Machine learning for penguin classification

Created	@July 24, 2023 10:26 PM
Tags	Project Python

1.1 Group Members:

1. Tobey Ho
2. Kristina Lau
3. Marlene Lin

1.2 Contributions:

- All three of us wrote data import (2.1), cleaning (2.3), and feature selection (4).
- Kristina led part 1 of both the exploratory data analysis (pair-wise scatterplots (3.4), summary table (3.1) and the modeling (nearest neighbors (5.4)). And split the data (2.3).
- Marlene wrote the decision boundary function (5.1), did part 2 of both data analysis (bivariate KDEs (3.3)) and modeling (neural network (5.3)), and led the discussion (6).
- Tobey led part 3 of both data analysis (boxplots (3.2)) and modeling (multinomial logistic regression (5.2)).
- We were responsible for the explanations of our own parts' figures and models.

2. DATA IMPORT AND CLEANING

To begin, we are going to download our data set and import the libraries we will need. Then, we will look over the first 5 rows of data to see the information we will be working with.

2.1 Import modules AND data

```
import pandas as pd
import numpy as np
import urllib
from matplotlib import pyplot as plt
import seaborn as sns
import matplotlib.patches as mpatches

from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from itertools import combinations
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, accuracy_score, ConfusionMatrixDisplay

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier

from sklearn.utils._testing import ignore_warnings
from sklearn.exceptions import ConvergenceWarning

from IPython.display import Image
```

```
url = "https://philchodrow.github.io/PIC16A/datasets/palmer_penguins.csv"
penguins = pd.read_csv(url)
penguins.head()
```

studyName	Sample Number	Species	Region	Island	Stage	Individual ID	Clutch Completion
0	PAL0708	1	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A1
1	PAL0708	2	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A2
2	PAL0708	3	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A1
3	PAL0708	4	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A2
4	PAL0708	5	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N3A1

2.2 Split Data

Now, we are going to hold out 20% of the data set as testing data and the remaining 80% will become training data. We want to split this data prior to cleaning so that we do not accidentally pollute our test set.

```
np.random.seed(1000)
train,test = train_test_split(penguins, test_size = 0.20)
```

2.3 Clean Data

Now that we have split our data, we can begin the cleaning process and this can be done by writing a function so that we can easily apply it to both training and testing data. The function will remove any unnecessary data (NaNs, the "." gender penguin, irrelevant columns, etc.) and transform any text data to numbers.

```
#how fit_transform encode!
sex_encoding =  {'MALE':0,"FEMALE":1}
species_encoding = {'Adelie':0, 'Chinstrap':1, 'Gentoo':2}
island_encoding = {'Biscoe':0,'Dream':1,'Torgersen':2}
def cleanpg(data, numeric):
    """
    cleans the data
    parameters:
        data: the data set
        numeric: bool, user input as to whether any strings will need to be
                 converted to integers
    returns: the cleaned data, predictor variables X, target variable y
    """
    # use only data that does not have the sex = "."
    data = data[data.Sex != "."]

    # shorten species name
    data["Species"] = (data["Species"].str.split()).str.get(0)

    # make a copy of the data so that we do not alter the original data
    clean = data.copy()

    # drop irrelevant data and NaN values
    clean = clean.drop(["studyName","Sample Number", "Region","Stage","Individual ID",
                        "Clutch Completion","Date Egg", "Comments"], axis = 1).dropna()

    # convert str values to integers
```

```
    if numeric:
        le = preprocessing.LabelEncoder()
        clean["Island"] = le.fit_transform(clean['Island'])
        clean["Species"] = le.fit_transform(clean['Species'])
        clean["Sex"] = le.fit_transform(clean['Sex'])

    # make predictor and target variables
    X = clean.drop(["Species"], axis = 1)
    y = clean["Species"]

    return (clean,X,y)
```

The function is now ready to be applied to our training and test data! Let's look at our new clean data.

```
# pass in all the data for cleaning - data exploration purpose
clean, X_null, y_null = cleanpg(penguins, numeric = False)
cleannum, X_num, y_num = cleanpg(penguins, numeric = True)
train_clean, Xtrain, ytrain = cleanpg(train, numeric = True)
test_clean, Xtest, ytest = cleanpg(test, numeric = True)
clean.head()
```

	Species	Island	Culmen Length (mm)	Culmen Depth (mm)	Flipper Length (mm)	Body Mass (g)	Sex
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	FEMALE
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	FEMALE
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	FEMALE
5	Adelie	Torgersen	39.3	20.6	190.0	3650.0	MALE
6	Adelie	Torgersen	38.9	17.8	181.0	3625.0	FEMALE

3. EXPLORATORY DATA ANALYSIS

3.1 Summary Table

Let's begin exploring our data by creating a summary table of all our features. We are going to group by our qualitative features of species, island, and sex.

```
groups = clean.groupby(["Species", "Island", "Sex"])[["Culmen Length (mm)",
        "Culmen Depth (mm)",
        "Flipper Length (mm)",
        "Body Mass (g)",
        "Delta 15 N (o/oo)",
        "Delta 13 C (o/oo)"]].mean().round(2)
groups
```

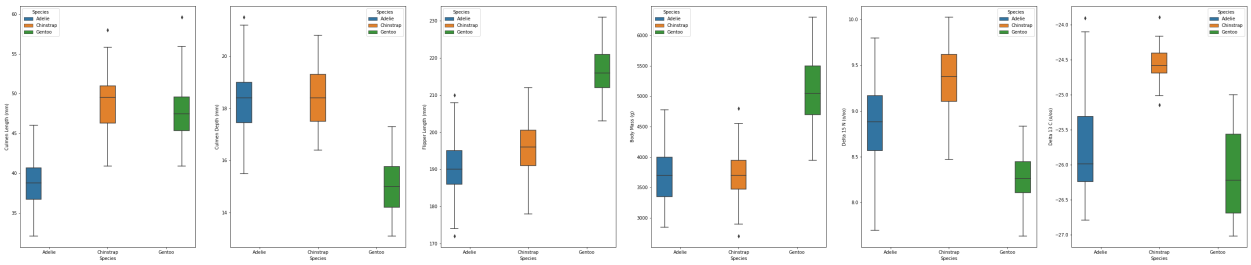
Species	Island	Sex	Culmen Length (mm)	Culmen Depth (mm)	Flipper Length (mm)	Body Mass (g)	Delta 15 N (o/oo)	Delta 13 C (o/oo)
Adelie	Biscoe	FEMALE	37.36	17.70	187.18	3369.32	8.77	-25.92
		MALE	40.59	19.04	190.41	4050.00	8.87	-25.92
	Dream	FEMALE	36.91	17.62	187.85	3344.44	8.91	-25.74
		MALE	40.01	18.84	192.52	4052.00	8.98	-25.76
	Torgersen	FEMALE	37.44	17.54	188.73	3390.91	8.66	-25.74
		MALE	40.75	19.34	195.76	4059.52	8.92	-25.84
Chinstrap	Dream	FEMALE	46.57	17.59	191.74	3527.21	9.25	-24.57
		MALE	51.07	19.25	199.73	3938.64	9.46	-24.55
Gentoo	Biscoe	FEMALE	45.56	14.24	212.71	4679.74	8.19	-26.20
		MALE	49.51	15.73	221.53	5488.75	8.30	-26.17

From our summary table, we can observe many relationships. For one, we can see which species reside on each island. Biscoe island has Adelie and Gentoo penguins, Dream island has Adelie and Chinstrap penguins, and Torgensen island only has Adelie penguins. Secondly, we can see that sex plays a role in our quantitative features within species such that females have smaller values as compared to males in all categories. We can also see trends for each species within our quantitative features. It seems that Gentoo penguins have the shallowest culmen depth, the longest flipper length, the largest body size, and the most negative delta 13 C values. Adelie penguins have the shortest culmen length and Chinstrap penguins have the highest delta 15 N value.

