Applied Data Science

L8. Classification

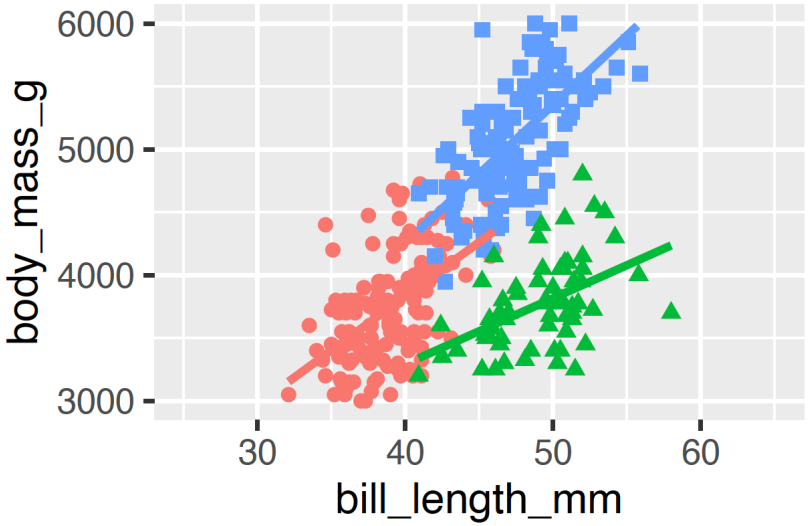
Irina Mohorianu

Head of Bioinformatics/ Scientific Computing @CSCI

Overview

Classifications. Definitions

- Classification: class supervised learning methods (data with labels), for which the output variable is categorical.
- we already discussed the logistic model for the Palmer penguins. In this lecture we will expand and generalise those ideas.



Horst AM, Hill AP, Gorman KB (2020). palmerpenguins: Palmer

https://allisonhorst.github.io/palmerpenguins/. doi:10.5281/zenodo.3960218.

Classifications. Definitions

Classification: take a $D-$dimensional input x and assign entried to one of the K discrete classes $C_k, k = 1, ..., K$. The input is presented as tuples $(x_1, y_1), ..., (x_n, y_n)$ with $\{y_1, ..., y_2\} \in \{C_1, ..., C_k\}$.

Decision regions, decision boundaries: the input space is divided into *decision regions*, separated by *decision boundaries*

Linear models for Classification: the decision boundaries are linear functions of x defined by $D - 1$ dimensional hyperplanes within the D dimensional space.

Linearly Separable Datasets: datasets whose classes can be separated by linear hyperplanes.

Classifications. Definitions

- for 2-class models, the output variable has 2 values, $t \in \{0, 1\}$. 0 corresponds to class C_1; 1 corresponds to class C_2.
- for $K > 2$ class, we rely on the one hot encoding (OHE).
- Class C_1 corresponds to $[1, 0, 0]$
- Class C_2 corresponds to $[0, 1, 0]$
- Class C_3 corresponds to $[0, 0, 1]$.

Classifications. Approaches

[Option 1] determine a discriminant function that assigns each vector x to a specific class.

[Option 2] Model the conditional probability distribution $p(C_k|x)$ during an inference stage; next use this in later stages.

- **model 2.1** model $p(C_k|x)$ directly by parametric models, optimize the parameters on a training set, assess on a test set
- **model 2.2 generative approach:** model the conditional class densities $p(x|C_k)$, the priors $p(C_k)$, then compute $p(C_k|x)$ using Bayes' theorem:

$$p(C_k|x) = \frac{p(x|C_k)p(C_k)}{p(x)}.$$

Classifications. Approaches

Input: t-uples $(x_1, y_1), ..., (x_n, y_n)$ with $\{y_1, ..., y_2\} \in \{C_1, ..., C_k\}$.

Output: define $P(Y|X)$

Both discriminative and generative models address the same question, from different angles.

[Discriminative model] makes predictions on unseen data based on conditional probabilities; used for classification and regression

[Generative model] focuses on the distribution within a dataset; it return a probability for a given example.

Classifications. Approaches

Examples of discriminative approaches:

- **Logistic regression**, Support Vector machines (SVMs)
- Standard neural networks
- Nearest Neighbours, Decision Trees and Random Forests

Examples of generative approaches:

- **Linear discriminant analysis**. Quadratic discriminant analysis.
- Naive Bayes, Bayesian networks
- Hidden Markov Models (HMMs)
- Generative Adversarial Networks (GANs)

Classification. Approaches

Robustness and computational costs:

- generative models are more sensitive to outliers compared to discriminative models
- discriminative models: the misclassification angle introduces penalties.
- in a context of missing data, generative models can estinate the posterior by marginalising the unseen variables; discriminative models require all features to be observed.
- discriminative models are computationally cheap (both as runtime and resources, and from an optimisation perspective) compared to generative models.
- generative models need fewer training points, but have stronger assumptions (**assumption of conditional independence**)

Discriminant functions. Definitions

To predict discrete class labels (as posterior probabilities in the range $(0, 1)$), we consider the generalization of the Bayesian regression model

$$y(x) = f(w^T x + w_0)$$

$f(\cdot)$ is known as an activation function.

