AVIREDDY NVSRK ROHAN

19BCE1180

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from matplotlib import cm
from sklearn.metrics import silhouette_samples
```

In [2]:

```
iris_data = pd.read_csv("/content/iris.csv")
X = iris_data[[column for column in iris_data.columns if column != 'species']]
y = iris_data['species']
```

In [3]:

```
iris_data.columns
```

Out[3]:

```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

clustering by kmeans++

In [ ]:

```
iris_data
```

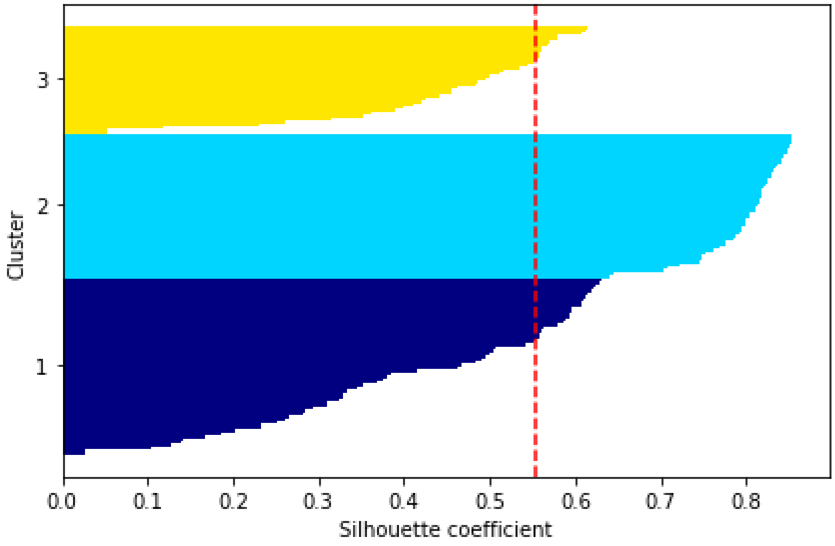
Out[10]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

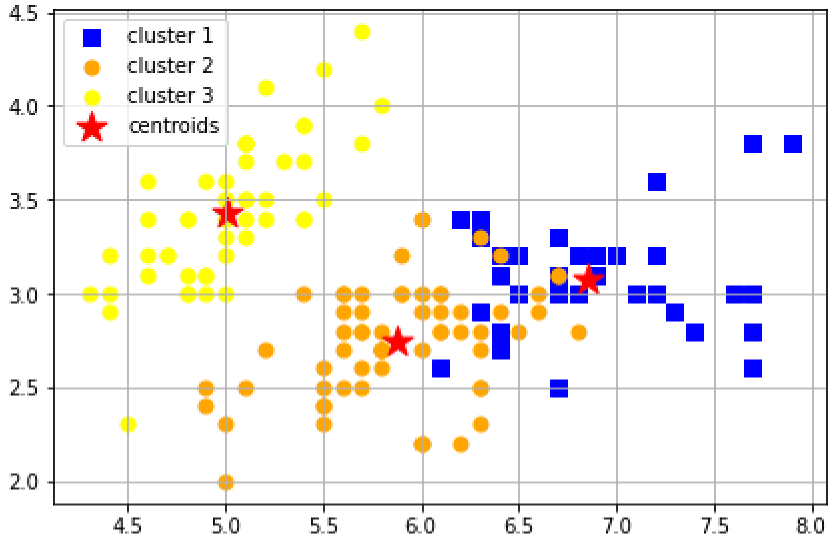
150 rows × 5 columns

In [4]:

```
km = KMeans(n_clusters=3,
            init='k-means++',
            n_init=10,
            max_iter=300,
            tol=1e-04,
            random_state=0)
y_km = km.fit_predict(X)
cluster_labels = np.unique(y_km)
n_clusters = cluster_labels.shape[0]
silhouette_vals = silhouette_samples(X, y_km, metric='euclidean')
y_ax_lower, y_ax_upper = 0, 0
yticks = []
for i, c in enumerate(cluster_labels):
    c_silhouette_vals = silhouette_vals[y_km == c]
    c_silhouette_vals.sort()
    y_ax_upper += len(c_silhouette_vals)
    color = cm.jet(float(i) / n_clusters)
    plt.barh(range(y_ax_lower, y_ax_upper), c_silhouette_vals, height=1.0,
             edgecolor='none', color=color)
    yticks.append((y_ax_lower + y_ax_upper) / 2.)
    y_ax_lower += len(c_silhouette_vals)

silhouette_avg = np.mean(silhouette_vals)
plt.axvline(silhouette_avg, color="red", linestyle="--")
plt.yticks(yticks, cluster_labels + 1)
plt.ylabel('Cluster')
plt.xlabel('Silhouette coefficient')
plt.tight_layout()
# plt.savefig('./figures/silhouette.png', dpi=300)
plt.show()
```



```
In [5]: km = KMeans(n_clusters=3,
                    init='random',
                    n_init=1,
                    max_iter=20,
                    tol=1e-04,
                    random_state=0)
        y_km = km.fit_predict(X)
        X = np.array(X)
        plt.scatter(X[y_km == 0,0],
                    X[y_km == 0,1],
                    s=50,
                    c='blue',
                    marker='s',
                    label='cluster 1')
        plt.scatter(X[y_km == 1,0],
                    X[y_km == 1,1],
                    s=50,
                    c='orange',
                    marker='o',
                    label='cluster 2')
        plt.scatter(X[y_km == 2,0],
                    X[y_km == 2,1],
                    s=50,
                    c='yellow',
                    marker='o',
                    label='cluster 3')
        plt.scatter(km.cluster_centers_[:, 0],
                    km.cluster_centers_[:, 1],
                    s=250,
                    marker='*',
                    c='red',
                    label='centroids')
        plt.legend()
        plt.grid()
        plt.tight_layout()
        #plt.savefig('./figures/centroids.png', dpi=300)
        plt.show()
```