3.2: Boxplots

To look at each qualitiative feature between species, we can use a boxplot! Boxplots can give us an idea of how spread out each variable is by giving us the mean, median, and max, if there are any notable outliers, and if the variable will be in a low or high range.

```
# list of quantitative features
numerics = ['Culmen Length (mm)','Culmen Depth (mm)',
            'Flipper Length (mm)', 'Body Mass (g)',
            'Delta 15 N (o/oo)', 'Delta 13 C (o/oo)']

# creates figure by looping through each quantitave feature
fig, ax = plt.subplots(1,len(numerics),figsize=(50,10), sharex = True)
for i in range(0,len(numerics)):
    sns.boxplot(data = clean, x = "Species", y = numerics[i], hue = "Species", ax = ax[i])
```

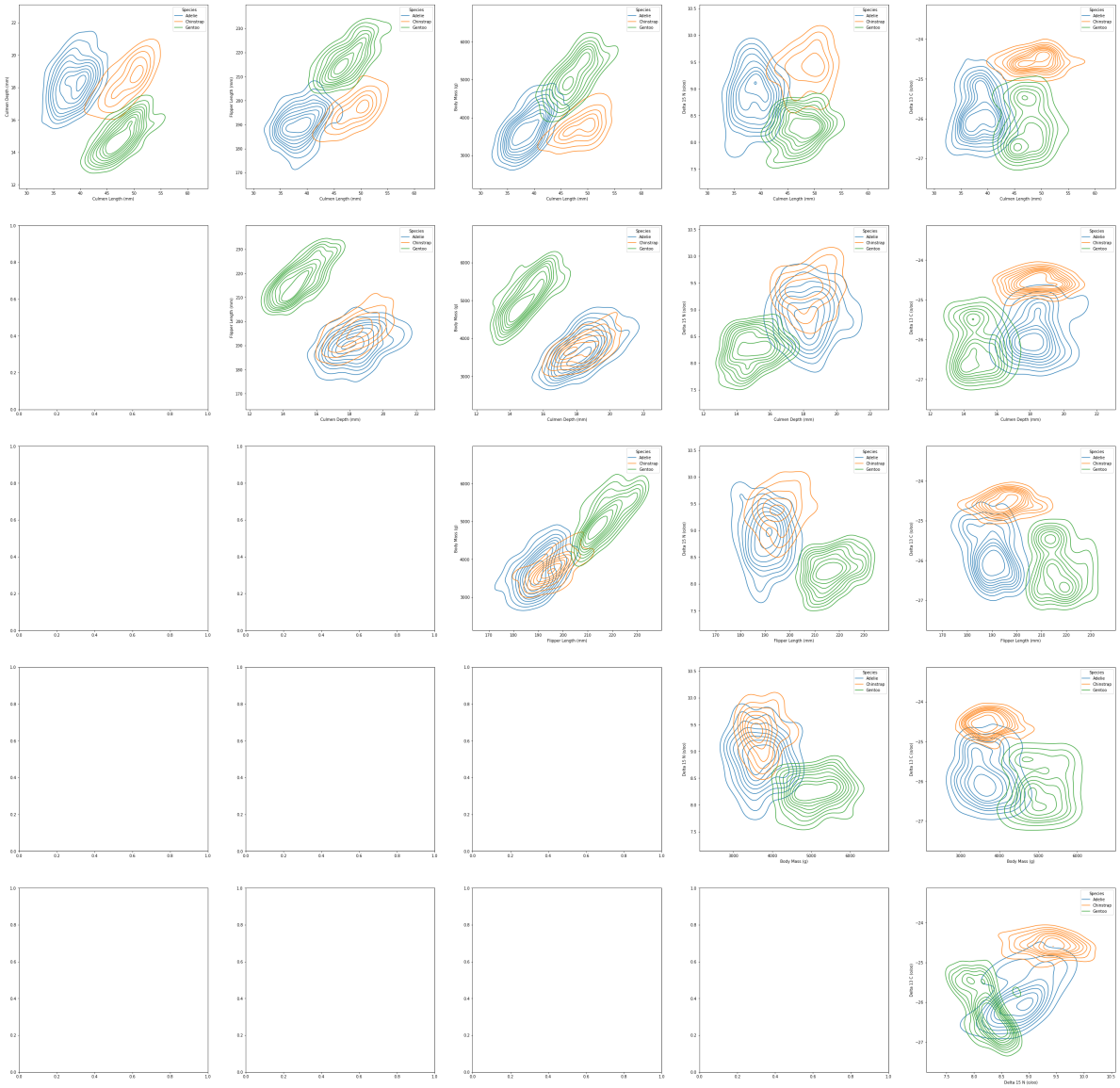


The boxplots correlate with our summary table and give us a good visualization of the trends mentioned above for each species.

3.3: Bivariate kernel density estimate (KDE) plots

The **bivariate distributions using kernel density estimation (KDE)** can help us visualize the distribution of observations of different groups in a dataset using continuous probability density curves in two dimensions. Relative to a histogram, KDE can produce a plot that is less cluttered and more interpretable, especially when drawing multiple distributions. By splitting the dataset based on species, and plotting the KDE plots of pairs of quantitative varaibles, we can get an intuition on which pair of quantitative variables would best distinguish each species.

```
f, ax = plt.subplots(len(numerics)-1,len(numerics)-1,figsize=(50,50))
for i1 in range(0,len(numerics)):
    for i2 in range(i1+1,len(numerics)):
        if i1 is not i2:
            # Draw a contour plot to represent each bivariate density
            plot = sns.kdeplot(
                data=clean,
                x=numerics[i1],
                y=numerics[i2],
                hue="Species",
                thresh=.1,ax = ax[i1,i2-1],legend=True,levels=10)
```



