

Machine learning algorithms for classification and regression on iris in Python

```
In [169... import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import datasets
import pandas as pd
import numpy as np

# Convert 'iris.data' numpy array to 'iris.dataframe' pandas dataframe
# complete the iris dataset by adding species
iris = datasets.load_iris()
iris = pd.DataFrame(
    data= np.c_[iris['data'], iris['target']],
    columns= iris['feature_names'] + ['target']
)

species = []

for i in range(len(iris['target'])):
    if iris['target'][i] == 0:
        species.append("setosa")
    elif iris['target'][i] == 1:
        species.append('versicolor')
    else:
        species.append('virginica')

iris['species'] = species
iris
```

```
Out[169]:
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	species
0	5.1	3.5	1.4	0.2	0.0	setosa
1	4.9	3.0	1.4	0.2	0.0	setosa
2	4.7	3.2	1.3	0.2	0.0	setosa
3	4.6	3.1	1.5	0.2	0.0	setosa
4	5.0	3.6	1.4	0.2	0.0	setosa
...
145	6.7	3.0	5.2	2.3	2.0	virginica
146	6.3	2.5	5.0	1.9	2.0	virginica
147	6.5	3.0	5.2	2.0	2.0	virginica
148	6.2	3.4	5.4	2.3	2.0	virginica
149	5.9	3.0	5.1	1.8	2.0	virginica

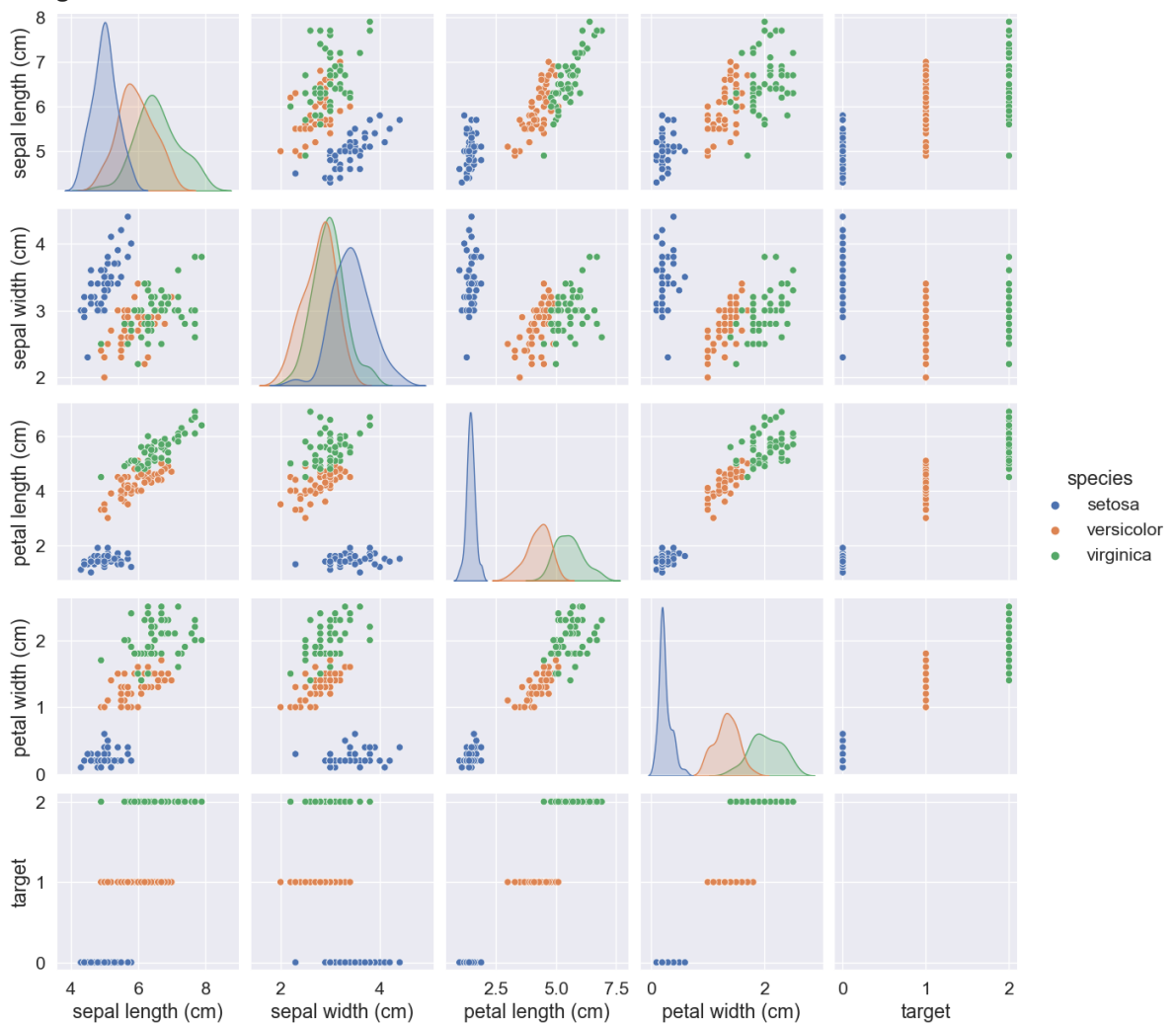
150 rows × 6 columns

1. Visualization

In [171]...

