pic16a_final_proj

June 2, 2021

1 Final Project Group 19:

Group Member:
- Benson Zu
- Flora Wang
- Yijun He

Group 19 HW7
work contribution statement
Benson Zu, Olivia He, Flora Wang.
For this homework, for a lot of parts we worked as a team together over zoom and Google Colab.
Below are the specific work distribution for each step:
1. Data Cleaning: Benson Zu, Flora Wang, Yijun He
2. Feature Selection: Flora Wang
3. Logistic Regressino Modeling: Benson Zu
4. Decision Tree Modelling: Yijun He
5. Support Vector machine Modelling: Flora Wang

2 1. Data Preparation

2.1 1.1 Import Data

[ ]:
```
#Import Modules
import pandas as pd
from matplotlib import pyplot as plt
from sklearn import tree, preprocessing
import numpy as np

#Import Raw Data
url = "https://philchodrow.github.io/PIC16A/datasets/palmer_penguins.csv"
penguins_raw = pd.read_csv(url)
```

Now we inspect the raw data to see what we can do with it.

[ ]:
```
penguins_raw = penguins_raw[penguins_raw.Sex != '.']
print(penguins_raw.shape)
penguins_raw.head()
```

```
(343, 17)
```

```
[ ]:   studyName  Sample Number  … Delta 13 C (o/oo)
Comments
0   PAL0708              1  …               NaN  Not enough blood for
isotopes.
1   PAL0708              2  …         -24.69454
NaN
2   PAL0708              3  …         -25.33302
NaN
3   PAL0708              4  …               NaN              Adult not
sampled.
4   PAL0708              5  …         -25.32426
NaN

[5 rows x 17 columns]
```

2.2 1.2 Split Data

2.2.1 1.2.1 Split into X and y

In classification machine learning, a model reads in some feature data set, then make decision (what class this data belongs to) and gives the target data. In this project, we wish to classify different species of penguins. Thus we split the features of the penguins into features data set (X), and the species into a target data set (y).

```
[ ]: X = penguins_raw.drop(["Species"],axis = 1)
     y = pd.DataFrame(penguins_raw['Species'])
     X.shape,y.shape
```

```
[ ]: ((343, 16), (343, 1))
```

2.2.2 1.2.2 Split into train and test sets

In Machine Learning, we first train our model by feeding it some training dataset, then evaluate the model by using the trained parameters to test on some testing dataset. Therefore, we split our data into training and testing sets randomly using the sklearn toolkit.

```
[ ]: #Split the training and testing sets
     import random
     from sklearn.model_selection import train_test_split
     random.seed(0)
     X_train, X_test, y_train, y_test = train_test_split(X,y,random_state = 0,␣
      ↪train_size = 0.7)
```

```
[ ]: # Check data shape
     X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
[ ]: ((240, 16), (103, 16), (240, 1), (103, 1))
```

[ ]:
```
# inspect data
X_train.head()
```

[ ]:
```
     studyName  Sample Number  … Delta 13 C (o/oo) Comments
33     PAL0708             34  …          -25.14591      NaN
322    PAL0910            103  …          -26.21651      NaN
106    PAL0910            107  …          -26.48973      NaN
34     PAL0708             35  …          -25.23061      NaN
230    PAL0708             11  …          -25.39330      NaN

[5 rows x 16 columns]
```

2.3 1.3 Data Cleaning

As we examined the raw data, we want to 1. remove some columns that won't be helpful for our prediction 2. format some columns so that they can be used for prediction

2.3.1 1.3.1 Check invariant Columns

[ ]:
```
#check columns that we think are the same for every row
print(set(penguins_raw['Stage']))
print(set(penguins_raw['Region']))
```

```
{'Adult, 1 Egg Stage'}
{'Anvers'}
```

2.3.2 1.3.2 Basic data cleaning functions

[ ]:
```
def  data_clean(df_input,df_output):
  '''
  Drop columns that are not features (such as individual ID)
  features that are the same for every row,
  and isotope ratios that have too many NA Values.
  Input: X and Y
  Return: Clean X and Y
  '''
  not_features = ['Comments', "Sample Number","studyName", \
                  "Individual ID", "Date Egg"]
  same_features = ['Region', "Stage"]
  isotope = ["Delta 15 N (o/oo)","Delta 13 C (o/oo)"]
  df_input = df_input.drop(not_features + same_features + isotope, axis= 1)

  #Drop Na Value
  df_input_no_na = pd.concat([df_input,df_output],axis =1).dropna()
  df_input = df_input_no_na.drop(["Species"],axis =1)
  df_output = df_input_no_na[["Species"]]
  return df_input,df_output
```

```
def concise_output (df):
  '''
  Make the species name to be concise as the first word
  '''
  df["Species"] = df["Species"].str.split().str.get(0)
  return df


def cat_to_num_input (df):
  '''
  Transform categorical data into numerical data
  '''
  from sklearn import preprocessing;
  le = preprocessing.LabelEncoder();

  x = df.select_dtypes(include=['object'])
  names = x.columns
  for i in names:
    df[i]=le.fit_transform(df[i]);
  return df
```

2.3.3 1.3.3 Training Data Cleaning

Version 1: Categorical Data Reserved

[ ]:
```
X_train_clean,y_train_clean = data_clean(X_train,y_train)
y_train_clean= concise_output(y_train_clean)
print(X_train.shape,y_train.shape)
print(X_train_clean.shape,y_train_clean.shape)
```

```
(240, 16) (240, 1)
(232, 7) (232, 1)
```

Version 2: Categorical Data Changed to Numerical

[ ]:
```
X_train_clean_cat_to_num = cat_to_num_input(X_train_clean)
y_train_clean_cat_to_num = cat_to_num_input(y_train_clean)
```

3 2. Select Features

3.1 2.1 Feature Selection

Since we need to use at least one qualitative feature according to the instruction, we decided to choose quanlitative and quantitative features separately.

[ ]:
```
```

```
categorical = ['Island', 'Clutch Completion', 'Sex'] #choose 1 from these
numerical = ['Culmen Length (mm)',        'Culmen Depth (mm)',        'Flipper␣
↪Length (mm)',        'Body Mass (g)'] #choose 2
```

[ ]:
```
#select categorical feature
from sklearn.feature_selection import mutual_info_classif
scores = mutual_info_classif(X_train_clean_cat_to_num[categorical],␣
↪y_train_clean, random_state=19)
scores
  #[5.19146404e-01, 3.06011632e-04, 8.88509390e-03]
  #according to the highest score, we choose 'Island'
```

[ ]:
```
array([5.19146404e-01, 3.06011632e-04, 8.88509390e-03])
```

[ ]:
```
#select numerical feature
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif

# feature extraction
test = SelectKBest(score_func=f_classif, k=2)
fit = test.fit(X_train_clean[numerical], y_train_clean)
# summarize scores
fit.scores_
  #[256.11629605, 287.45389304, 368.74338002, 264.36692772]
  #according to the highest score, we choose 'Culmen Length (mm)','Flipper␣
↪Length (mm)'
```

[ ]:
```
array([256.11629605, 287.45389304, 368.74338002, 264.36692772])
```

[ ]:
```
cols_selected = ['Culmen Length (mm)','Flipper Length (mm)','Island']
X_train_selected = X_train_clean_cat_to_num[cols_selected]
```

3.2 2.2 Feature Modification

3.2.1 2.2.1 Normalized Variables prepared

In the next step, we normalize our feature data, so that the parameters for each feature can be chosen by evenly evaluating by the model.

[ ]:
```
#Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
cols_scale = ['Culmen Length (mm)','Flipper Length (mm)']
X_train_selected_norm=X_train_selected
X_train_selected_norm[cols_scale] =  sc.
↪fit_transform(X_train_selected[cols_scale])
X_train_selected_norm.head()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/usr/local/lib/python3.7/dist-packages/pandas/core/indexing.py:1734:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  isetter(loc, value[:, i].tolist())
```

[ ]:

	Culmen Length (mm)	Flipper Length (mm)	Island
33	-0.634232	-1.266292	1
322	0.560964	0.957681	0
106	-1.070574	-0.190176	0
34	-1.487944	-0.477140	1
230	-0.634232	0.885940	0

3.2.2 2.2.1 Dummy Variables prepared

We change the categorical data into one hot encoding, so that it doesn't indicate an order of importance.