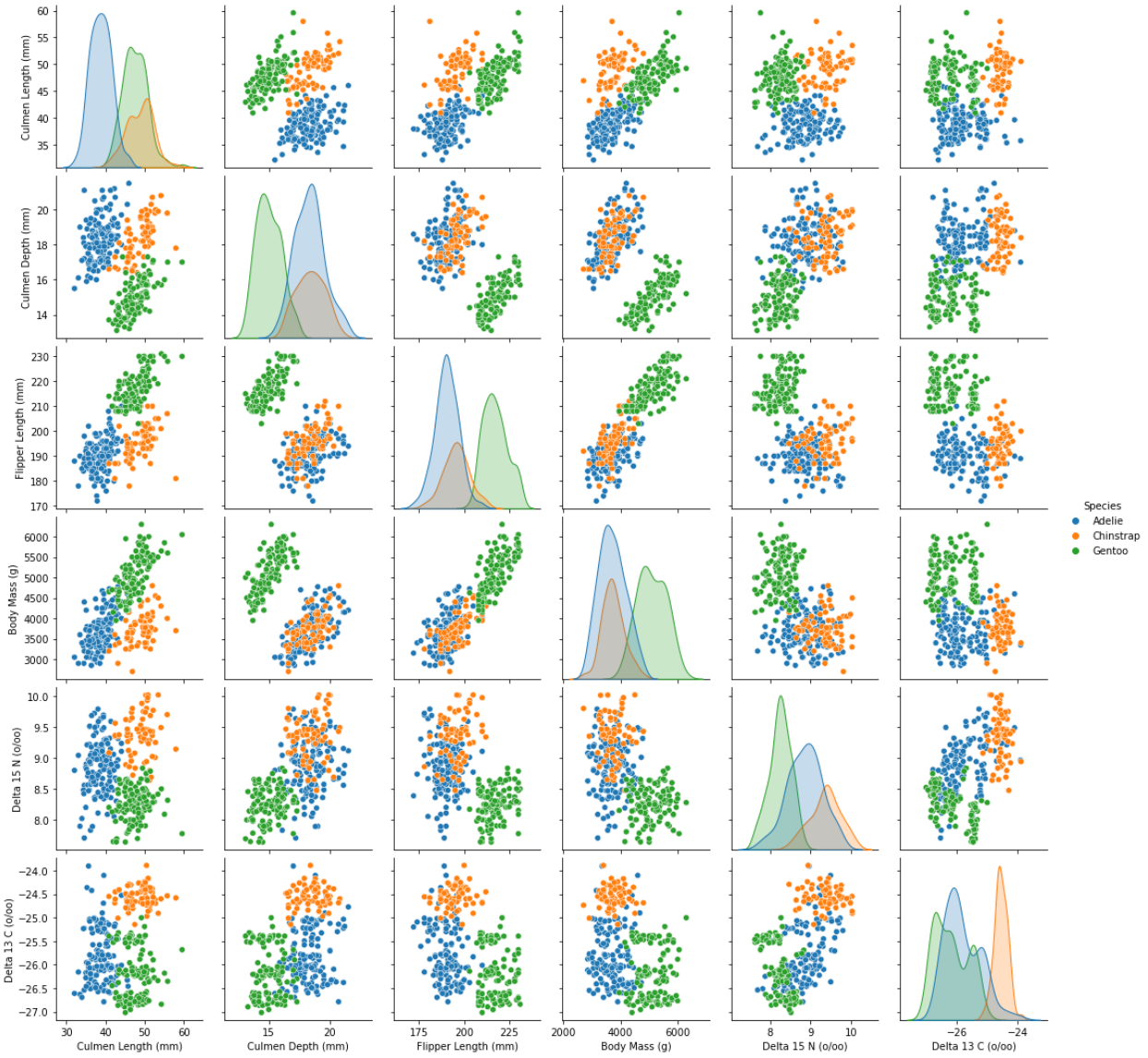
The KDE plots above show that some pairs of the quatitative variables could be used to better distinguish among species. While a bivariate KDE is three-dimensional, plotting the level curves/contour lines could 'flatten' the plot into 2D for easier visualization. The groups of level curves of different colors display the joint-density distributions of the x- and y- variables for each of the three species. Since the shape of the cluster is determined by both the mean and variance of the distributions, if two clusters are more separated from each other, it means the overall observations of the two variables are more different between the two clusters. Therefore, by looking at which pairs of the quantitative variables better separate the clusters, we can be informed on feature selection for the upcoming machine learning.

Some of the pairs that work well: **Culmen length & depth, culmen length & flipper length**

3.4: Pairwise Scatter Plot

Pairwise scatter plots allow us to compare multiple combinations of two features to better understand which set of features should be used in the model. By splitting the dataset based on species and plotting each quantitative feature against each other, we can see which features are able to distinguish each species the best.

```
sns.pairplot(clean, hue="Species")
```



From the plot, it's really easy to see that **flipper length and culmen length** or **culmen length and culmen depth** gives us really distinct clusters of each species. Thus, these could be good candidates for features to use in our model.

4. FEATURE SELECTION¶

We need to use features that would best distinguish each species in our models, so writing a function that can help each model search for the right features would be useful! We need 2 quantitative features and 1 categorical feature, so the total number of combinations is C(6,2) * 2 = 15 * 2 = 30. Since the number of combinations and our dataset are small, an exhaustive search for automated feature selection wouldn't take very long. Although we could use the qualitative features that showed good clusters in our data exploration, these are mainly visual observations and do not tell us statistically which features are the strongest predictors of species. By using a function to manually check all the combinations, we would have more accurate statistics as to which features we should be used in our models.

4.1 Exhaustive Feature Search Function¶

```
# creating a list of all possible combinations
```

```
# all pairs
quant_combos = list(combinations(numerics,2))
pg_combos = []

#loops over pairs to append qualitative elements and add combos to pg_combos list
for i in quant_combos:
    sex_combos = list(i)
    island_combos = list(i)
    sex_combos.append("Sex")
    island_combos.append("Island")
    pg_combos.append(sex_combos)
    pg_combos.append(island_combos)
```

```
@ignore_warnings(category = ConvergenceWarning)
def find_feature(model, combos,X,y):
    '''
    Finds the best feature selection for a ML model
    Inputs: ML model and a list of the feature selection combinations
    Outputs: Optimized feature select, its cv score, and amount of total combinations
    '''
    best_combo = []
    best_score = 0
    for cols in combos:
        x = cross_val_score(model, X[cols], y, cv = 5).mean()
        if x > best_score:
            best_combo = cols
            best_score = x
    return best_combo, np.round(best_score, 3)
```

Features for logistic regression model:¶

```
LG = LogisticRegression(solver = "saga")
best_combo_log, best_score_log = find_feature(LG, pg_combos,Xtrain,ytrain)
best_combo_log, best_score_log
```

```
(['Culmen Length (mm)', 'Culmen Depth (mm)', 'Island'], 0.965)
```

Features or K nearest neighbors:

```
nbrs = KNeighborsClassifier()`

`best_combo_knn, best_score_knn = find_feature(nbrs, pg_combos,Xtrain,ytrain)`

`best_combo_knn, best_score_knn`
```

```
(['Culmen Length (mm)', 'Culmen Depth (mm)', 'Sex'], 0.977)
```

```
# to see if the best combination from the logistic regression and neural networks models would
# be a decent fit for the K nearest neighbors model
poss = ['Culmen Length (mm)', 'Culmen Depth (mm)', 'Island']
cross_val_score(nbrs, Xtrain[poss], ytrain, cv = 5).mean()
```

```
0.9727752639517344
```

Features for neural networks:¶

```
mlp = MLPClassifier()
best_combo_nn, best_score_nn = find_feature(mlp, pg_combos,Xtrain,ytrain)
best_combo_nn, best_score_nn
```

```
(['Culmen Length (mm)', 'Culmen Depth (mm)', 'Island'], 0.969)
```

After performing exhaustive search on all possible combinations of predictor variables, ['Culmen Length (mm)', 'Culmen Depth (mm)', 'Island'] turned out to be the best combination for the logistic regression and neural networks models with a CV score of 0.965 and 0.957, respectively. The best combination for K nearest neighbors was ['Culmen Length (mm)', 'Culmen Depth (mm)', 'Sex'] with a CV score of 0.977, but since the other two models shared a common set of best features and these features also scored pretty high for the K nearest neighbors model (0.973), it made sense to use **['Culmen Length (mm)', 'Culmen Depth (mm)', 'Island']** for all of the models. Additionally, it would make it easier for us to compare each model if we used the same features for consistency.

```
best_combo = ['Culmen Length (mm)', 'Culmen Depth (mm)', 'Island']
Xtrain = Xtrain[best_combo]
Xtest = Xtest[best_combo]
```

5. MODELING

We decided to use Multinomial Logistic Regression, Neural Networks, and K-Nearest Neighbors as our models. Because we need to visualize all three of these with decision region plots, it would be easier to write a function that can be applied to each model.