```
plt.figure(figsize=(6,3))
sns.pairplot(iris, hue='species')
plt.show()
```

<Figure size 600x300 with 0 Axes>



2. splitting the dataset into training and test sets

In [141]...

```
X = iris.iloc[:, 0:4]
y = iris.iloc[:, 4]
class_names = iris.iloc[:, 5]

from sklearn.model_selection import train_test_split
import random

random.seed(2023)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=40, train_s:
```

2.1 Saving a copy of the different datasets in .csv files

In [168]...

```
# 2. save entire dataset, training and testing datasets
# save a copy of the dataset in .csv
iris.to_csv('C:/Users/julia/OneDrive/Desktop/github/24. Machine learning toolbox Py
iris.to_csv('C:/Users/julia/OneDrive/Desktop/github/24. Machine learning toolbox Py
index = False)
```

```
iris.to_csv('C:/Users/julia/OneDrive/Desktop/github/24. Machine learning toolbox Py  
index = False)
```

3. Some metrics

TP = true positive, TN = true negative, FP = false positive, FN = false negative

Accuracy. The number of samples correctly classified out of all the samples present in the (test) set.

$$Accuracy = \frac{TP + TN}{(TP + TN + FP + FN)}$$

Precision (for the positive class). The number of samples actually belonging to the positive class out of all the samples that were predicted to be of the positive class by the model.

$$Precision = \frac{TP}{(TP + FP)}$$

Recall (for the positive class). The number of samples predicted correctly to be belonging to the positive class out of all the samples that actually belong to the positive class.

$$Recall = \frac{TP}{(TP + NP)}$$

F1-Score (for the positive class). The harmonic mean of the precision and recall scores obtained for the positive class.

$$F1 - score = \frac{2 * Precision * Recall}{(Precision + Recall)}$$

1. Naive Bayes classifier

1.1 train the model

```
In [143... from sklearn.naive_bayes import GaussianNB  
  
# create a Gaussian NB classifier  
nb_model = GaussianNB()  
  
# fit the model to the iris dataset  
nb_model.fit(X_train,y_train)  
  
# make predictions on test set  
y_pred_nb = nb_model.predict(X_test)
```

1.2 Confusion matrix

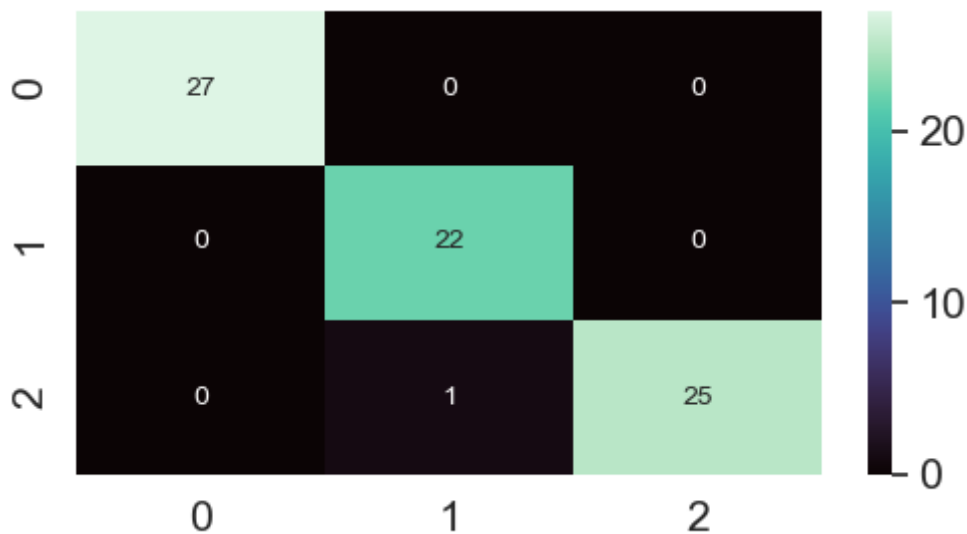
Now that we have predictions, we can compute a confusion matrix and the accuracy of our trained SVM classifier on the testing set.