[ ]:

```
# use the 3 features
#change island data into one hot encoding
X_train_selected_dummy_norm = pd.get_dummies(X_train_selected_norm, columns =␣
↪['Island'])
X_train_selected_dummy_norm.head()
```

[ ]:

	Culmen Length (mm)	Flipper Length (mm)	Island_0	Island_1	Island_2
33	-0.634232	-1.266292	0	1	0
322	0.560964	0.957681	1	0	0
106	-1.070574	-0.190176	1	0	0
34	-1.487944	-0.477140	0	1	0
230	-0.634232	0.885940	1	0	0

3.2.3 2.2.3 Test Data Modification for Evaluation

[ ]:

```
# Basic Cleaning of the data
X_test_clean,y_test_clean = data_clean(X_test,y_test)
y_test_clean = concise_output(y_test_clean)
```

```
y_test_clean_cat_to_num = cat_to_num_input(y_test_clean)

# Categorical to numeric
X_test_clean_cat_to_num = cat_to_num_input(X_test_clean)

#Feature Selection
X_test_selected = X_test_clean_cat_to_num[cols_selected]

#Dummies and Normalization Preparation
X_test_selected_norm=X_test_selected
X_test_selected_norm[cols_scale] =  sc.
 ↪fit_transform(X_test_selected[cols_scale])
X_test_selected_norm.head()

#X_test_selected_dummy_norm = pd.get_dummies(X_test_selected_norm, columns =␣
 ↪['Island'])
#X_test_selected_dummy_norm.head()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:14:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/usr/local/lib/python3.7/dist-packages/pandas/core/indexing.py:1734:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  isetter(loc, value[:, i].tolist())
```

[ ]:

	Culmen Length (mm)	Flipper Length (mm)	Island
92	-1.614916	-1.028596	1
281	0.477072	1.543602	0
132	-1.134787	-0.456997	1
280	0.322745	0.614752	0
6	-0.774691	-1.314396	2

[ ]:

```
X_test_selected_dummy_norm = pd.get_dummies(X_test_selected_norm, columns =␣
 ↪['Island'])
X_test_selected_dummy_norm.head()
```

[ ]:

	Culmen Length (mm)	Flipper Length (mm)	Island_0	Island_1	Island_2
92	-1.614916	-1.028596	0	1	0
281	0.477072	1.543602	1	0	0
132	-1.134787	-0.456997	0	1	0
280	0.322745	0.614752	1	0	0
6	-0.774691	-1.314396	0	0	1

4 3. Logistic Regression

4.1 3.1 Modeling

4.1.1 3.2.1 Find the best parameter

[ ]:
```
# use the GridSearchMethods
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
```

[ ]:
```
# Avoid warning message
from warnings import simplefilter
from sklearn.exceptions import DataConversionWarning
simplefilter("ignore", category=DataConversionWarning)
```

[ ]:
```
random.seed(19)
param_grid = [
    {
     'penalty' : ['l1', 'l2'],
    'C' : np.logspace(-4, 4, 20),
    'solver' : ['liblinear']
     }
]
## Build the template for training
model_logistic = GridSearchCV(LogisticRegression(), param_grid, cv=4,
                              scoring='accuracy');
# Train the model
model_logistic.fit(X_train_selected_dummy_norm,y_train_clean_cat_to_num);
#print the best parameter
print("Best parameters set found on development set:")
print(model_logistic.best_params_)
```

```
Best parameters set found on development set:
{'C': 1.623776739188721, 'penalty': 'l2', 'solver': 'liblinear'}
```

4.2 3.2 Evaluation

[ ]:
```
#print the scores of training
print("The accuracy score for traning is: ")
```

```
print(model_logistic.
 ↪score(X_train_selected_dummy_norm,y_train_clean_cat_to_num))
print("The accuracy score for testing is: ")
print(model_logistic.score(X_test_selected_dummy_norm,y_test_clean_cat_to_num))
```

```
The accuracy score for traning is:
0.978448275862069
The accuracy score for testing is:
0.9801980198019802
```

4.3 3.3 Inspection

4.3.1 3.2.1 Confusion matrices

[ ]:
```
#training data's confusion matrix
from sklearn.metrics import confusion_matrix
#training data's confusion matrix
y_train_pred=model_logistic.predict(X_train_selected_dummy_norm)
c_logistic_train = confusion_matrix(y_train_clean_cat_to_num, y_train_pred)
print(c_logistic_train)
```

```
[[94  2  0]
 [ 3 46  0]
 [ 0  0 87]]
```

[ ]:
```
#testing data's confusion matrix
y_test_pred=model_logistic.predict(X_test_selected_dummy_norm)
c_logistic_test = confusion_matrix(y_test_clean_cat_to_num, y_test_pred)
print(c_logistic_test)
```

```
[[48  1  1]
 [ 0 19  0]
 [ 0  0 32]]
```

What are falsely classified? For the traning dataset, we misclassified 3 Adelie as a Chinstrap penguin, and 2 Chinstrap as Adelie penguins. For the testing dataset, we misclassified 1 Chinstrap as a Adelie penguin ,and 1 Adelie as a Gentoo penguin,

4.3.2 3.2.2 Possible Reasons

[ ]:
```
#change target data from a column vector to a 1d array
y_train_1d = np.ravel(y_train_clean_cat_to_num)
y_test_1d = np.ravel(y_test_clean_cat_to_num)
```

3.2.2.1 Inspect the wrongly predicted Training data

[ ]:
```
#inspect the wrongly specified training data
false_index_train = y_train_1d != y_train_pred
mistakes_train = X_train_selected_norm[false_index_train]
```

```
print("Training: true:"+ str(y_train_1d[false_index_train]))
print("          pred:"+ str(y_train_pred[false_index_train]))
mistakes_train
```

```
Training: true:[1 1 0 0 1]
          pred:[0 0 1 1 0]
```

[ ]:

	Culmen Length (mm)	Flipper Length (mm)	Island
184	-0.330690	-1.051068	1
182	-0.634232	-1.051068	1
43	-0.027149	-0.405399	1
99	-0.197891	-0.692363	1
172	-0.349662	-1.481515	1

[ ]:

```
# inspect Adelie data in the training dataset
adelie = y_train_1d==0
X_train_selected_norm[adelie].iloc[:10] #check 10 rows
```

[ ]:

	Culmen Length (mm)	Flipper Length (mm)	Island
33	-0.634232	-1.266292	1
106	-1.070574	-0.190176	0
34	-1.487944	-0.477140	1
97	-0.748060	-0.405399	1
85	-0.558347	-0.548881	1
45	-0.880860	-0.835845	1
134	-1.165430	-1.051068	1
108	-1.165430	-1.481515	0
90	-1.620743	0.025048	1
46	-0.596290	-1.409774	1

[ ]:

```
# inspect Chinstrap data in the training dataset
chinstrap = y_train_1d==1
X_train_selected_norm[chinstrap].iloc[:10] #check 10 rows
```

[ ]:

	Culmen Length (mm)	Flipper Length (mm)	Island
159	1.338789	-0.333658	1
166	0.314336	-0.835845	1
156	1.604389	-0.333658	1
215	2.192501	0.383753	1
199	0.902448	0.742458	1
158	0.352279	-1.696738	1
153	1.092162	-0.405399	1
213	0.485078	-0.907586	1
219	1.130104	-0.261917	1
198	1.111133	-0.835845	1

Explaination to the wrong classification between `Adelie and Chinstrap`:

* Based on the inspection of the wrongly classified traning set, we found that logistic model wrongly predicted some true Chinstrap to Adelie species.
* Inspecting the dataset, we find that the Chinstrap typically has a positive normalized cumlem length and Adelie typically has a negative normalized cumlem length.
* Therefore, for potential outliers of Chinstrap with negative normalized culem length (like data `184, 182, 172`), is more likely to be classified as Adelie.
* For a similar reason, when the normalized culem length is not negative enough,(like data `43,99`), they are more likely to be wrongly classified as Christrap.

