Détection automatique des fake news à partir de donnees textuelles (Fake News Detection)

Groupe 5

BELOT Mathieu : 21904568 \ CANHOTO Mickael : 22304810\ DEURVEILHER Jean Louis : 22102662\ FONTAINE Emmanuel : 22312091\ NGUYEN Thi-Christine : 21704571

Ce projet s'inscrit dans le contexte de l'apprentissage supervisé, i.e. les données possèdent des labels. Il vise à trouver les modèles les plus performants pour prédire si des articles de presse sont vrais ou faux. Les articles contiennent des assertions (une assertion est une proposition que l'on avance et que l'on soutient comme vraie) faites, par exemple, par des hommes politiques. \ Nous allons pour cela réaliser une suite de traitement afin d'obtenir une classification satisfaisante

---

Table des matières

Installation

Tout d'abord, nous allons télécharger les librairies nécessaires pour notre futurs traitement et récupérer les doneés sur lequel nous allons travailler.

In [ ]:
```
# Installation des librairies
!pip install pandas numpy scikit-learn nltk matplotlib imblearn contractions
```

In [ ]:
```
# Importation des librairies
import warnings # enlevé les warnings
warnings.filterwarnings("ignore", category=FutureWarning)

# Manipulation dataset
import pandas as pd # Lecture Dataset
import sys # Récupération dataset
import numpy as np # Array
import matplotlib.pyplot as plt
import seaborn as sns
from statistics import median

# Stopwords
from nltk.corpus import stopwords
import re # Regular expression
import nltk
import contractions
import inflect
import difflib

# SKLearn, classification
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split, cross_val_score, KFold, cross
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.decomposition import LatentDirichletAllocation
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE

## Naive Bayes classifieur
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import ComplementNB
```

Nous récupérons ensuite la base de données

In [ ]:
```
# Ajout du google drive
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

In [ ]:
```
# Chemin de la base de donneés
my_local_drive='/content/gdrive/My Drive/dataset/'
sys.path.append(my_local_drive)
%cd $my_local_drive

%pwd
```

In [ ]:
```
# Importation de la base de donneés
df=pd.read_csv('HAI817_Projet_train.csv', sep=',')
display (df.head())
```

	public_id	text	title	our rating
0	5a228e0e	Distracted driving causes more deaths in Canad...	You Can Be Fined $1,500 If Your Passenger Is U...	false
1	30c605a1	Missouri politicians have made statements afte...	Missouri lawmakers condemn Las Vegas shooting	mixture
2	c3dea290	Home Alone 2: Lost in New York is full of viol...	CBC Cuts Donald Trump's 'Home Alone 2' Cameo O...	mixture
3	f14e8eb6	But things took a turn for the worse when riot...	Obama's Daughters Caught on Camera Burning US ...	false
4	faf024d6	It's no secret that Epstein and Schiff share a...	Leaked Visitor Logs Reveal Schiff's 78 Visits ...	false

---

Ingénierie des donneés

Le pré-traitement des données consiste à nettoyer, normaliser et transformer les données brutes afin de les préparer de manière optimale pour l'analyse ou l'apprentissage automatique.

Stopwords

ils permettent de filtrer les mots les plus courants et peu informatifs, ce qui réduit la dimensionnalité des données et améliore l'efficacité des algorithmes de traitement de texte.

In [ ]:
```
nltk.download('averaged_perceptron_tagger')
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('omw-1.4')

stop_words = set(stopwords.words('english'))
```

In [ ]:

```
# Fonction de clean du text paramètrable
def clean_text(text,
               lowercase=False,
               removestopwords=False,
               removedigits=False,
               digits2str=False,
               rooting=False,
               lemmatisation=False,
               tagging=False,
               replace_contractions=False,
               remove_punctuation=False
               ):
  X=str(text)