Decision hyperplane: $y(x) = const$ so that $w^T x + w_0 = const$

i.e. the decision hyperplane is linear in x, even if $f(\cdot)$ is nonlinear.

The model is non-linear in w, as it was for the simple regression models.

These models are generalised linear models.

Discriminant functions. Definitions

A discriminant function takes x and assigns it directly to one of the K classes C_k.

For two classes a linear discriminant function is

$$y(x) = w^T x + w_0.$$

where w is the weight vector and the w_0 is the bias ($-w_0$ is also called a threshold).

x is assigned to C_1 if $y(x) \geq 0$; to C_2 otherwise.

Decision boundary: $y(x) = 0$, a $D - 1$ dimensional hyperplane.

Discriminant functions. Definitions

Consider two points x_A and x_B on the decision surface.

$y(x_A) = 0$ and $y(x_B) = 0$; therefore $w^T(x_A - x_B) = 0$.

i.e. w is orthogonal to every vector on the decision surface.

w **determines the orientation of the decision surface.**

Consider the normal vector $\overrightarrow{OA}$ from the origin to the decision surface.

x is the projection of the origin O on the decision surface.

$$\overrightarrow{OA} = x = \alpha \frac{w}{||w||_{w^T x + w_0 = 0}} \implies w^T \alpha \frac{w}{||w||} + w_0 = 0$$

$$(OA) = \alpha = -\frac{w_0}{||w||}$$

the bias parameter w_0 determines the location of the decision surface from the origin.

Logistic regression. Why

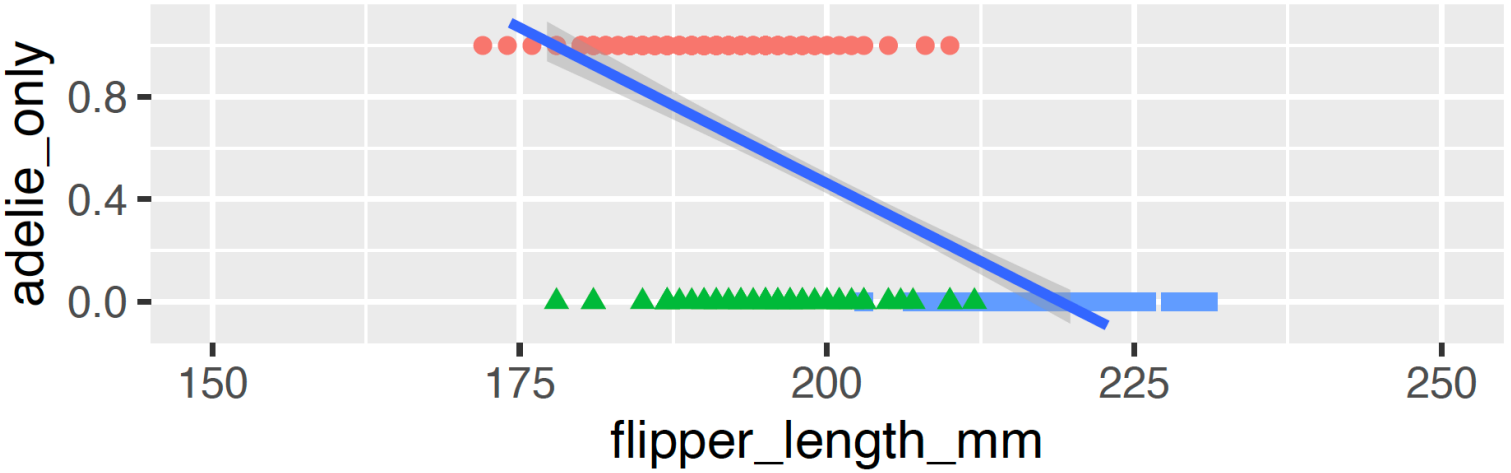
Input: continuous (or categorical) predictors

Output: species for the Palmer penguins

For convenience, we focus on the Adelie vs complement i.e. 0/1 coding.

