SY09 Printemps 2022
TP 08 — Éléments de théorie bayésienne de la décision

Dans ce TP, on souhaite étudier les stratégies de Neyman-Pearson et de Bayes pour résoudre un problème de décision, dans le cas où les distributions conditionnelles voire les probabilités a priori sont connues.

1 Partie théorique

1.1 Problème et modélisation

On considérera un problème très simplifié d'identification de produits chimiques à partir de leur temps de dégradation. Plus particulièrement, on supposera être en présence de $g = 2$ produits, présents en proportions initiales π_1 (classe ω_1) et π_2 (classe ω_2).

On suppose pouvoir tester le temps de dégradation des produits selon deux protocoles ; on notera X^1 et X^2 les temps de dégradation selon le premier et le second protocole, respectivement. Pour un même produit, *on supposera indépendants ces temps X^1 et X^2 mesurés par chacun des deux protocoles*. On suppose en outre que les distributions de temps de dégradation sont modélisés par des lois exponentielles :

$$X^1 \underset{\omega_1}{\sim} \mathcal{E}(\lambda_1), \quad X^2 \underset{\omega_1}{\sim} \mathcal{E}(\lambda_2);$$
$$X^1 \underset{\omega_2}{\sim} \mathcal{E}(\theta_1), \quad X^2 \underset{\omega_2}{\sim} \mathcal{E}(\theta_2).$$

1.2 Questions préliminaires

1 Quelle est la densité jointe du vecteur aléatoire $\boldsymbol{X} = (X^1, X^2)^T$ dans chacune des classes ?

En utilisant l'indépendance conditionnelle, et en notant f_{kj} la densité conditionnelle de X^j dans la classe ω_k et f_k la densité jointe de $\boldsymbol{X}$ dans la classe ω_k, on a

$$f_1(\boldsymbol{x}) = f_{k1}(x_1) f_{k2}(x_2) = \lambda_1 \lambda_2 \exp(-\lambda_1 x_1 - \lambda_2 x_2) = \lambda_1 \lambda_2 \exp(-\lambda^T \boldsymbol{x}),$$

avec $\lambda = (\lambda_1, \lambda_2)^T$. De même, on a

$$f_2(\boldsymbol{x}) = \theta_1 \theta_2 \exp(-\theta^T \boldsymbol{x}),$$

avec $\theta = (\theta_1, \theta_2)^T$.

2 Montrer que la frontière de décision obtenue en appliquant la stratégie de Neyman-Pearson est une droite dont on précisera les paramètres.

On a

$$\delta(\boldsymbol{x}) = \omega_1 \Leftrightarrow \frac{f_1(\boldsymbol{x})}{f_2(\boldsymbol{x})} \geq s,$$

avec

$$\Pr\left(\frac{f_1(\boldsymbol{X})}{f_2(\boldsymbol{X})} < s | Z = \omega_1\right) = \alpha^*.$$

La frontière de décision correspond donc bien à l'équation d'une droite :

$$\frac{f_1(\boldsymbol{x})}{f_2(\boldsymbol{x})} < s \Leftrightarrow (\theta - \lambda)^T \boldsymbol{x} - \ln\left(\frac{\theta_1\theta_2}{\lambda_1\lambda_2}s\right) < 0.$$

3 Quelle frontière de décision obtient-on si l'on applique la stratégie de Bayes ?

On a

$$\delta(\boldsymbol{x}) = \omega_1 \Leftrightarrow \frac{f_1(\boldsymbol{x})}{f_2(\boldsymbol{x})} \geq \frac{\pi_2}{\pi_1} \Leftrightarrow (\theta - \lambda)^T \boldsymbol{x} - \ln\left(\frac{\theta_1\theta_2\pi_2}{\lambda_1\lambda_2\pi_1}\right) \geq 0.$$

4 Indépendamment des hypothèses de distribution formulées dans cet exercice, justifier rigoureusement l'estimation de la probabilité d'erreur de Bayes $\epsilon^\star$ par la moyenne empirique des erreurs commises par la règle de Bayes $\delta^\star$ sur un échantillon donné.

