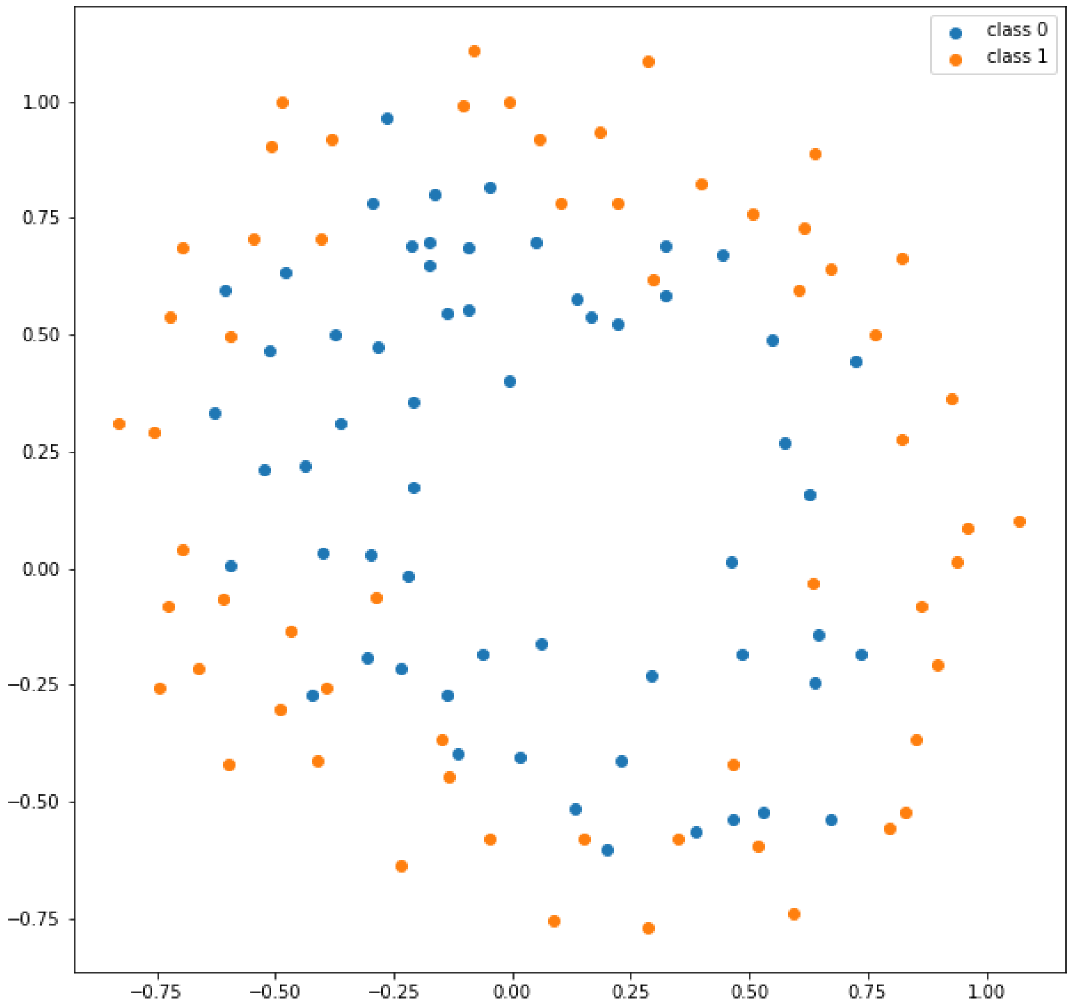
[1]:
```
import pandas
import matplotlib.pyplot as plt
import numpy as np
import math
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LogisticRegression
import seaborn as sns
```

[2]:
```
data = pandas.read_csv('D:\ML\ML_HW2\Data\Quality.csv')
```

[3]:
```
c0 = data[data['class'] == 0]
c1 = data[data['class'] == 1]
```

[4]:
```
plt.figure(figsize=(10,10))

plt.scatter('x' ,'y', data = c1)
plt.scatter('x' ,'y', data = c0)
plt.legend(["class 0" , "class 1"])
plt.show()
```



```
[5]: class LogisticRegression:
         w = np.array
         b = 0
         def sigmoid(self,z):
             return 1.0/(1 + np.exp(-z))

         def loss(self,y, y_hat):
             loss = -np.mean(y*(np.log(y_hat)) - (1-y)*np.log(1-y_hat))
             return loss
         def gradients(self,X, y, y_hat):

             m = X.shape[0]
             dw = (1/m)*np.dot(X.T, (y_hat - y))
             db = (1/m)*np.sum((y_hat - y))

             return dw, db

         def normalize(self,X):
             m, n = X.shape
             for i in range(n):
                 X = (X - X.mean(axis=0))/X.std(axis=0)
             return X

         def train(self,X, y, bs = 20, epochs = 100, lr = 0.01):
             m, n = X.shape
             w = np.zeros((n,1))
             b = 0
             y = y.values.reshape(m,1)
             x = self.normalize(X)
             losses = []
             for epoch in range(epochs):
                 for i in range((m-1)//bs + 1):
                     start_i = i*bs
                     end_i = start_i + bs
                     xb = X[start_i:end_i]
                     yb = y[start_i:end_i]
                     y_hat = self.sigmoid(np.dot(xb, w) + b)
                     dw, db = self.gradients(xb, yb, y_hat)
                     w -= lr*dw
                     b -= lr*db
                 l = self.loss(y, self.sigmoid(np.dot(X, w) + b))
                 losses.append(l)
             self.w = w
             self.b = b
             return w, b, losses

         def predict(self, X):
             x = self.normalize(X)
             preds = np.dot(X, self.w) + self.b
             pred_class = []
             pred_class = [1 if i > 0.5 else 0 for i in preds]

             return np.array(preds)
```

[6]:
```
poly1 = PolynomialFeatures(degree  = 5, include_bias=False)

#Extract Polynomial features
poly1.fit(data[['x','y']])

#initialize log reg
reg1 = LogisticRegression(max_iter = 1000)

#transform to 27 dimensions
data_transformed = poly1.transform(data[['x','y']])

#Fit the data and get coefficients
reg1.fit(data_transformed,data['class'])

print("The Accuracy is:")
print(reg1.score(data_transformed,data['class']))
```

```
The Accuracy is:
0.8389830508474576
```

[7]:
```
xx, yy = np.meshgrid(np.arange(-1, 1, 0.001),
                     np.arange(-1, 1, 0.001))

Z = reg1.predict(poly1.transform(np.c_[xx.ravel(),yy.ravel()]))
plt.figure(figsize=(12,8))
plt.scatter('x' ,'y', data = c1)
plt.scatter('x' ,'y', data = c0)
plt.legend(["class 0" , "class 1"])

Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, cmap=plt.cm.Paired)

plt.show()
```