We could use the simple linear regression

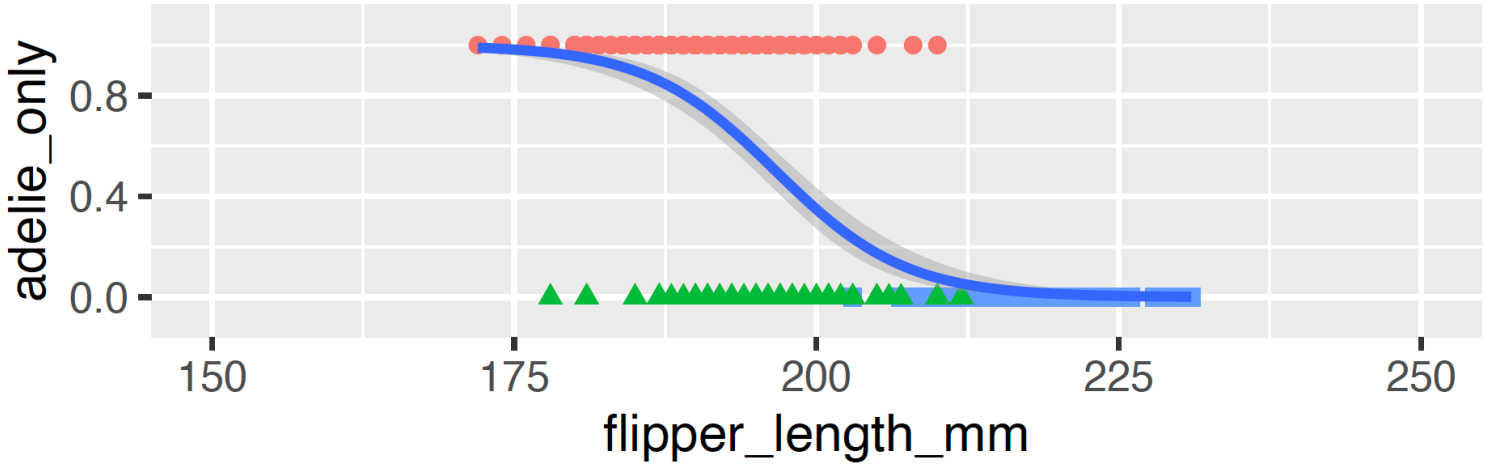
$$p(X) = \beta_0 + \beta_1 X.$$



Logistic regression. How

Using the predicted β_0 and β_1 we don't capture the focus labels i.e. are the penguins of the Adelie species or not (one hot encoding was used, as described in the Regression II).

To avoid this, we can model $p(X)$ using a function that produces 0/1 output.



The logistic function

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}$$

Logistic regression. Maximum likelihood

To fit the logistic function we use *maximum likelihood*.

With a bit of manipulation we derive

$$\frac{p(X)}{1-p(X)} = e^{\beta_0+\beta_1 x}$$

The LHS is called *odds*.

Applying log we obtain

$$log\frac{p(X)}{1-p(X)} = \beta_0 + \beta_1 x$$

The LHS is the *log odds* or *logit*.

The logistic regression model has a logit that is linear in X.

Logistic regression. Maximum likelihood

The coefficients β_0 and β_1 must be estimated from the data.

We pick the β_0 and β_1 to maximise the likelihood of the observed data.

To do this, we use a *likelihood function*

$$\ell(\beta_0, \beta) = \prod_{i:y_i=1} p(x_i) \prod_{i:y_i=0} (1 - p(x_i))$$

Logistic regression. Maximum likelihood

```
X = sm.add_constant(my_penguins['bill_length_mm'])
y = my_penguins['adelie_only']

# Logistic regression model
model_adelie = sm.Logit(y, X)
result_adelie = model_adelie.fit()

print(result_adelie.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.135507
         Iterations 9
                           Logit Regression Results
==============================================================================
Dep. Variable:            adelie_only   No. Observations:                  333
Model:                          Logit   Df Residuals:                      331
Method:                           MLE   Df Model:                            1
Date:                Tue, 24 Oct 2023   Pseudo R-squ.:                  0.8023
Time:                        14:55:18   Log-Likelihood:                -45.124
converged:                       True   LL-Null:                       -228.29
Covariance Type:            nonrobust   LLR p-value:                 1.180e-81
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         50.2432      6.918      7.263      0.000      36.685      63.802
bill_length_mm    -1.1717      0.162     -7.255      0.000      -1.488      -0.855
==============================================================================
```

Many aspects of logistic regression are similar to linear regression.

We can measure the accuracy of coefficient estimates by computing standard errors.

Logistic regression. Multiple logistic regression

The binary response can be predicted using multiple predictors.

$$log\frac{p(X)}{1-p(X)} = \beta_0 + \beta_1 X_1 + ... + \beta_p X_p,$$

where $X = (X_1, ..., X_p)$ are p predictors.

$$p(X) = \frac{e^{\beta_0+\beta_1 X_1+...+\beta_p X_p}}{1 + e^{\beta_0+\beta_1 X_1+...+\beta_p X_p}}$$

