Iris Classifier by Various Models

Okino Kamali Leiba

In [ ]:

```
import pandas as pd, numpy as np, matplotlib.pyplot as plt, seaborn as sns, os
```

In [ ]:

```
data_set = "C:/Users/Owner/source/vsc_repo/confusion_matrix_cookbook/iris_confusion
iris_data = pd.read_csv(data_set, engine="c", delimiter=",", encoding="utf-8", head
```

In [ ]:

```
from io import StringIO
python_data = open(data_set).read()
lst_com = [list_item.split(",") for list_item in python_data.splitlines()]
# data_clip = pd.read_clipboard(python_data)
# data_table = pd.read_table(python_data)
data_csv = pd.read_csv(StringIO(python_data), header=0, sep=",", engine="c", linete

# https://matthewrocklin.com/blog/work/2017/10/16/streaming-dataframes-1

pd.DataFrame(lst_com)
```

Out[ ]:

	0	1	2	3	4	5
0	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	1	5.1	3.5	1.4	0.2	Iris-setosa
2	2	4.9	3.0	1.4	0.2	Iris-setosa
3	3	4.7	3.2	1.3	0.2	Iris-setosa
4	4	4.6	3.1	1.5	0.2	Iris-setosa
...	...	...	...	...	...	...
146	146	6.7	3.0	5.2	2.3	Iris-virginica
147	147	6.3	2.5	5.0	1.9	Iris-virginica
148	148	6.5	3.0	5.2	2.0	Iris-virginica
149	149	6.2	3.4	5.4	2.3	Iris-virginica
150	150	5.9	3.0	5.1	1.8	Iris-virginica

151 rows × 6 columns

Exploratory Data Analysis

In [ ]:

```
iris_data.index
```

Out[ ]:

```
RangeIndex(start=0, stop=150, step=1)
```

In [ ]:

```
iris_data.columns
```

Out[ ]:

```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

In [ ]:

```
iris_data.dtypes
```

Out[ ]:

```
Id                 int64
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
PetalWidthCm     float64
Species           object
dtype: object
```

In [ ]:

```
iris_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

In [ ]:

```
iris_data["Species"].unique()
```

Out[ ]:

```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

In [ ]:

```
iris_data.head(5)
```

Out[ ]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [ ]:

```
iris_data.tail(5)
```

Out[ ]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

Set Figure Theme

In [ ]:

```
custom_params = {'figure.facecolor': 'orange',
 'axes.labelcolor': '.15',
 'xtick.direction': 'out',
 'ytick.direction': 'out',
 'xtick.color': '.15',
 'ytick.color': '.15',
 'axes.axisbelow': True,
 'grid.linestyle': '-',
 'text.color': '.15',
 'font.family': ['sans-serif'],
 'font.sans-serif': ['Arial',
  'DejaVu Sans',
  'Liberation Sans',
  'Bitstream Vera Sans',
  'sans-serif'],
 'lines.solid_capstyle': 'round',
 'patch.edgecolor': 'w',
 'patch.force_edgecolor': True,
 'image.cmap': 'rocket',
 'xtick.top': True,
 'ytick.right': True,
 'axes.grid': True,
 'axes.facecolor': '#EAEAF2',
 'axes.edgecolor': 'black',
 'grid.color': 'white',
 'axes.spines.left': False,
 'axes.spines.bottom': False,
 'axes.spines.right': False,
 'axes.spines.top': False,
 'xtick.bottom': True,
 'ytick.left': True}
sns.set_theme(style="ticks", rc=custom_params)
```

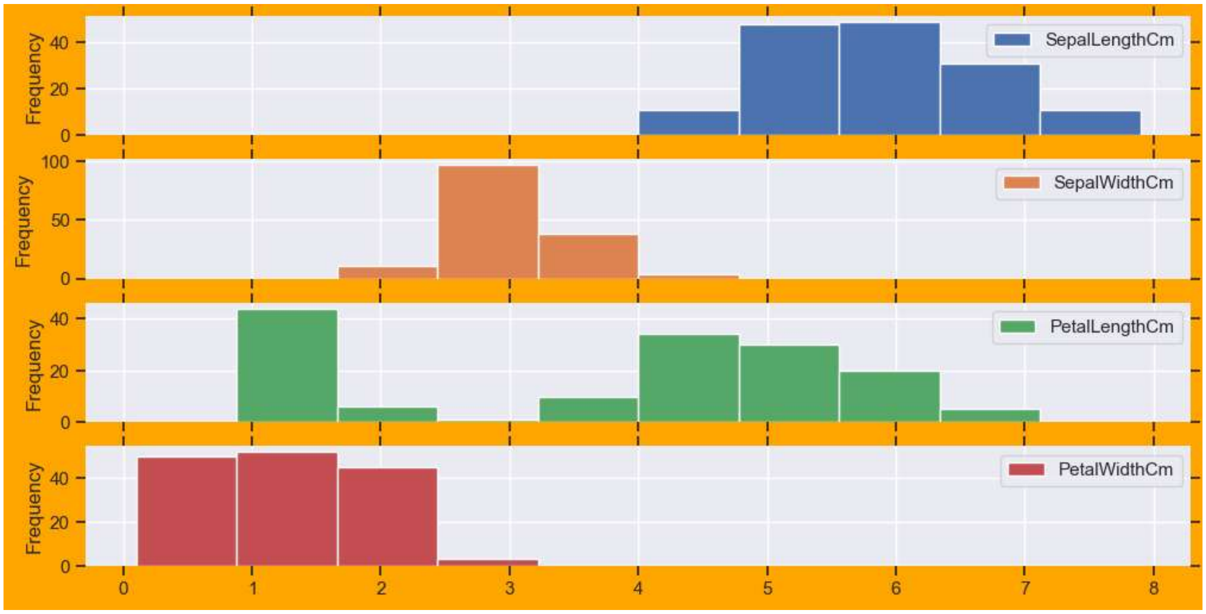
Data Transformation and Preparation

In [ ]:

