Les objectifs de ce TP

1) Faire une classification binaire à l'aide de `LogisticRegression` du module `sklearn.linear_model`
2) Séparer et normaliser les données
3) Evaluer la qualité du modèle de classification par des fonctions du module `sklearn.metrics`
4) Données non linéairement séparables
5) Données en dimension supérieure à 1
6) Classification multiple avec `LogisticRegression` et `KNeighborsClassifier` du module `sklearn.neighbors`.

Q1. Nous utilisons la librairie `sklearn.datasets` pour générer des données simulées sur lesquelles nous appliquons le modèle de classification par régression logistique. Nous utilisons, en particulier, la fonction `make_blobs` afin de générer des données pour la classification. Le code suivant vous permet de générer des données linéairement séparables.

```
from sklearn.datasets import make_blobs
x, y = make_blobs(n_samples=1000,random_state=3, centers=2)
```

Pour afficher un résumé des données :

```
from collections import Counter
print(x.shape, y.shape)
print(Counter(y))
```

Pour ploter les données

```
from matplotlib import pyplot as plt
axes=plt.axes()  #axes.grid()
plt.scatter(x[:,0],x[:,1], c=y.ravel(), s=50,cmap="RdBu", vmin=-.2,
               vmax=1.2,edgecolor="white", linewidth=1)
axes.set(aspect="equal",ylim=(-5, 5), xlabel="$X_1$", ylabel="$X_2$")
plt.show()
```

Q2. Séparer les données en deux parties (training and test) en utilisant la fonction `train_test_split` du module `sklearn.model_selection`

Q3. Le module `sklearn.preprocessing` contient des fonctions de normalisation des données. Nous considérons ici les deux types de normalisation : la standardisation par la fonction `StandardScaler` et le range scaling par la fonction `MinMaxScaler` .

Vous pouvez faire le data splitting avant la normalisation ou l'inverse.

```
#Standardization
from sklearn import preprocessing
scaler = preprocessing.StandardScaler().fit(x_train)
x_scaled_train = scaler.transform(x_train)
x_scaled_test = scaler.transform(x_test)

#Scaling features to a range
min_max_scaler = preprocessing.MinMaxScaler()
x_minmax = min_max_scaler.fit_transform(x)
```

Q4. La méthode de régression logistique est implémentée dans le module **`sklearn.linear_model`** sous le nom **`LogisticRegression`**.

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(x_train, y_train)  # apprentissage
y_train_pred = model.predict(x_train) # prediction de training
y_test_pred = model.predict(x_test) # prediction de test
```

Q5. Il existe plusieurs critères pour évaluer la qualité d'une classification. En voici quelques-uns :

1) **`accuracy`**

```
from sklearn import metrics
accuracy = metrics.accuracy_score(y_test, y_test_pred)
accuracy_percentage = 100 * accuracy
print("pourcentage des bien classes ",accuracy_percentage)
```

2) **`classification_report`**

```
print(metrics.classification_report(y_test, y_test_pred))
```

3) **`cohen_kappa_score`**

```
print(metrics.cohen_kappa_score(y_test, y_test_pred))
```

4) **`confusion_matrix`**

```
confmat = metrics.confusion_matrix(y_test, y_test_pred)
print("Matrice de confusion ",confmat)
```

5) **`Specificity, Sensibility, precision`**

```
tn, fp, fn, tp = confusion_matrix(y_test, y_test_pred).ravel()
specificity = tn / (tn+fp)
#completer ici
```

6) **roc_auc_score**

```
probas = model.predict_proba(x_test)
print('auc=', metrics.roc_auc_score(y_test, probas[:,1]))
fpr0, tpr0, _ = metrics.roc_curve(y_test, probas[:, 1])
fig, ax = plt.subplots(1, 1, figsize=(4,4))
ax.plot([0, 1], [0, 1], 'k--')
ax.plot(fpr0, tpr0, label=model.classes_[0])
ax.set_title('Courbe ROC - classifieur')
ax.text(0.5, 0.3, "plus mauvais que\nle hasard dans\ncette zone")
ax.set_xlabel("FPR")
ax.set_ylabel("TPR");
ax.legend();
```

Q6. Les données que nous considérons dans cette question sont non linéairement séparable. Tourner votre code avec ces deux ensembles de données. Qu'est-ce que vous constatez ? Proposer une solution pour améliorer la qualité de la classification (utiliser la fonction `PolynomialFeatures`).

La fonction **`PolynomialFeatures`** du module **`sklearn.preprocessing`** génère une nouvelle matrice de features composée de toutes les combinaisons polynômiales des features de degré inférieur ou égale au degré indiqué. Pour degree=2 avec deux features x1 et x2, cette fonction renvoie [1, x1, x2, x1x2, $x1^2$, $x2^2$].

```
from sklearn.preprocessing import PolynomialFeatures
polynomial_features= PolynomialFeatures(degree=1) # polynomial degree
X = polynomial_features.fit_transform(x)
```

```
#Ensemble 1
from sklearn.datasets import make_gaussian_quantiles
x, y = make_gaussian_quantiles(n_samples=500, n_classes=2)
```

```
#Ensemble 2
from sklearn.datasets import make_moons
x,y = make_moons(n_samples=500,noise=0.3,random_state=0)
```

Q7. Nous considérons, dans cette question, des données avec un nombre de features supérieur à 2. Tourner votre code avec ces nouvelles données. Qu'est-ce que vous constatez ? Modifier le code pour qu'il soit adapté à ces données.

```
#Ensemble 3
from sklearn.datasets import make_classification
x, y = make_classification(n_samples=1000, n_features=5,
         n_informative=2, n_redundant=2, n_classes=2, random_state=4)
```

Q8. Utiliser la méthode de régularisation afin d'améliorer votre classification.

```
#L2 Regularization, beta=1/C
model = LogisticRegression(penalty='l2', C=10)
```

Q9. Nous considérons l'ensemble des données IRIS multiclasses, sur lequel nous appliquons les deux modèles : `LogisticRegression` et `KNeighborsClassifier`. Comparer les performances de ces deux méthodes.

```
from sklearn.datasets import load_iris
x, y = load_iris( return_X_y=True)
model = LogisticRegression(multi_class='ovr')
```

```
# k-plus-proches voisins
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=3)
```

Q10. Programmer la méthode de descente du gradient du modèle de régression logistique. Quel est l'effet du learning rate sur la convergence ?

Q11. Programmer la méthode des k-plus-proches voisins.