100 XP

Exercise - Train and evaluate multiclass classification models

10 minutes

Sandbox activated! Time remaining: 54 min

You have used 3 of 10 sandboxes for today. More sandboxes will be available tomorrow.

Runtime File Edit View Comments

Run all azureml_py38

We would love to hear your feedback on the notebooks experience! Please take a few minutes to complete our survey.

Multiclass Classification

In the last notebook, we looked at binary classification. This works well when the data observations belong to one of two classes or categories, such as "True" or "False". When the data is categorized into more than two classes, you'll need a multiclass classification algorithm.

Multiclass classification can be a combination of multiple binary classifiers. There are two ways to approach this problem:

- **One vs Rest (OVR)**, in which a classifier is created for each possible class value, with a positive outcome for cases where the prediction is *this* class, and negative predictions for

```
import pandas as pd

# load the training dataset
!wget https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to-machine-le
penguins = pd.read_csv('penguins.csv')

# Display a random sample of 10 observations
sample = penguins.sample(10)
sample
```

[21] ✓ <1 sec

```
--2024-02-09 23:06:31--  https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to-machine-learning/main/Data/ml-basics/penguins.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.111.133, 185.199.108.133, 185.199.109.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.111.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 7086 (6.9K) [text/plain]
Saving to: ‘penguins.csv.1’

penguins.csv.1      100%[===================>]   6.92K  --.-KB/s    in 0s

2024-02-09 23:06:31 (66.9 MB/s) - ‘penguins.csv.1’ saved [7086/7086]
```

	CulmenLength	CulmenDepth	FlipperLength	BodyMass	Species
178	44.5	14.3	216.0	4100.0	1
259	53.4	15.8	219.0	5500.0	1
215	54.3	15.7	231.0	5650.0	1
248	49.4	15.8	216.0	4925.0	1
97	40.3	18.5	196.0	4350.0	0
49	42.3	21.2	191.0	4150.0	0
104	37.9	18.6	193.0	2925.0	0
131	43.1	19.2	197.0	3500.0	0
189	44.4	17.3	219.0	5250.0	1
227	48.6	16.0	230.0	5800.0	1

The dataset contains the following columns:

- **CulmenLength**: The length in mm of the penguin's culmen (bill).
- **CulmenDepth**: The depth in mm of the penguin's culmen.
- **FlipperLength**: The length in mm of the penguin's flipper.
- **BodyMass**: The body mass of the penguin in grams.
- **Species**: An integer value that represents the species of the penguin.

The **Species** column is the label we want to train a model to predict. The dataset includes three possible species, which are encoded as 0, 1, and 2. The actual species names are

```
penguin_classes = ['Adelie', 'Gentoo', 'Chinstrap']
print(sample.columns[0:5].values, 'SpeciesName')
for index, row in penguins.sample(10).iterrows():
    print('[', row[0], row[1], row[2], row[3], int(row[4]),']', penguin_classes[int(row[
```

[22] ✓ <1 sec

```
['CulmenLength' 'CulmenDepth' 'FlipperLength' 'BodyMass' 'Species'] SpeciesName
[ 42.8 14.2 209.0 4700.0 1 ] Gentoo
[ 38.8 20.0 190.0 3950.0 0 ] Adelie
[ 47.6 18.3 195.0 3850.0 2 ] Chinstrap
[ 38.1 17.0 181.0 3175.0 0 ] Adelie
[ 36.2 17.3 187.0 3300.0 0 ] Adelie
[ 38.9 18.8 190.0 3600.0 0 ] Adelie
[ 46.6 17.8 193.0 3800.0 2 ] Chinstrap
[ 48.2 15.6 221.0 5100.0 1 ] Gentoo
[ 46.9 14.6 222.0 4875.0 1 ] Gentoo
[ 53.4 15.8 219.0 5500.0 1 ] Gentoo
```

Now that we know what the features and labels in the data represent, let's explore the dataset. First, let's see if there are any missing (*null*) values.

```
# Count the number of null values for each column
penguins.isnull().sum()
```