```
iris_data = iris_data.drop("Id", axis=1, errors="ignore", inplace=False)
```

In [ ]:

```
iris_data.plot.hist(subplots=True, figsize=(12,6),);
```



In [ ]:

```
x = iris_data.drop("Species", axis=1, inplace=False, errors="ignore")
# X scale-min_max = (X – X min) / (X max – X min)
X = (x - np.min(x)) / (np.max(x) -np.min(x))
# X scale_MAS = x / max(abs|x|)
# X = x / np.max(np.abs(x))
# X z-score = (x - mean / (x - std)  *normal distribution
#   X = (x - np.mean(x)) / (x - np.std(x))
y = iris_data["Species"]
```

```
c:\ProgramData\Anaconda3\envs\conda_env\lib\site-packages\numpy\core\fromnumeric.p
y:84: FutureWarning: In a future version, DataFrame.min(axis=None) will return a s
calar min over the entire DataFrame. To retain the old behavior, use 'frame.min(ax
is=0)' or just 'frame.min()'
  return reduction(axis=axis, out=out, **passkwargs)
c:\ProgramData\Anaconda3\envs\conda_env\lib\site-packages\numpy\core\fromnumeric.p
y:84: FutureWarning: In a future version, DataFrame.max(axis=None) will return a s
calar max over the entire DataFrame. To retain the old behavior, use 'frame.max(ax
is=0)' or just 'frame.max()'
  return reduction(axis=axis, out=out, **passkwargs)
c:\ProgramData\Anaconda3\envs\conda_env\lib\site-packages\numpy\core\fromnumeric.p
y:84: FutureWarning: In a future version, DataFrame.min(axis=None) will return a s
calar min over the entire DataFrame. To retain the old behavior, use 'frame.min(ax
is=0)' or just 'frame.min()'
  return reduction(axis=axis, out=out, **passkwargs)
```

In [ ]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.09, random_st
```

Random Forest Classifier

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings("ignore")
```

```
param_grid = {"n_estimators" : [50, 100, 150, 200], "max_depth" : [2, 4, 6, 8, 10],
"log2"], "random_state" : [0,42]}
search_grid = GridSearchCV(RandomForestClassifier(), param_grid, cv = 5, scoring =
search_grid.fit(X_train, y_train)
print(search_grid.best_params_)

rf = RandomForestClassifier(n_estimators=50, max_depth=2, max_features='sqrt', min_
rf.fit(X_train, y_train)
print("Random Forest Classifier: ", rf.score(X_test, y_test))
```