  # contractions
  if replace_contractions:
    X = contractions.fix(X)


  # espaces entre les chiffres
  X = re.sub(r'(\d)\s+(\d)', r'\1\2', X)
  # les caractères uniques
  X = re.sub(r'\s+[a-zA-Z]\s+', ' ', X)
  # espaces multiples en un seul espace
  X = re.sub(r'\s+', ' ', X, flags=re.I)

  if remove_punctuation:
    X = re.sub(r'[^\w\s]',' ', X)

  tokens = nltk.word_tokenize(X)
  # lowercase
  if lowercase:
    tokens = [token.lower() for token in tokens]

  # punctuation
  if remove_punctuation:
      table = str.maketrans('', '', string.punctuation)
      words = [token.translate(table) for token in tokens]
  else:
    words = [token for token in tokens]

  # non alphanum
  words = [word for word in words if word.isalnum()]

  # chiffres
  if removedigits:
    words = [word for word in words if not word.isdigit()]
  elif digits2str:
      words = [ inflect.engine().number_to_words(word) if word.isdigit() else word

  # suppression des stopwords
  if removestopwords:
    words = [word for word in words if not word in stop_words]

  # lemmatisation
  if lemmatisation:
    lemmatizer=nltk.stem.WordNetLemmatizer()
    words = [lemmatizer.lemmatize(word)for word in words]
```

```
    # racinisation
    if rooting:
      ps = nltk.stem.PorterStemmer()
      words=[ps.stem(word) for word in words]

    if tagging:
      words = nltk.pos_tag(words)
      return words

    X = ' '.join(words)

    return X
```

In [ ]:
```
normalized = df.copy()
normalized['text'] = normalized['text'].apply(lambda text: clean_text(text, True, T
```

Comparatif avant et après traitement

In [ ]:
```
print("Texte original")
display(df['text'].head())
print("\nTexte post-traitement")
display(normalized['text'].head())
```

```
Texte original
0    Distracted driving causes more deaths in Canad...
1    Missouri politicians have made statements afte...
2    Home Alone 2: Lost in New York is full of viol...
3    But things took a turn for the worse when riot...
4    It’s no secret that Epstein and Schiff share a...
Name: text, dtype: object
Texte post-traitement
0    distracted driving cause death canada impaired...
1    missouri politician made statement mass shooti...
2    home alone lost new york full violence opinion...
3    thing took turn worse riot police fired tear g...
4    secret epstein schiff share long history perve...
Name: text, dtype: object
```

Vectorisation

La vectorisation implique la conversion de données textuelles en représentations numériques, afin de permettre aux algorithmes d'apprentissage automatique de comprendre et de traiter le langage humain.

In [ ]:
```
# Configuration du vecteur TF-IDF avec des n-grammes
ngram_range = (1, 2)  # 2-grammes
vectorizer = TfidfVectorizer(ngram_range=ngram_range, min_df=0.005, max_df=0.9)

# vectorizer = TfidfVectorizer()
normalizedText = normalized['text'].copy()

# Ajustement TF-IDF
vectorizedText = vectorizer.fit_transform(normalizedText)

scaler = StandardScaler()
scaled = scaler.fit_transform(vectorizedText.toarray())

vectorized = pd.DataFrame(data=scaled, columns = vectorizer.get_feature_names_out()

display(vectorized)
```

	11th	13th	17th	18th	1930s	1960s	1970s	1980s	19
0	-0.087865	-0.064967	-0.066399	-0.067301	-0.076922	-0.070895	-0.097219	-0.101692	-0.091
1	-0.087865	-0.064967	-0.066399	-0.067301	-0.076922	-0.070895	-0.097219	-0.101692	-0.091
2	-0.087865	-0.064967	-0.066399	-0.067301	-0.076922	-0.070895	-0.097219	-0.101692	-0.091
3	-0.087865	-0.064967	-0.066399	-0.067301	-0.076922	-0.070895	-0.097219	-0.101692	-0.091
4	-0.087865	-0.064967	-0.066399	-0.067301	-0.076922	-0.070895	-0.097219	-0.101692	-0.091
...	...	...	...	...	...	...	...	...	
1259	-0.087865	-0.064967	-0.066399	-0.067301	-0.076922	-0.070895	-0.097219	-0.101692	-0.091
1260	-0.087865	-0.064967	-0.066399	-0.067301	-0.076922	-0.070895	-0.097219	-0.101692	-0.091
1261	-0.087865	-0.064967	-0.066399	-0.067301	-0.076922	-0.070895	-0.097219	-0.101692	-0.091
1262	-0.087865	-0.064967	-0.066399	-0.067301	-0.076922	-0.070895	-0.097219	-0.101692	-0.091
1263	-0.087865	-0.064967	-0.066399	-0.067301	-0.076922	-0.070895	-0.097219	-0.101692	-0.091

1264 rows × 8576 columns

Topic Modelling

Le topic modelling est une technique d'apprentissage automatique non supervisée qui identifie et extrait des thèmes ou des sujets latents à partir d'un ensemble de documents textuels.

In [ ]:
```python
# Vecteur de caractéristiques
vectorizer = CountVectorizer(max_features=1000, stop_words='english')
X = vectorizer.fit_transform(df['text'])