3.2.2.2 Inspect the wrongly predicted Testing data

[ ]:
```
#inspect the wrongly specified training data
false_index_test = y_test_1d != y_test_pred
mistakes_test = X_test_selected_norm[false_index_test]

print("Testing  true:"+ str(y_test_1d[false_index_test]))
print("       pred:"+ str(y_test_pred[false_index_test]))
mistakes_test
```

```
Testing  true:[0 0]
       pred:[1 2]
```

[ ]:

	Culmen Length (mm)	Flipper Length (mm)	Island
49	-0.191678	-0.599897	1
101	-0.414595	0.257503	0

[ ]:
```
# inspect Gentoo data in the training dataset
gentoo = y_train_1d==2
X_train_selected_norm[gentoo].iloc[:10] #check 10 rows
```

[ ]:

	Culmen Length (mm)	Flipper Length (mm)	Island
322	0.560964	0.957681	0
230	-0.634232	0.885940	0
306	-0.159948	1.172905	0
334	0.371250	1.101164	0
221	1.092162	2.033797	0
291	0.409193	1.388128	0
235	0.959362	1.101164	0
228	-0.178920	0.527235	0
312	0.238450	0.742458	0
335	2.059701	2.033797	0

Explaination to the wrong classification between `Adelie and Gentoo`:
* Based on the inspection of the wrongly classified traning set, we found that logistic model wrongly predicted some true Adelie to Christrap or Gentoo species.
* The reason of misclassification of true Adelie to Christrap (data `49`) is the same as the `3.2.2.1`: the culemn length may not negative enough to convience the model to classify it as a Adelie.
* Inspecting the Gentoo dataset, we find that the it typically has a positive normalized Flipper Length length and Adelie typically has a negative normalized Flipper length.

* Therefore, for potential outliers of Adelie with posisitve flipper length (like data 101), is more likely to be classified as Gentoo.

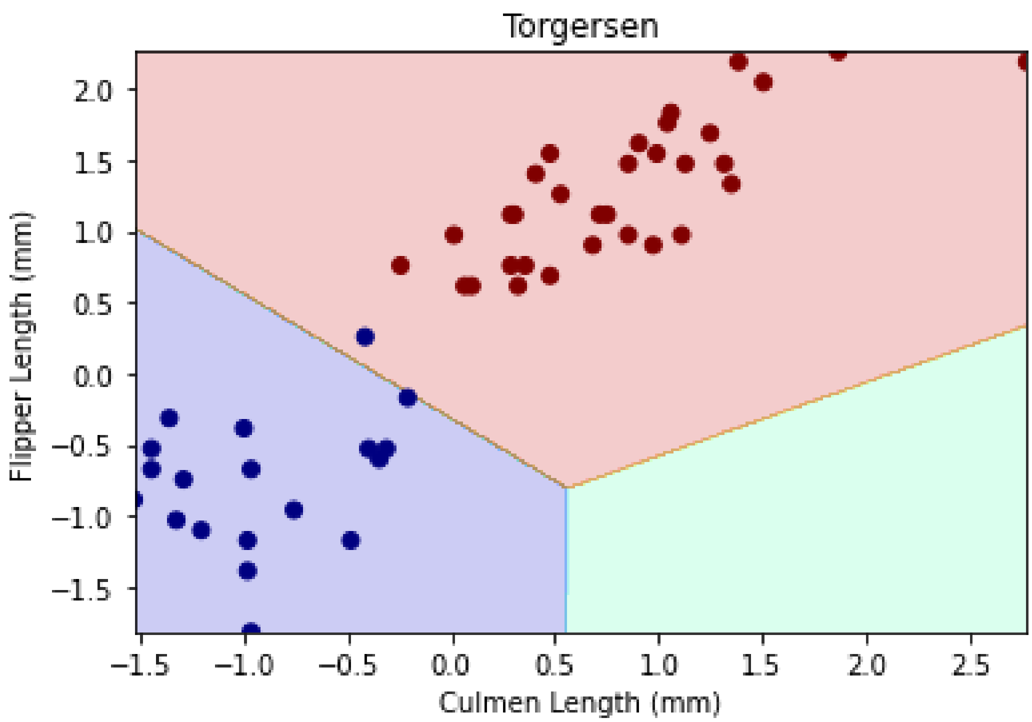
4.4 3.4 Visualization

[ ]:
```
def new_plot_regions(c,X,y,Island):
    """
    The function is used to plot the region plots of classificaiton models
    , which has more than 3 paramters and one of them is Island.
    It takes input:
    c: training models
    X: input data
    y: output data
    Island: 0 or 1 or 2
    output: a regions plot
    """
    X_f = X[X["Island"] == Island]
    y_f = y[X["Island"] == Island]
    print("There are ",len(X_f), "cases in the island", Island)
    x0 = X_f['Culmen Length (mm)']
    x1 = X_f['Flipper Length (mm)']
    grid_x = np.linspace(x0.min(),x0.max(),501)
    grid_y = np.linspace(x1.min(),x1.max(),501)
    xx, yy = np.meshgrid(grid_x, grid_y)
    XX = xx.ravel()
    YY = yy.ravel()
    ones = [1]*251001
    zeros = [0]*251001
    XX_list = XX.tolist()
    YY_list = YY.tolist()

    input_data0 = list(zip(XX_list,YY_list,ones,zeros,zeros))
    input_data1 = list(zip(XX_list,YY_list,zeros,ones,zeros))
    input_data2 = list(zip(XX_list,YY_list,zeros,zeros,ones))

    if Island == 0:
        input_data = input_data0
        title = "Torgersen"
    elif Island == 1:
        input_data = input_data1
        title = "Dream"
    elif Island == 2:
        input_data = input_data2
        title = "Biscoe"

    random.seed(19)
    p = c.predict(input_data)
```

```
    p = p.reshape(xx.shape)

    fig, ax = plt.subplots(1)
    # use contour plot to visualize the predictions
    ax.contourf(xx, yy, p, cmap = "jet", alpha = 0.2, vmin = 0, vmax = 2)
    # plot the data
    ax.scatter(x0, x1, c = np.array(y_f), cmap = "jet", vmin = 0, vmax = 2)

    ax.set(xlabel = "Culmen Length (mm)",
        ylabel = "Flipper Length (mm)",title=title)
```

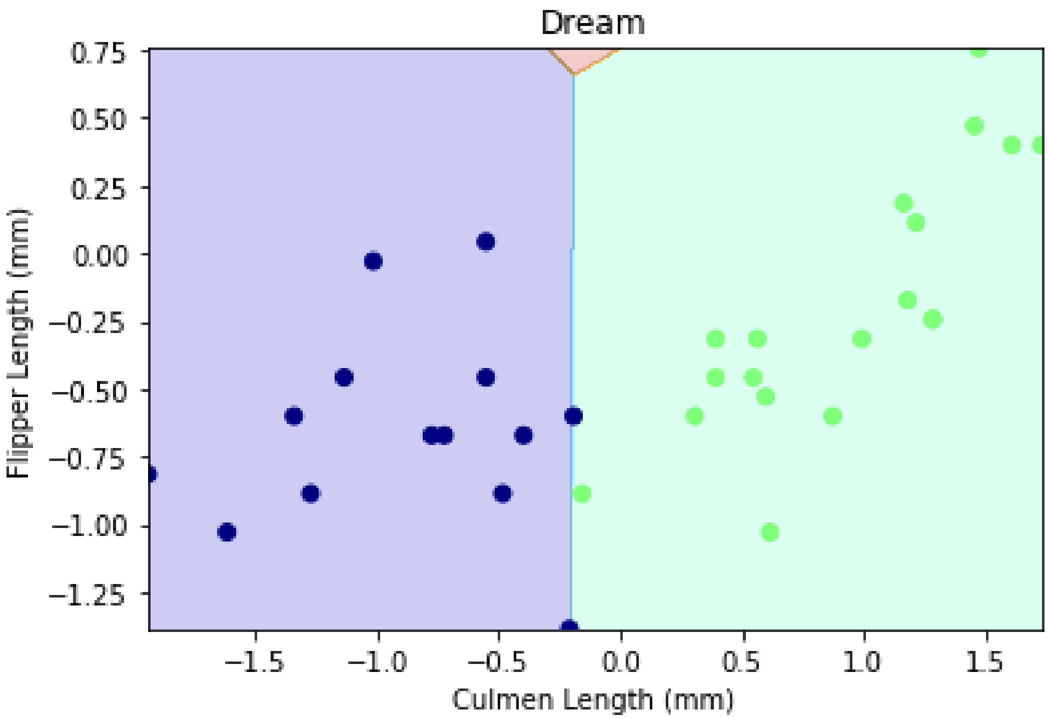
[ ]:
```
new_plot_regions(model_logistic,X_test_selected_norm,y_test_clean_cat_to_num,0)
```

```
There are  51 cases in the island 0
```



