Balanced Dataset Classification

Source: https://machinelearningmastery.com/machine-learning-in-python-step-by-step/

You will focus more on how to code without being distracted on installation of python version and libraries. Therefor, it is advisable to use *Google Collab* or *Anaconda.*

Let's begin our journey!

Building machine learning project usually involved several well known steps which are :
1- Prepare data
2- Preprocessing data
3- Build model
4- Predict
5- Show results

1 - Prepare data

- Identify data needed
- Load data
- Understand the data

Identify data needed

In this project, our problem is to classify iris flower(Iris Setosa, Iris Versicolor, Iris Virginica) with given its features (sepal length, sepal width, petal length, petal width)
So we need a set of data about the flower (Iris dataset).

Load data

- In order to start, we need to import some libraries. This is the library that we will use in this project.

```
# Load libraries
import pandas
from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

Next, we load the dataset.

```
# Load dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = pandas.read_csv(url, names=names)
```

There are several ways to load dataset.

1 - Direct from internet

```
 url =
"https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris
.csv"
names = ['sepal-length', 'sepal-width', 'petal-length',
'petal-width', 'class']
dataset = pandas.read_csv(url, names=names)
```

2 - From local folder

```
from google.colab import files #this one for google collab only
uploaded = files.upload()
dataset = pd.read_csv(io.BytesIO(uploaded['filename.csv']),
names=names)
dataset = dataset.drop([])
OR
URL = "filename.csv"
dataset = pandas.read_csv(URL, names = names)
```

Note:Write uploaded file name into *'filename.csv'*.

Understand the data

We need to understand the data like how the data looks like in the program, how many group of data and its class distribution, and etc.

- Find out how many rows and columns of the dataset. Rows mean how many data that the data have and column is for the feature.

```
# shape output: (150 row,5 column)
print(dataset.shape)
```

```
(150, 5)
```

- Take a peek how the data looks like. You should see first 5 rows of data.

```
# head
print(dataset.head())
```

	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

- Summary for each features.

```
# descriptions
print(dataset.describe())
```

```
       sepal-length  sepal-width  petal-length  petal-width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.054000      3.758667     1.198667
std        0.828066     0.433594      1.764420     0.763161
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000
```

Count: you can see all data is filled. None of the features has null data since it has same number.

Min and max : the range value is in between 0 to 8 only.

- Find out how many data for each group.

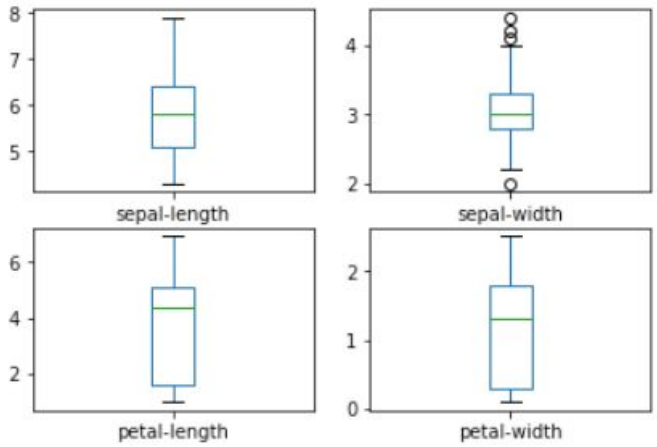
```
# class distribution
print(dataset.groupby('class').size())
```

```
class
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

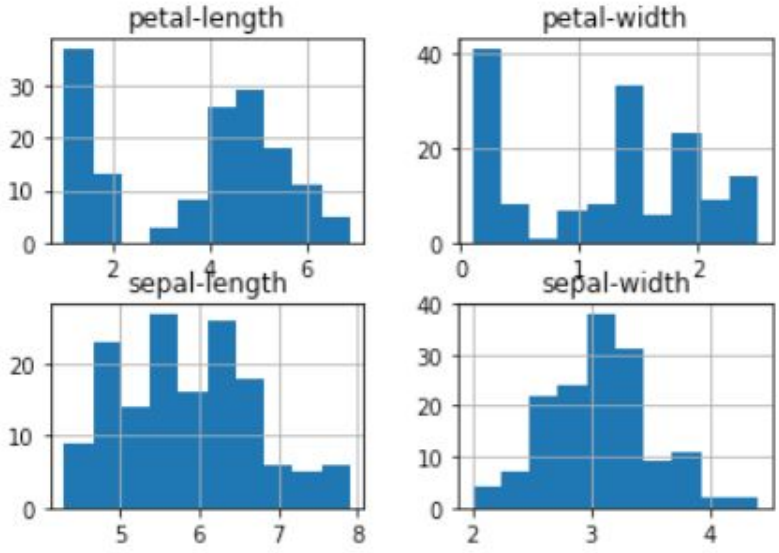
For this data, the data is divided equally for each group. In real problem, data usually imbalanced. Thus a few more steps need to add before proceed building a model. You can find out more in https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/

- You also can visualize the data for further understanding.