# Modèle LDA
lda_model = LatentDirichletAllocation(n_components=10, random_state=42)
lda_model.fit(X)

# Afficher les mots clés des topics
feature_names = vectorizer.get_feature_names_out()
for topic_idx, topic in enumerate(lda_model.components_):
    print(f"Topic {topic_idx}:")
    print(" ".join([feature_names[i] for i in topic.argsort()[:-10 - 1:-1]]))

# Identifier les topics sous-représentés
topic_distribution = lda_model.transform(X)
topic_counts = topic_distribution.sum(axis=0)
average_documents_per_topic = median(topic_counts)
seuil = 0.5 * average_documents_per_topic # 50% du nombre moyen de documents par to

print("\nMoyenne de docs par topic : ", int(average_documents_per_topic))

underrepresented_topics = np.where(topic_counts < seuil)[0]
if (underrepresented_topics.size == 0) :
   print("\nAucun topics sous-représentés")
else :
  print("\nTopics sous-représentés:", underrepresented_topics)
```

```
Topic 0:
said eu cent people year uk government britain 000 court
Topic 1:
new state year students education school york need schools children
Topic 2:
food 2020 20 services tax party works pizza care chick
Topic 3:
said people government just like country think ve state right
Topic 4:
covid coronavirus 19 virus china people ballots election cases masks
Topic 5:
health vaccine nhs cancer people patients said children study doctors
Topic 6:
people said year rise million poverty level 000 years report
Topic 7:
climate change global warming ice world sea years scientists said
Topic 8:
trump president said state biden campaign states election american house
Topic 9:
news uk pictures 2021 police march getty pa london verifyerrors

Moyenne de docs par topic :  117

Topics sous-représentés: [2 9]
```

In [ ]:

```python
# Fonction pour compter les topics dans le dataframe
def count_topics_in_dataframe(df, vectorizer, lda_model):
    # Dictionnaire
    topic_counts = {topic_idx: 0 for topic_idx in range(lda_model.n_components)}

    # Parcourir chaque phrase dans le df
    for text in df['text']:

        text_vectorized = vectorizer.transform([text])
        topic_distribution = lda_model.transform(text_vectorized)

        # topic principal
        main_topic = np.argmax(topic_distribution)

        # Compteur d'articles par topics
        topic_counts[main_topic] += 1

    return topic_counts

topics_counts = count_topics_in_dataframe(df, vectorizer, lda_model)
# Affichage des comptes de topics
for topic_idx, count in topics_counts.items():
    print(f"Topic {topic_idx}: {count} occurrences")
```

```
Topic 0: 166 occurrences
Topic 1: 79 occurrences
Topic 2: 24 occurrences
Topic 3: 202 occurrences
Topic 4: 117 occurrences
Topic 5: 125 occurrences
Topic 6: 51 occurrences
Topic 7: 154 occurrences
Topic 8: 314 occurrences
Topic 9: 32 occurrences
```

Upsampling

L'upsampling est une technique utilisée dans le domaine de l'apprentissage automatique pour équilibrer les classes déséquilibrées en augmentant le nombre d'instances de la classe minoritaire.

In [ ]:
```
# Encodage des étiquettes
label = preprocessing.LabelEncoder()
label.fit(df["our rating"])
Y_train = label.transform(df["our rating"])
X_with_topics = np.hstack((X.toarray(), topic_distribution))

print("Encodage :")
for cls, idx in zip(label.classes_, label.transform(label.classes_)):
    print(f" '{cls}' est encodée en {idx}")