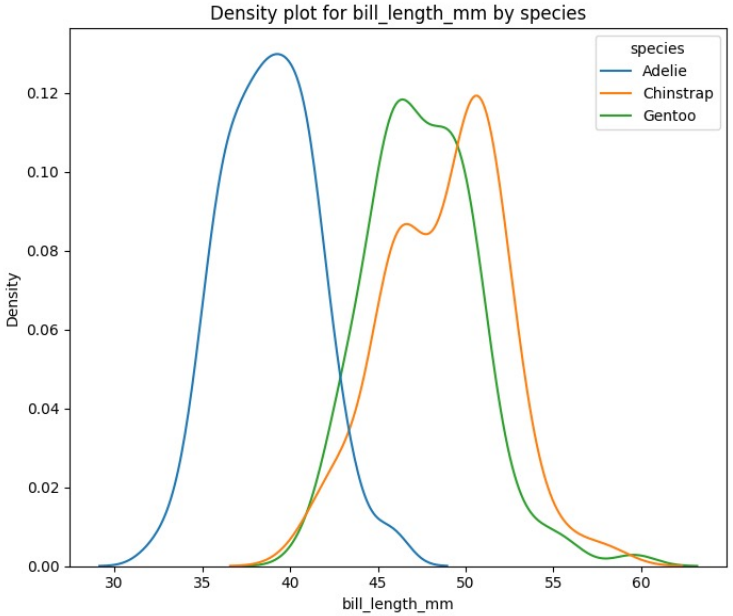
The Maximum likelihood method can be used to estimate the $\beta_0, \beta_1, ..., \beta_p$.

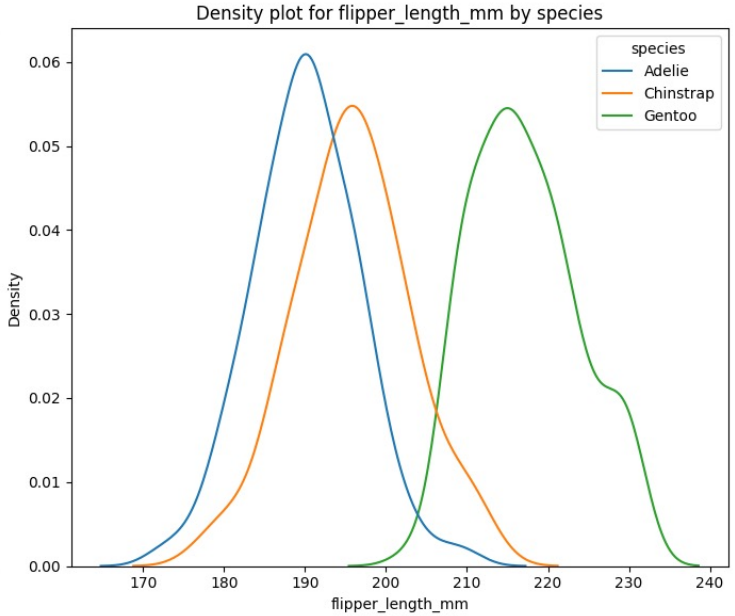
Logistic regression. Multiple logistic regression

```
X = sm.add_constant(my_penguins[['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm']])
y = my_penguins['adelie_only']

# Multiple Logistic regression model
model_adelie_multiple = sm.Logit(y, X)
result_adelie_multiple = model_adelie_multiple.fit()

print(result_adelie_multiple.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.023446
         Iterations 13
                           Logit Regression Results
==============================================================================
Dep. Variable:            adelie_only   No. Observations:                  333
Model:                          Logit   Df Residuals:                      329
Method:                           MLE   Df Model:                            3
Date:                Tue, 24 Oct 2023   Pseudo R-squ.:                  0.9658
Time:                        14:58:20   Log-Likelihood:                -7.8074
converged:                       True   LL-Null:                       -228.29
Covariance Type:            nonrobust   LLR p-value:                 2.964e-95
==============================================================================
                     coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const             10.7013     15.855      0.675      0.500     -20.373      41.776
bill_length_mm    -2.8383      0.971     -2.924      0.003      -4.741      -0.936
bill_depth_mm      4.6222      1.755      2.634      0.008       1.183       8.062
flipper_length_mm  0.1521      0.094      1.616      0.106      -0.032       0.337
==============================================================================
```

Logistic regression. Multiple logistic regression






```python
import matplotlib.pyplot as plt

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(14, 6))

sns.kdeplot(data=my_penguins, x='bill_length_mm', hue='species', common_norm=False, ax=axes[0])
axes[0].set_title("Density plot for bill_length_mm by species")

sns.kdeplot(data=my_penguins, x='flipper_length_mm', hue='species', common_norm=False, ax=axes[1])
axes[1].set_title("Density plot for flipper_length_mm by species")

plt.tight_layout()
plt.show()
```

Logistic regression. Multinomial logistic regression

We have more than 2 classes present in the dataset.

The analysis so far was conducted on Adelie vs non-Adelie.

We select a class to serve as baseline.