```
In [144... from sklearn.metrics import confusion_matrix

cm_nb = confusion_matrix(y_test, y_pred_nb)
cm_nb

df_cm_nb = pd.DataFrame(cm_nb, range(len(class_names.unique())), range(len(class_names.unique())))

plt.figure(figsize=(6,3))
sns.set(font_scale=1.4) # for label size
sns.heatmap(df_cm_nb, annot=True, annot_kws={"size": 10}, cmap = sns.color_palette("magma", 3))
plt.show()
```



1.3 Accuracy of the Naive Bayes classifier

```
In [145... from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

accuracy_test_nb = round(accuracy_score(y_test, y_pred_nb)* 100, 2)
accuracy_train_nb = round(nb_model.score(X_train, y_train)* 100, 2)
precision_nb = precision_score(y_test, y_pred_nb, average = 'micro')
recall_nb = recall_score(y_test, y_pred_nb, average = 'micro')
f1_nb = f1_score(y_test, y_pred_nb, average = 'micro')

print("Accuracy testing: %.3f" % accuracy_test_nb)
print("Accuracy training: %.3f" % accuracy_train_nb)
print('precision_NB : %.3f' % precision_nb)
print('recall_NB: %.3f' % recall_nb)
print('f1-score_NB : %.3f' % f1_nb)
```

```
Accuracy testing: 98.670
Accuracy training: 93.330
precision_NB : 0.987
recall_NB: 0.987
f1-score_NB : 0.987
```

2. Random forest classifier

2.1 train the model

```
In [146... from sklearn.ensemble import RandomForestClassifier

# create a Gaussian RF classifier
rf_model = RandomForestClassifier(n_estimators=1000)

# fit the model to the iris dataset
rf_model.fit(X_train,y_train)

# make predictions on test set
y_pred_rf = rf_model.predict(X_test)
```

2.2 Confusion matrix and accuracy

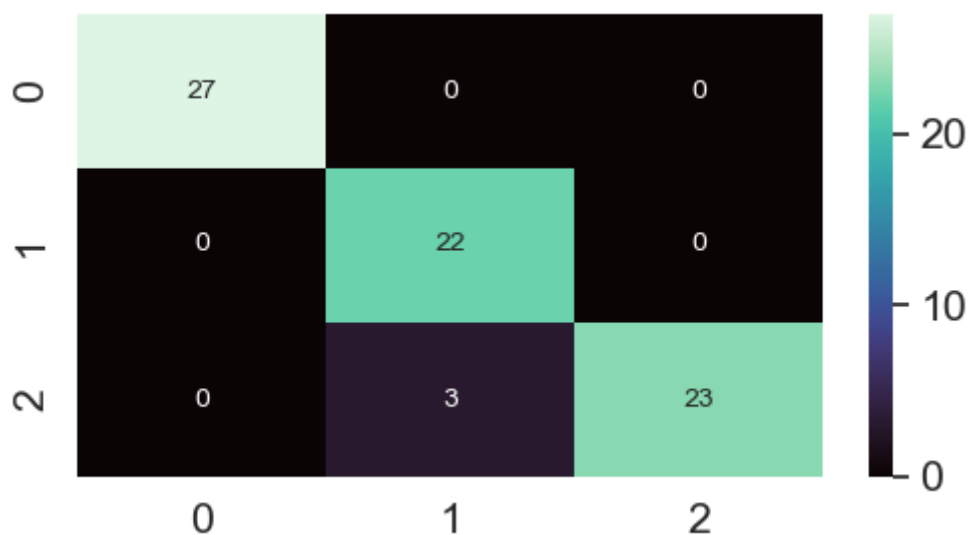
Now that we have predictions, we can compute a confusion matrix and the accuracy of our trained SVM classifier on the testing set.