# Oversampler
oversampler = SMOTE()
# X_train_resampled, Y_train_resampled = oversampler.fit_resample(vectorized, Y_tra
X_train_resampled, Y_train_resampled = oversampler.fit_resample(X_with_topics, Y_tr

# Comptage avant/après sampling
unique_classes_before, counts_before = np.unique(Y_train, return_counts=True)
print("\nAvant l'upsampling:")
for cls, count in zip(unique_classes_before, counts_before):
    print(f"\tClasse {cls}: {count} exemples")

unique_classes_after, counts_after = np.unique(Y_train_resampled, return_counts=Tru
print("\nAprès l'upsampling:")
for cls, count in zip(unique_classes_after, counts_after):
    print(f"\tClasse {cls}: {count} exemples")
```

```
Encodage :
 'false' est encodée en 0
 'mixture' est encodée en 1
 'other' est encodée en 2
 'true' est encodée en 3

Avant l'upsampling:
        Classe 0: 578 exemples
        Classe 1: 358 exemples
        Classe 2: 117 exemples
        Classe 3: 211 exemples

Après l'upsampling:
        Classe 0: 578 exemples
        Classe 1: 578 exemples
        Classe 2: 578 exemples
        Classe 3: 578 exemples
```

Classification

La classification va pouvoir nous permettre déterminer la véracité de l'article en fonction de leurs caractéristiques.

Pour obtenir le meuilleur résultat possible, nous allons pour établir 4 méthodes de classification :

- Naïve Bayes
- SVC (Support Vector Clustering)
- Decision Tree
- KNN (k-nearest neighbors)

Puis nous allons les tester sur 3 taches de classification :

- {VRAI} vs. {FAUX} vs. {MIXTE} vs. {AUTRE} (quatre classes)
- {VRAI} vs. {FAUX} (deux classes)
- {VRAI ou FAUX} vs. {AUTRE} (deux classes)

Nous pourrons ensuite évualer chaque classification puis les comparer avec les autres.

Fonction d'affichage des courbes

In [ ]:
```
def plot_curves_confusion(confusion_matrix, class_names):
    plt.figure(1, figsize=(16, 6))
    plt.gcf().subplots_adjust(left=0.125, bottom=0.2, right=1, top=0.9, wspace=0.25

    # Matrice de confusion
    plt.subplot(1, 3, 3)
    sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap='Blues', xticklabels=cl
    plt.xlabel('Predicted', fontsize=12)
    plt.title("Confusion matrix")
    plt.ylabel('True', fontsize=12)
    plt.show()


def plot_curves(scores):
    plt.figure(1, figsize=(16, 6))
    plt.gcf().subplots_adjust(left=0.125, bottom=0.2, right=1, top=0.9, wspace=0.25

    # Plot loss
    plt.subplot(121)
    plt.title('Cross Entropy Loss')
    plt.plot(scores, color='blue')
    plt.ylabel('Loss')
    plt.xlabel('Fold')

    # Plot accuracy
    plt.subplot(122)
    plt.title('Classification Accuracy')
    plt.plot(1 - scores, color='red')
    plt.ylabel('Error Rate')
    plt.xlabel('Fold')

    plt.show()

def plot_curves_results(naive_scores, svc_scores, decision_scores, knn_scores):
  classifiers = ['Naive Bayes', 'SVC', 'Decision Tree', 'KNN']

  fold_scores = [naive_scores, svc_scores, decision_scores, knn_scores]

  # Scores moyens
  plt.figure(figsize=(8, 5))
  mean_scores = [score.mean() for score in fold_scores]
  plt.bar(classifiers, mean_scores, color=['blue', 'orange', 'green', 'red'])
  plt.title('Scores moyens des classifieurs')
  plt.xlabel('Classifieurs')
  plt.ylabel('Score moyen')
  plt.show()