- Cross-validation to choose complexity parameters.
- Evaluation on unseen testing data, including a confusion matrix.
- A visualization of decision regions for the model, with one plot corresponding to each value of the qualitative variable. You are not permitted to use the mlxtend package to construct your decision regions. Your colors must be consistent between your decision region plots. You must also provide a readable legend and correct axis labels.
- Discussion of the mistakes made by each model. Your discussion should give intuition for why the model fails in certain cases, using the decision regions to illustrate your explanation.

5.1 Decision boundary function¶

```
def dec_bound(var,X_in,y_in,mdl,encoding,colormap):
    '''
    plot the decision boundary of the model (mdl) on the cleaned data
      (X_in, predictors, y_in, target) and variables(var)
    INPUT:
        X_in: predictors
        y_in: target
        mdl: model
        var: a list of variables, the first two quantitative, the third qualitative
        encoding: dictionary, how the categorical variable is encoded to numeric,
              key is the number, value is the original var.
        colormap: dict, color data points based on species,
              key: 0, 1, 2, based on species_encoding, value: color, string.
    '''
    #categorical/qualitative
    catl =  penguins[var[2]].unique()
    fig,ax = plt.subplots(1,len(var),figsize=(25,10))
    # fit the model on real data
    mdl.fit(X_in, y_in)
    x = X_in.to_numpy()
    y = y_in.to_numpy()
    for i in range(len(catl)):
        # plot the real data points corresponding to each of the categorical var
        X = X_in[X_in[var[2]] == i]
        Y = y_in[X_in[var[2]] == i]
        # scatter plot coordinates are based on the values
        # of the first two quanlitative variables
        ax[i].scatter(X[var[0]], X[var[1]], c = Y.map(colormap), cmap = "ocean")
        # simulated data - grid
        x_min, x_max = x[:, 0].min() - 1, x[:, 0].max() + 1
        y_min, y_max = x[:, 1].min() - 1, x[:, 1].max() + 1
        xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02), np.arange(y_min, y_max, 0.02))
        #ravel() turns back to 1 dimensional array
        XX = xx.ravel()
        YY = yy.ravel()
        # using the np.c_ attribute to join together the two parts of the grid.
        XY = np.c_[XX, YY]
        # append island value to the end of each 2d array in the list XY
```

```
        XY = [np.append(twos, i) for twos in XY]
        # turn the resulting predictions p back into 2d
        p = mdl.predict(XY)
        p = p.reshape(xx.shape)
        # use contour plot to visualize the prediction boundary
        ax[i].contourf(xx, yy, p, cmap = "ocean", alpha = 0.2)
        #label graphs
        ax[i].set(xlabel = var[0],
        ylabel = var[1],
        title = var[2] + " " + list(encoding.keys())[i])

#see model evaluation part of neural networks for plotting the legends and title
```

5.2 Multinomial Logistic Regression

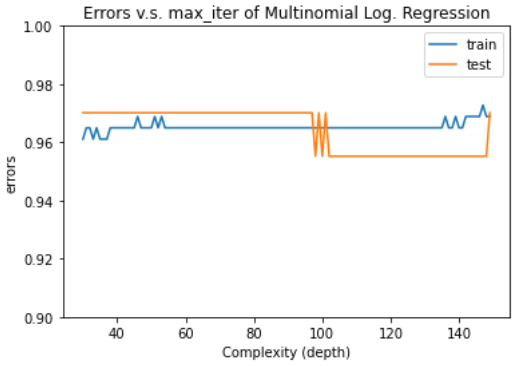
Logistic Regression models the probabilities of a binomial event using a logistic curve function. This translates to a multinomial case when using multiple inputs, like the qualities of a penguin to evaluate a parametrized probability in the model.

Cross Validation

```
@ignore_warnings(category = ConvergenceWarning)
def find_iter(lrange):
    score_list =[]
    test_list = []
    best_iter = None
    iter_score = 0
    for d in lrange:
        LG = LogisticRegression(solver = 'saga', max_iter = d,random_state=10)
        LG.fit(Xtrain,ytrain)
        x = LG.score(Xtrain, ytrain)
        test = LG.score(Xtest, ytest)
        cv = cross_val_score(LG,X_num[best_combo],y_num,cv=5).mean()
        score_list.append(x)
        test_list.append(test)
        if cv > iter_score:
            best_iter = d
            iter_score = x
    fig,ax = plt.subplots(1)
    ax.plot(lrange,score_list,label="train")
    ax.plot(lrange,test_list,label="test")
    ax.set(xlabel = "Complexity (depth)", ylabel = "errors", title = "Errors v.s. max_iter of Multinomial Log. Regression")
    ax.legend()
    plt.ylim(0.9,1)
    return best_iter, iter_score
```

```
lrange = np.arange(30,150)
best_iter, iter_score = find_iter(lrange)
best_iter, iter_score
```

```
(57, 0.9649805447470817)
```



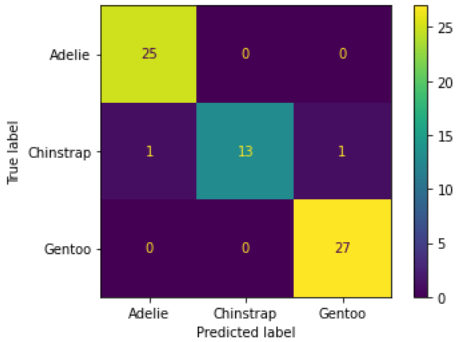
```
#fits training data and scores the train and test data
LG = LogisticRegression(solver = 'saga', max_iter = best_iter,random_state=10)
LG.fit(Xtrain,ytrain)
LG_train_score = LG.score(Xtrain,ytrain)
LG_test_score = LG.score(Xtest,ytest)
print("score on training " + str(LG_train_score))
print("score on testing " + str(LG_test_score))
cv1 = cross_val_score(LG,X_num[best_combo],y_num,cv=5).mean()
print("cross validation score " + str(cv1))
```

```
score on training 0.9649805447470817
score on testing 0.9701492537313433
cross validation score 0.9629326923076924
```