[23] ✓ <1 sec

```
CulmenLength     2
CulmenDepth      2
FlipperLength    2
BodyMass         2
Species          0
dtype: int64
```

It looks like there are some missing feature values, but no missing labels. Let's dig a little deeper and see the rows that contain nulls.

```
# Show rows containing nulls
penguins[penguins.isnull().any(axis=1)]
```

```
penguins[penguins.isnull().any(axis=1)]
```

[24] ✓ <1 sec

	CulmenLength	CulmenDepth	FlipperLength	BodyMass	Species
3	NaN	NaN	NaN	NaN	0
271	NaN	NaN	NaN	NaN	1

There are two rows that contain no feature values at all (*NaN* means "not a number"), so these won't be useful in training a model. Let's discard them from the dataset.

```
# Drop rows containing NaN values
penguins=penguins.dropna()
#Confirm there are now no nulls
penguins.isnull().sum()
```

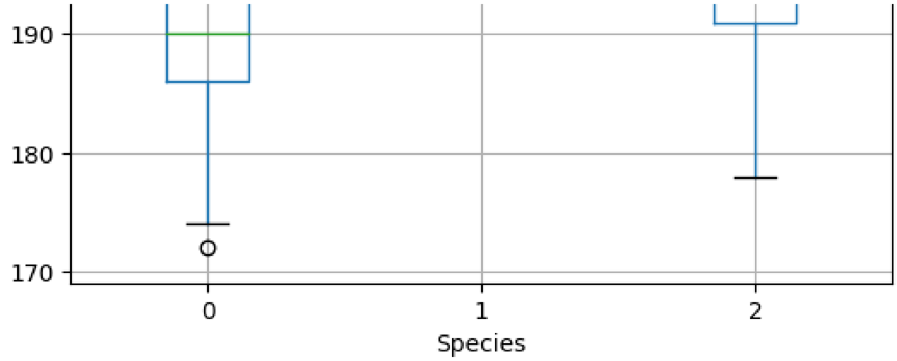
[25] ✓ <1 sec

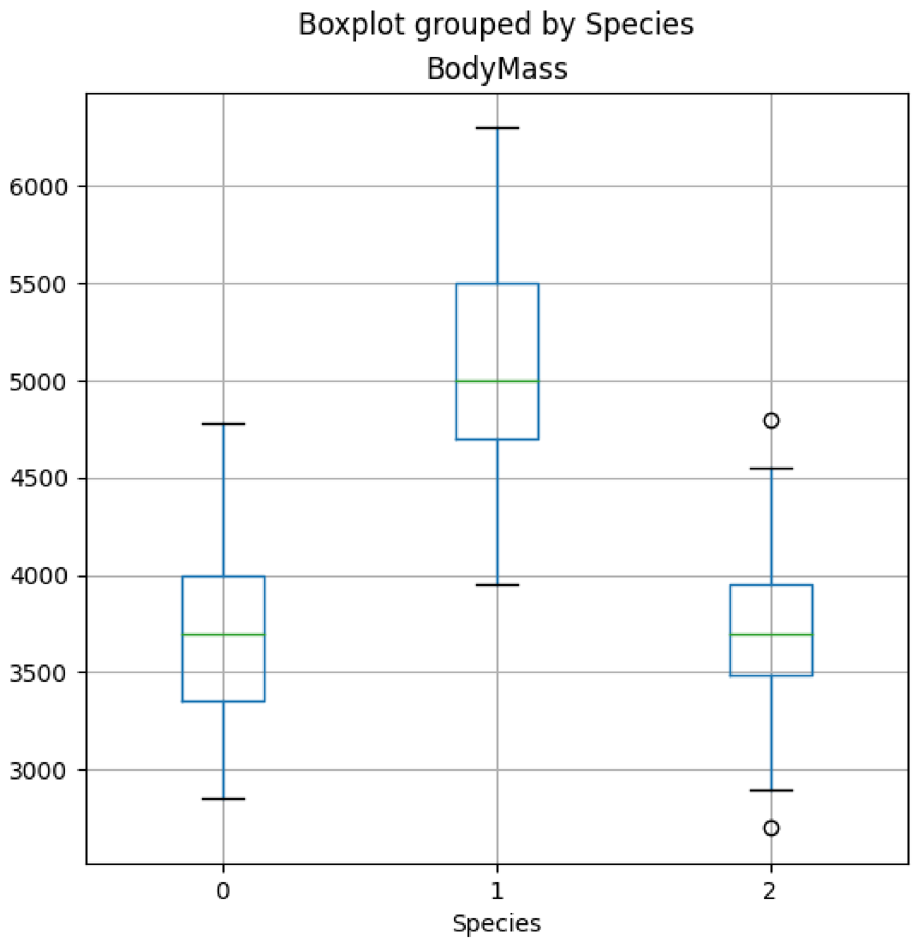
```
CulmenLength     0
CulmenDepth      0
FlipperLength    0
BodyMass         0
Species          0
dtype: int64
```

