AUTHOR
Jamie Hackney

PUBLISHED
February 16, 2024

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from matplotlib.patches import Patch
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
from sklearn.feature_selection import SelectKBest, chi2, f_classif, mutual_info_classif
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix

train_url = "https://raw.githubusercontent.com/PhilChodrow/ml-notes/main/data/palmer-pen
df_train = pd.read_csv(train_url)

le = LabelEncoder()
le.fit(df_train["Species"])

def prepare_data(df):
  df = df.drop(["studyName", "Sample Number", "Individual ID", "Date Egg", "Comments", "
  df = df[df["Sex"] != "."]
  df = df.dropna()
  y = le.transform(df["Species"])
  df = df.drop(["Species"], axis = 1)
  df = pd.get_dummies(df)
  return df, y

X_train, y_train = prepare_data(df_train)
X_train.head()
```

	Unnamed: 0	Culmen Length (mm)	Culmen Depth (mm)	Flipper Length (mm)	Body Mass (g)	Delta 15 N (o/oo)	Delta 13 C (o/oo)	Island_Biscoe	Island_Dream	Island_Torgersen	Stage 1 Egg
0	182	40.9	16.6	187.0	3200.0	9.08458	-24.54903	False	True	False	True
1	192	49.0	19.5	210.0	3950.0	9.53262	-24.66867	False	True	False	True
2	223	50.0	15.2	218.0	5700.0	8.25540	-25.40075	True	False	False	True
3	234	45.8	14.6	210.0	4200.0	7.79958	-25.62618	True	False	False	True
4	185	51.0	18.8	203.0	4100.0	9.23196	-24.17282	False	True	False	True

Feature Selection

```
# FIND THE BEST OVERALL 3 FEATURES
X_train_k_best_features_m = SelectKBest(mutual_info_classif, k=3).fit(X_train, y_train)
X_train_k_best_features_m.get_feature_names_out()
```

```
array(['Unnamed: 0', 'Flipper Length (mm)', 'Delta 13 C (o/oo)'],
      dtype=object)
```

```
# FIND THE BEST 2 NUMERICAL FEATURES
numerical_cols = ['Unnamed: 0', 'Culmen Length (mm)', 'Culmen Depth (mm)', 'Flipper Leng
X_train_k_best_features_f = SelectKBest(f_classif, k=2).fit(X_train[numerical_cols], y_t
X_train_k_best_features_f.get_feature_names_out()
```

```
array(['Unnamed: 0', 'Flipper Length (mm)'], dtype=object)
```

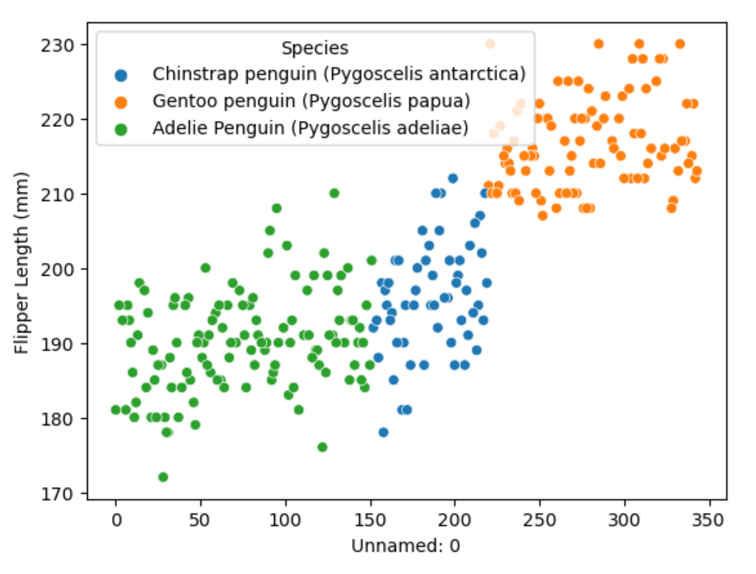
```
# FIND THE BEST CATEGORICAL FEATURE
categorical_cols = ['Island_Biscoe', 'Island_Dream', 'Island_Torgersen', 'Stage_Adult, 1
X_train_k_best_features_c = SelectKBest(chi2, k=3).fit(X_train[categorical_cols], y_trai
X_train_k_best_features_c.get_feature_names_out()
```

