AUTHOR
Jamie Hackney

PUBLISHED
February 16, 2024

Abstract

The goal of this project is to explore the capabilities of various machine learning algorithms for predicting penguin species in the Palmer penguins data set. To do so, the work was split into three main sections: an exploration, modelling, and an evaluation. The exploration section is meant to develop an understanding of the dataset through various visualizations as well as summary tables of the dataset. The idea is that a better understanding of the dataset would lead to better feature selection for the modelling section. Specifically, SciKitLearn's feature selection tools were employed to find the features that were the most helpful. Once the features were selected, several models were built, including a Logistic Regression, Decision Tree, Support Vector Machine, and Random Forest. 5-Fold cross validation was used to both evaluate the preformance of the models and find the best parameters. Finally, the models were evaluated on the test set, with a Random Forest acheiving an accuracy of 100%.

```
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()
```

	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_Adult, 1 Egg Stage	Clu Cor
0	40.9	16.6	187.0	3200.0	9.08458	-24.54903	False	True	False	True	Fals
1	49.0	19.5	210.0	3950.0	9.53262	-24.66867	False	True	False	True	Fals
2	50.0	15.2	218.0	5700.0	8.25540	-25.40075	True	False	False	True	Fals
3	45.8	14.6	210.0	4200.0	7.79958	-25.62618	True	False	False	True	Fals
4	51.0	18.8	203.0	4100.0	9.23196	-24.17282	False	True	False	True	Fals

Feature Selection

```
# FIND THE BEST OVERALL 3 FEATURES
X_train_k_best_features_m = SelectKBest(mutual_info_classif, k=4).fit(X_train, y_train)
X_train_k_best_features_m.get_feature_names_out()
```

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

```
# FIND THE BEST 2 NUMERICAL FEATURES
numerical_cols = ['Culmen Length (mm)', 'Culmen Depth (mm)', 'Flipper Length (mm)', 'Bod
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(['Culmen Length (mm)', 'Flipper Length (mm)'], dtype=object)
```

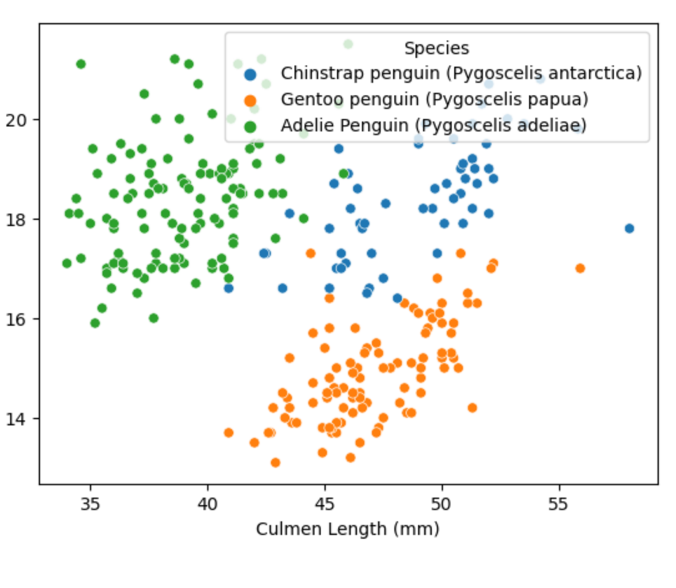
```
# FIND THE BEST CATEGORICAL FEATURE
categorical_cols = ['Island_Biscoe', 'Island_Dream', 'Island_Torgersen', 'Stage_Adult, 1
X_train_k_best_features_c = SelectKBest(chi2, k=3).fit(X_train[categorical_cols], y_trai
X_train_k_best_features_c.get_feature_names_out()
```

```
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 'Culmen Length (mm)', 'Culmen Depth (mm)', and 'Flipper Length (mm)'. 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 'Culmen Length (mm)' 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

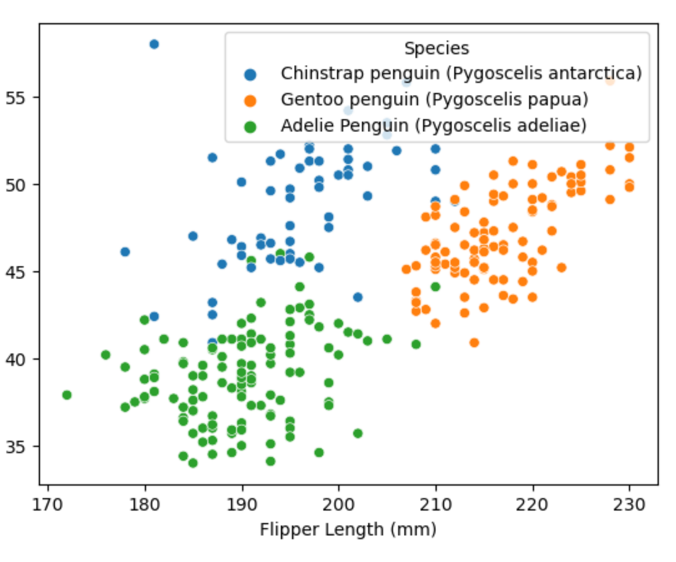
```
flipper_unnamed = sns.scatterplot(df_train, x = "Culmen Length (mm)", y = "Culmen Depth
```



After selecting my variables I went to visualize some of them. I first started with the relationship between 'Culmen Length (mm)' and 'Flipper Depth (mm)' and species. From looking at the graph I saw that the three species have pretty unique relatinships between the culmen length and depth.

'Unnamed: 0' is no longer in the dataset, but when it was… I think it represented 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 = "Culmen Length
```