Now that we've dealt with the missing values, let's explore how the features relate to the label by creating some box charts.

```
from matplotlib import pyplot as plt
%matplotlib inline

penguin_features = ['CulmenLength','CulmenDepth','FlipperLength','BodyMass']
penguin_label = 'Species'
for col in penguin_features:
    penguins.boxplot(column=col, by=penguin_label, figsize=(6,6))
    plt.title(col)
plt.show()
```

[26] ✓ 1 sec





From the box plots, it looks like species 0 and 2 (Adelie and Chinstrap) have similar data profiles for culmen depth, flipper length, and body mass, but Chinstraps tend to have longer culmens. Species 1 (Gentoo) tends to have fairly clearly differentiated features from the others, which should help us train a good classification model.

Prepare the data

Just as for binary classification, before training the model, we need to separate the features and label, and then split the data into subsets for training and validation. We'll also apply a

```
from sklearn.model_selection import train_test_split

# Separate features and labels
penguins_X, penguins_y = penguins[penguin_features].values, penguins[penguin_label].valu
```

```
# Split data 70%-30% into training set and test set
x_penguin_train, x_penguin_test, y_penguin_train, y_penguin_test = train_test_split(peng
                                                                                    test
                                                                                    rand
                                                                                    stra

print ('Training Set: %d, Test Set: %d \n' % (x_penguin_train.shape[0], x_penguin_test.s
```

[27] ✓ <1 sec

```
Training Set: 239, Test Set: 103
```

Train and evaluate a multiclass classifier

Now that we have a set of training features and corresponding training labels, we can fit a multiclass classification algorithm to the data to create a model. Most Scikit-Learn classification algorithms inherently support multiclass classification. We'll try a logistic

```
from sklearn.linear_model import LogisticRegression

# Set regularization rate
reg = 0.1

# train a logistic regression model on the training set
multi_model = LogisticRegression(C=1/reg, solver='lbfgs', multi_class='auto', max_iter=1
print (multi_model)
```

[28] ✓ <1 sec

```
LogisticRegression(C=10.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=10000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Now we can use the trained model to predict the labels for the test features, and compare the predicted labels to the actual labels.

```
penguin_predictions = multi_model.predict(x_penguin_test)
print('Predicted labels: ', penguin_predictions[:15])
print('Actual labels   : ', y_penguin_test[:15])
```

[29] ✓ <1 sec

```
Predicted labels:  [0 1 0 2 2 1 1 1 0 2 2 1 2 1 2]
Actual labels   :  [0 1 2 2 2 1 1 1 0 2 2 1 2 1 2]
```

Let's look at a classification report.