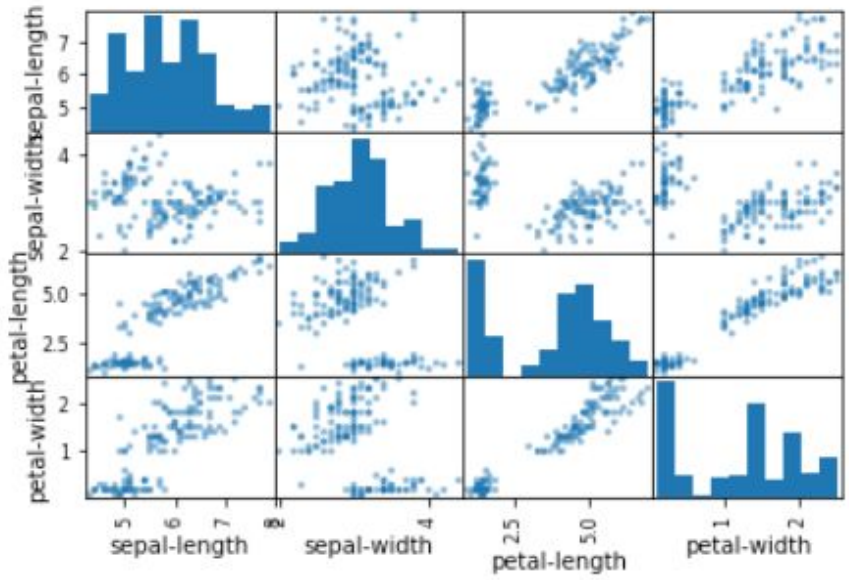
```
# box and whisker plots
dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
plt.show()
```




```
# histograms
dataset.hist()
plt.show()
```



```
# scatter plot matrix
scatter_matrix(dataset)
plt.show()
```



Here, we can see the relation between features

2- Preprocessing data

Next, we will split the data for training and validation.

```
# Split-out validation dataset
array = dataset.values
X = array[:,0:4]
Y = array[:,4]
validation_size = 0.20
seed = 7
X_train, X_validation, Y_train, Y_validation = model_selection.train_test_split(
    X, Y, test_size=validation_size, random_state=seed)
```

The data is split into 80% training and 20% validation.
There are four important variables involve in model learning and prediction which are;
X_train
Y_train
X_validation
Y_validation.

I explained it in table below about the split.

<table>
<tr><td>Index</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>Features</td><td>sepal-length</td><td>sepal-width</td><td>petal-length</td><td>petal-width</td><td>Class
Iris Versicolor
Iris Setosa
Iris Virginica</td></tr>
<tr><td rowspan="2">--
|
|
|
|
150 rows ----|
|
|
|
|
|
|
|
---</td><td colspan="3">X_train (80%)

<u>X is the features</u>

Train data is use for model to learn pattern of the entire data.</td><td colspan="2">Y_train (80%)

<u>Y is the output</u></td></tr>
<tr><td colspan="3">X_validation (20%)

Validation data is data that will exclude during training of the model. The data is used to check performance of the model either the model predict correctly or not.</td><td colspan="2">Y_validation (20%)</td></tr>
</table>

Note:
random_state is the initial shuffling of training data after each epoch(iteration). The value(or seed) can be any integer but it is advisable to put same seed instead of using *pseudo random generator*.

3- Build model

For this project, we can classify flowers in 6 methods which are Logistic Regression(LR), Linear Discrimination Analysis(LDA), K-Nearest Neighbour(KNN), DecisionTreeClassifier, Gaussian Naive Bayes and Support Vector Machine.
In real-life problem, if you have several choices of model to use, you can evaluate models performance first.

For this project, we use K-fold cross validation. K can be any number, in this case we set 10 (10-fold cross validation).