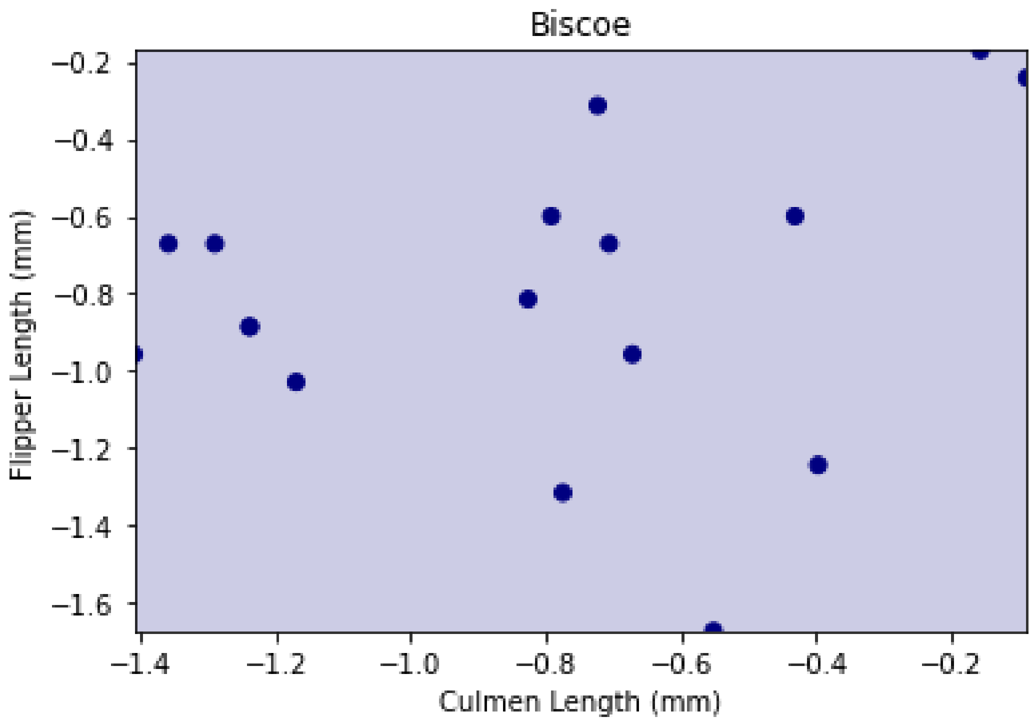
[ ]:
```
new_plot_regions(model_logistic,X_test_selected_norm,y_test_clean_cat_to_num,1)
```

```
There are  33 cases in the island 1
```



[ ]:
```
new_plot_regions(model_logistic,X_test_selected_norm,y_test_clean_cat_to_num,2)
```

```
There are  17 cases in the island 2
```



5 4. Decision Tree

5.1 4.1 Modeling

5.1.1 4.1.1 Find the best parameter

[ ]:
```
from sklearn.model_selection import cross_val_score
from sklearn import tree
T = tree.DecisionTreeClassifier(max_depth= 3)

# Use K-fold cross-validation to estimate the optimal complexity of a model.
cv_scores = cross_val_score(T, X_train_selected_norm, y_train_clean , cv=10)
cv_scores
```

[ ]:
```
array([1.        , 0.91666667, 0.91304348, 1.        , 0.86956522,
       0.91304348, 0.95652174, 1.        , 0.91304348, 0.91304348])
```

[ ]:
```
# Find the best max_depth parameter that results in highest cv_score
best_score = 0

for d in range(1,30):
    T = tree.DecisionTreeClassifier(max_depth = d,random_state=1)
```

```
    cv_score = cross_val_score(T, X_train_selected_norm, y_train_clean, cv=10).
↪mean()
    if cv_score > best_score:
        best_depth = d
        best_score = cv_score

print("Best Depth:",best_depth)
```

```
Best Depth: 4
```

5.1.2 4.1.2 Modeling

[ ]:
```
T = tree.DecisionTreeClassifier(max_depth= best_depth)
T
```

[ ]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=4, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

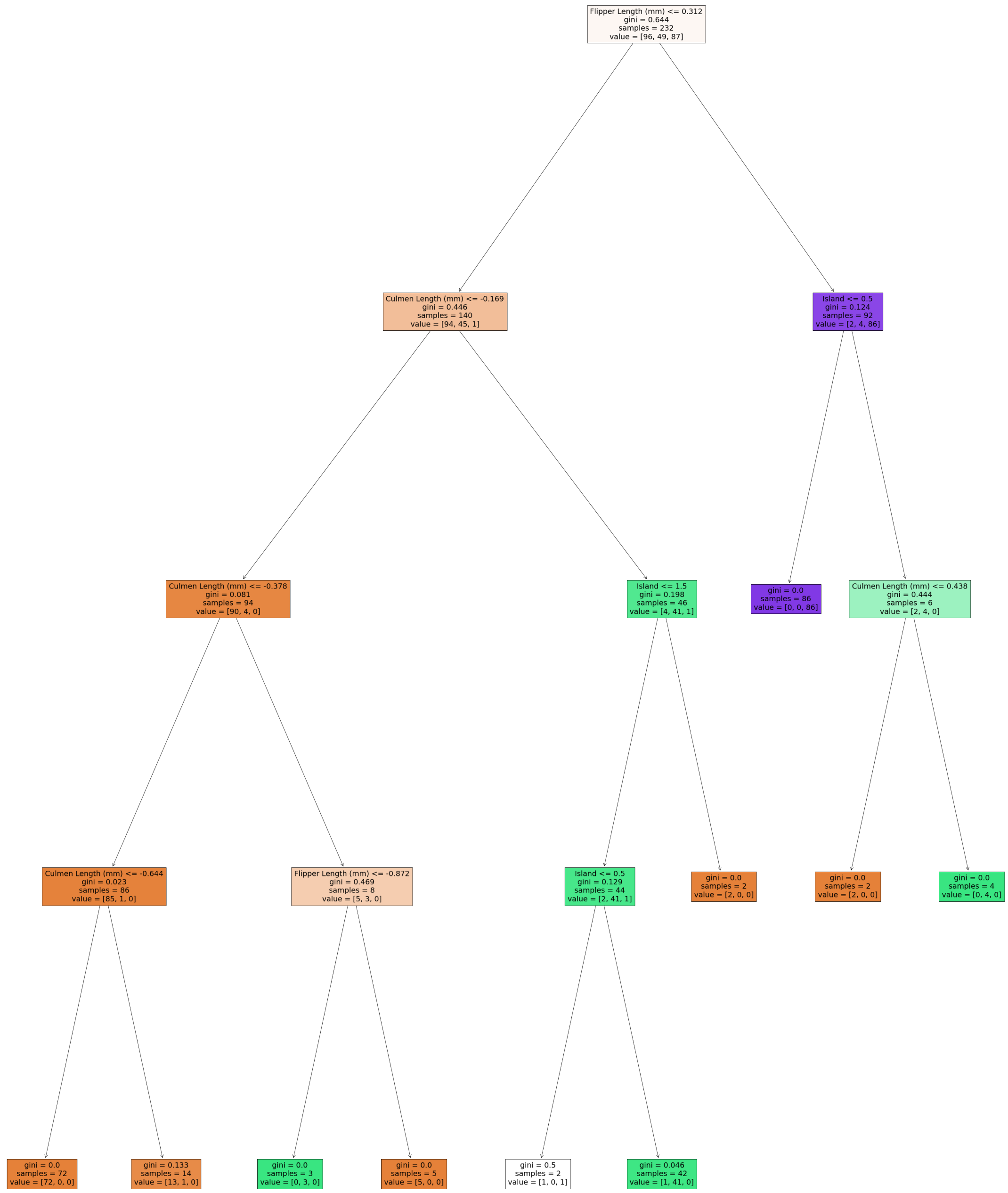
5.2 4.2 Evaluation

[ ]:
```
T.fit(X_train_selected_norm, y_train_clean)
T.score(X_train_selected_norm, y_train_clean),T.score(X_test_selected_norm,␣
↪y_test_clean)
```

[ ]:
```
(0.9870689655172413, 0.9900990099009901)
```

Using best_depth as parameter, the model scores on training and testing data are very close

[ ]:
```
# plot the decision tree
fig, ax = plt.subplots(1, figsize = (50, 70))
p = tree.plot_tree(T,
                   filled = True,
                   feature_names = cols_selected)
```