Without loss of generality, we select class K.

$$Pr(Y = k|X = x) = \frac{e^{\beta_{k0}+\beta_{k1}x_1+...+\beta_{kp}x_p}}{1 + \sum_{l=1}^{K-1} e^{\beta_{l0}+\beta_{l1}x_1+...+\beta_{lp}x_p}}$$

for the classes $k = 1, ...K - 1$

and

$$Pr(Y = K|X = x) = \frac{1}{1 + \sum_{l=1}^{K-1} e^{\beta_{l0}+\beta_{l1}x_1+...+\beta_{lp}x_p}}$$

Generative models. Context

Logistic regression focuses on directly modeling $Pr(Y - k|X = x)$ using the logistic function i.e. we model the conditional distribution of the response Y given the predictor(s) X.

The alternative is to rely on Bayes' theorem and model the distributions of the predictors X separately for each of the response classes.

Remarks:

- when there is substantial separation between (two) classes, the parameter estinates for the logistic regression model are quite unstable
- if the distribution of predictors X is approximately normal for each class, and the sample size is small, then the generative models are more robust.
- the generative models can be extended naturally to more than two classes.

Generative models. Context

Goal: classify an observation into one of K classes.

Let π_k be the *prior* probability that a randomly chosen observation comes form the kth class.

Let $f_k(X) = Pr(X|Y = k)$ be the *density function* of X for an observation from the kth class.

Bayes' theorem states that

$$Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}.$$

Let $p_k(x) = Pr(Y = k|X = x)$ be the posterior probability that an observation $X = x$ belongs to the kth class.

Generative models.
Linear Discriminant Analysis (p=1)

We assume $p = 1$ i.e. we only have one predictor.

Assumptions on $f_k(x)$:

- $f_k(x)$ is Gaussian.

$$f_k(x) = \frac{1}{\sqrt{2\pi}\sigma_k} \exp(-\frac{1}{2\sigma_k^2}(x - \mu_k)^2)$$

  where μ_k and σ_k^2 are the mean and variance for the kth class.

- we further assume constant variance across classes

$$\sigma_1^2 = ... = \sigma_k^2 = \sigma^2$$

Generative models.
Linear Discriminant Analysis (p=1)

The posterior probability will then be:

$$p_k(x) = \frac{\pi_k \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{1}{2\sigma_k^2}(x-\mu_k)^2)}{\sum_{l=1}^{K} \pi_l \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{1}{2\sigma_k^2}(x-\mu_l)^2)}$$

where π_k is the prior probability that an observation is in the kth class.

Taking the log and rearranging the terms

$$\delta_k(x) = x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + log(\pi_k)$$

The entry will be assigned to the class with the largest $\delta_k(x)$.

Generative models.
Linear Discriminant Analysis (p=1)

If $K = 2$ and $\pi_1 = \pi_2$ then the Bayes classifier assigns the observation to class 1 if $2(\mu_1 - \mu_2) > \mu_1^2 - \mu_2^2$, and to class 2 otherwise.

The Bayes decision boundary is the point for which $\delta_1(x) = \delta_2(x)$ i.e.

$$x = \frac{\mu_1^2 - \mu_2^2}{2(\mu_1 - \mu_2)} = \frac{\mu_1 + \mu_2}{2}$$

Generative models.
Linear Discriminant Analysis

The *Linear Discriminant analysis* (LDA) approximates the Bayes classifier by plugging the estimated for π_k, μ_k and σ^2 into

$$\delta_k(x) = x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + log(\pi_k)$$

The estimates for the $\hat{\mu_k}, \hat{\sigma}^2$ are:

$$\hat{\mu_k} = \frac{1}{n_k} \sum_{i:y_i=k} x_1$$

$$\hat{\sigma}^2 = \frac{1}{n-k} \sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \hat{\mu_k})^2$$

In the absence of any additional information, LDA astimates μ_k using the proprotion of training observations in the kth class $\hat{\pi_k} = \frac{n_k}{n}$.

Generative models.
Linear Discriminant Analysis

The LDA classifier assigns an observation $X = c$ to the class for which

$$\hat{\delta}_k(x) = x\frac{\hat{\mu}_k}{\hat{\sigma}^2} - \frac{\hat{\mu}_k{}^2}{\hat{\sigma}^2} + log(\hat{\pi}_k)$$

The LDA can be extended to multiple predictors.

We assume that $X = (X_1, ..., X_p)$ is drawn from a multivariate Gaussian
i.e. each individual predictor follows are one-dimensional normal distribution.