```
array(['Island_Biscoe', 'Island_Dream', 'Island_Torgersen'], dtype=object)
```

My first task was to find the 3 best features to use for classification, which I opted to do with scikit-learns SelectKBest function. This takes in a scoring function, as well as the training features and target variable. I first ran this with k=3 and mutual information scoring to see which 3 features, among all numerical and categorical features, have the lowest dependency. This is desirable because, since we can only use 3 features, we want each feature to give us new information about the target variable. If the three features are highly dependent, then they are redundant in their information. Using mutual information scoring, I saw that the 3 features with the lowest mutual dependency were 'Unnamed: 0', 'Flipper Length (mm)', and 'Delta 13 C (o/oo)'. I was a little confused as to what the 'Unnamed: 0' column was, but I continued on for now (more on that later). These 3 features were all numerical, and since I knew I needed a combination of 2 numerical and 1 categorical feature, I split my feature selection between the numerical features and the categorical features. When scoring the numerical features, I opted to use analysis of variance F-value to find the degree of linear dependency. I again saw that 'Unnamed: 0' and 'Flipper Length (mm)' were the least linearly dependent features. When scoring the categorical features, I used chi-squared to measure the dependency between each feature and the target variable so see which was most relevant for classification. I found that the 'Island' feature, which had been split into 'Island_Biscoe', 'Island_Dream', and 'Island_Torgersen' during data processing were the most informative.

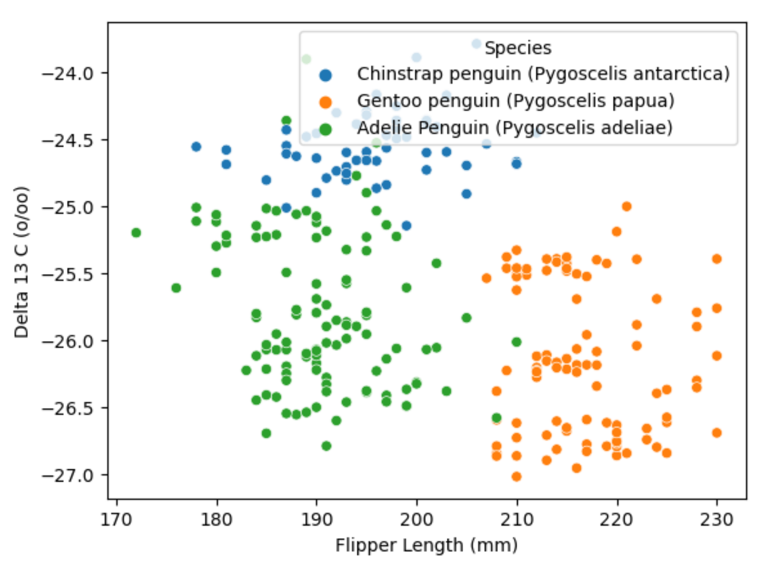
Visualizations

```
flipper_unnamed = sns.scatterplot(df_train, x = "Unnamed: 0", y = "Flipper Length (mm)",
```



After selecting my variables I went to visualize some of them. I first started with the relationship between 'Flipper Length (mm)' and 'Unnamed: 0', and after looking at the graph I realized that 'Unnamed: 0' was probably each penguin's ID from the data collection which got converted into a feature. Moreover, the ID's seemed to in order of species; in other words, Adelie Penguins are ID's 0 - 150ish, Gentoos are 150ish - 215ish, and Chinstraps are 215ish - 350. This meant that the data are perfectly linearly seperable by ID alone, and therefore Support Vector Machines, Decision Trees, and even a Perceptron should be able to perfectly classify the training data by ID alone, and possibly the testing data as well if the testing data ID's match this distribution.

```
flipper_delta = sns.scatterplot(df_train, x = "Flipper Length (mm)", y = "Delta 13 C (o/
```



After realizing the 'Unnamed: 0' column was probably just an ID column, I wanted to visualize the Flipper Lenth with the next best numerical feature, which was 'Delta 13 C (o/oo)'. The graph above showed me that while the data was not perfectly linearly seperable with respect to these 2 features, there were still pretty distinct trends in the data. Specifically, the rectangular shape of the regions made me think that a Decision Tree would preform well.