Flipper Length (mm) <= 0.312
gini = 0.644
samples = 232
value = [96, 49, 87]
Culmen Length (mm) <= -0.169
gini = 0.446
samples = 140
value = [94, 45, 1]
Island <= 0.5
gini = 0.124
samples = 92
value = [2, 4, 86]
Culmen Length (mm) <= -0.378
gini = 0.081
samples = 94
value = [90, 4, 0]
Island <= 1.5
gini = 0.198
samples = 46
value = [4, 41, 1]
gini = 0.0
samples = 86
value = [0, 0, 86]
Culmen Length (mm) <= 0.438
gini = 0.444
samples = 6
value = [2, 4, 0]
Culmen Length (mm) <= -0.644
gini = 0.023
samples = 86
value = [85, 1, 0]
Flipper Length (mm) <= -0.872
gini = 0.469
samples = 8
value = [5, 3, 0]
Island <= 0.5
gini = 0.129
samples = 44
value = [2, 41, 1]
gini = 0.0
samples = 2
value = [2, 0, 0]
gini = 0.0
samples = 2
value = [2, 0, 0]
gini = 0.0
samples = 4
value = [0, 4, 0]
gini = 0.0
samples = 72
value = [72, 0, 0]
gini = 0.133
samples = 14
value = [13, 1, 0]
gini = 0.0
samples = 3
value = [0, 3, 0]
gini = 0.0
samples = 5
value = [5, 0, 0]
gini = 0.5
samples = 2
value = [1, 0, 1]
gini = 0.046
samples = 42
value = [1, 41, 0]

[ ]:
```
T_20 = tree.DecisionTreeClassifier(max_depth = 20)
T_20.fit(X_train_selected_norm, y_train_clean)
T_20.score(X_train_selected_norm, y_train_clean),T_20.
↪score(X_test_selected_norm, y_test_clean)
```

[ ]:
```
(1.0, 0.9801980198019802)
```

Compare to other parameters, the model shows perfect fitting on training data but a lower fit on testing data, which indicates overfitting.

5.3 4.3 Inspection

5.3.1 4.3.1 Confusion Matrix

[ ]:
```
# confusion matrix of training data
dt_yhat_train=T.predict(X_train_selected_norm)
cm_dt_train=confusion_matrix(y_train_clean,dt_yhat_train)
print(cm_dt_train)
```

```
[[95  1  0]
 [ 1 48  0]
 [ 1  0 86]]
```

[ ]:
```
# confusion matrix of testing data
dt_yhat_test=T.predict(X_test_selected_norm)
cm_dt_test=confusion_matrix(y_test_clean,dt_yhat_test)
print(cm_dt_test)
```

```
[[49  1  0]
 [ 0 19  0]
 [ 0  0 32]]
```

What are falsely classified?

For the traning dataset, we misclassified 1 Chinstrap penguin as Adelie penguin, 1 Adelie penguin as Chinstrap, and 1 Getoo penguin as Adelie.

For the testing dataset, we misclassified 1 Chinstrap as a Adelie penguin ,and 1 Adelie as a Chinstrap penguin.

5.3.2 4.3.2 Possible Reasons

4.3.2.1 inspect the wrongly specified training data

[ ]:
```
false_index_train = y_train_1d != dt_yhat_train
mistakes_train = X_train_selected_norm[false_index_train]

print("Training: true:"+ str(y_train_1d[false_index_train]))
print("          pred:"+ str(dt_yhat_train[false_index_train]))
mistakes_train
```

```
Training: true:[1 0 2]
          pred:[0 1 0]
```

[ ]:

	Culmen Length (mm)	Flipper Length (mm)	Island
182	-0.634232	-1.051068	1
43	-0.027149	-0.405399	1
318	0.788620	0.096789	0

[ ]:
```
#inspect Adelie data in the training dataset
adelie = y_train_1d==0
X_train_selected_norm[adelie].iloc[:10] #check 10 rows
```

[ ]:

	Culmen Length (mm)	Flipper Length (mm)	Island
33	-0.634232	-1.266292	1
106	-1.070574	-0.190176	0
34	-1.487944	-0.477140	1
97	-0.748060	-0.405399	1
85	-0.558347	-0.548881	1
45	-0.880860	-0.835845	1
134	-1.165430	-1.051068	1
108	-1.165430	-1.481515	0
90	-1.620743	0.025048	1
46	-0.596290	-1.409774	1

[ ]:
```
#inspect Chinstrap data in the training dataset
chinstrap = y_train_1d==1
X_train_selected_norm[chinstrap].iloc[:10] #check 10 rows
```

[ ]:

	Culmen Length (mm)	Flipper Length (mm)	Island
159	1.338789	-0.333658	1
166	0.314336	-0.835845	1
156	1.604389	-0.333658	1
215	2.192501	0.383753	1
199	0.902448	0.742458	1
158	0.352279	-1.696738	1
153	1.092162	-0.405399	1
213	0.485078	-0.907586	1
219	1.130104	-0.261917	1
198	1.111133	-0.835845	1

[ ]:
```
# inspect Gentoo data in the training dataset
gentoo = y_train_1d==2
X_train_selected_norm[gentoo].iloc[:10] #check 10 rows
```

[ ]:

	Culmen Length (mm)	Flipper Length (mm)	Island
322	0.560964	0.957681	0
230	-0.634232	0.885940	0

```
306          -0.159948          1.172905       0
334           0.371250          1.101164       0
221           1.092162          2.033797       0
291           0.409193          1.388128       0
235           0.959362          1.101164       0
228          -0.178920          0.527235       0
312           0.238450          0.742458       0
335           2.059701          2.033797       0
```

According to the first 10 rows in Chinstrap data, we observe that "Culman Length (mm)" are all positive.

However, this feature of the misclassified Chinstrap penguin (No.182) is negative.

Thus, it is reasonable to indicate that the sign of "Culman Length (mm)" is a decisive factor for prediction.

[ ]:
```
small_flip = np.asarray(X_train_selected_norm['Flipper Length (mm)'])<0.1
X_train_selected_norm[np.logical_and(small_flip, gentoo)]
```

[ ]:
```
     Culmen Length (mm)  Flipper Length (mm)  Island
318             0.78862             0.096789       0
```

As for the misclassification of (No.318), we find that it is the only Gentoo penguin with small flipper (less than 0.1 mm).

Therefore, the magnitude of flipper length might be a decisive indicator for prediction.

[ ]:
```
neg_flip = np.asarray(X_train_selected_norm['Flipper Length (mm)'])<-3
X_train_selected_norm[np.logical_and(neg_flip, adelie)]
```

[ ]:
```
Empty DataFrame
Columns: [Culmen Length (mm), Flipper Length (mm), Island]
Index: []
```

[ ]:
```
X_train_selected_norm[np.logical_and(neg_flip, chinstrap)]
```

[ ]:
```
Empty DataFrame
Columns: [Culmen Length (mm), Flipper Length (mm), Island]
Index: []
```

As for the misclassification of (No.43), since it has large negative value of normalized flipper length(-0.40 mm), both Adelie and Chinstrap do not have similar case.

Therefore, it is more reasonable to be treated as an outlier of the data.

4.3.2.2 inspect the wrongly specified testing data

[ ]:
```
false_index_test = y_test_1d != dt_yhat_test
mistakes_test = X_test_selected_norm[false_index_test]
```

```
print("true:"+ str(y_test_1d[false_index_test]))
print("pred:"+ str(dt_yhat_test[false_index_test]))
mistakes_test
```

```
true:[0]
pred:[1]
```

[ ]:

	Culmen Length (mm)	Flipper Length (mm)	Island
37	-0.208825	-1.385846	1

[ ]:

```
negative_cul = np.asarray(X_train_selected_norm['Culmen Length (mm)'])<0
X_train_selected_norm[np.logical_and(negative_cul, chinstrap)]
```

[ ]:

	Culmen Length (mm)	Flipper Length (mm)	Island
216	-0.140977	0.025048	1
184	-0.330690	-1.051068	1
182	-0.634232	-1.051068	1
174	-0.197891	-1.051068	1
172	-0.349662	-1.481515	1

As for the misclassified Adelie penguin (No.37), when comparing to other Chinstrap penguins with negative "Culman Length (mm)", their magnitude of "Flipper Length (mm)" are similar (between -1.05 and -1.50, except for No.216, which could be treated as an outlier). Therefore, both the sign of culman length and the magnitude of flipper length might affect the prediction.