cluster labels

```
In [ ]: y_km
```

```
Out[12]: array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
                0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0,
                0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1], dtype=int32)
```

CLUSTER VS ACTUAL LABELS

```
In [9]: y_num = [2 if label == 'Iris-setosa' else 1 if label == 'Iris-versicolor' else 0
```

In [11]:

```
from sklearn.metrics import classification_report
cr = classification_report(y_num, y_km, digits=3)
print("Classification Report\n\n", cr)
```

```
Classification Report

               precision    recall  f1-score   support

           0      1.000     0.260     0.413       150
           1      0.000     0.000     0.000         0
           2      0.000     0.000     0.000         0

    accuracy                          0.260       150
   macro avg      0.333     0.087     0.138       150
weighted avg      1.000     0.260     0.413       150


/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0
in labels with no true samples. Use `zero_division` parameter to control this b
ehavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0
in labels with no true samples. Use `zero_division` parameter to control this b
ehavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0
in labels with no true samples. Use `zero_division` parameter to control this b
ehavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

model buliding

In [6]:

```
from sklearn.linear_model import LogisticRegression
```

In [12]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y_km, test_size=0.3, rando

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
```

In [13]:

```
model = LogisticRegression()
model.fit(X_train_std, y_train)

y_pred = model.predict(X_test_std)
print('Total Examples: {}\nMisclassified examples: {}'.format((y_test.size), (y_t
```

```
Total Examples: 45
Misclassified examples: 1
```

In [16]:

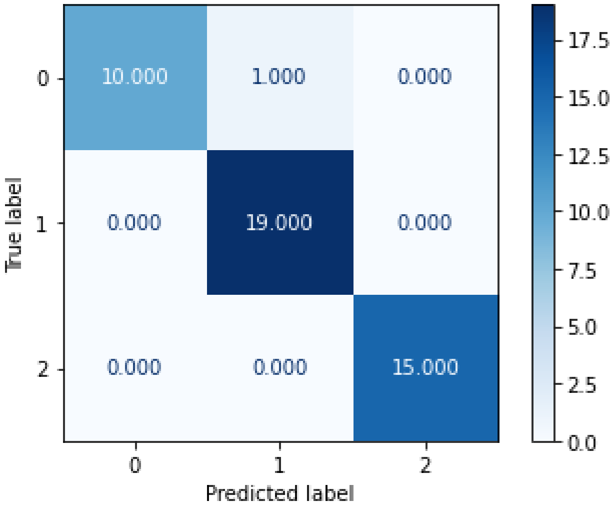
```
cr = classification_report(y_test, y_pred, digits=3)
print(cr)
```

```
              precision    recall  f1-score   support

           0      1.000     0.909     0.952        11
           1      0.950     1.000     0.974        19
           2      1.000     1.000     1.000        15

    accuracy                          0.978        45
   macro avg      0.983     0.970     0.976        45
weighted avg      0.979     0.978     0.978        45
```

In [17]:

```
cm = skm.plot_confusion_matrix(model, X_test_std, y_test, cmap='Blues', values_fo
print(cm)
```

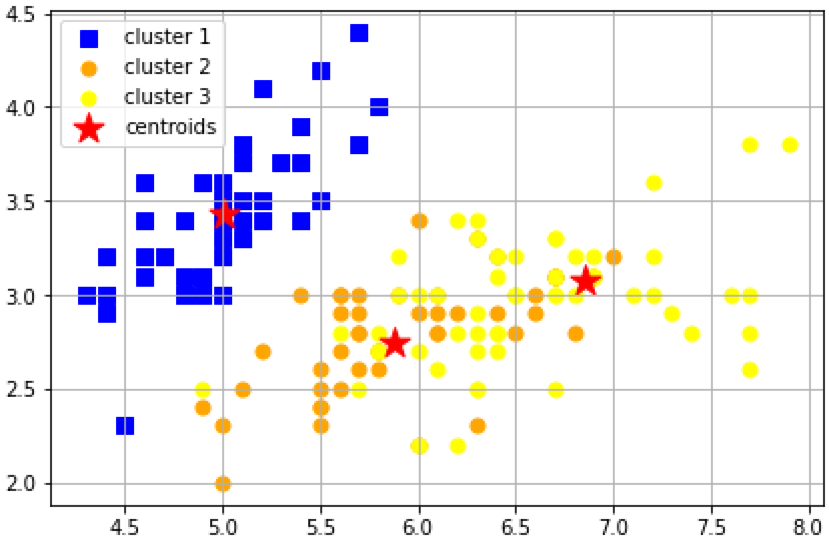
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7fa2
e2a51f50>