On commencera par rappeler la définition du taux d'erreur de Bayes donnée en cours :

$$\varepsilon^\star = \varepsilon(\delta^\star) = \int_{\mathcal{X}} \varepsilon(\delta^\star|\boldsymbol{x}) f(\boldsymbol{x}) d\boldsymbol{x} = \mathbb{E}_{\boldsymbol{X}}\left[\varepsilon(\delta^\star|\boldsymbol{x})\right],$$

où l'erreur conditionnelle $\varepsilon(\delta^\star|\boldsymbol{x})$ est elle-même définie comme l'espérance d'erreur (espérance calculée par rapport à la distribution de la variable de classe Z) :

$$\varepsilon(\delta^\star|\boldsymbol{x}) = \Pr(\delta^\star(\boldsymbol{X}) \not\equiv Z|\boldsymbol{X} = \boldsymbol{x}) = \mathbb{E}_Z\left[\mathbb{1}_{[\delta^\star(\boldsymbol{x})\not\equiv Z]}\right],$$

avec $\mathbb{1}_{[\delta^\star(\boldsymbol{X})\not\equiv Z]}$ la variable indicatrice d'une erreur conditionnelle à l'événement $\boldsymbol{X} = \boldsymbol{x}$. Le taux d'erreur de Bayes est donc l'espérance, calculée par rapport à la distribution jointe du couple $(\boldsymbol{X}, Z)$, de l'indicatrice d'erreur :

$$\varepsilon(\delta^\star) = \mathbb{E}_{\boldsymbol{X},Z}\left[\mathbb{1}_{[\delta^\star(\boldsymbol{X})\not\equiv Z]}\right].$$

Soulignons que cette dernière formule peut également être retrouvée à partir de la notion de risque d'une règle de décision, définie en cours comme l'espérance du coût des décisions prises par cette règle :

$$r(\delta^\star) = \mathbb{E}_{\boldsymbol{X},Z}\left[c(\delta^\star(\boldsymbol{X})|Z)\right];$$

on retombe bien sur l'expression précédente en considérant des coûts 0/1.

Pour estimer cette espérance, on peut donc utiliser un échantillon généré suivant la distribution jointe $\Pr(\boldsymbol{X}, Z)$, sur lequel on calculera la moyenne des erreurs de classement par la règle de Bayes $\delta^\star$. La propriété de convergence de la moyenne empirique (loi faible des grands nombres) permet d'obtenir, pour un échantillon suffisamment grand, une estimation raisonnablement proche du taux (théorique) d'erreur de Bayes $\varepsilon(\delta^\star)$.

5 Toujours indépendamment des hypothèses de distribution formulées dans cet exercice, on veut en déduire l'expression de l'erreur de Bayes pour les problèmes de classification binaires ($g = 2$).

5 a Montrer que l'erreur théorique de tout classifieur binaire peut s'écrire

$$\varepsilon(\delta) = \int_{\mathcal{R}_2} \pi_1 f_1(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} + \int_{\mathcal{R}_1} \pi_2 f_2(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}.$$

Rappelons que l'on a

$$\varepsilon(\delta) = \mathbb{E}_{\boldsymbol{X},Z}\left[\mathbb{1}_{\delta(\boldsymbol{X})\not\equiv Z}\right] = \mathbb{E}_{\boldsymbol{X}}\left[\varepsilon(\delta|\boldsymbol{x})\right], \quad \text{avec} \quad \varepsilon(\delta|\boldsymbol{x}) = \mathbb{E}_Z\left[\mathbb{1}_{[\delta(\boldsymbol{x})\not\equiv Z]}\right].$$

Dans le cas de deux classes, le calcul de l'espérance d'erreur donne :

$$\begin{aligned}\varepsilon(\delta|\boldsymbol{x}) &= \sum_z \mathbb{1}_{\delta(\boldsymbol{X})\neq Z}\mathbb{P}(Z=z|\boldsymbol{X}=\boldsymbol{x}),\\ &= \mathbb{1}_{\delta(\boldsymbol{x})\neq 1}\mathbb{P}(Z=1|\boldsymbol{X}=\boldsymbol{x}) + \mathbb{1}_{\delta(\boldsymbol{x})\neq 0}\mathbb{P}(Z=0|\boldsymbol{X}=\boldsymbol{x}).\end{aligned}$$