```
In [147... from sklearn.metrics import confusion_matrix

cm_rf = confusion_matrix(y_test, y_pred_rf)
cm_rf

df_cm_rf = pd.DataFrame(cm_rf, range(len(class_names.unique())), range(len(class_names.unique())))

plt.figure(figsize=(6,3))
sns.set(font_scale=1.4) # for label size
sns.heatmap(df_cm_rf, annot=True, annot_kws={"size": 10}, cmap = sns.color_palette("magma", 3))
plt.show()
```



2.3 Accuracy of the Random forest classifier

```
In [148... from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

accuracy_test_rf = round(accuracy_score(y_test, y_pred_rf)* 100, 2)
accuracy_train_rf = round(rf_model.score(X_train, y_train)* 100, 2)
precision_rf = precision_score(y_test, y_pred_rf, average = 'micro')
recall_rf = recall_score(y_test, y_pred_rf, average = 'micro')
f1_rf = f1_score(y_test, y_pred_rf, average = 'micro')
```

```

print("Accuracy testing: %.3f" % accuracy_test_rf)
print("Accuracy training: %.3f" % accuracy_train_rf)
print('precision_rf : %.3f' %precision_rf)
print('recall_rf: %.3f' %recall_rf)
print('f1-score_rf : %.3f' %f1_rf)

```

```

Accuracy testing: 96.000
Accuracy training: 100.000
precision_rf : 0.960
recall_rf: 0.960
f1-score_rf : 0.960

```

3. Multinomial Logistic Regression classifier

3.1 train the model

In [149...

```

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# create a Logistic regression model
lr_model = LogisticRegression(solver='lbfgs', max_iter=400, multi_class = 'multinomial')

# fit the model to the iris dataset
lr_model.fit(X_train, y_train)

# make predictions on test set
y_pred_lr = lr_model.predict(X_test)

```

3.2 Confusion matrix and accuracy

Now that we have predictions, we can compute a confusion matrix and the accuracy of our trained SVM classifier on the testing set.

In [150...

```

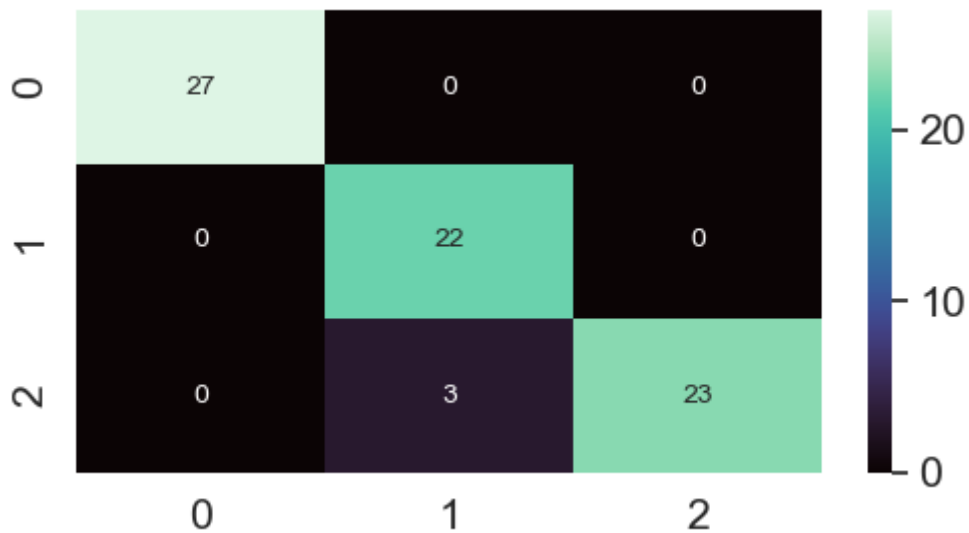
from sklearn.metrics import confusion_matrix
cm_lr = confusion_matrix(y_test, y_pred_lr)
cm_lr

df_cm_lr = pd.DataFrame(cm_lr, range(len(class_names.unique())), range(len(class_names.unique())))

plt.figure(figsize=(6,3))
sns.set(font_scale=1.4) # for label size
sns.heatmap(df_cm_lr, annot=True, annot_kws={"size": 10}, cmap = sns.color_palette("magma", as_cmap=True))

plt.show()

```



3.3 Accuracy of the Multinomial Logistic Regression classifier

```
In [151... from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

accuracy_test_lr = round(accuracy_score(y_test, y_pred_lr)* 100, 2)
accuracy_train_lr = round(lr_model.score(X_train, y_train)* 100, 2)
precision_lr = precision_score(y_test, y_pred_lr, average = 'micro')
recall_lr = recall_score(y_test, y_pred_lr, average = 'micro')
f1_lr = f1_score(y_test, y_pred_lr, average = 'micro')

print("Accuracy testing: %.3f" % accuracy_test_lr)
print("Accuracy training: %.3f" % accuracy_train_lr)
print('precision_lr : %.3f' % precision_lr)
print('recall_lr: %.3f' % recall_lr)
print('f1-score_lr : %.3f' % f1_lr)