```
from sklearn. metrics import classification_report

print(classification_report(y_penguin_test, penguin_predictions))
```

[30] <1 sec

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97        45
           1       1.00      1.00      1.00        37
           2       0.95      0.90      0.93        21

    accuracy                           0.97       103
   macro avg       0.97      0.96      0.96       103
weighted avg       0.97      0.97      0.97       103
```

As with binary classification, the report includes *precision* and *recall* metrics for each class. However, while with binary classification we could focus on the scores for the *positive* class; in this case, there are multiple classes so we need to look at an overall metric (either the macro or weighted average) to get a sense of how well the model performs across all three classes.

You can get the overall metrics separately from the report using the Scikit-Learn metrics score classes, but with multiclass results you must specify which average metric to use for precision and recall.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score

print("Overall Accuracy:",accuracy_score(y_penguin_test, penguin_predictions))
print("Overall Precision:",precision_score(y_penguin_test, penguin_predictions, average=
print("Overall Recall:",recall_score(y_penguin_test, penguin_predictions, average='macro
```

[31] <1 sec

```
Overall Accuracy: 0.970873786407767
Overall Precision: 0.9688405797101449
Overall Recall: 0.9608465608465608
```

Now let's look at the confusion matrix for our model.

```
from sklearn.metrics import confusion_matrix

# Print the confusion matrix
mcm = confusion_matrix(y_penguin_test, penguin_predictions)
print(mcm)
```

[32] <1 sec

```
[[44  0  1]
 [ 0 37  0]
 [ 2  0 19]]
```

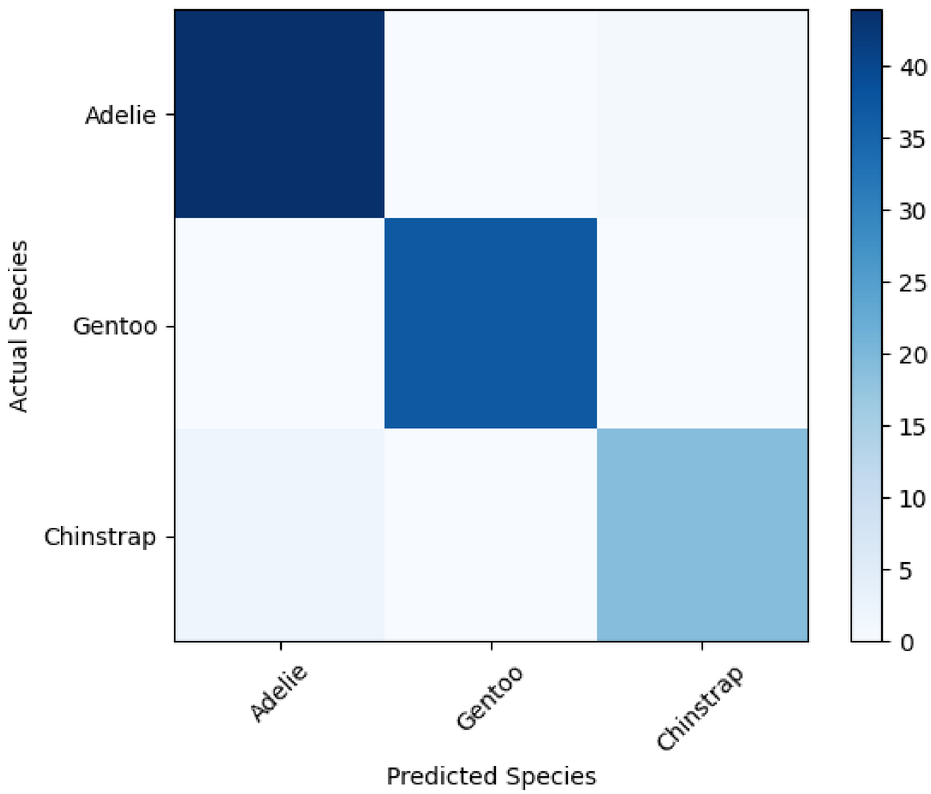
+ Code + Markdown

The confusion matrix shows the intersection of predicted and actual label values for each class, where the diagonal intersections from top-left to bottom-right indicate the number of correct predictions.

When dealing with multiple classes, it's generally more intuitive to visualize this as a heat map.

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

plt.imshow(mcm, interpolation="nearest", cmap=plt.cm.Blues)
plt.colorbar()
tick_marks = np.arange(len(penguin_classes))
plt.xticks(tick_marks, penguin_classes, rotation=45)
plt.yticks(tick_marks, penguin_classes)
plt.xlabel("Predicted Species")
plt.ylabel("Actual Species")
plt.show()
```