On utilise cette expression dans le calcul de l'erreur $\varepsilon(\delta)$:

$$\begin{aligned}\varepsilon(\delta) = \mathbb{E}_{\boldsymbol{X}}\left[\varepsilon(\delta|\boldsymbol{x})\right] &= \int_{\mathcal{X}} \varepsilon(\delta|\boldsymbol{x}) f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x},\\ &= \int_{\mathcal{X}} \mathbb{1}_{\delta(\boldsymbol{x})\neq 1}\mathbb{P}(Z=1|\boldsymbol{X}=\boldsymbol{x}) f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} + \int_{\mathcal{X}} \mathbb{1}_{\delta(\boldsymbol{x})\neq 0}\mathbb{P}(Z=0|\boldsymbol{X}=\boldsymbol{x}) f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x},\\ &= \int_{\mathcal{R}_2} \pi_1 f_1(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} + \int_{\mathcal{R}_1} \pi_2 f_2(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x},\end{aligned}$$

5 b Montrer que dans le cas du classifieur de Bayes, cette erreur s'écrit

$$\varepsilon(\delta^\star) = \int_{\mathcal{X}} \min\left(\pi_1 f_1(\boldsymbol{x}), \pi_2 f_2(\boldsymbol{x})\right)\mathrm{d}\boldsymbol{x},$$

avec f_1 et f_2 les densités conditionnelles et π_1 et π_2 les proportions des classes.

Dans le cas du classifieur de Bayes, les régions de décision $\mathcal{R}_1$ et $\mathcal{R}_2$ sont formellement définies par

$$\begin{aligned}\mathcal{R}_1 &= \Big\{\boldsymbol{x} : \delta^\star(\boldsymbol{x}) \neq 0\Big\} = \Big\{\boldsymbol{x} : \pi_1 f_1(\boldsymbol{x}) \geq \pi_2 f_2(\boldsymbol{x})\Big\},\\ \mathcal{R}_2 &= \Big\{\boldsymbol{x} : \delta^\star(\boldsymbol{x}) \neq 1\Big\} = \Big\{\boldsymbol{x} : \pi_1 f_1(\boldsymbol{x}) < \pi_2 f_2(\boldsymbol{x})\Big\}.\end{aligned}$$

En regroupant les deux intégrales, on obtient donc finalement

$$\varepsilon(\delta^\star) = \int_{\mathcal{X}} \min\left(\pi_1 f_1(\boldsymbol{x}), \pi_2 f_2(\boldsymbol{x})\right)\mathrm{d}\boldsymbol{x}.$$

2 Partie pratique

2.1 Protocole de simulation

On cherche à générer un échantillon de taille n suivant le modèle génératif décrit ci-dessus. Intuitivement, pour chaque nouvel individu, il faudrait (1) déterminer sa classe puis (2) générer un vecteur aléatoire (composante par composante, les variables X^1 et X^2 étant indépendantes conditionnellement à la classe) suivant les paramètres correspondants.

On propose donc le protocole de simulation suivant :

1. calculer le nombre de points n_1 présents dans la classe ω_1 : on verra n_1 comme la réalisation d'une v.a. $N_1 \sim \mathcal{B}(n, \pi_1)$;
2. générer n_1 vecteurs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n_1}$ en concaténant n_1 réalisations de variables aléatoires $X_1^1 \sim \mathcal{E}(\lambda_1)$ et $X_1^2 \sim \mathcal{E}(\lambda_2)$;
3. générer $n_2 = n - n_1$ vecteurs $\boldsymbol{x}_{n_1+1}, \ldots, \boldsymbol{x}_n$ en concaténant n_2 réalisations de variables aléatoires $X_2^1 \sim \mathcal{E}(\theta_1)$ et $X_2^2 \sim \mathcal{E}(\theta_2)$.