It means dataset will be split into 10 parts, 9 parts will be train and test on 1 part. This process will be repeat for all train-test splits combinations.
The accuracy percentage is calculated by $\frac{correct\ classification}{total\ data} \times 100\%$

You can find out more about cross validation at
https://towardsdatascience.com/cross-validation-explained-evaluating-estimator-performance-e51e5430ff85

```
# Test options and evaluation metric
seed = 7
scoring = 'accuracy'

# Spot Check Algorithms
models = []
models.append(('LR', LogisticRegression(solver='liblinear', multi_class='ovr')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))
# evaluate each model in turn
results = []
names = []
for name, model in models:
  kfold = model_selection.KFold(n_splits=10, random_state=seed)
  cv_results = model_selection.cross_val_score(model, X_train, Y_train, cv=kfold,
                                               scoring=scoring)
  results.append(cv_results)
  names.append(name)
  msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
  print(msg)
```

```
LR: 0.966667 (0.040825)
LDA: 0.975000 (0.038188)
KNN: 0.983333 (0.033333)
CART: 0.983333 (0.033333)
NB: 0.975000 (0.053359)
SVM: 0.991667 (0.025000)
```

As you can see here, SVM is the highest estimated accuracy score. It means that, SVM have the least mistake in classification. For approval, let's classify the flowers by using all methods.

4 - Predict and Show results

- **#LR**

```
# Make prediction using LR

lr = LogisticRegression(solver='liblinear', multi_class='ovr')
lr.fit(X_train, Y_train)
predictions = lr.predict(X_validation)

print('Logistic Regression')
print("Accuracy score : %s" % accuracy_score(Y_validation, predictions))
print("Confusion Matrix ")
print(confusion_matrix(Y_validation, predictions))
print("Report")
print(classification_report(Y_validation, predictions))

#if you want to print the value of test data (X_validation)
#for i in range(len(X_validation)):
  #print("%s : %s      : (%s)" % (X_validation[i], Y_validation[i], predictions[i]))

print("\n wrongly classified")
for j in range(len(X_validation)):
  if Y_validation[j] != predictions[j]:
    print("%s : %s      : (%s)" % (X_validation[j], Y_validation[j], predictions[j]))
```

```
Logistic Regression
Accuracy score : 0.8
Confusion Matrix
[[ 7  0  0]
 [ 0  7  5]
 [ 0  1 10]]
Report
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       0.88      0.58      0.70        12
 Iris-virginica       0.67      0.91      0.77        11

      micro avg       0.80      0.80      0.80        30
      macro avg       0.85      0.83      0.82        30
   weighted avg       0.83      0.80      0.80        30


 wrongly classified
[5.4 3.0 4.5 1.5] : Iris-versicolor      : (Iris-virginica)
[5.6 3.0 4.5 1.5] : Iris-versicolor      : (Iris-virginica)
[6.7 3.0 5.0 1.7] : Iris-versicolor      : (Iris-virginica)
[6.0 3.4 4.5 1.6] : Iris-versicolor      : (Iris-virginica)
[5.9 3.2 4.8 1.8] : Iris-versicolor      : (Iris-virginica)
[7.2 3.0 5.8 1.6] : Iris-virginica      : (Iris-versicolor)
```

- **#LDA**

```
# Make predictions by LDA
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, Y_train)
predictions = lda.predict(X_validation)

print('LDA')
print("Accuracy score : %s" % accuracy_score(Y_validation, predictions))
print("Confusion Matrix ")
print(confusion_matrix(Y_validation, predictions))
print("Report")
print(classification_report(Y_validation, predictions))

#if you want to print the value of test data (X_validation)
#for i in range(len(X_validation)):
  #print("%s : %s      : (%s)" % (X_validation[i], Y_validation[i], predictions[i]))

print("\n wrongly classified")
for j in range(len(X_validation)):
  if Y_validation[j] != predictions[j]:
    print("%s : %s      : (%s)" % (X_validation[j], Y_validation[j], predictions[j]))
```

```
LDA
Accuracy score : 0.9666666666666667
Confusion Matrix
[[ 7  0  0]
 [ 0 11  1]
 [ 0  0 11]]
Report
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       1.00      0.92      0.96        12
 Iris-virginica       0.92      1.00      0.96        11

      micro avg       0.97      0.97      0.97        30
      macro avg       0.97      0.97      0.97        30
   weighted avg       0.97      0.97      0.97        30


 wrongly classified
[5.9 3.2 4.8 1.8] : Iris-versicolor      : (Iris-virginica)
```