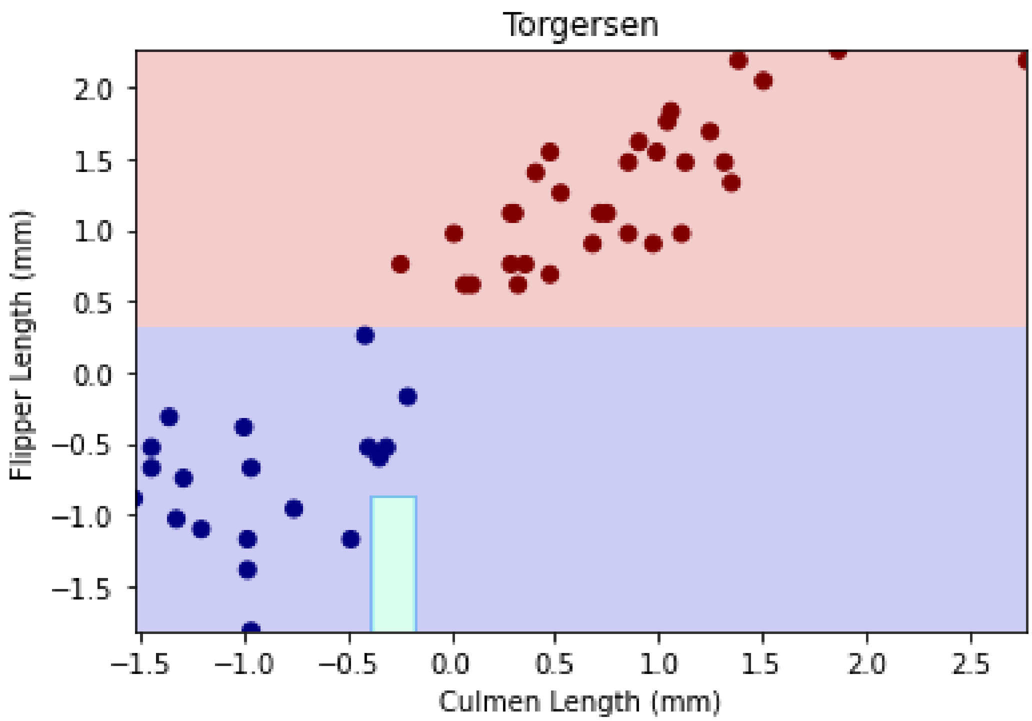
5.4 4.4 Visualization

[ ]:

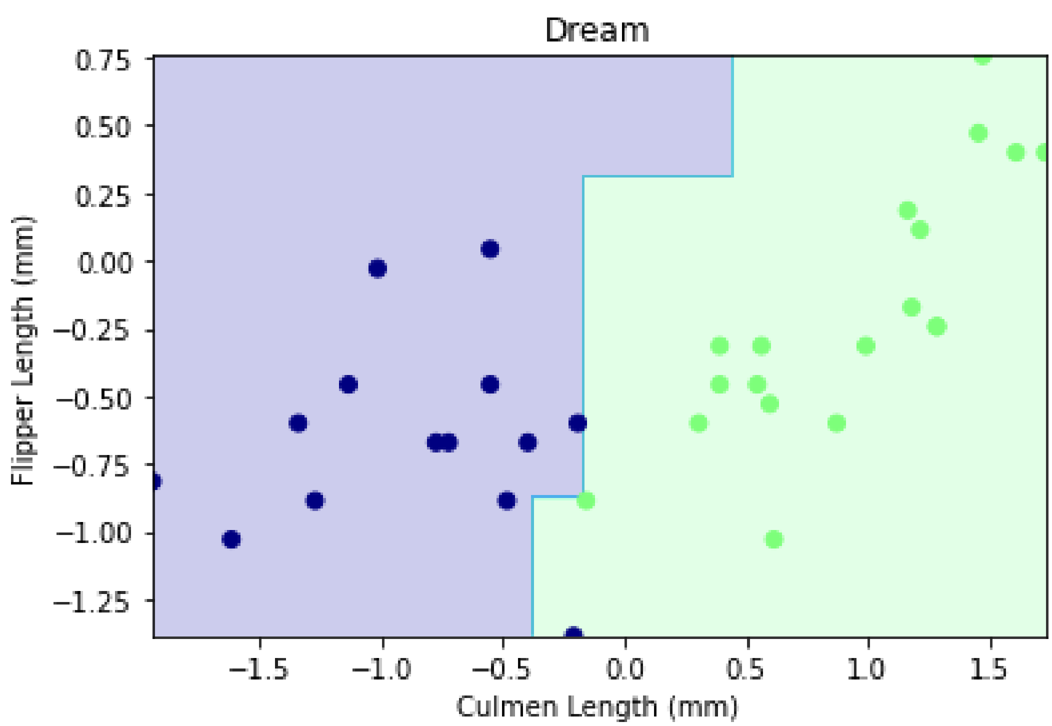
```
def new2_plot_regions(c, X, y,Island):
    X_f = X[X["Island"] == Island]
    y_f = y[X["Island"] == Island]
    # for convenience, give names to the two
    # columns of the data
    x0 = X_f['Culmen Length (mm)']
    x1 = X_f['Flipper Length (mm)']

    # create a grid
    grid_x = np.linspace(x0.min(),x0.max(),501)
    grid_y = np.linspace(x1.min(),x1.max(),501)
    xx, yy = np.meshgrid(grid_x, grid_y)

    # extract model predictions, using the
    # np.c_ attribute to join together the
    # two parts of the grid.
    # array.ravel() converts an multidimensional
    # array into a 1d array, and we use array.reshape()
    # to turn the resulting predictions p
    # back into 2d
```

```
    XX = xx.ravel()
    YY = yy.ravel()
    XX_list = XX.tolist()
    YY_list = YY.tolist()

    twos = [2]*251001
    ones = [1]*251001
    zeros = [0]*251001

    if Island == 0:
        input_data = list(zip(XX_list,YY_list,zeros))
        title = "Torgersen"
    elif Island == 1:
        input_data = list(zip(XX_list,YY_list,ones))
        title = "Dream"
    elif Island == 2:
        input_data = list(zip(XX_list,YY_list,twos))
        title = "Biscoe"

    random.seed(19)
    p = c.predict(input_data)
    p = p.reshape(xx.shape)

    fig, ax = plt.subplots(1)
    # use contour plot to visualize the predictions
    ax.contourf(xx, yy, p, cmap = "jet", alpha = 0.2, vmin = 0, vmax = 2)
    # plot the data
    ax.scatter(x0, x1, c = np.array(y_f), cmap = "jet", vmin = 0, vmax = 2)

    ax.set(xlabel = "Culmen Length (mm)",
        ylabel = "Flipper Length (mm)",title=title)
```

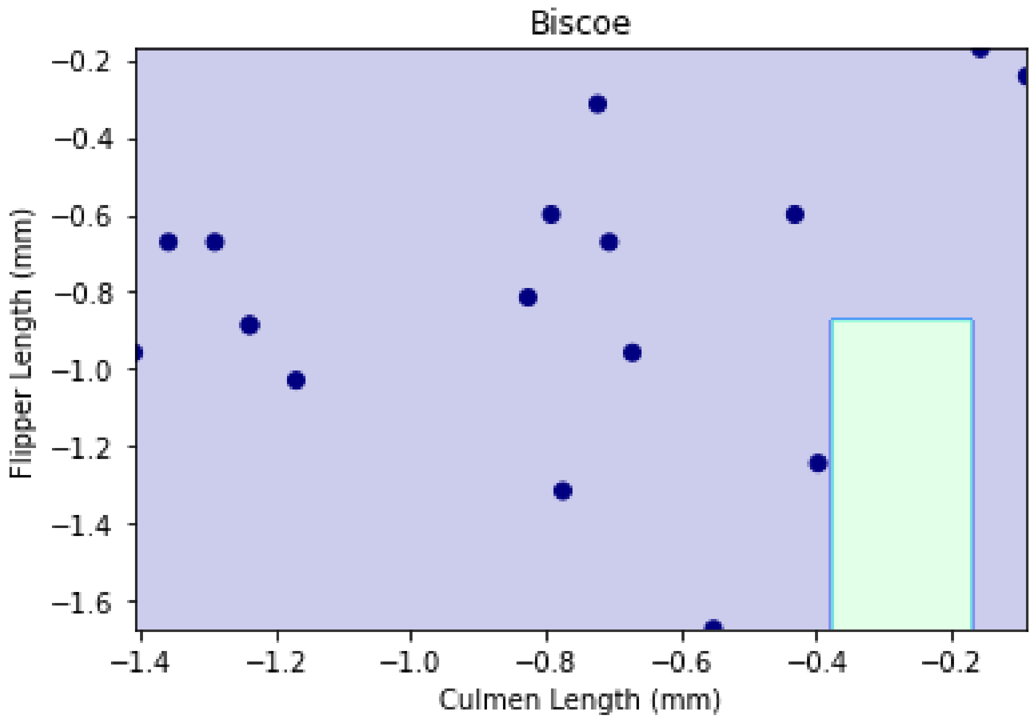
[ ]:
```
new2_plot_regions(T, X_test_selected_norm, y_test_clean, 0)
```




```
[ ]: new2_plot_regions(T, X_test_selected_norm, y_test_clean, 1)
```



[ ]:
```
new2_plot_regions(T, X_test_selected_norm, y_test_clean, 2)
```



6 5. Support Vector Machine

Support Vector Machine model works by finding threshholds that classify each data into a group. **How it works:** As we classify by threshold, if we determine the threshold by finding the midpoint between the edges of features of each group, then the model will be very sensitive to outliers, which will cause inaccurate prediction. Therefore we handle outliers by ignoring them when determining threshold, and we call the ignored data "support vectors".