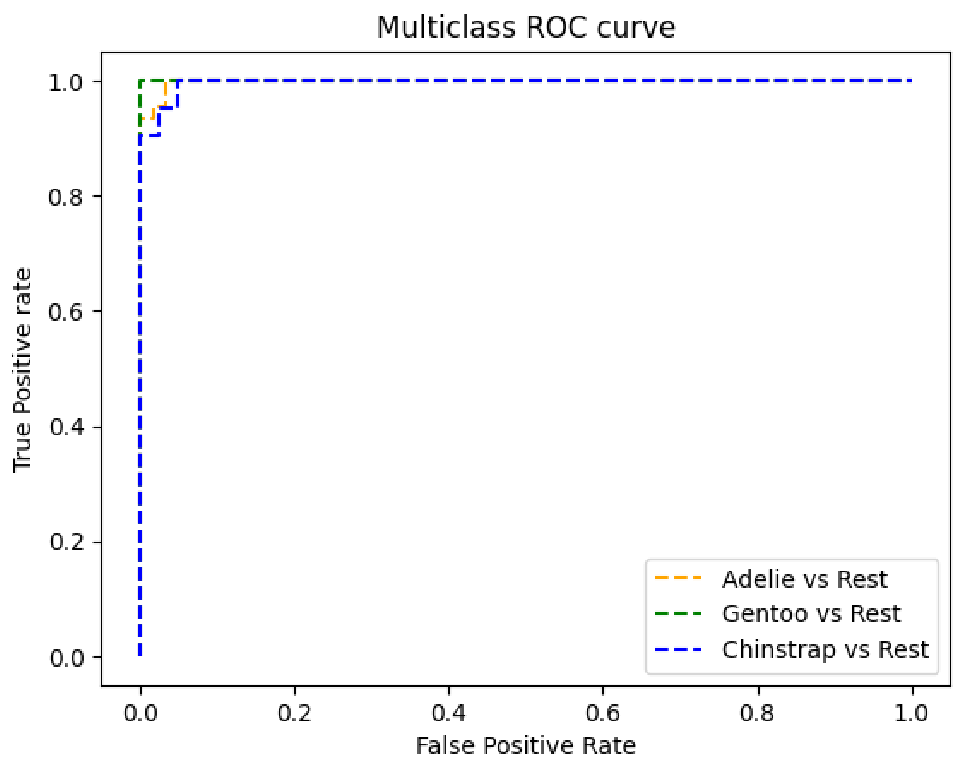
[33] ✓ <1 sec



The darker squares in the confusion matrix plot indicate high numbers of cases, and you can hopefully see a diagonal line of darker squares indicating cases where the predicted and actual label are the same.

In the case of a multiclass classification model, a single ROC curve showing true positive rate vs false positive rate is not possible. However, you can use the rates for each class in a One vs Rest (OVR) comparison to create a ROC chart for each class.

[34] ✓ <1 sec



To quantify the ROC performance, you can calculate an aggregate area under the curve score that is averaged across all of the OVR curves.