- **#KNN**

```
# Make predictions by KNN
knn = KNeighborsClassifier()
knn.fit(X_train, Y_train)
predictions = knn.predict(X_validation)

print('KNN')
print("Accuracy score : %s" % accuracy_score(Y_validation, predictions))
print("Confusion Matrix ")
print(confusion_matrix(Y_validation, predictions))
print("Report")
print(classification_report(Y_validation, predictions))

#if you want to print the value of test data (X_validation)
#for i in range(len(X_validation)):
  #print("%s : %s      : (%s)" % (X_validation[i], Y_validation[i], predictions[i]))

print("\n wrongly classified")
for j in range(len(X_validation)):
  if Y_validation[j] != predictions[j]:
    print("%s : %s      : (%s)" % (X_validation[j], Y_validation[j], predictions[j]))
```

```
KNN
Accuracy score : 0.9
Confusion Matrix
[[ 7  0  0]
 [ 0 11  1]
 [ 0  2  9]]
Report
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       0.85      0.92      0.88        12
 Iris-virginica       0.90      0.82      0.86        11

      micro avg       0.90      0.90      0.90        30
      macro avg       0.92      0.91      0.91        30
   weighted avg       0.90      0.90      0.90        30


 wrongly classified
[4.9 2.5 4.5 1.7] : Iris-virginica      : (Iris-versicolor)
[6.7 3.0 5.0 1.7] : Iris-versicolor      : (Iris-virginica)
[6.0 2.2 5.0 1.5] : Iris-virginica      : (Iris-versicolor)
```

- **#CART**

```
# Make predictions by Decision Tree
dt = DecisionTreeClassifier()
dt.fit(X_train, Y_train)
predictions = dt.predict(X_validation)

print('Decision Tree')
print("Accuracy score : %s" % accuracy_score(Y_validation, predictions))
print("Confusion Matrix ")
print(confusion_matrix(Y_validation, predictions))
print("Report")
print(classification_report(Y_validation, predictions))

#if you want to print the value of test data (X_validation)
#for i in range(len(X_validation)):
  #print("%s : %s      : (%s)" % (X_validation[i], Y_validation[i], predictions[i]))

print("\n wrongly classified")
for j in range(len(X_validation)):
  if Y_validation[j] != predictions[j]:
    print("%s : %s      : (%s)" % (X_validation[j], Y_validation[j], predictions[j]))
```

```
Decision Tree
Accuracy score : 0.8666666666666667
Confusion Matrix
[[ 7  0  0]
 [ 0 10  2]
 [ 0  2  9]]
Report
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       0.83      0.83      0.83        12
 Iris-virginica       0.82      0.82      0.82        11

      micro avg       0.87      0.87      0.87        30
      macro avg       0.88      0.88      0.88        30
   weighted avg       0.87      0.87      0.87        30


 wrongly classified
[6.7 3.0 5.0 1.7] : Iris-versicolor      : (Iris-virginica)
[5.9 3.2 4.8 1.8] : Iris-versicolor      : (Iris-virginica)
[6.0 2.2 5.0 1.5] : Iris-virginica      : (Iris-versicolor)
[7.2 3.0 5.8 1.6] : Iris-virginica      : (Iris-versicolor)
```

- **#GaussianNB**

```
# Make predictions by Decision Tree
dt = DecisionTreeClassifier()
dt.fit(X_train, Y_train)
predictions = dt.predict(X_validation)

print('Decision Tree')
print("Accuracy score : %s" % accuracy_score(Y_validation, predictions))
print("Confusion Matrix ")
print(confusion_matrix(Y_validation, predictions))
print("Report")
print(classification_report(Y_validation, predictions))

#if you want to print the value of test data (X_validation)
#for i in range(len(X_validation)):
  #print("%s : %s      : (%s)" % (X_validation[i], Y_validation[i], predictions[i]))

print("\n wrongly classified")
for j in range(len(X_validation)):
  if Y_validation[j] != predictions[j]:
    print("%s : %s      : (%s)" % (X_validation[j], Y_validation[j], predictions[j]))
```

```
Decision Tree
Accuracy score : 0.8666666666666667
Confusion Matrix
[[ 7  0  0]
 [ 0 10  2]
 [ 0  2  9]]
Report
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       0.83      0.83      0.83        12
 Iris-virginica       0.82      0.82      0.82        11

      micro avg       0.87      0.87      0.87        30
      macro avg       0.88      0.88      0.88        30
   weighted avg       0.87      0.87      0.87        30


 wrongly classified
[6.7 3.0 5.0 1.7] : Iris-versicolor      : (Iris-virginica)
[5.9 3.2 4.8 1.8] : Iris-versicolor      : (Iris-virginica)
[6.0 2.2 5.0 1.5] : Iris-virginica      : (Iris-versicolor)
[7.2 3.0 5.8 1.6] : Iris-virginica      : (Iris-versicolor)
```