Confusion Matrix

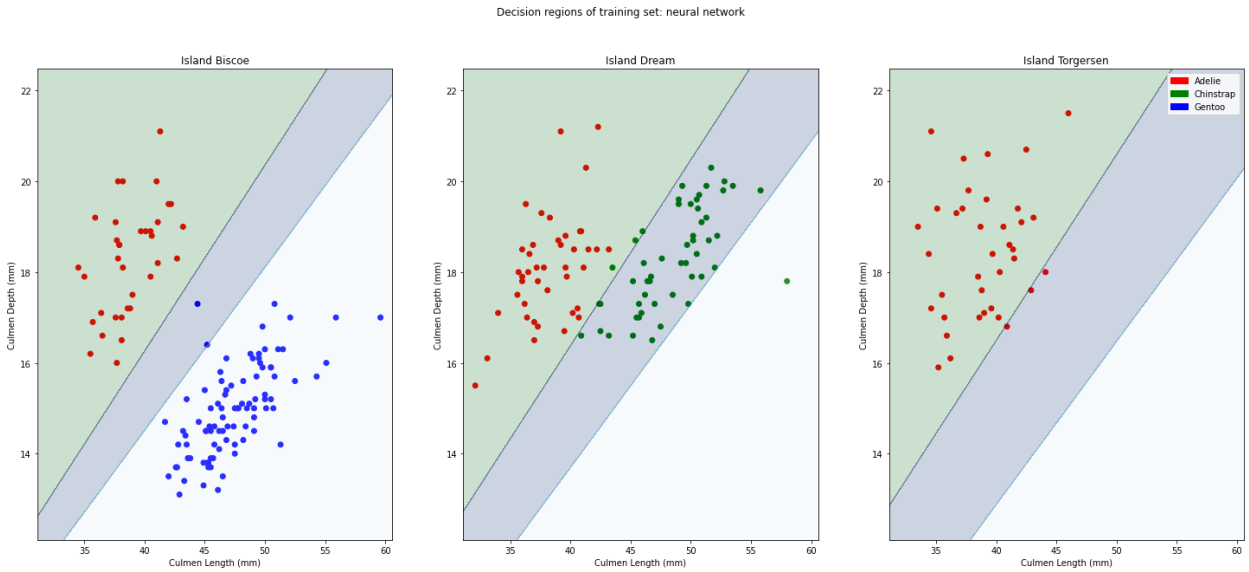
```
#model evaluation on test sets: confusion matrix
y_test_pred1 = LG.predict(Xtest)
accuracy1 = accuracy_score(ytest, y_test_pred1)
print("Accuracy score: " + str(accuracy1))
cm = confusion_matrix(ytest,y_test_pred1)
disp1 = ConfusionMatrixDisplay(cm, display_labels=penguins['Species'].str.split().str.get(0).unique())
disp1.plot()
```

```
Accuracy score: 0.9701492537313433
```

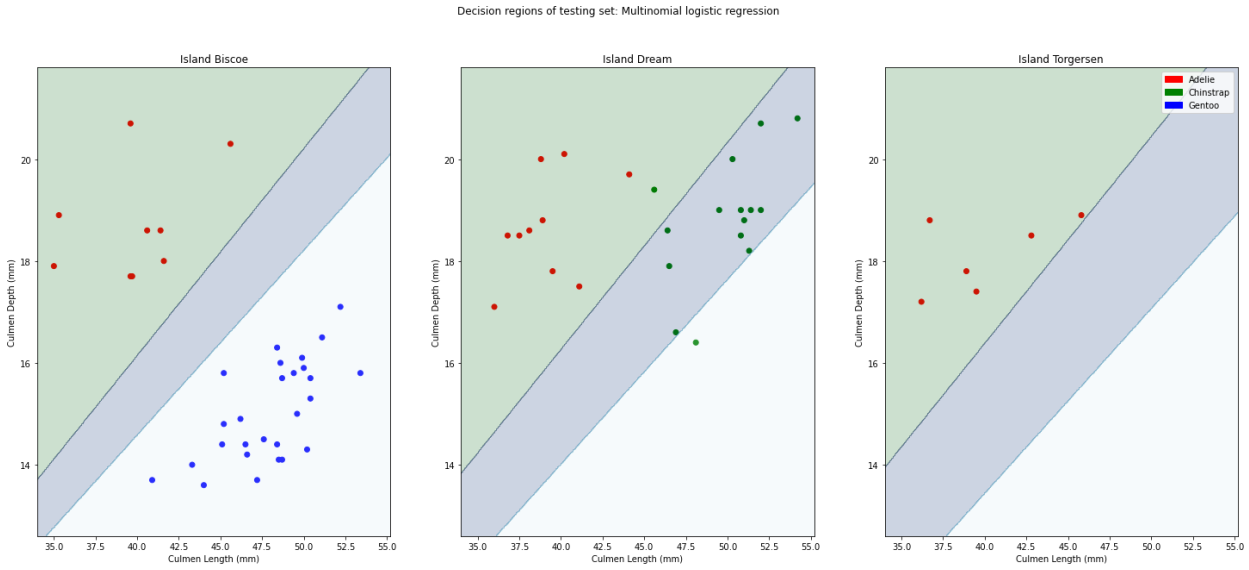


Decision Regions

```
colormap = {0:"red", 1: "green", 2:"blue"}

dec_bound(best_combo,Xtrain[best_combo],ytrain,LG,island_encoding,colormap)
#create a legend
colors = list(colormap.values())
species = list(species_encoding.keys())
plt.legend(handles=[mpatches.Patch(color= colors[0], label=species[0]),
                    mpatches.Patch(color= colors[1], label=species[1]),
                    mpatches.Patch(color= colors[2], label=species[2])], loc='upper right')
plt.suptitle("Decision regions of training set: neural network")
```




```
colormap = {0:"red", 1: "green", 2:"blue"}

dec_bound(best_combo,Xtest[best_combo],ytest,LG,island_encoding,colormap)
#create a legend
colors = list(colormap.values())
species = list(species_encoding.keys())
plt.legend(handles=[mpatches.Patch(color= colors[0], label=species[0]),
                    mpatches.Patch(color= colors[1], label=species[1]),
                    mpatches.Patch(color= colors[2], label=species[2])], loc='upper right')
plt.suptitle("Decision regions of testing set: Multinomial logistic regression")
```



Discussion of Multinomial Logistic Regression

- The Multinomial Logistic Regression model looks relatively accurate, as the accuracy score is 97%. The training error is 96.5%, and 97% for testing error/accuracy, 96.3% for cv
- The best max-iter for the model is found to be 69.

- From the confusion matrix, the model wrongly predicted 1 Chinstrap as Adelie, and 1 Chinstrap as Gentoo on testing sets (unforeseen data).
- From the decision region plots, it seems all decision boundaries are linear, and the misclassified penguins have culmen length and depth measurements close to the boundaries.

5.3 Neural Networks

Description: （Reference: Sklearn documentation: 1.17. Neural network models (supervised))

- Multi-layer Perceptron (MLP) is a supervised learning algorithm that learns a function f(): - > where m and n are the dimension of the input and the output respectively. In this case, both m and n equal 3. Given a set of features and a target, MLP can learn a non-linear function approximator for either classification or regression with the hidden layers between input and output layers. $R^m \rightarrow R^n$

Advantages:

- Capability to learn non-linear models.
- Support multi-label classification with softmax functions which give the probabilities of an input belonging to each of the output class

Disadvantages:

- MLP requires tuning a number of hyperparameters such as the number of hidden neurons, layers, and iterations.
- MLP is sensitive to feature scaling and could perform better with min-max normalization

Cross Validation¶

```
#training the model on the training sets, tuning for hyperparamters with grid search

mlp = MLPClassifier(random_state=1)
parameter_space = {'hidden_layer_sizes': [(10,30,10),(20,),(10,5,3),],
    'activation': ['identity','logistic','tanh', 'relu'],
    'solver': ['sgd', 'adam','ibfgs'],
    'alpha': [0.0001, 0.01, 0.05, 0.1,1],
    'learning_rate': ['constant','invscaling','adaptive'],
    'max_iter': [100,500,1500,2000,3000]}
#mlp = GridSearchCV(mlp, parameter_space, n_jobs=-1, cv=5)
#mlp.fit(Xtrain, ytrain)
#print('Best parameters found:\n', mlp.best_params_)
# to save time, the best params from the commented out section is:
best_params =  {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (10, 5, 3), 'learning_rate': 'constant', 'max_iter': 3000, 'so
best_params
```

```
{'activation': 'tanh',
 'alpha': 0.0001,
 'hidden_layer_sizes': (10, 5, 3),
 'learning_rate': 'constant',
 'max_iter': 3000,
 'solver': 'adam'}
```