Accuracy testing: 96.000
Accuracy training: 98.670
precision_lr : 0.960
recall_lr: 0.960
f1-score_lr : 0.960
```

4. Support Vector Machines classifier

4.1 train the model

```
In [152... from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# create a SVM model
svm_model = SVC(kernel = 'linear', random_state = 0)

# fit the model to the iris dataset
svm_model.fit(X_train, y_train)

# make predictions on test set
y_pred_svm = svm_model.predict(X_test)
```

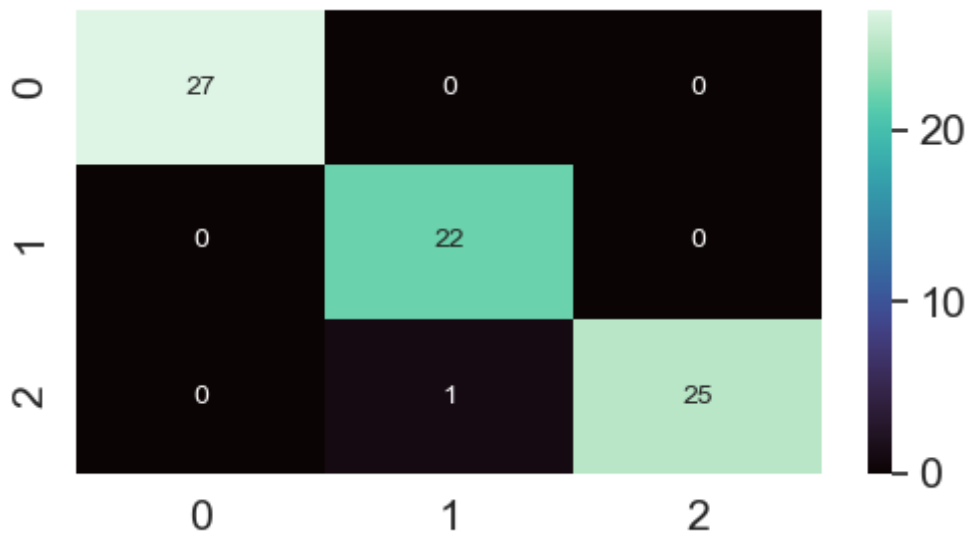
4.2 Confusion matrix and accuracy

Now that we have predictions, we can compute a confusion matrix and the accuracy of our trained SVM classifier on the testing set.

```
In [153... from sklearn.metrics import confusion_matrix
cm_svm = confusion_matrix(y_test, y_pred_svm)
cm_svm

df_cm_svm = pd.DataFrame(cm_svm, range(len(class_names.unique())), range(len(class_names.unique())))

plt.figure(figsize=(6,3))
sns.set(font_scale=1.4) # for label size
sns.heatmap(df_cm_svm, annot=True, annot_kws={"size": 10}, cmap = sns.color_palette("magma", as_cmap=True))
plt.show()
```



4.3 Accuracy of the SVM classifier

```
In [154... from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

accuracy_test_svm = round(accuracy_score(y_test, y_pred_svm)* 100, 2)
accuracy_train_svm = round(svm_model.score(X_train, y_train)* 100, 2)
precision_svm = precision_score(y_test, y_pred_svm, average = 'micro')
recall_svm = recall_score(y_test, y_pred_svm, average = 'micro')
f1_svm = f1_score(y_test, y_pred_svm, average = 'micro')

print("Accuracy testing: %.3f" % accuracy_test_svm)
print("Accuracy training: %.3f" % accuracy_train_svm)
print('precision_svm : %.3f' % precision_svm)
print('recall_svm: %.3f' % recall_svm)
print('f1-score_svm : %.3f' % f1_svm)
```

```
Accuracy testing: 98.670
Accuracy training: 98.670
precision_svm : 0.987
recall_svm: 0.987
f1-score_svm : 0.987
```

5. K-Nearest Neighbors classifier

5.1 train the model