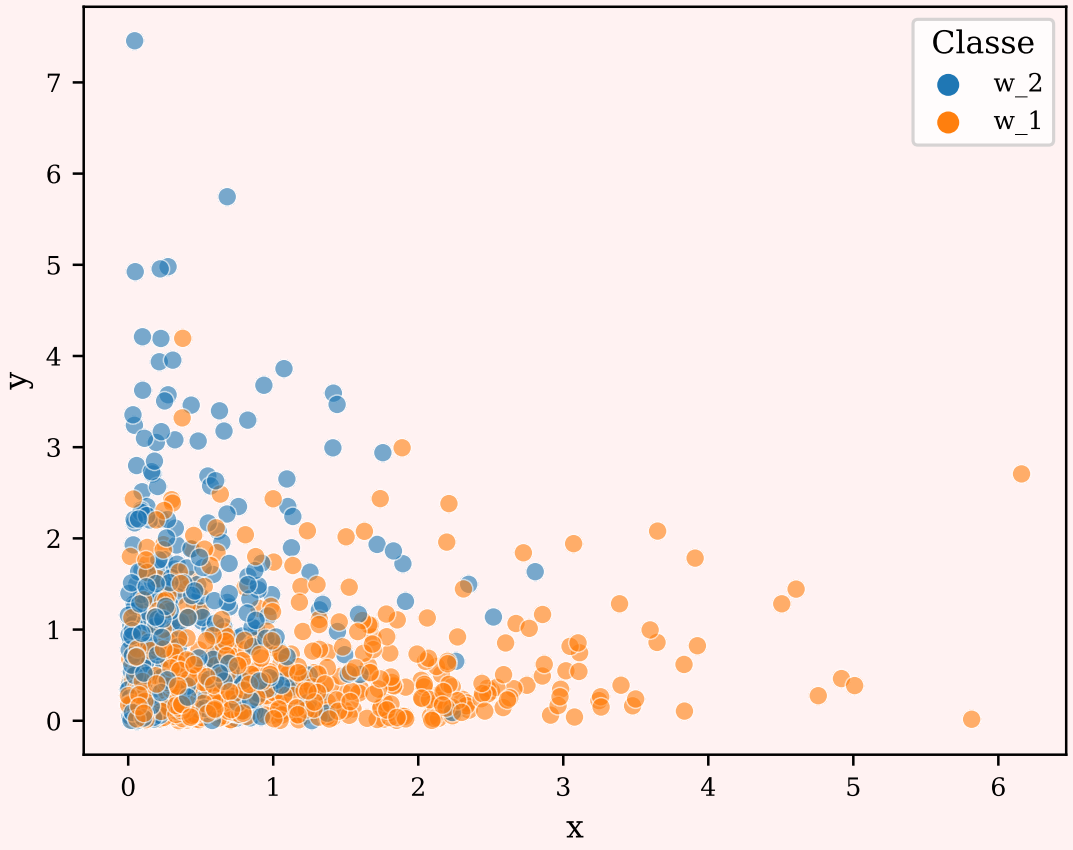
Implémentation

6 Implémenter ce protocole de simulation, et générer un échantillon et ses étiquettes de taille $n = 1000$, en utilisant $\pi_1 = 0.6$, $\lambda_1 = 1$, $\lambda_2 = 2$, $\theta_1 = 2$, $\theta_2 = 1$.

```
In [1]: rng = np.random.default_rng()

        pi1 = 0.6
        pi2 = 1 - pi1
        lambda1 = 1
        lambda2 = 2
        theta1 = 2
        theta2 = 1

        n = 1000
        n1 = rng.binomial(n, pi1, 1)[0]
        n2 = n - n1

        X1 = np.column_stack(
            (
                rng.exponential(scale=1/lambda1, size=n1),
                rng.exponential(scale=1/lambda2, size=n1),
            )
        )
        X2 = np.column_stack(
            (
                rng.exponential(scale=1/theta1, size=n2),
                rng.exponential(scale=1/theta2, size=n2),
            )
        )
        X = np.concatenate((X1, X2), axis=0)

        z = np.repeat(["w_1", "w_2"], [n1, n2])

        from sklearn.utils import shuffle
        X, z = shuffle(X, z)
```

7 Afficher le jeu de données avec l'information de classe. Quel serait un bon séparateur linéaire ?

```
In [2]: data = pd.DataFrame(dict(x=X[:, 0], y=X[:, 1], Classe=z))

        sns.scatterplot(x="x", y="y", hue="Classe", data=data, alpha=.6)
        plt.show()
```



La droite $y = x$ semble être un bon séparateur linéaire.