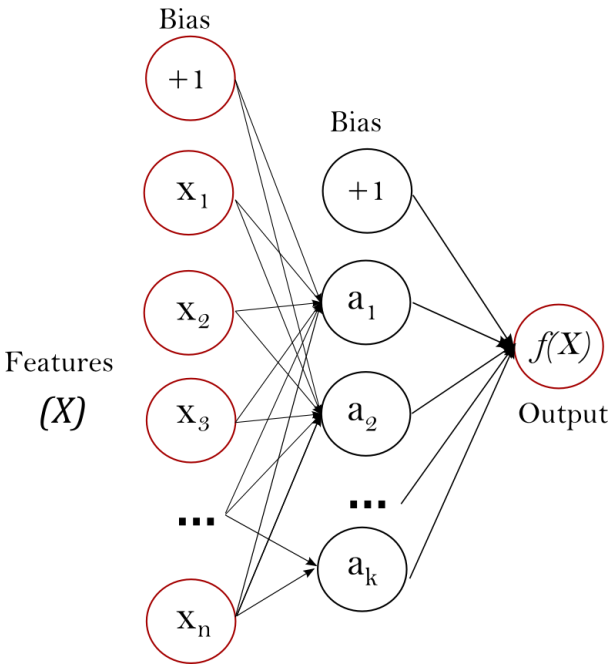
```
mlp = MLPClassifier(activation = 'tanh', alpha = 0.0001, hidden_layer_sizes = (10,5,3),
                    learning_rate = 'constant', solver = 'adam',random_state=1,max_iter=3000)
mlp.fit(Xtrain,ytrain)
cv2 = cross_val_score(mlp,X_num[best_combo],y_num,cv=5).mean()
print("score on training " + str(mlp.score(Xtrain,ytrain)))
print("score on testing " + str(mlp.score(Xtest,ytest)))
print("cross validation score " + str(cv2))
```

```
score on training 1.0
score on testing 0.9850746268656716
cross validation score 0.996875
```

Grid search is a model (estimator) optimization method that performs exhaustive search in the given hyperparameter space for optimal performance (best mean cross-validated score). For a typical multi-layer perceptron (one-layer)

(References: MLP documentation, Grid Search sklearn documentation, Google Developer ML Glossary)

Some of the important hyperparameters include: A hidden layer in the MLP is consist of weights, bias, and activation func In the above one-layer MLP, the red layer provides a linear transformation of the input features X, where each neuron corresponds to weights and bias (top). The summation of all neurons in the red layer is than passed into an activation function (black layer) that is capbale of introducing non-linearity into the model to capture more arbitrary features more flexibly. The activation function maps the inputs to the ouputs which are then feed into the next layer (if there is any).



Hidden layer size:

- notice that there are two dimensions, total number of layers, number of neurons in each layer. The overall architecture of the layers could affect training accuracy, although there is no guarantee that a wider (more neurons in each)/deeper (more layers) network gives better results. it depends!
- the parameter hidden_layer_sizes is a tuple where the ith element represents the number of neurons in the ith hidden layer.

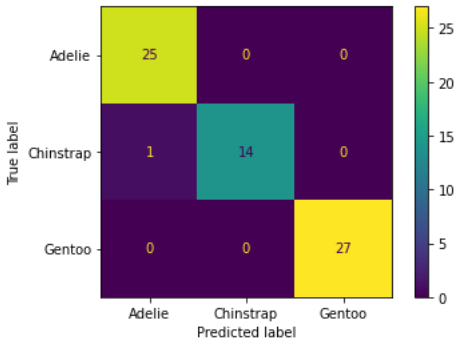
General hyperparameters:

- solver/optimizer:
  - decides how the model weights/bias are update in the training process
  - options given by the documentation
- activation:
  - explained above
  - options given by the documentation
- learning_rate:
  - step size in gradient descent, smaller rate covers more of the curve when searching for optimum but would take longer training time, options given by the documentation
  - options given by the documentation
- alpha:
  - the strength of the L2 regularization/penalty term that could decourage overfitting. a larger alpha encourage smaller weights and give smoother decision boundaries.
- max_iter:
  - Maximum number of iterations. the solver would iterate until convergence to an optimal solution under this constraint

Confusion Matrix

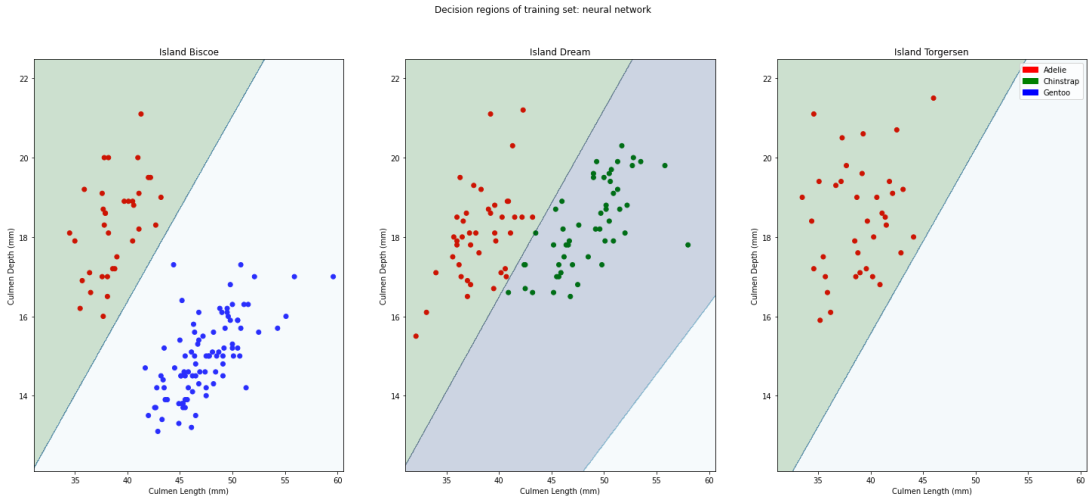
```
#model evaluation on test sets: confusion matrix
y_test_pred2 = mlp.predict(Xtest)
accuracy2 = accuracy_score(ytest, y_test_pred2)
print("Accuracy score: " + str(accuracy2))

cm = confusion_matrix(ytest,y_test_pred2)
disp2 = ConfusionMatrixDisplay(cm, display_labels=penguins['Species'].str.split().str.get(0).unique())
disp2.plot()
```

```
Accuracy score: 0.9850746268656716
```