```
{'max_depth': 2, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_spli
t': 2, 'n_estimators': 50, 'random_state': 0}
Random Forest Classifier:  0.9285714285714286
```

In [ ]:

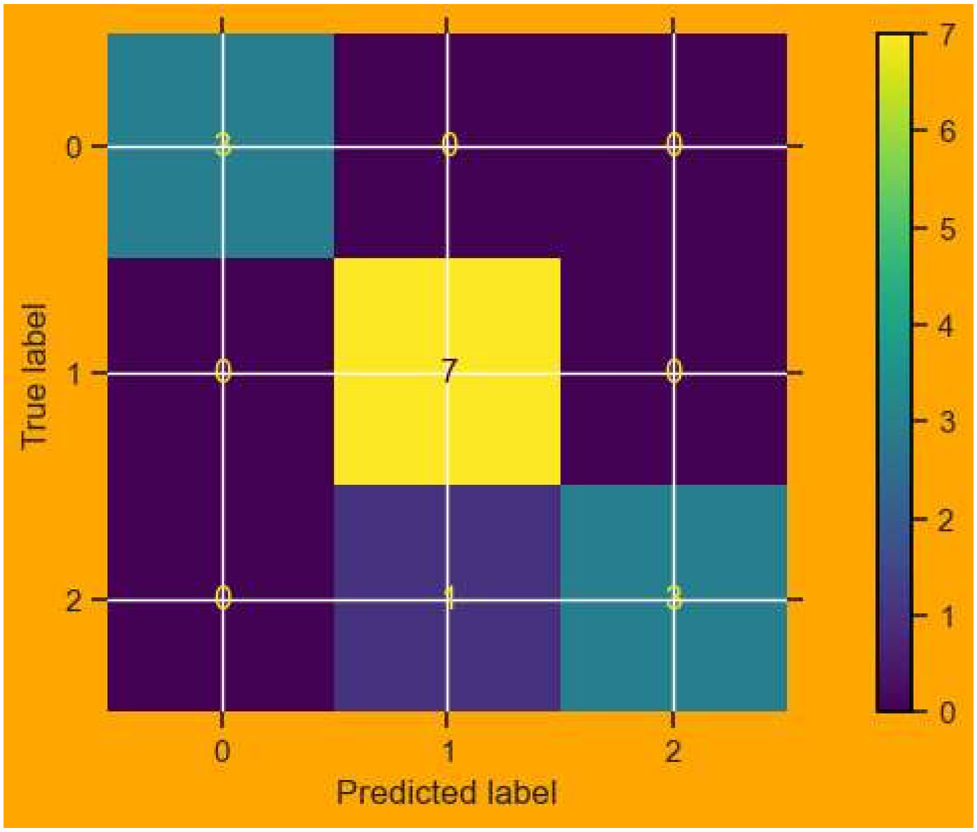
```
from sklearn.metrics import confusion_matrix
y_predict_rf = rf.predict(X_test)

cm_rf = confusion_matrix(y_test, y_predict_rf)
cm_rf
```

Out[ ]:

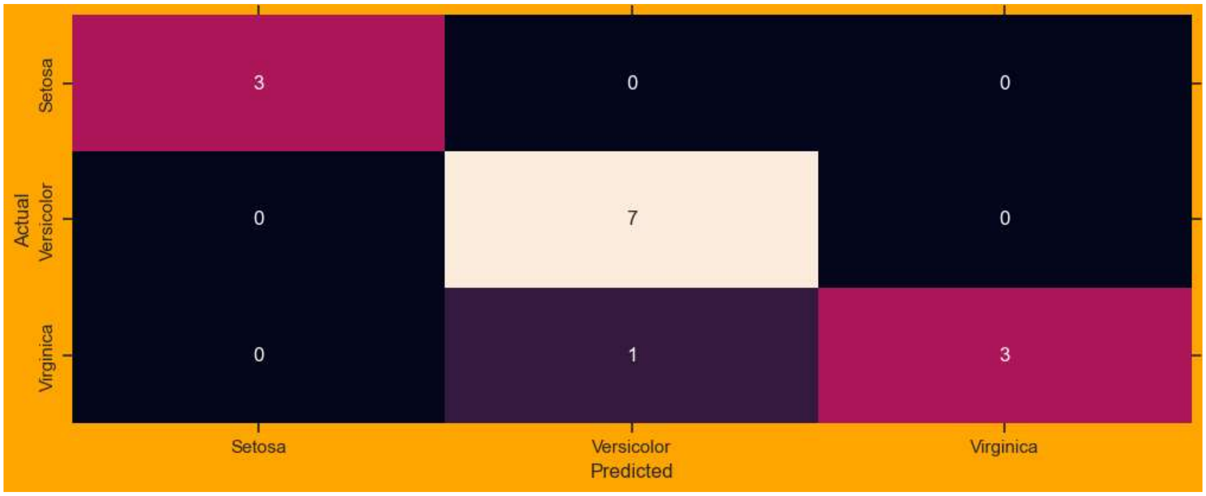
```
array([[3, 0, 0],
       [0, 7, 0],
       [0, 1, 3]], dtype=int64)
```

In [ ]:

```
from sklearn.metrics import ConfusionMatrixDisplay
fig, axe = plt.subplots(figsize=(10,4), constrained_layout=True, dpi=100)
# cm_display = ConfusionMatrixDisplay(cm_rf).plot()
cm_display = ConfusionMatrixDisplay(cm_rf)
cm_display.plot(ax=axe);
```



```
In [ ]: plt.figure(figsize=(10,4), constrained_layout=True, dpi=100)
        axe = sns.heatmap(cm_rf, annot=True, fmt=".3g", cbar= False, xticklabels=["Setosa",
        axe.set_xlabel("Predicted");
        axe.set_ylabel("Actual");
```



Support Vector Classifier

```
In [ ]: from sklearn.svm import SVC
        from sklearn.model_selection import GridSearchCV

        param_grid = {"kernel" : ["linear", "poly", "rbf", "sigmoid"], "degree" : [3, 6, 9,
        grid_search = GridSearchCV(SVC(), param_grid, refit=True, cv=5, scoring="accuracy")
        grid_search.fit(X_train, y_train)
        print(grid_search.best_params_)

        svc = SVC(kernel="linear", random_state=0, degree=3)
        svc.fit(X_train, y_train)
        print("Support Vector Classifier: ", svc.score(X_test, y_test))
```

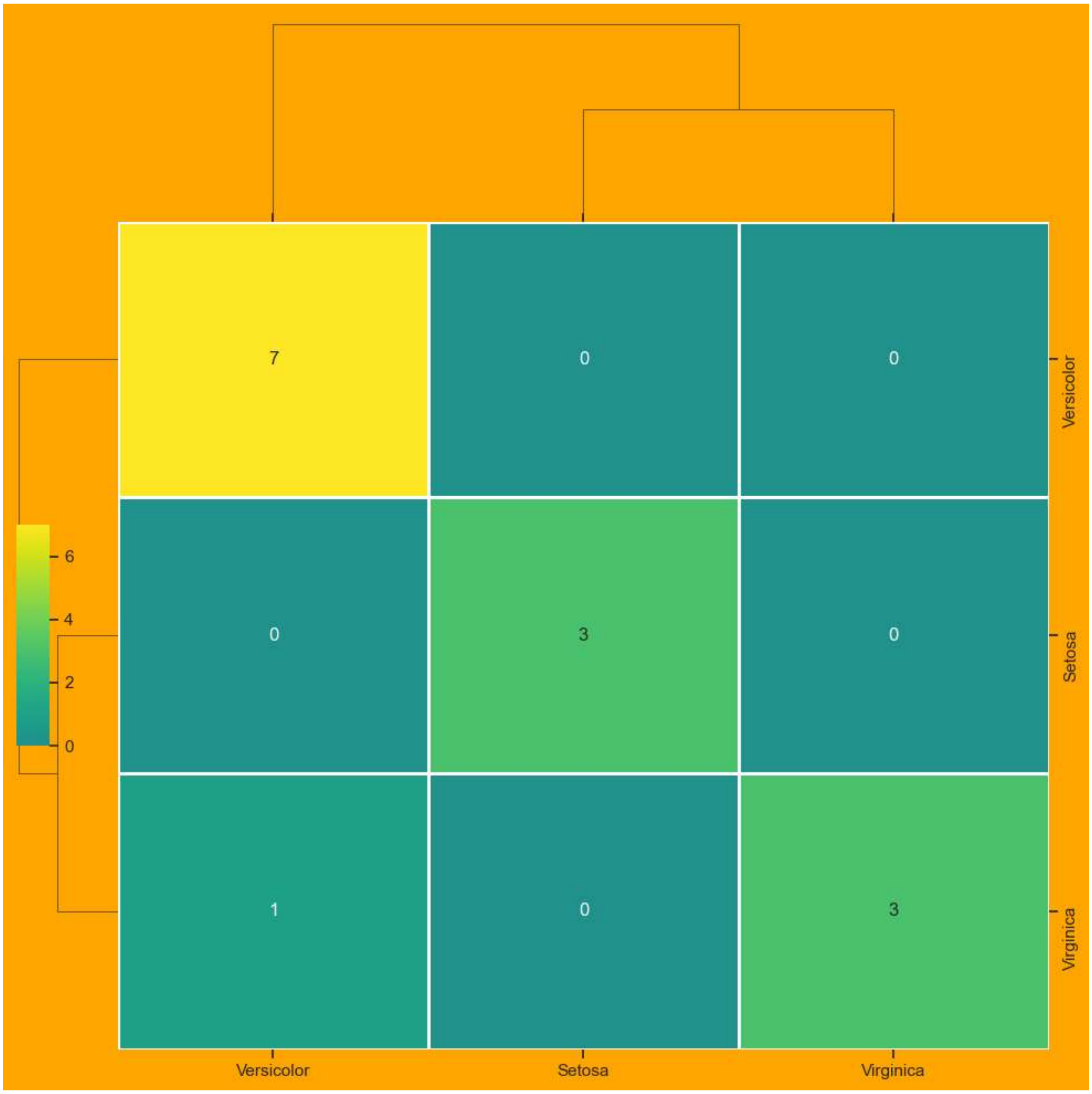
```
{'degree': 3, 'kernel': 'linear', 'random_state': 0}
Support Vector Classifier:  0.9285714285714286
```