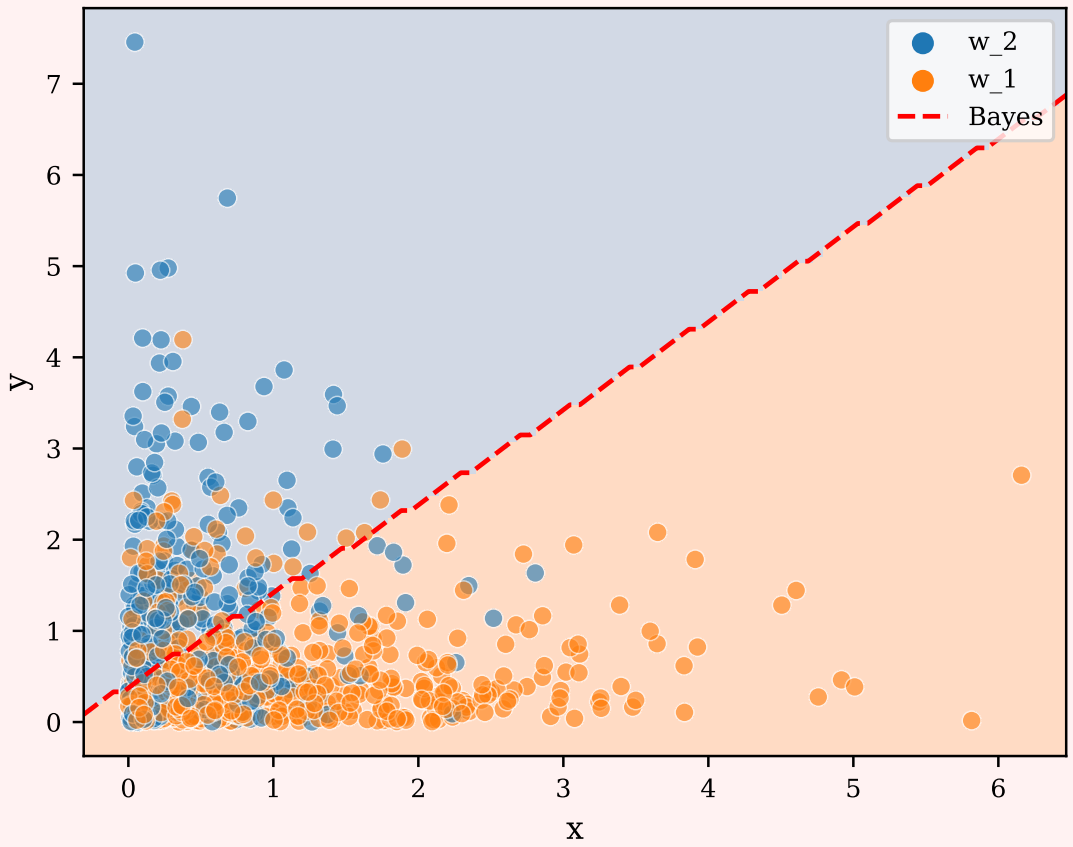
8 Estimer le taux d'erreur de Bayes au moyen de cet échantillon. On pourra utiliser la fonction `np.apply_along_axis`.

```
In [3]: def delta(row):
            x, y = row
            cste = np.log(theta1 * theta2 * pi2 / lambda1 / lambda2 / pi1)
            d = (theta1 - lambda1) * x + (theta2 - lambda2) * y - cste
            return "w_1" if d >= 0 else "w_2"

        pred = np.apply_along_axis(delta, 1, X)

        taux_Bayes = np.mean(pred != z)
        taux_Bayes
Out [3]: 0.294
```

9 Tracer la frontière de Bayes correspondante. On utilisera la fonction `add_decision_boundary` du TP07 en lui donnant directement une fonction de prédiction (et pas un objet `scikit-learn`) et en précisant l'argument `model_classes`.

```
def predict(X):
    return ...
```

```
In [4]: def predict(X):
            return np.apply_along_axis(delta, 1, X)

        sns.scatterplot(x="x", y="y", hue="Classe", data=data, alpha=.6)
        add_decision_boundary(predict, label="Bayes", model_classes=["w_1",
        ↪  "w_2"])
        plt.show()
```



10 En utilisant le protocole d'évaluation des performances vu précédemment, appliquer la stratégie des K plus proches voisins à ce problème de décision. Comparer les performances obtenues au taux d'erreur de Bayes.