Generative models.
Linear Discriminant Analysis

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Splitting the data
train_data, test_data = train_test_split(my_penguins, test_size=0.2, stratify=my_penguins['species'], random_state=42)

scaler = StandardScaler()

# Fitting the scaler only on training data and transforming the training data
train_transform = pd.DataFrame(scaler.fit_transform(train_data.select_dtypes(include=['float64', 'int64'])),
                               columns=train_data.select_dtypes(include=['float64', 'int64']).columns,
                               index=train_data.index)

# Use the same scaler to transform the test data
test_transform = pd.DataFrame(scaler.transform(test_data.select_dtypes(include=['float64', 'int64'])),
                              columns=test_data.select_dtypes(include=['float64', 'int64']).columns,
                              index=test_data.index)

# Ensure non-numeric columns remain in the transformed datasets
train_transform = pd.concat([train_data.select_dtypes(exclude=['float64', 'int64']), train_transform], axis=1)
test_transform = pd.concat([test_data.select_dtypes(exclude=['float64', 'int64']), test_transform], axis=1)
```

Generative models.
Linear Discriminant Analysis

```python
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

X_train = train_transform[['bill_length_mm', 'bill_depth_mm', 'body_mass_g', 'flipper_length_mm']]
y_train = train_transform['species']

# LDA model
model_lda4 = LinearDiscriminantAnalysis()
model_lda4.fit(X_train, y_train)

print("Prior Probabilities of Groups:")
print(model_lda4.priors_)

print("\nGroup Means:")
group_means_df = pd.DataFrame(model_lda4.means_,
                              columns=['bill_length_mm', 'bill_depth_mm', 'body_mass_g', 'flipper_length_mm'],
                              index=model_lda4.classes_)
print(group_means_df)

print("\nCoefficients of Linear Discriminants:")
coefficients_df = pd.DataFrame(model_lda4.scalings_,
                               columns=[f"LD{i}" for i in range(1, model_lda4.scalings_.shape[1] + 1)],
                               index=['bill_length_mm', 'bill_depth_mm', 'body_mass_g', 'flipper_length_mm'])
print(coefficients_df)
```

```
Prior Probabilities of Groups:
[0.43984962 0.20300752 0.35714286]

Group Means:
           bill_length_mm  bill_depth_mm  body_mass_g  flipper_length_mm
Adelie          -0.947213       0.606648    -0.638284          -0.769145
Chinstrap        0.894786       0.638094    -0.581931          -0.386155
Gentoo           0.657953      -1.109841     1.116879           1.166762

Coefficients of Linear Discriminants:
                     LD1       LD2
bill_length_mm    -0.446191 -2.219359
bill_depth_mm      2.017616 -0.027558
body_mass_g       -1.165135  1.218901
flipper_length_mm -1.115116  0.263158
```

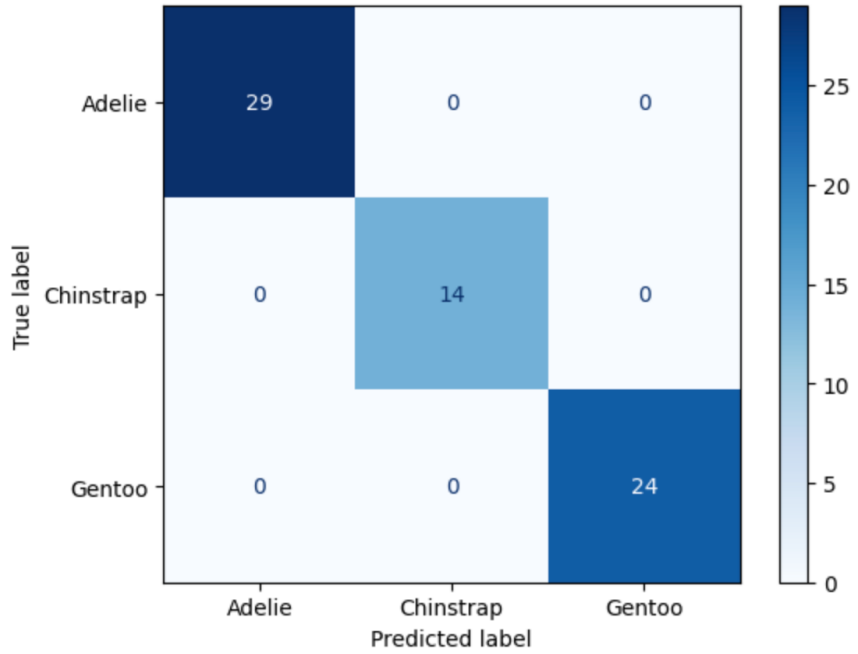
Generative models.
Linear Discriminant Analysis

```python
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

X_train = train_transform[['bill_length_mm', 'bill_depth_mm', 'body_mass_g', 'flipper_length_mm']]
y_train = train_transform['species']

# LDA model
model_lda4 = LinearDiscriminantAnalysis()
model_lda4.fit(X_train, y_train)

print("Prior Probabilities of Groups:")
print(model_lda4.priors_)

print("\nGroup Means:")
group_means_df = pd.DataFrame(model_lda4.means_,
                              columns=['bill_length_mm', 'bill_depth_mm', 'body_mass_g', 'flipper_length_mm'],
                              index=model_lda4.classes_)
print(group_means_df)

print("\nCoefficients of Linear Discriminants:")
coefficients_df = pd.DataFrame(model_lda4.scalings_,
                               columns=[f"LD{i}" for i in range(1, model_lda4.scalings_.shape[1] + 1)],
                               index=['bill_length_mm', 'bill_depth_mm', 'body_mass_g', 'flipper_length_mm'])
print(coefficients_df)
```