6.1 5.1 Modeling

6.1.1 5.1.1 Find the best parameters

[ ]:
```
#import necessary modules
from sklearn import svm
from sklearn.model_selection import GridSearchCV
```

[ ]:
```
#use gridsearch to find the best parameters:
  #C: regularizor used to prevent overfitting
  #gamma: parameter in the kernal function
  #kernel: the kerel function we'll use
```

```
random.seed(0)
param_grid = [
      {'C': [0.5, 1, 10, 100],
       'gamma': ['scale', 1, 0.1, 0.01, 0.001, 0.0001],
       'kernel': ['rbf', 'sigmoid', 'linear']
      }]

#create the model object with the best parameters
clf_svm = GridSearchCV(svm.SVC(), param_grid, cv=5,
                           scoring='accuracy');
```

6.1.2 5.1.2 Train the model

[ ]:
```
#change target data from a column vector to a 1d array
y_train_1d = np.ravel(y_train_clean_cat_to_num)
y_test_1d = np.ravel(y_test_clean_cat_to_num)
```

[ ]:
```
#train the model
clf_svm.fit(X_train_selected_dummy_norm,y_train_1d)
```

[ ]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid=[{'C': [0.5, 1, 10, 100],
                          'gamma': ['scale', 1, 0.1, 0.01, 0.001, 0.0001],
                          'kernel': ['rbf', 'sigmoid', 'linear']}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

[ ]:
```
#check the best parameters we found
print(clf_svm.best_params_)
```

```
{'C': 0.5, 'gamma': 0.1, 'kernel': 'rbf'}
```

6.2 5.2 Evaluation

[ ]:
```
#check scores of the model
print(clf_svm.score(X_train_selected_dummy_norm,y_train_1d))
print(clf_svm.score(X_test_selected_dummy_norm,y_test_1d))
```

```
0.9741379310344828
0.9801980198019802
```

The scores are high and relatively close to each other, thus we believe that there is no issue of overfitting.

6.3 5.3 Inspection

6.3.1 5.3.1 Confusion Matrices

[ ]:
```
#training data's confusion matrix
yhat_train=clf_svm.predict(X_train_selected_dummy_norm)
cm_svm_train = confusion_matrix(y_train_1d, yhat_train)
print(cm_svm_train)
```

```
[[95  1  0]
 [ 5 44  0]
 [ 0  0 87]]
```

[ ]:
```
#testing data's confusion matrix
yhat_test=clf_svm.predict(X_test_selected_dummy_norm)
cm_svm_test = confusion_matrix(y_test_1d, yhat_test)
print(cm_svm_test)
```

```
[[49  0  1]
 [ 1 18  0]
 [ 0  0 32]]
```

What are falsely classified? For the traning dataset, we misclassified 1 Adelie as a Chinstrap penguin, and 5 Chinstrap as Adelie penguins. For the testing dataset, we misclassified 1 Adelie as a Gentoo penguin, and 1 Chinstrap as a Adelie penguin

6.3.2 5.3.2 Possible Reasons

[ ]:
```
#inspect the wrongly specified training data
false_index_train = y_train_1d != yhat_train
mistakes_train = X_train_selected_norm[false_index_train]

print("true:"+ str(y_train_1d[false_index_train]))
print("pred:"+ str(yhat_train[false_index_train]))
mistakes_train
```

```
true:[1 1 1 0 1 1]
pred:[0 0 0 1 0 0]
```

[ ]:

	Culmen Length (mm)	Flipper Length (mm)	Island
216	-0.140977	0.025048	1
184	-0.330690	-1.051068	1
182	-0.634232	-1.051068	1
43	-0.027149	-0.405399	1
174	-0.197891	-1.051068	1
172	-0.349662	-1.481515	1

[ ]:
```python
#inspect the wrongly specified testing data
false_index_test = y_test_1d != yhat_test
mistakes_test = X_test_selected_norm[false_index_test]

print("true:"+ str(y_test_1d[false_index_test]))
print("pred:"+ str(yhat_test[false_index_test]))
mistakes_test
```

```
true:[1 0]
pred:[0 2]
```

[ ]:

	Culmen Length (mm)	Flipper Length (mm)	Island
206	-0.157383	-0.885696	1
101	-0.414595	0.257503	0

[ ]:
```python
#inspect Adelie data in the training dataset
adelie = y_train_1d==0
X_train_selected_norm[adelie].iloc[:10] #check 10 rows
```

[ ]:

	Culmen Length (mm)	Flipper Length (mm)	Island
33	-0.634232	-1.266292	1
106	-1.070574	-0.190176	0
34	-1.487944	-0.477140	1
97	-0.748060	-0.405399	1
85	-0.558347	-0.548881	1
45	-0.880860	-0.835845	1
134	-1.165430	-1.051068	1
108	-1.165430	-1.481515	0
90	-1.620743	0.025048	1
46	-0.596290	-1.409774	1

[ ]:
```python
#inspect Chinstrap data in the training dataset
chinstrap = y_train_1d==1
X_train_selected_norm[chinstrap].iloc[:10] #check 10 rows
```

[ ]:

	Culmen Length (mm)	Flipper Length (mm)	Island
159	1.338789	-0.333658	1
166	0.314336	-0.835845	1
156	1.604389	-0.333658	1
215	2.192501	0.383753	1
199	0.902448	0.742458	1
158	0.352279	-1.696738	1
153	1.092162	-0.405399	1
213	0.485078	-0.907586	1
219	1.130104	-0.261917	1
198	1.111133	-0.835845	1

1. For the misclassification between Adelie and Chinstrap: The problem is probably at the

training stage. The support vector machine model handles outliers, thus in the training stage, it might have decided that the wrongly specified rows are outliers according to the threshhold it finds. In the training dataset, the normalized Culmen Length feature for Adelie penguins are typically negative, while for Chinstrap penguins are typically positive. As the wrongly labeled Chinstrap penguins actually have negative normalized Culmen Length feature (row 216, 184, 182, 174, 172 in train and 206 in test), the weight the model put on it might have affected the prediction.

[ ]:
```
negative_cul = np.asarray(X_train_selected_norm['Culmen Length (mm)'])<0
X_train_selected_norm[np.logical_and(negative_cul, chinstrap)]
```

[ ]:

	Culmen Length (mm)	Flipper Length (mm)	Island
216	-0.140977	0.025048	1
184	-0.330690	-1.051068	1
182	-0.634232	-1.051068	1
174	-0.197891	-1.051068	1
172	-0.349662	-1.481515	1

The above code shows that, the only cases when Chinstrap penguins have negative normalized culment length are the cases that are wrongly specified. This is evidence that the SVM model thinks these cases are outliers

2. For the misclassificaiton between Adelie and Gentoo:

[ ]:
```
#inspect Gentoo data in the training dataset
gentoo = y_train_1d==2
X_train_selected_norm[gentoo].iloc[:10] #check 10 rows
```

[ ]:

	Culmen Length (mm)	Flipper Length (mm)	Island
322	0.560964	0.957681	0
230	-0.634232	0.885940	0
306	-0.159948	1.172905	0
334	0.371250	1.101164	0
221	1.092162	2.033797	0
291	0.409193	1.388128	0
235	0.959362	1.101164	0
228	-0.178920	0.527235	0
312	0.238450	0.742458	0
335	2.059701	2.033797	0

We can tell that the normalized Flipper Length of Gentoo penguins are generally far above 0, and the normalized Culmen Length are generally positive. It might have misclassified row 101 because its normalized Flipper Length is relatively close to 0 (0.25), and its normalized Flipper Length is negative (-0.41).