```
frequency_tab = pd.crosstab(df_train['Species'], df_train['Island'])
frequency_tab
```

Island	Biscoe	Dream	Torgersen
Species			
Adelie Penguin (Pygoscelis adeliae)	33	45	42
Chinstrap penguin (Pygoscelis antarctica)	0	57	0
Gentoo penguin (Pygoscelis papua)	98	0	0

Finally, to get an understanding of the relationship between the Island and the species, I decided to make a frequency table showing the counts of each specie on each Island. The Pandas crosstab function makes this really easy to do. I saw that while Adelie penguins are found on all three islands, Chinstraps are found only on the Dream Island and Gentoos are found only on the Biscoe Island.

Building the Models

```
predictor_cols = ['Unnamed: 0', 'Flipper Length (mm)', 'Island_Dream', 'Island_Biscoe',
```

```
LR = LogisticRegression(max_iter=10000)
LR.fit(X_train[predictor_cols], y_train)
LR.score(X_train[predictor_cols], y_train)
```

```
1.0
```

```
scores = []
gammas = 10**np.arange(float(-5), float(5))

for gamma in gammas:
    SVM = SVC(gamma=gamma)
    SVM.fit(X_train[predictor_cols], y_train)

    cv_scores_SMV = cross_val_score(SVM, X_train[predictor_cols], y_train, cv=5)
    scores.append((gamma, cv_scores_SMV.mean()))

print(scores)

SVM = SVC(gamma=0.001)
SVM.fit(X_train[predictor_cols], y_train)
SVM.score(X_train[predictor_cols], y_train)
```

```
[(1e-05, 0.9765460030165911), (0.0001, 0.9804675716440421), (0.001, 0.988310708898944), (
```

```
0.9921875
```

```
scores = []
depths = list(range(1, 100, 1))

for depth in depths:
    DT = DecisionTreeClassifier(max_depth=depth)
    DT.fit(X_train[predictor_cols], y_train)

    cv_scores_DT = cross_val_score(DT, X_train[predictor_cols], y_train, cv=5)
    scores.append((depth, cv_scores_DT.mean()))

print(scores)

DT = DecisionTreeClassifier(max_depth=2)
DT.fit(X_train[predictor_cols], y_train)
DT.score(X_train[predictor_cols], y_train)
```

```
[(1, 0.7812971342383107), (2, 0.9922322775263952), (3, 0.9922322775263952), (4, 0.9922322
```

```
1.0
```

```
RF = RandomForestClassifier()
RF.fit(X_train[predictor_cols], y_train)
RF.score(X_train[predictor_cols], y_train)
```

```
1.0
```

Using 'Unnamed: 0', 'Flipper Length (mm)', and 'Island' (which was split into 3 separate features during the data processing), I fit a Logistic Regression, Support Vector Machine, Decision Tree, and Random Forest. I saw that the Logistic Regression algorithm was not converging with the default numer of iterations (100), so I increased that until it converged at 100% accuracy. I fit the Support Vector Machine with a range of gamma values from 0.00001 to 10000 and evaluated each with 5-fold cross validation. I achieved the highest cross validation score with a gamma of 0.001, so I used that and achieved an accuracy score of 0.9921875 on the training data. I did the same with the Decision Tree except with the max_depth parameter. With that the cross validation accuracy remained the same after max_depths greater than 2, so thats what I used and achieved an accuract score of 1.0 on the training data. Finally, the Random Forest with default parameters also achieved an accuracy score of 1.0 on the training data.

Evaluating Model on Testing Data

```
test_url = "https://raw.githubusercontent.com/PhilChodrow/ml-notes/main/data/palmer-peng
test = pd.read_csv(test_url)