```
Prior Probabilities of Groups:
[0.43984962 0.20300752 0.35714286]

Group Means:
           bill_length_mm  bill_depth_mm  body_mass_g  flipper_length_mm
Adelie          -0.947213       0.606648    -0.638284          -0.769145
Chinstrap        0.894786       0.638094    -0.581931          -0.386155
Gentoo           0.657953      -1.109841     1.116879           1.166762

Coefficients of Linear Discriminants:
                     LD1       LD2
bill_length_mm    -0.446191 -2.219359
bill_depth_mm      2.017616 -0.027558
body_mass_g       -1.165135  1.218901
flipper_length_mm -1.115116  0.263158
```

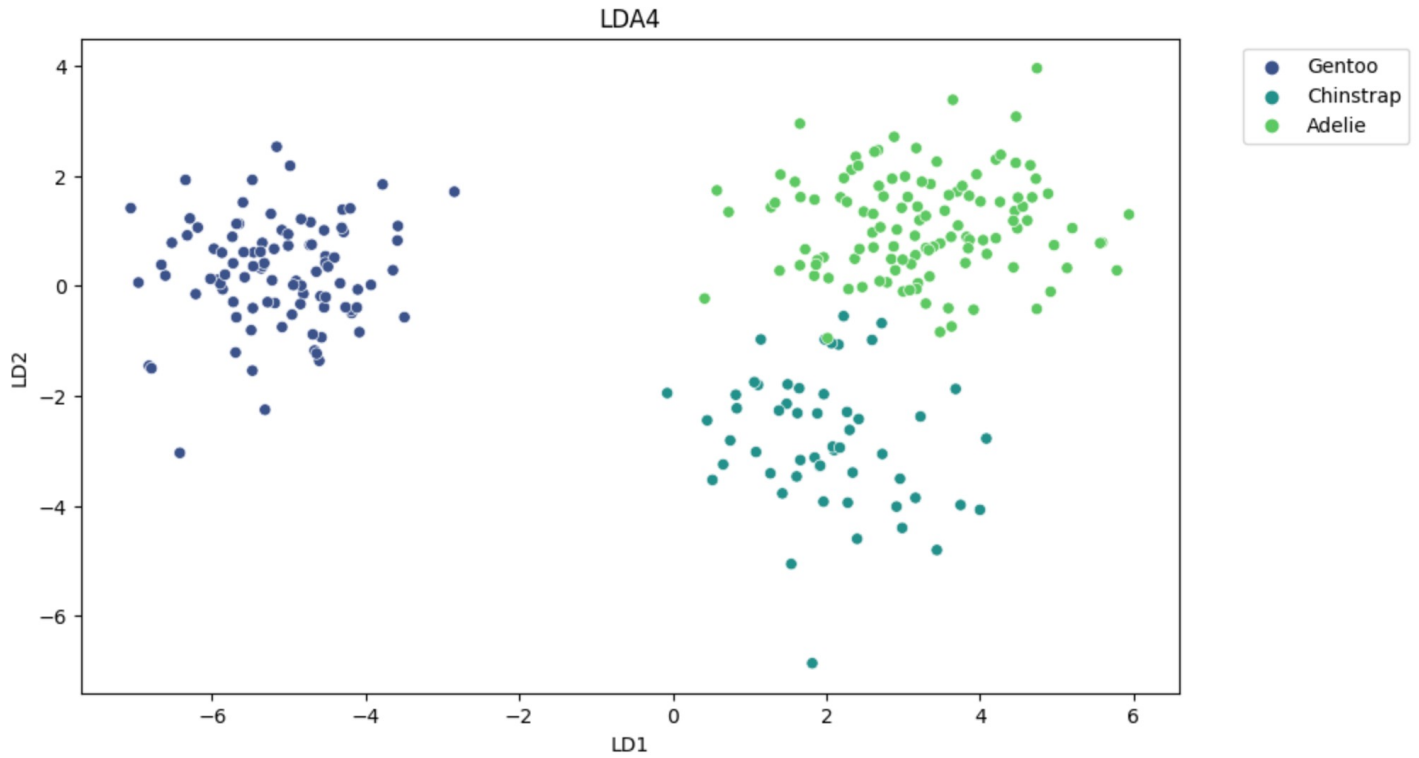
Generative models.
Linear Discriminant Analysis

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Making predictions on the test data
predictions_lda4 = model_lda4.predict(test_transform[['bill_length_mm', 'bill_depth_mm', 'body_mass_g', 'flipper_length_mm']])

# Calculate and display confusion matrix
cm_lda4 = confusion_matrix(test_transform['species'], predictions_lda4, labels=model_lda4.classes_)

cm_display = ConfusionMatrixDisplay(cm_lda4, display_labels=model_lda4.classes_)
cm_display.plot(cmap='Blues')
plt.show()
```