[ ]:
```
small_flip = np.asarray(X_train_selected_norm['Flipper Length (mm)'])<0.3
X_train_selected_norm[np.logical_and(small_flip, gentoo)]
```

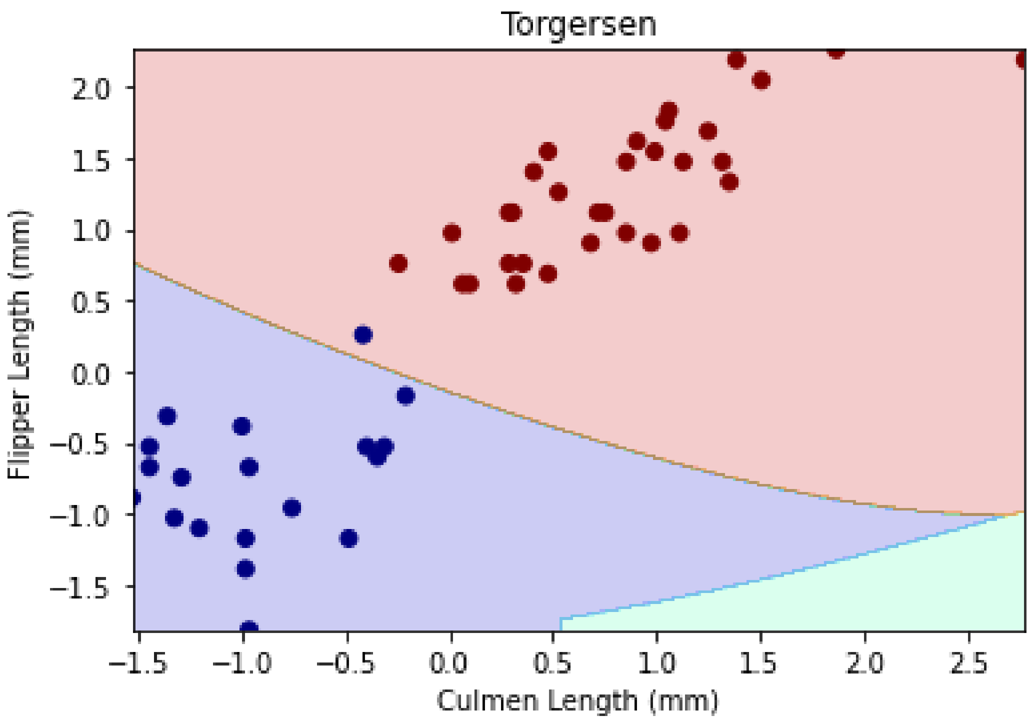
```
[ ]:      Culmen Length (mm)  Flipper Length (mm)  Island
 318             0.78862             0.096789       0
```

The above code shows that, there is only 1 case in the training set that Gentoo has Flipper Length less than 0.3, and its Culmen Length is positive. Thus the misclassified test data might have been an outlier.

6.4 5.4 Visualization

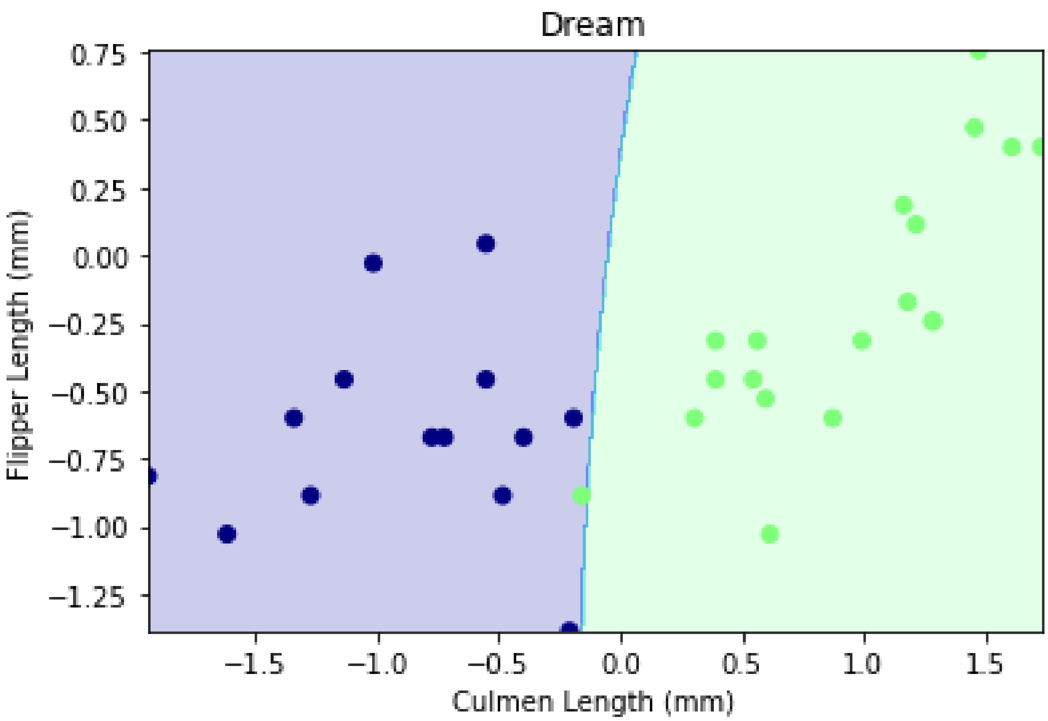
```
[ ]: new_plot_regions(clf_svm, X_test_selected_norm, y_test_clean_cat_to_num, 0)
```

```
There are  51 cases in the island 0
```



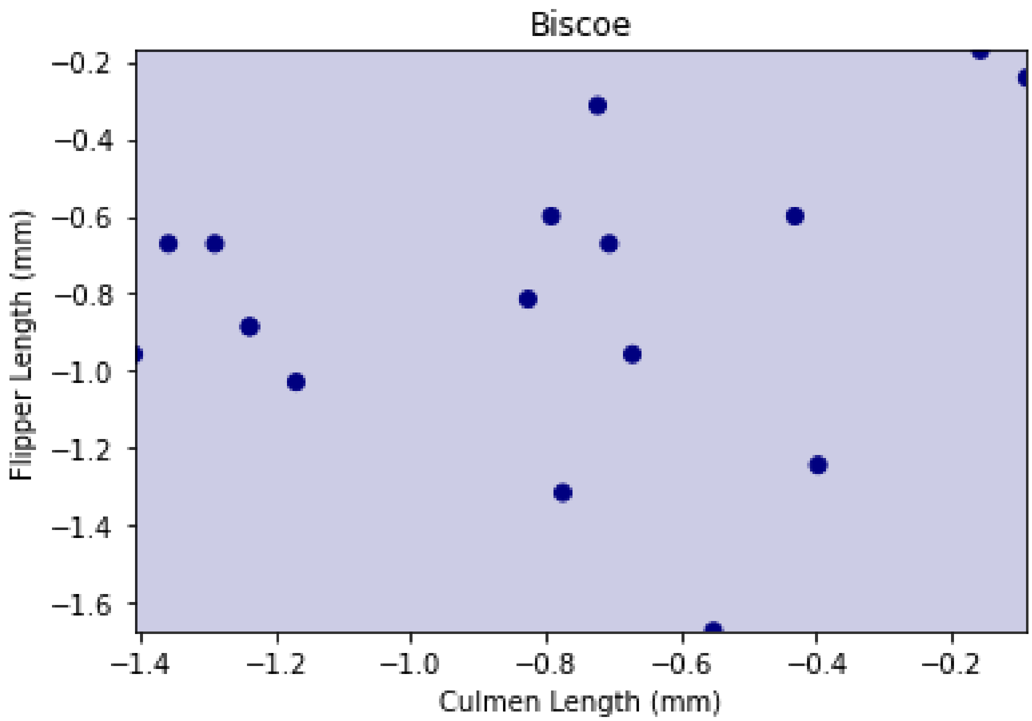

```
[ ]: new_plot_regions(clf_svm, X_test_selected_norm, y_test_clean_cat_to_num, 1)
```

```
There are  33 cases in the island 1
```



[ ]:
```
new_plot_regions(clf_svm, X_test_selected_norm, y_test_clean_cat_to_num, 2)
```

```
There are  17 cases in the island 2
```


Biscoe
Flipper Length (mm)
Culmen Length (mm)
−0.2
−0.4
−0.6
−0.8
−1.0
−1.2
−1.4
−1.6
−1.4
−1.2
−1.0
−0.8
−0.6
−0.4
−0.2

```
[ ]:
```