In [5]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score

X_train, X_test, z_train, z_test = train_test_split(X, z,
↪  test_size=0.33)

n_neighbors_list = np.unique(np.round(np.geomspace(1, 500,
↪  100)).astype(int))
param_grid = {"n_neighbors": n_neighbors_list}

cls = KNeighborsClassifier()
search = GridSearchCV(cls, param_grid, scoring="accuracy", cv=10)
search.fit(X_train, z_train)
search.best_params_
```

Out [5]:
```
{'n_neighbors': 81}
```

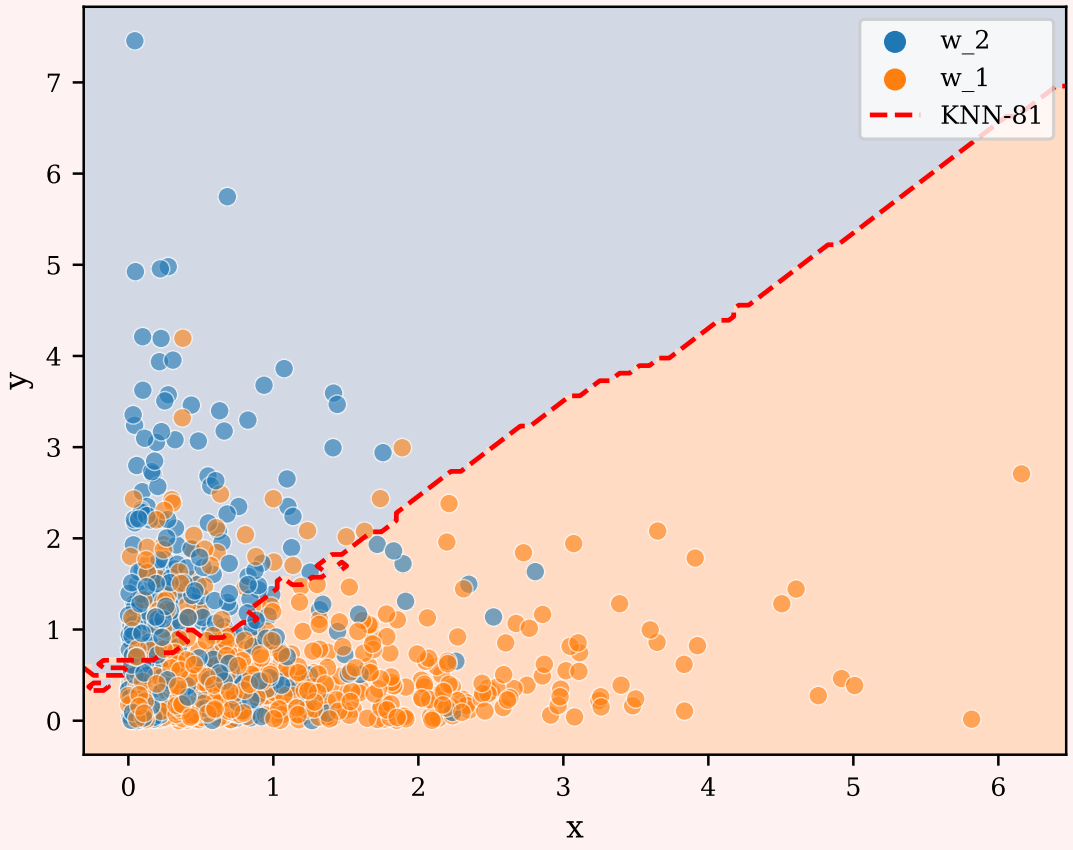
In [6]:
```
z_pred = search.predict(X_test)
miscls_KNN = 1 - accuracy_score(z_pred, z_test)
miscls_KNN
```

Out [6]:
```
0.27575757575757576
```

In [7]:
```
z_pred = search.predict(X_train)
miscls_KNN_train = 1 - accuracy_score(z_train, z_pred)
miscls_KNN_train
```

Out [7]:
```
0.29850746268656714
```

In [8]:
```
sns.scatterplot(x="x", y="y", hue="Classe", data=data, alpha=.6)
add_decision_boundary(search,
↪  label=f"KNN-{search.best_params_['n_neighbors']}")
plt.show()
```



On constate que le taux d'erreur obtenu avec les K plus proches voisins est proche du taux d'erreur de Bayes estimé précédemment. En revanche, l'erreur d'apprentissage est plus faible mais c'est une estimation biaisée, optimiste, de l'erreur théorique du classifieur.

Calcul numérique de l'erreur de Bayes.

11 Créer une function en Python prenant en argument une abscisse et une ordonnée et renvoyant la fonction sous le signe intégrale.

On pourra utiliser la fonction `expon` définie dans le module `scipy.stats` pour calculer la densité d'une loi exponentielle.