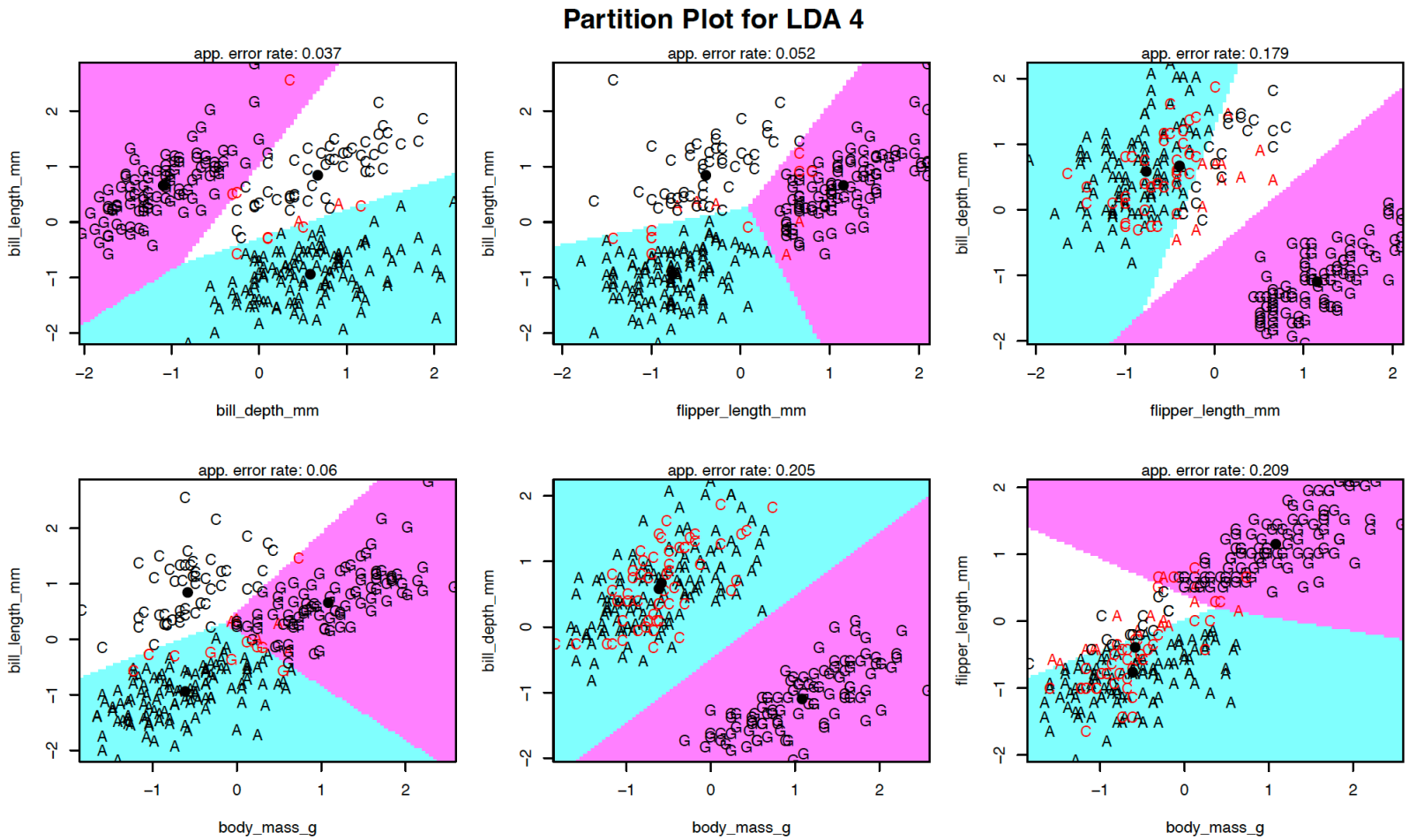


Generative models.
Linear Discriminant Analysis

```
# Projecting data onto the first two linear discriminants
lda_projection = model_lda4.transform(train_transform[['bill_length_mm', 'bill_depth_mm', 'body_mass_g', 'flipper_length_mm']])

plot_df = pd.DataFrame(lda_projection[:, :2], columns=['LD1', 'LD2'])
plot_df['species'] = train_transform['species'].values

plt.figure(figsize=(10, 6))
sns.scatterplot(x='LD1', y='LD2', hue='species', data=plot_df, palette='viridis')
plt.title('LDA4')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2)
plt.show()
```



Generative models.
Linear Discriminant Analysis



Quotes

"It's tough to make predictions, especially about the future."

— Yogi Berra

— Gareth James, Daniela Witten, Trevor Hastie, Rob Tibshirani