I then wanted to visualize the relationship between 'Flipper Length (mm)' and 'Culment Length (mm)' and species. This plot shows that again their is a pretty unique relationship for each species between the two measurements.

```
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 = ['Culmen Length (mm)', 'Culmen Depth (mm)', 'Island_Dream', 'Island_Bis
```

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

```
0.99609375
```

```
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, 0.42187028657616893), (0.0001, 0.7266214177978884), (0.001, 0.9217948717948717),

0.94140625
```

```
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, 0.7147058823529412), (2, 0.9294871794871794), (3, 0.9687028657616894), (4, 0.9647812

0.95703125
```

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

```
1.0
```

Using 'Culmen Length (mm)', 'Culmen Depth (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 99.6% 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 0.1, so I used that and achieved an accuracy of 94.1% 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 5, so thats what I used and achieved an accuracy of 95.7% on the training data. Finally, the Random Forest with default parameters also achieved an accuracy of 100% 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)
RF.score(X_test[predictor_cols], y_test)
```

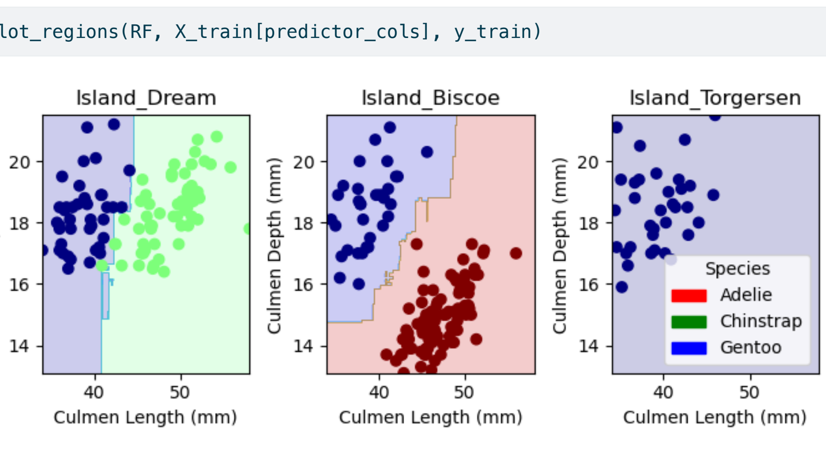
```
1.0
```

I used a Random Forest on the testing data and achieved an accuracy of 100%

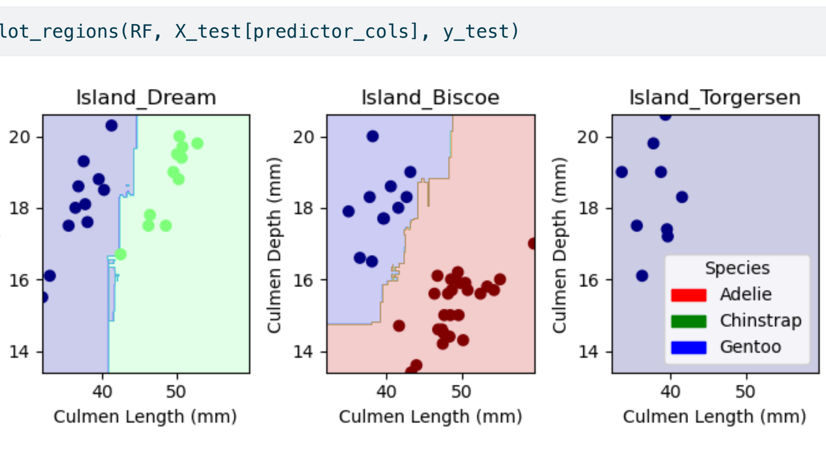
Plot Decision Region

```
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(RF, X_train[predictor_cols], y_train)
```




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



I plotted the decision regions for the Random Forest on the training data and the testing data, and saw that for the Dream and Biscoe Islands, it was able to find a decision boundary between the species. Since only Gentoo penguins are found on the Torgersen Island, the Randome Forest did not have to find a decision boundary. For the other islands, the decision boundary it found was pretty choppy, which makes sense for how Random Forest's work. It does, however, raise questions of overfitting on this dataset.

The Confusion Matrix

```
y_test_pred = RF.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.

Discussion

In this blog post I explored the capabilities of Logistic Regression, Support Vector Machine, Decision Tree, and Random Forest models for classifying pengiun species in the Palmer penguin data set. I used SciKitLearn's feature selection tools to find the best 3 features, which were the Culmen Depth, Culmen Length, and Island. While all models preformed well, scoring above 90% accuracy on the trainin data, only a Random Forest was able to achieve 100% accuracy.