Multi-cell-type classification on PBMC-33K dataset

In [1]:
```
%load_ext autoreload
%autoreload 2
%config Completer.use_jedi = False
```

In [2]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import umap

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import *
```

1. Load dataset

In [4]:
```
## Let's load the filtered dataset as anndata object
X = np.load('../data/pbmc_33k/33k_multi_Tcells.npy',  mmap_mode='r')
y = np.load('../data/pbmc_33k/33k_multi_Tcells_lbl.npy', mmap_mode='r')
gene_names = pd.read_csv('../data/pbmc_33k/33k_gene_idx.csv')
```

In [5]:
```
target_names = ['Tcm/Naive cytotoxic T cells', 'Tcm/Naive helper T cells',
                'Tem/Effector helper T cells', 'Tem/Trm cytotoxic T cells']
```

In [6]:
```
X.shape
```

Out[6]:
```
(8992, 32738)
```

In [7]:
```
set(y)
```

Out[7]:
```
{0, 1, 2, 3}
```

In [8]:
```
# making sure the length of the data and labels match
assert X.shape[0] == len(y)
```

2. Task

In this tutorial we will again work with PBMC-33K dataset which is now filtered to four cell types: Tcm/Naive cytotoxic T cells, Tem/Trm cytotoxic T cells, Tem/Effector helper T cells and Tcm/Naive helper T cells. List of markers for T-cells

1. Please first visualize how data looks like now compared to binary case using PCA and UMAP. Use data pre-processing steps when needed.
2. Then compare the following algorithms on how well they differentiate between four classes of T-cells. You can use reduced PCA=100 as your input to accelarate training. Please generate following plots to illustrate perfomance comparison.

   - Logistic Regression
   - KNN (k=3)
   - KNN (k=10)
   - Decision Tree

In [9]:
```
mask = np.where((np.sum(X > 0, axis = 0)))[0]
X = X[:, mask]
```

In [10]:
```
## let's check the new shape of the data
X.shape
```

Out[10]:
```
(8992, 17438)
```

In [11]:
```
gene_names = gene_names.iloc[mask]
```

In [12]:
```
## scale each gene to unit variance.

X_sc = StandardScaler().fit_transform(X)
X_sc
```

Out[12]:
```
array([[-0.0105462 , -0.02358727, -0.01826859, ..., -0.0483826 ,
        -0.16524202, -0.15694271],
       [-0.0105462 , -0.02358727, -0.01826859, ..., -0.0483826 ,
        -0.16524202, -0.15694271],
       [-0.0105462 , -0.02358727, -0.01826859, ..., -0.0483826 ,
        -0.16524202, -0.15694271],
       ...,
       [-0.0105462 , -0.02358727, -0.01826859, ..., -0.0483826 ,
        -0.16524202,  6.199944  ],
       [-0.0105462 , -0.02358727, -0.01826859, ..., -0.0483826 ,
        -0.16524202,  6.199944  ],
       [-0.0105462 , -0.02358727, -0.01826859, ..., -0.0483826 ,
        -0.16524202, -0.15694271]], dtype=float32)
```

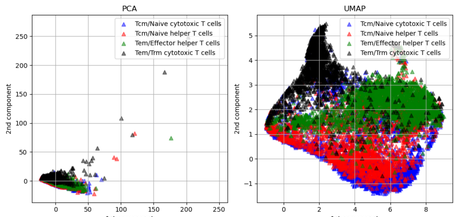
In [13]:
```
X_sc.shape
```

Out[13]:
```
(8992, 17438)
```

In [14]:
```
## Let's use PCA to reduce dimensionality even lower
X_sc_pca = PCA(n_components=100, random_state=42).fit_transform(X_sc)