```
In [ ]: from sklearn.metrics import confusion_matrix
        y_predict_svc = svc.predict(X_test)

        cm_svc = confusion_matrix(y_test, y_predict_svc)
        cm_svc
```

```
Out[ ]: array([[3, 0, 0],
               [0, 7, 0],
               [0, 1, 3]], dtype=int64)
```

```
In [ ]: plt.figure(figsize=(5,3), constrained_layout=True, dpi=100)
        sns.clustermap(cm_svc,  xticklabels=["Setosa","Versicolor","Virginica"], yticklabel
        plt.show()
```

```
<Figure size 500x300 with 0 Axes>
```

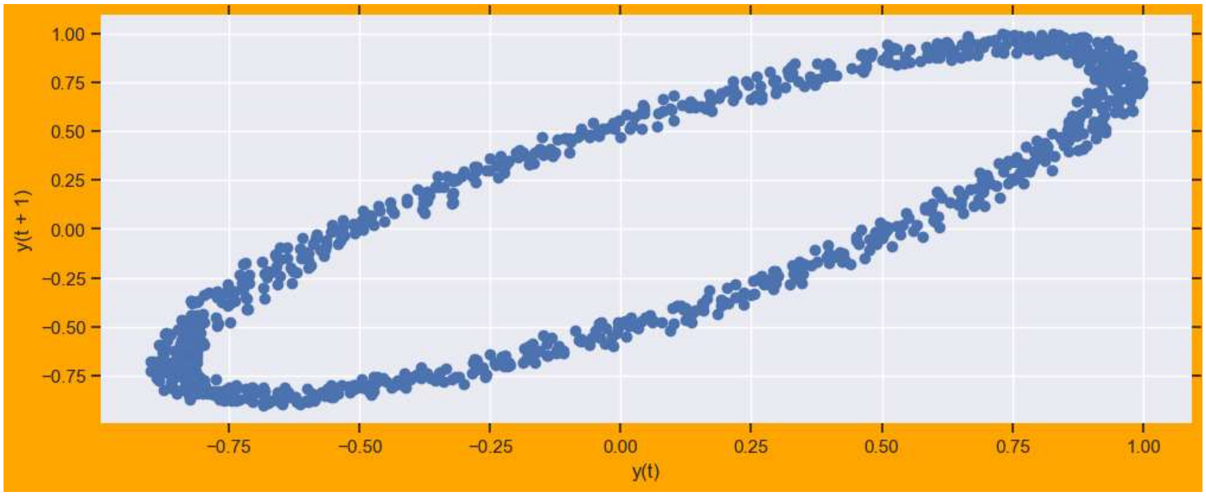


In [ ]:

```
from pandas.plotting import lag_plot

plt.figure(figsize=(10,4), constrained_layout=True, dpi=100)

data = pd.Series(0.1 * np.random.rand(1000) + 0.9 * np.sin(np.linspace(-99 * np.pi,

lag_plot(data)
```

Out[ ]:

```
<Axes: xlabel='y(t)', ylabel='y(t + 1)'>
```


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
y(t + 1)
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
y(t)