```
In [9]: from scipy.stats import expon

        pi1 = 0.6
        pi2 = 1 - pi1
        lambda1 = 1
        lambda2 = 2
        theta1 = 2
        theta2 = 1

        def err_function(x, y):
            f1x = expon(scale=1/lambda1).pdf(x)
            f1y = expon(scale=1/lambda2).pdf(y)
            f2x = expon(scale=1/theta1).pdf(x)
            f2y = expon(scale=1/theta2).pdf(y)

            return min(pi1 * f1x * f1y, pi2 * f2x * f2y)
```

12 En utilisant la fonction `dblquad` d'intégration numérique d'une intégrale double, donner un encadrement de l'erreur de Bayes à 10^{-3} près.

```
from scipy.integrate import dblquad
```

```
In [10]: from scipy.integrate import dblquad
         dblquad(err_function, 0, np.inf, 0, np.inf, epsabs=1e-3)
Out [10]: (0.31110590440142405, 0.0009941577509708348)
```

Calcul numérique de l'erreur d'un classifieur

Dans cette partie, on se propose de calculer numériquement l'erreur théorique d'un classifieur binaire δ quelconque. D'après la question 5, on sait que l'erreur de δ peut s'écrire

$$\varepsilon(\delta) = \int_{\mathcal{R}_1} \pi_2 f_2(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} + \int_{\mathcal{R}_2} \pi_1 f_1(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}.$$

En d'autres termes, il suffit d'identifier les deux régions de décision complémentaires de δ, en utilisant le mécanisme de classement implémenté par le classifieur ; puis d'intégrer la bonne densité conditionnelle sur chacune des régions pour calculer son erreur théorique.

Nous allons utiliser la méthode de Simpson pour calculer numériquement ces intégrales. Pour cela, il faut commencer par discrétiser l'espace sur lequel on va intégrer. Une discrétisation uniforme ne présentant guère de sens pour notre problème, on choisit une discrétisation basée sur des fractiles exponentiels.

13 À l'aide de `np.linspace`, générer une discrétisation uniforme des ordres de fractiles entre 0 et 1 exclu.

```
In [11]: alpha = np.linspace(0, 1, 500, endpoint=False)
```

14 À l'aide de la méthode `ppf` calculant les fractiles, en déduire une discrétisation exponentielle des abcisses et des ordonnées.

```
In [12]: rv = expon(scale=1/lambda1)
         x = rv.ppf(alpha)
         y = x
```

Il faut à présent calculer en chacun des points repérés par leur abscisse et leur ordonnée les deux densités possibles $f_1(\boldsymbol{x})$ et $f_2(\boldsymbol{x})$.

15 Créer un tableau bidimensionnel des densités pour f_1. Faire de même pour f_2.

```
In  [13]: f1x = expon(scale=1/lambda1)
          f1y = expon(scale=1/lambda2)
          Y1, X1 = np.meshgrid(f1y.pdf(y), f1x.pdf(x))
          f1 = pi1 * Y1 * X1

          f2x = expon(scale=1/theta1)
          f2y = expon(scale=1/theta2)
          Y2, X2 = np.meshgrid(f2y.pdf(y), f2x.pdf(x))
          f2 = pi2 * Y2 * X2
```

Il faut maintenant identifier les deux régions de décision et prédisant la classe pour chaque point de la discrétisation bidimensionnelle.

16 Créer un tableau bidimensionnel donnant la classe pour chaque point de la discrétisation bidimensionnelle.

```
In  [14]: Y, X = np.meshgrid(y, x)
          xy = np.column_stack((X.ravel(), Y.ravel()))
          Z = search.predict(xy).reshape(X.shape)
```

17 Créer un tableau bidimensionnel calculant la valeur de la fonction à intégrer en utilisant la prédiction de la classe.

On pourra utiliser la fonction `np.where`.

```
In  [15]: err_KNN = np.where(Z == "w_1", f2, f1)
          err_bayes = np.minimum(f1, f2)
```

18 Intégrer numériquement la fonction obtenue. On utilisera deux intégrations successives avec la méthode de Simpson.

```
from scipy.integrate import simps
simps(simps(<tableau bidim à intégrer>, <discrétisation des y>),
↪  <discrétisation des x>)
```

```
In  [16]: from scipy.integrate import simps
          simps(simps(err_bayes, y), x)
Out [16]: 0.31110739225169803
In  [17]: simps(simps(err_KNN, y), x)
Out [17]: 0.3125809850243686
```