X_sc_umap = (umap.UMAP()).fit_transform(X_sc_pca)
```

```
OMP: Info #276: omp_set_nested routine deprecated, please use omp_set_max_active_levels instead.
```

In [15]:
```
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))

target_classes = range(0, 4)
colors = ("blue", "red", "green", "black")
markers = ("*", "^", "^", "^")

for target_class, color, marker in zip(target_classes, colors, markers):
    ax1.scatter(
        x=X_sc_pca[y == target_class, 0],
        y=X_sc_pca[y == target_class, 1],
        color=color,
        label=f"{target_names[target_class]}",
        alpha=0.5,
        marker=marker,
    )

    ax2.scatter(
        x=X_sc_umap[y == target_class, 0],
        y=X_sc_umap[y == target_class, 1],
        color=color,
        label=f"{target_names[target_class]}",
        alpha=0.5,
        marker=marker,
    )

ax1.set_title("PCA")
ax2.set_title("UMAP")

for ax in (ax1, ax2):
    ax.set_xlabel("1st component")
    ax.set_ylabel("2nd component")
    ax.legend(loc="upper right")
    ax.grid()

_ = plt.tight_layout()
```



In [16]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(
                                X_sc_pca, y, test_size=0.30, random_state=42)
```

In [17]:
```
from sklearn.preprocessing import LabelBinarizer

label_binarizer = LabelBinarizer().fit(y_train)
y_onehot_test = label_binarizer.transform(y_test)
y_onehot_test.shape  # (n_samples, n_classes)
```

Out[17]:
```
(2698, 4)
```

In [18]:
```
# Create different classifiers.
classifiers = {
    "L2_logistic": LogisticRegression(
        penalty="l2", solver="saga", multi_class="ovr"),
    "KNN_3": KNeighborsClassifier(3),
    "KNN_10": KNeighborsClassifier(10),
    "Decision_tree": DecisionTreeClassifier(max_depth=5, random_state=42),
}
```

In [19]:
```
probas = {}
accuracies = {}
for index, (name, classifier) in enumerate(classifiers.items()):
    classifier.fit(x_train, y_train)

    y_pred = classifier.predict(x_test)
    accuracies[name] = accuracy_score(y_test, y_pred)
    # get probabilities
    probas = classifier.predict_proba(x_test)
    probas[name] = probas
```

In [20]:
```
fig, ax = plt.subplots()

bar_colors = ['tab:blue', 'tab:blue', 'tab:green', 'tab:orange']
ax.bar(accuracies.keys(), accuracies.values(), color=bar_colors)
plt.ylim((0, 1))
ax.set_ylabel('accuracy')
ax.set_title('Comparison of accuracy scores')
for i in range(len(accuracies.keys())):
    plt.annotate(str(np.round(list(accuracies.values())[i], 3)),
                 xy=(list(accuracies.keys())[i], list(accuracies.values())[i]),
                 ha='center', va='bottom')

plt.show()
```


In [22]:
```
y_onehot_test
```

Out[22]:
```
array([[0, 0, 0, 1],
       [0, 1, 0, 0],
       [0, 0, 0, 1],
       ...,
       [0, 1, 0, 0],
       [0, 0, 1, 0],
       [0, 1, 0, 0]])
```

In [24]:
```
#from itertools import cycle

#fig, ax = plt.subplots(figsize=(6, 6))
fig, axs = plt.subplots(2, 2, figsize=(8, 8))
matplotlib.rcParams.update({'font.size': 6})
row, col = 0, 0

for name, pred in  probas.items():
    RocCurveDisplay.from_predictions(
        y_onehot_test.ravel(),
        pred.ravel(),
        name="micro-average OvR",
        color="red",
        ax = axs[row][col],
        #plot_chance_level=True,
    )

    colors = ("aqua", "darkorange", "cornflowerblue", "green")
    for class_id, color in zip(range(n_classes), colors):
        RocCurveDisplay.from_predictions(
            y_onehot_test[:, class_id],
            pred[:, class_id],
            name=f"ROC curve for {target_names[class_id]}",
            color=color,
            ax = axs[row][col],
            #plot_chance_level=(class_id == 2),
        )


    axs[row][col].set_title(f"{name} AUROC ")

    row +=1
    if row%2 == 0: col += 1
```


In [ ]:
```

```