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureW
arning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_
matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class m
ethods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_
estimator.
  warnings.warn(msg, category=FutureWarning)
```



Gaussian mixture clustering

```
In [8]: from sklearn.mixture import GaussianMixture
```

```
In [18]: sc = GaussianMixture(n_components=3, covariance_type='full')
         preds = sc.fit_predict(X)
         X = np.array(X)
         plt.scatter(X[preds == 0,0],
                     X[preds == 0,1],
                     s=50,
                     c='blue',
                     marker='s',
                     label='cluster 1')
         plt.scatter(X[preds == 1,0],
                     X[preds == 1,1],
                     s=50,
                     c='orange',
                     marker='o',
                     label='cluster 2')
         plt.scatter(X[preds == 2,0],
                     X[preds == 2,1],
                     s=50,
                     c='yellow',
                     marker='o',
                     label='cluster 3')
         plt.scatter(km.cluster_centers_[:, 0],
                     km.cluster_centers_[:, 1],
                     s=250,
                     marker='*',
                     c='red',
                     label='centroids')
         plt.legend()
         plt.grid()
         plt.tight_layout()
         #plt.savefig('./figures/centroids.png', dpi=300)
         plt.show()
```



cluster labels to classifier

In [21]:

```
print('Cluster labels: %s' % preds)
```

```
Cluster labels: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 2 1
 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

In [22]:

```
y_num2 = [0 if label == 'Iris-setosa' else 1 if label == 'Iris-versicolor' else 2
print(y_num2[:10])
print(preds[:10])
```

```
[2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
[0 0 0 0 0 0 0 0 0 0]
```

cluster predicted vs actual labels

In [29]:

```
import sklearn.metrics as skm
cr = skm.classification_report(y_num2, preds, digits=3)
print("Classification Report\n\n", cr)
```

```
Classification Report

               precision    recall  f1-score   support

           0      0.000     0.000     0.000         0
           1      0.000     0.000     0.000         0
           2      1.000     0.367     0.537       150

    accuracy                          0.367       150
   macro avg      0.333     0.122     0.179       150
weighted avg      1.000     0.367     0.537       150


/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0
in labels with no true samples. Use `zero_division` parameter to control this b
ehavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0
in labels with no true samples. Use `zero_division` parameter to control this b
ehavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:
UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0
in labels with no true samples. Use `zero_division` parameter to control this b
ehavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [24]:

```
X_train, X_test, y_train, y_test = train_test_split(X, preds, test_size=0.3, ranc

sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

model2 = LogisticRegression()
model2.fit(X_train_std, y_train)

y_pred = model2.predict(X_test_std)
print('Total Examples: {}\nMisclassified examples: {}'.format((y_test.size), (y_t
```

```
Total Examples: 45
Misclassified examples: 1
```

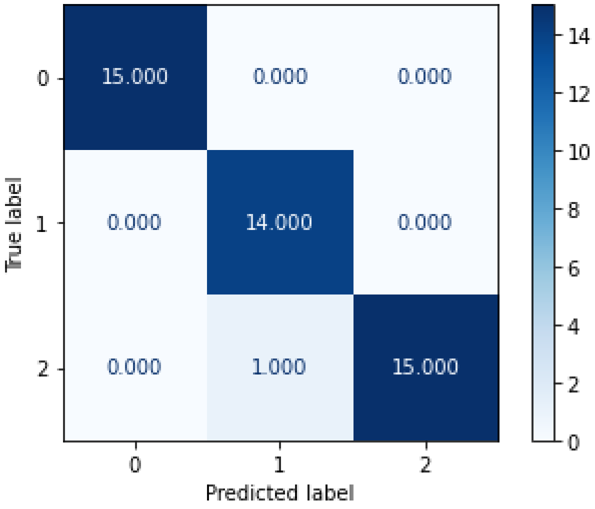
In [25]:

```
cr = classification_report(y_test, y_pred, digits=3)
print(cr)
```

```
              precision    recall  f1-score   support

           0      1.000     1.000     1.000        15
           1      0.933     1.000     0.966        14
           2      1.000     0.938     0.968        16

    accuracy                          0.978        45
   macro avg      0.978     0.979     0.978        45
weighted avg      0.979     0.978     0.978        45
```

```
In [28]: cm = skm.plot_confusion_matrix(model2, X_test_std, y_test, cmap='Blues', values_
         print(cm)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7fa2
e07e4090>

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureW
arning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_
matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class m
ethods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_
estimator.
  warnings.warn(msg, category=FutureWarning)
```


True label
Predicted label