X_test, y_test = prepare_data(test)
DT.score(X_test[predictor_cols], y_test)
```

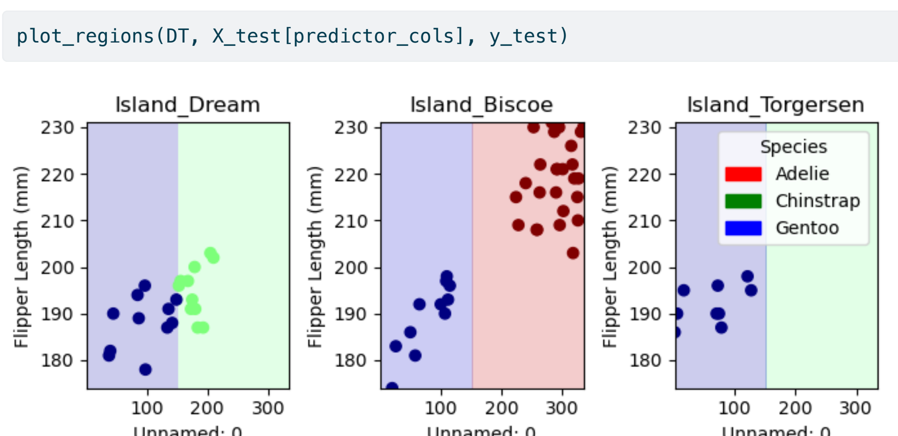
```
1.0
```

I used a Decision Tree on the testing data and achieved an accuracy score of 1.0.

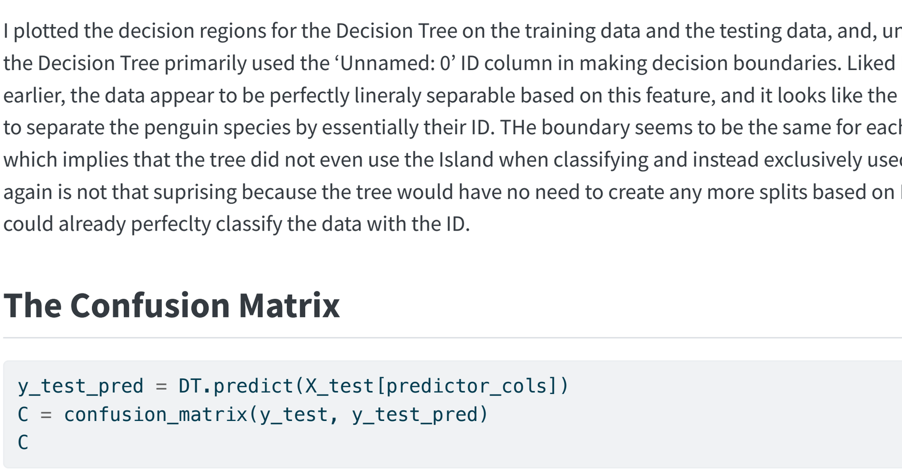
Plot Decision Region

```
def plot_regions(model, X, y):

    x0 = X[X.columns[0]]
    x1 = X[X.columns[1]]
    qual_features = X.columns[2:]

    fig, axarr = plt.subplots(1, len(qual_features), figsize = (7, 3))

    # 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)

    XX = xx.ravel()
    YY = yy.ravel()

    for i in range(len(qual_features)):
      XY = pd.DataFrame({
          X.columns[0] : XX,
          X.columns[1] : YY
      })

      for j in qual_features:
        XY[j] = 0

      XY[qual_features[i]] = 1

      p = model.predict(XY)
      p = p.reshape(xx.shape)


      # use contour plot to visualize the predictions
      axarr[i].contourf(xx, yy, p, cmap = "jet", alpha = 0.2, vmin = 0, vmax = 2)

      ix = X[qual_features[i]] == 1
      # plot the data
      axarr[i].scatter(x0[ix], x1[ix], c = y[ix], cmap = "jet", vmin = 0, vmax = 2)

      axarr[i].set(xlabel = X.columns[0],
            ylabel  = X.columns[1],
            title = qual_features[i])

      patches = []
      for color, spec in zip(["red", "green", "blue"], ["Adelie", "Chinstrap", "Gentoo"]
        patches.append(Patch(color = color, label = spec))

      plt.legend(title = "Species", handles = patches, loc = "best")

      plt.tight_layout()
```

```
plot_regions(DT, X_train[predictor_cols], y_train)
```




```
plot_regions(DT, X_test[predictor_cols], y_test)
```



I plotted the decision regions for the Decision Tree on the training data and the testing data, and, unsuprisingly, the Decision Tree primarily used the 'Unnamed: 0' ID column in making decision boundaries. Liked I talked about earlier, the data appear to be perfectly lineraly separable based on this feature, and it looks like the tree learned to separate the penguin species by essentially their ID. THe boundary seems to be the same for each island, which implies that the tree did not even use the Island when classifying and instead exclusively used the ID. This again is not that suprising because the tree would have no need to create any more splits based on Island if it could already perfeclty classify the data with the ID.

The Confusion Matrix

```
y_test_pred = DT.predict(X_test[predictor_cols])
C = confusion_matrix(y_test, y_test_pred)
C
```

```
array([[31,  0,  0],
       [ 0, 11,  0],
       [ 0,  0, 26]])
```

The confusion matrix shows again the 100% accuracy we were seeing.