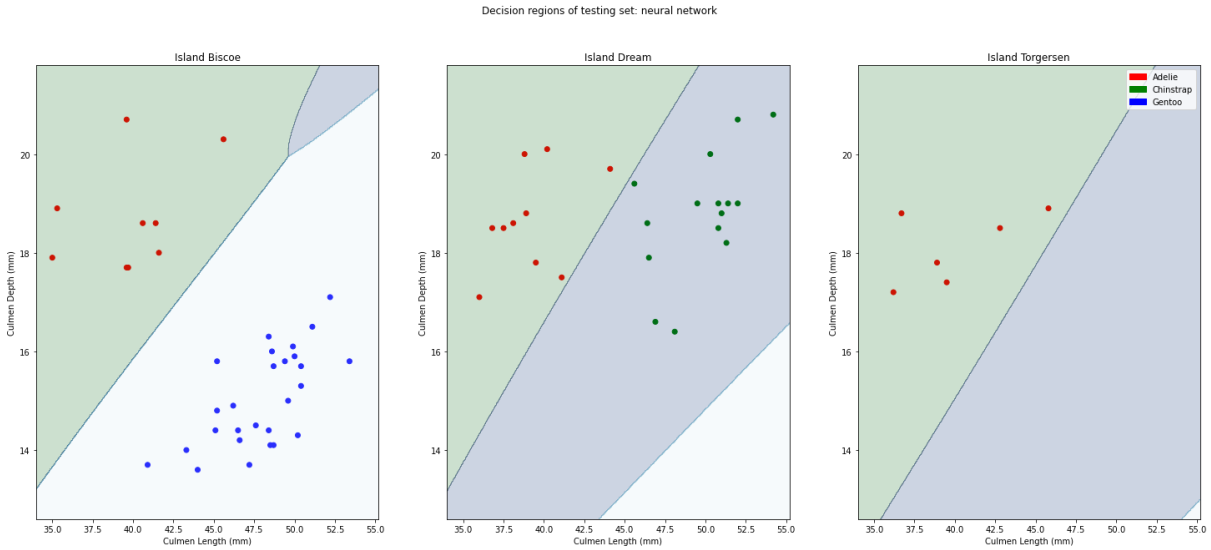


Decision Regions¶

```
colormap = {0:"red", 1: "green", 2:"blue"}
dec_bound(best_combo,Xtrain,ytrain,mlp,island_encoding,colormap)

#create a legend
colors = list(colormap.values())
species = list(species_encoding.keys())
plt.legend(handles=[mpatches.Patch(color= colors[0], label=species[0]),
                    mpatches.Patch(color= colors[1], label=species[1]),
                    mpatches.Patch(color= colors[2], label=species[2])], loc='upper right')


plt.suptitle("Decision regions of training set: neural network")
```



```
dec_bound(best_combo,Xtest,ytest,mlp,island_encoding,colormap)
#not in the function
#create a legend
colors = list(colormap.values())
species = list(species_encoding.keys())
plt.legend(handles=[mpatches.Patch(color= colors[0], label=species[0]),
                    mpatches.Patch(color= colors[1], label=species[1]),
                    mpatches.Patch(color= colors[2], label=species[2])], loc='upper right')


plt.suptitle("Decision regions of testing set: neural network")
```



Discussion of Neural Networks

- The Neural Network model achieved a final accuracy score of 98.5%, with training errors of 1 (no mistakes) and testing error/accuracy of 98.5% cross validation score is 99.7%
- The best hyperparameters are {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (10, 5, 3), 'learning_rate': 'constant', 'max_iter': 3000, 'solver': 'adam'}. Although further tuning the layer size could perhaps increase the model performance on unseen data, but it took too long to tune so I skipped it.
- From the confusion matrix, the model wrongly predicted 1 Chinstrap as Adelie
- From the decision region plots, it seems all decisions boundaries are linear.
- The neural network models could perhaps perform better with more datasets, predictors, and further tunings of the hyperparameters (especially scaling/normalization)

5.4 K-Nearest Neighbors

Cross Validation

```
from sklearn.model_selection import GridSearchCV
#create new a knn model
nbrs = KNeighborsClassifier()
#create a dictionary of all values we want to test for n_neighbors
nd = {'n_neighbors' : np.arange(1, 25)}
#use gridsearch to test all values for n_neighbors
nbrs = GridSearchCV(nbrs, nd, cv=5)
#fit model to data
nbrs.fit(Xtrain, ytrain)
nbrs.best_params_
```

```
{'n_neighbors': 2}
```

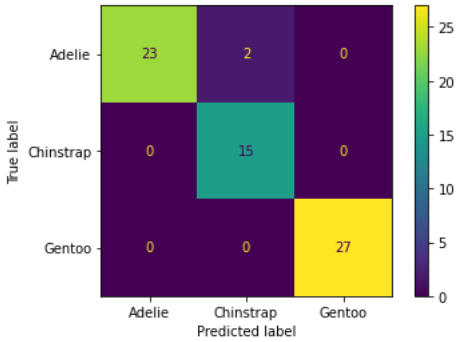
```
nbrs = KNeighborsClassifier(n_neighbors=2)
nbrs.fit(Xtrain,ytrain)
cv3 = cross_val_score(nbrs,X_num[best_combo],y_num,cv=5).mean()
print("score on training " + str(nbrs.score(Xtrain,ytrain)))
print("score on testing " + str(nbrs.score(Xtest,ytest)))
print("cross validation score " + str(cv3))
```

```
score on training 0.9883268482490273
score on testing 0.9701492537313433
cross validation score 0.9783653846153847
```

Confusion Matrix

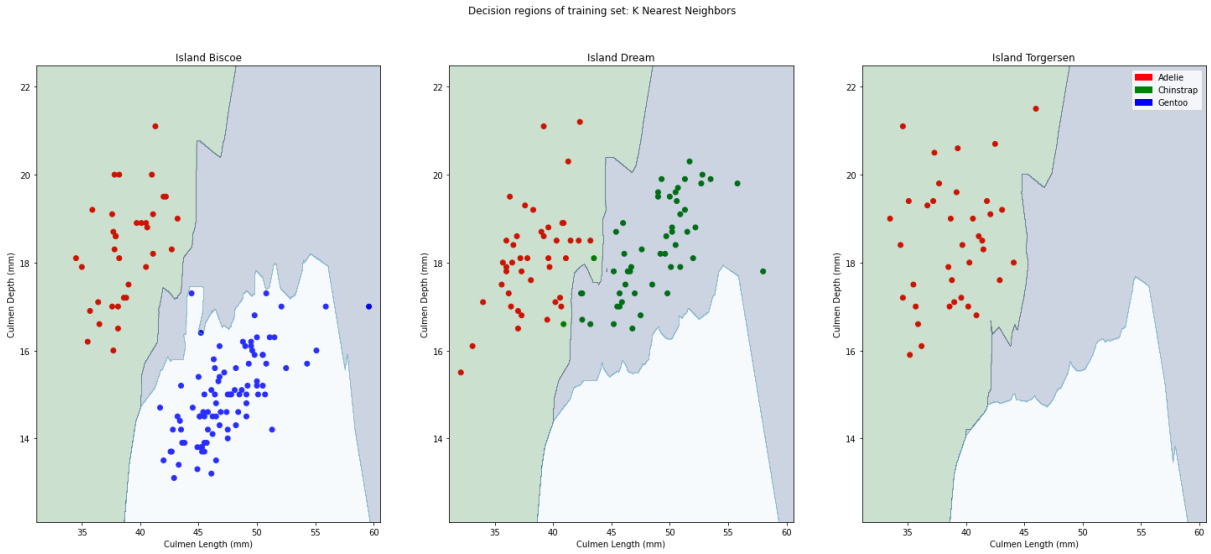
```
#model evaluation on test sets: confusion matrix
y_test_pred3 = nbrs.predict(Xtest)
accuracy3 = accuracy_score(ytest, y_test_pred3)
print("Accuracy score: " + str(accuracy3))

cm = confusion_matrix(ytest,y_test_pred3)
disp3 = ConfusionMatrixDisplay(cm, display_labels=penguins['Species'].str.split().str.get(0).unique())
disp3.plot()
```

```
Accuracy score: 0.9701492537313433
```