- **#SVM**

```
# Make predictions by SVM
svm = SVC(gamma='auto')
svm.fit(X_train, Y_train)
predictions = svm.predict(X_validation)

print('SVM')
print("Accuracy score : %s" % accuracy_score(Y_validation, predictions))
print("Confusion Matrix ")
print(confusion_matrix(Y_validation, predictions))
print("Report")
print(classification_report(Y_validation, predictions))

#if you want to print the value of test data (X_validation)
#for i in range(len(X_validation)):
  #print("%s : %s      : (%s)" % (X_validation[i], Y_validation[i], predictions[i]))

print("\n wrongly classified")
for j in range(len(X_validation)):
  if Y_validation[j] != predictions[j]:
    print("%s : %s      : (%s)" % (X_validation[j], Y_validation[j], predictions[j]))
```

```
SVM
Accuracy score : 0.9333333333333333
Confusion Matrix
[[ 7  0  0]
 [ 0 10  2]
 [ 0  0 11]]
Report
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       1.00      0.83      0.91        12
 Iris-virginica       0.85      1.00      0.92        11

      micro avg       0.93      0.93      0.93        30
      macro avg       0.95      0.94      0.94        30
   weighted avg       0.94      0.93      0.93        30

[6.7 3.0 5.0 1.7] : Iris-versicolor      : (Iris-virginica)
[5.9 3.2 4.8 1.8] : Iris-versicolor      : (Iris-virginica)
```

Note: Confusion Matrix is where we can see how many correct and incorrect classification occur.

Actual Class >>> Prediction Class v v	Iris Setosa	Iris Versicolor	Iris Virginica
Iris Setosa			
Iris Versicolor			
Iris Virginica			

So above is the results for all six method in classification of Iris. We can summarized as below.

Methods	Accuracy score from K-fold	Accuracy score	Wrongly classified
LR	0.9667	0.8	6
LDA	0.9750	0.9667	1
KNN	0.9833	0.9	3
CART	0.9833	0.8667	4
Gaussian NB	0.9750	0.8333	5
SVM	0.991667	0.9333	2

According to accuracy_score, the highest accuracy is LDA and it is prove that LDA only make 1 mistake compare to the others.

However, SVM got the highest in its cross validation accuracy score, but it prove that it make 2 mistakes.

Note:
Why cross_val_score and accuracy_score different?

Answer:
They're going to be different because in cross_val_score, you obtain an accuracy for each of your folds and average them. For each CV fold, your training and tests set are different; so, you obtain different accuracy values for each of them, and it enables you to calculate standard deviation of your accuracies, which is enclosed in parentheses in your image. accuracy_score of sklearn.metrics library calculates the accuracy based on the inputs y_pred and y_true. For example, if you input your entire training set, you'll get accuracy of your entire training set, which is of course slightly different than your CV score.
(Source:
https://stats.stackexchange.com/questions/393992/what-is-difference-between-accuracy-score-and-cross-val-score)

Conclusion

It is okay if you are not understand entirely for the first time as long as you successfully follow and run the tutorial from start to the end. You will understand little by little in machine learning as you dive in. Keep asking question and find the answer or ask the expert.

Try to implement on your problem and see how it works.

Practice:
Classify students marks into fail or pass. A student classify as pass when mathematics mark is 50 percent and above, and science mark is 70 percent and above. A student is fail when not fulfill the pass criteria.

Note: In machine learning, you don't have to make any condition in the model like, when math is more than 50 percent and science is more than 70 percent, status is pass. Machine learning model will read pattern on dataset given and the model will decide by their own on what the output will be.
You can refer to https://github.com/RufaidahOmar/Balanced-Dataset-Classification.git
for answer.

Good Luck.