[35] ✓ <1 sec

```
Average AUC: 0.9981999902100828
```

Preprocess data in a pipeline

Again, just like with binary classification, you can use a pipeline to apply preprocessing steps to the data before fitting it to an algorithm to train a model. Let's see if we can improve the penguin predictor by scaling the numeric features in a transformation step before training. We'll also try a different algorithm, a *support vector machine*, just

[36] ✓ <1 sec

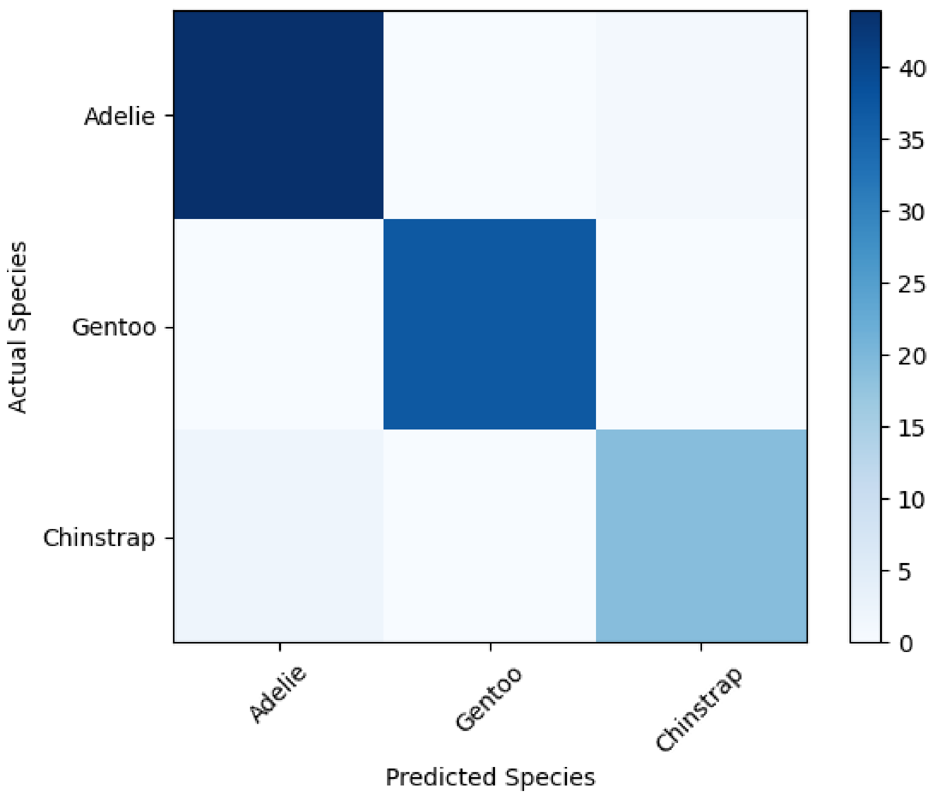
```
StandardScaler(copy=True,

with_mean=True,
```

```
                  gamma='scale', kernel='rbf', max_iter=-1, probability=True,
                  random_state=None, shrinking=True, tol=0.001,
                  verbose=False))],
         verbose=False)
```

Now let's evaluate the new model.

[37] <1 sec

```
Overall Accuracy: 0.9805825242718447
Overall Precision: 0.9767195767195768
Overall Recall: 0.9767195767195768
Average AUC: 0.9990361381446328
```



Use the model with new data observations

Save our newest trained model so we can use it again later.

[38] <1 sec

```
['./penguin_model.pkl']
```

Now let's use the model to predict the class of a new penguin observation.

[39]

<1 sec

```
New sample: [  50.4   15.3  224.  5550. ]
Predicted class is Gentoo
```

You can also submit a batch of penguin observations to the model, and get back a prediction for each one.

[40]

<1 sec

```
New samples:
[[  49.5   18.4  195.  3600. ]
 [  38.2   20.1  190.  3900. ]]
2 (Chinstrap)
0 (Adelie)
```

Summary

Classification is one of the most common forms of machine learning, and by following the basic principles we've discussed in this notebook you should be able to train and evaluate classification models with Scikit-Learn. It's worth spending some time investigating classification algorithms in more depth, and a good starting point is the

learn-notebooks-5d4f7662-84ac-416d-8605-3ecf30ac1799 Compute connected Viewing Kernel idle azureml_py38

Next unit: Knowledge check

Continue >