```
In [159... from sklearn.neighbors import KNeighborsClassifier

# create a KNN model
knn_model = KNeighborsClassifier(n_neighbors = 5, weights = 'distance')

# fit the model to the iris dataset
knn_model.fit(X_train, y_train)

# make predictions on test set
y_pred_knn = knn_model.predict(X_test)
```

5.2 Confusion matrix

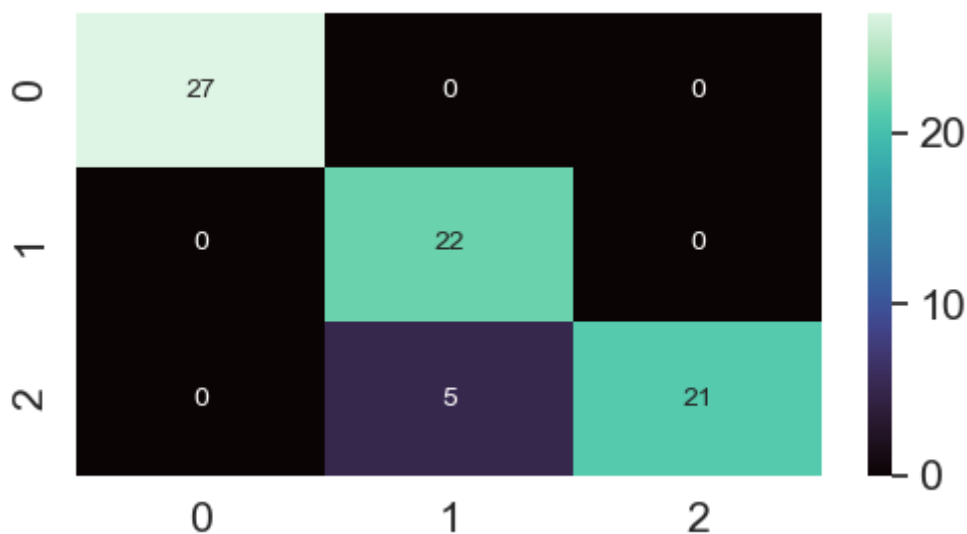
Now that we have predictions, we can compute a confusion matrix and the accuracy of our trained SVM classifier on the testing set.

```
In [156... from sklearn.metrics import confusion_matrix
cm_knn = confusion_matrix(y_test, y_pred_knn)
cm_knn

df_cm_knn = pd.DataFrame(cm_knn, range(len(class_names.unique())), range(len(class_names.unique())))

plt.figure(figsize=(6,3))
sns.set(font_scale=1.4) # for label size
sns.heatmap(df_cm_knn, annot=True, annot_kws={"size": 10}, cmap = sns.color_palette("magma", as_cmap=True))

plt.show()
```



5.3 Accuracy of the KNN classifier

```
In [157... from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

accuracy_test_knn = round(accuracy_score(y_test, y_pred_knn)* 100, 2)
accuracy_train_knn = round(knn_model.score(X_train, y_train)* 100, 2)
precision_knn = precision_score(y_test, y_pred_knn, average = 'micro')
recall_knn = recall_score(y_test, y_pred_knn, average = 'micro')
f1_knn = f1_score(y_test, y_pred_knn, average = 'micro')

print("Accuracy testing: %.3f" % accuracy_test_knn)
print("Accuracy training: %.3f" % accuracy_train_knn)
print('precision_knn : %.3f' % precision_knn)
```

```
print('recall_knn: %.3f' %recall_knn)
print('f1-score_knn : %.3f' %f1_knn)
```

Accuracy testing: 93.330
Accuracy training: 100.000
precision_knn : 0.933
recall_knn: 0.933
f1-score_knn : 0.933

6. Neural Network (MLP) classifier

6.1 train the model

```
In [160... from sklearn.neural_network import MLPClassifier

mlp_model = MLPClassifier(solver='lbfgs', alpha=1e-5,
                          hidden_layer_sizes=(5, 2), random_state=1)
# fit the model to the iris dataset
mlp_model.fit(X_train, y_train)

# make predictions on test set
y_pred_mlp = mlp_model.predict(X_test)
```

6.2 Confusion matrix

Now that we have predictions, we can compute a confusion matrix and the accuracy of our trained NN classifier on the testing set.

```
In [163... from sklearn.metrics import confusion_matrix
cm_mlp = confusion_matrix(y_test, y_pred_mlp)
cm_mlp

df_cm_mlp = pd.DataFrame(cm_mlp, range(len(class_names.unique())), range(len(class_
df_cm_mlp
```

```
Out[163]:
```

	0	1	2
0	0	27	0
1	0	22	0
2	0	26	0

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```