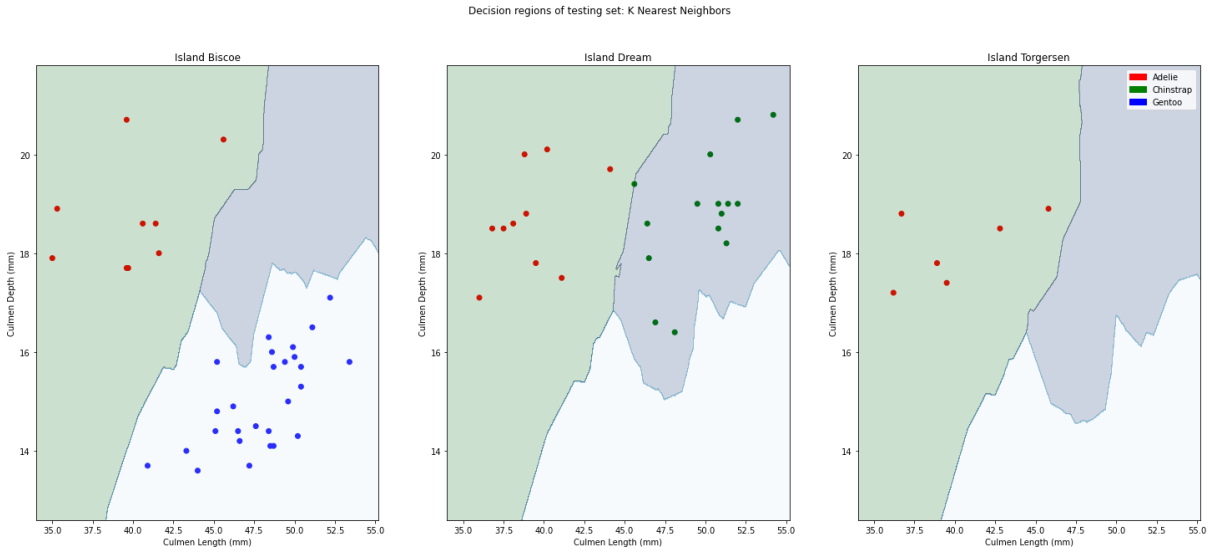


Decision Regions¶

```
#decision boundary
colormap = {0:"red", 1: "green", 2:"blue"}

dec_bound(best_combo,Xtrain,ytrain,nbrs,island_encoding,colormap)
#create a legend
colors = list(colormap.values())
species = list(species_encoding.keys())
plt.legend(handles=[mpatches.Patch(color= colors[0], label=species[0]),
                    mpatches.Patch(color= colors[1], label=species[1]),
                    mpatches.Patch(color= colors[2], label=species[2])], loc='upper right')

plt.suptitle("Decision regions of training set: K Nearest Neighbors")
```




```
#decision boundary
colormap = {0:"red", 1: "green", 2:"blue"}

dec_bound(best_combo,Xtest,ytest,nbrs,island_encoding,colormap)
#create a legend
colors = list(colormap.values())
species = list(species_encoding.keys())
plt.legend(handles=[mpatches.Patch(color= colors[0], label=species[0]),
                    mpatches.Patch(color= colors[1], label=species[1]),
                    mpatches.Patch(color= colors[2], label=species[2])], loc='upper right')

plt.suptitle("Decision regions of testing set: K Nearest Neighbors")
```



Discussion of KNN

- The KNearest Neighbors model looks relatively accurate, as the accuracy score is 97.0%.
- The best number of neighbors is 2
- From the confusion matrix, the model wrongly predicted 2 Adelie penguins to be Chinstrap penguins.

- In the decision regions above for training, 1 gentoo penguin was predicted to be a chinstrap penguin on biscoe island. 2 chinstrap penguins were predicted to be adelie penguins and 1 was predicted to be a gentoo penguin on dream island. on torgensen, 2 adelie were predicted to be chinstrap penguins.

6.1 Result overview

```
cv = [cv1,cv2,cv3]
acc = [accuracy1,accuracy2,accuracy3]
pd.DataFrame({"model":["MLR","MLP","KNN"],"cross_val":cv,"accuracy":acc})
```

	model	cross_val	accuracy
0	MLR	0.962933	0.970149
1	MLP	0.996875	0.985075
2	KNN	0.978365	0.970149

- To select the best combination of features (two quantitative & one qualitative predictors), we performed exhaustive searches on all possible combinations for each of the three models respectively. The optimal combinations are the same for all 3 models: ['Culmen Length (mm)', 'Culmen Depth (mm)', 'Island']
- the MLR and KNN models have the same final accuracy scores of 0.970, while MLP has 0.985.
- In terms of classifying unforseen data, both MLP and KNN make two mistakes while MLP makes 1. MLR misclassifies two Chinstrap, one as Adelie, the other as Gentoo, KNN misclassifies two Adelie as Chinstrap, and MLP misclassifies one Chinstrap as Adelie.
- The decision regions of KNN turn out to be different from both MLR and MLP, which have mostly linear boundaries.
- For this particular classification task and the given dataset, we prefer **K Nearest Neighbors** over MLR and MLP because of its has the highest cross-val score and accuracy. While accuracy states that the model performs classification well with the particular testing set we have, the cross validation score is a relatively unbiased estimated of the model's generalized performance. Coming to tuning, training, and visualizing decision-regions, MLp is not as time-efficient as MLR and KNN. And we have performed determined the optimal numeber of iterations to be 69 based on mean cross validation scores.

6.2 Model Improvement

- Model performance when more data are availble.
  - Experimentally, to figure out which model performs more accurately at classifying the penguins, we can evaluate the accuracy scores against increasing sizes of the training data.
  - Theoretically, multinomial logistic regression could work well with relatively small amount of data, which means there is less likely to be improvement when the amount of data increases. In contrast, neural networks need a larger amount of data to attain higher accuracy.
- Model performance when different data are availble
  - Experimentally, we could alternate the total numbers of measurements we include as our predictors, albeit the multicolinearity among them could induce overfitting as we train our models, thus doing little/the opposite to improve the overall classification accuracy.
  - In real world, complex measurement, missingness, and dependence have been important topics in statistical research. Many classification or pattern identification tasks are based on much messier datasets that require in-depth cleaning and reparametrization.
- They are more than just data points, they are penguins! (Reference: Britannica)
  - We should pay closer attention to the penguin-side of the data as well, understanding how the tasks are done originally as a biological question.

- There are currently around 18-21 types penguins in total and, perhaps surprisingly, the majority of them reside on islands rather than temperate zone islands rather than Antartica. Taking a closer look at the phylogenetic tree of penguins, penguins' appearant differences in crests, foots, fur-colors, body-size, and habitats are used to divide them into different Genres.
- (Reference: Avibase:) Under the Genus Pygoscelis, nicknamed "brush-tailed penguins", we have Adélie, chinstrap, and gentoo. Although some reports that the gentoo could be further divided into 4 sub-species depending on their habitats.
- A closer look at the respective species names give us the insight that these penguins could be distinguished by their appearances quite easily. Chinstrap is most easily identified because of the narrow black band under its head as if its wearing a black helmet. While Adelie penguins have completely black heads, Gentoo a splash of white above the eyes. That makes classification easier! Without the need to take body measurements or perform genetic sequencing. (Image source: WWF)

From left to right: Adelie, Chinstrap, Gentoo.