  # Scores pour chaque fold
  plt.figure(figsize=(7, 8))
  for i, (classifier, scores) in enumerate(zip(classifiers, fold_scores), start=1):
      plt.subplot(2, 2, i)
      plt.boxplot(scores)
      plt.title(f'Scores pour {classifier}')
      plt.xlabel('Fold')
      plt.ylabel('Score')

  plt.tight_layout()
  plt.show()
```

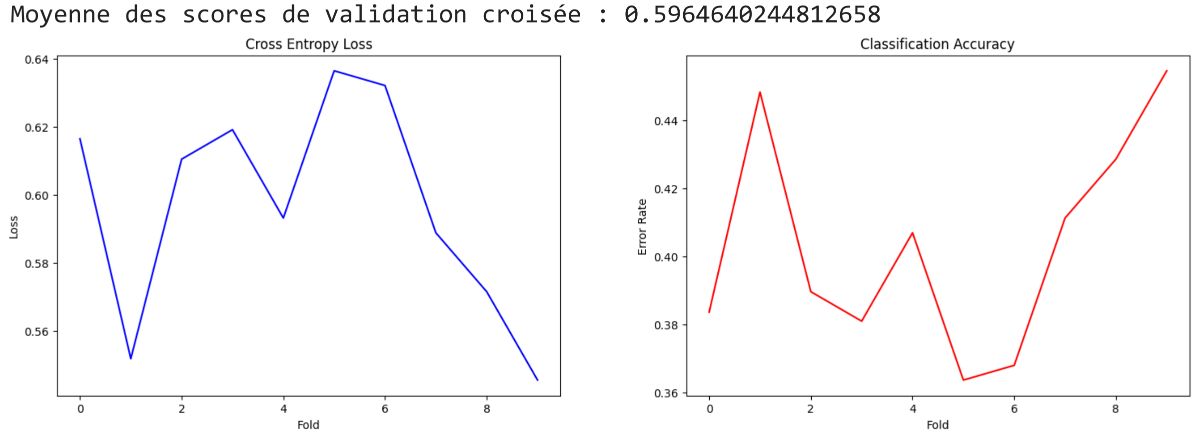
{VRAI} vs. {FAUX} vs. {MIXTE} vs. {AUTRE}

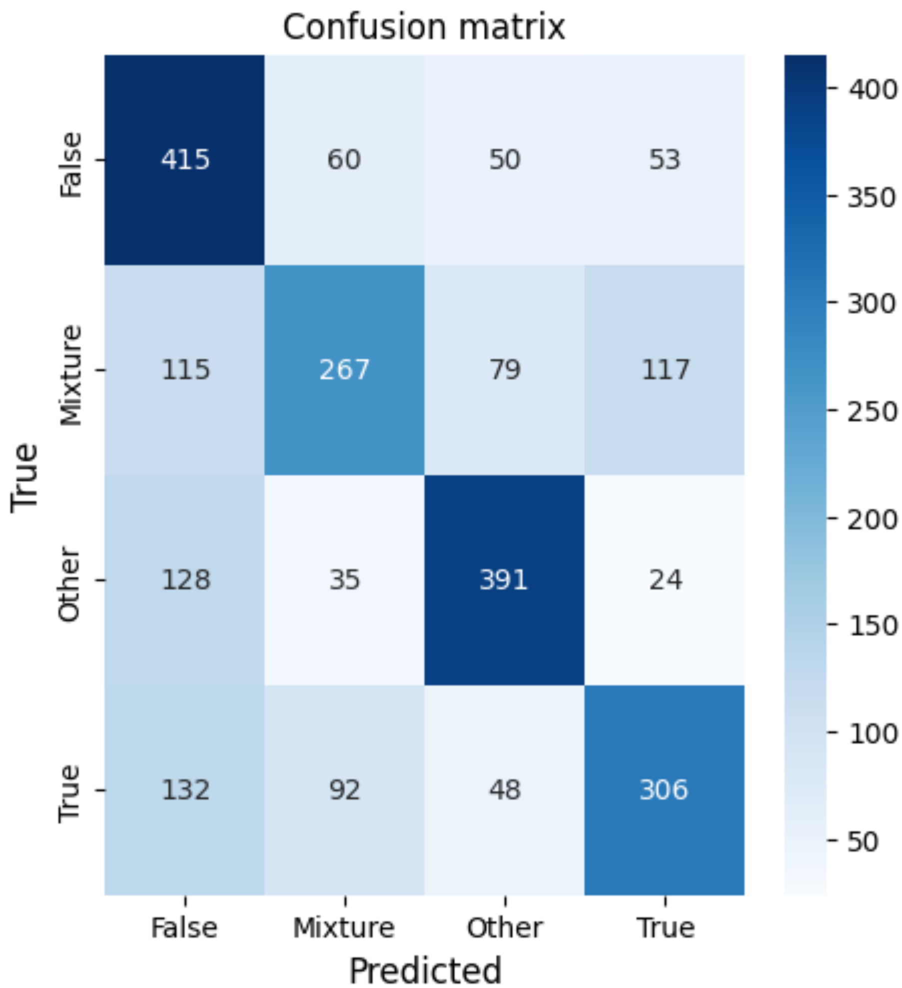
Naïve Bayes

In [ ]:

```
#X_train, X_test, y_train, y_test = train_test_split(X_train_resampled, Y_train_res

kfold = KFold(n_splits=10, shuffle=True, random_state=42)  # Définir le nombre de p
naive_bayes_classifier = MultinomialNB()

# Effectuer la validation croisée k-fold
naive_scores = cross_val_score(naive_bayes_classifier, X_train_resampled, Y_train_r

# Effectuer la validation croisée k-fold et obtenir les prédictions
y_pred_cv = cross_val_predict(naive_bayes_classifier, X_train_resampled, Y_train_re

# Calculer la matrice de confusion
conf_matrix = confusion_matrix(Y_train_resampled, y_pred_cv)

# Afficher les scores de validation croisée
print("Scores de validation croisée :", naive_scores)
print("Moyenne des scores de validation croisée :", naive_scores.mean())

plot_curves(naive_scores)
plot_curves_confusion(conf_matrix, ['False', 'Mixture', 'Other', 'True'])
```

```
Scores de validation croisée : [0.61637931 0.55172414 0.61038961 0.61904762 0.59307
359 0.63636364
 0.63203463 0.58874459 0.57142857 0.54545455]
Moyenne des scores de validation croisée : 0.5964640244812658
```



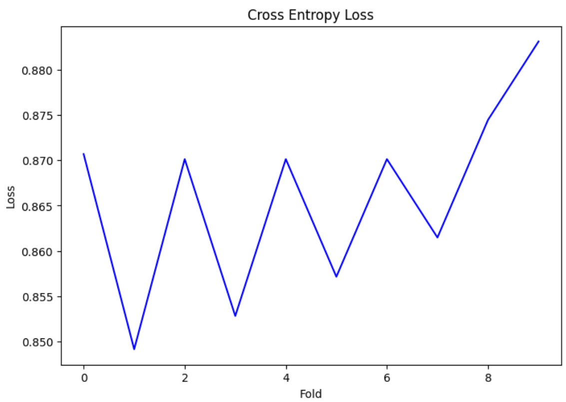


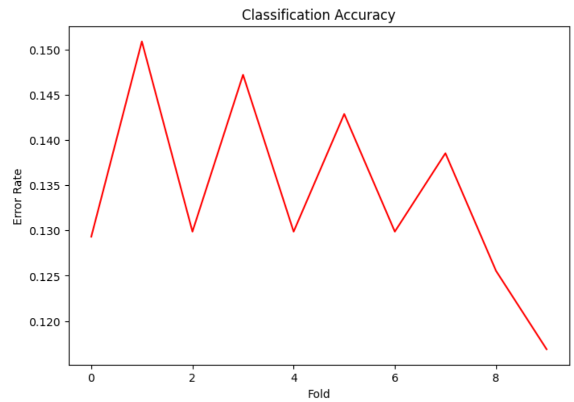
SVC

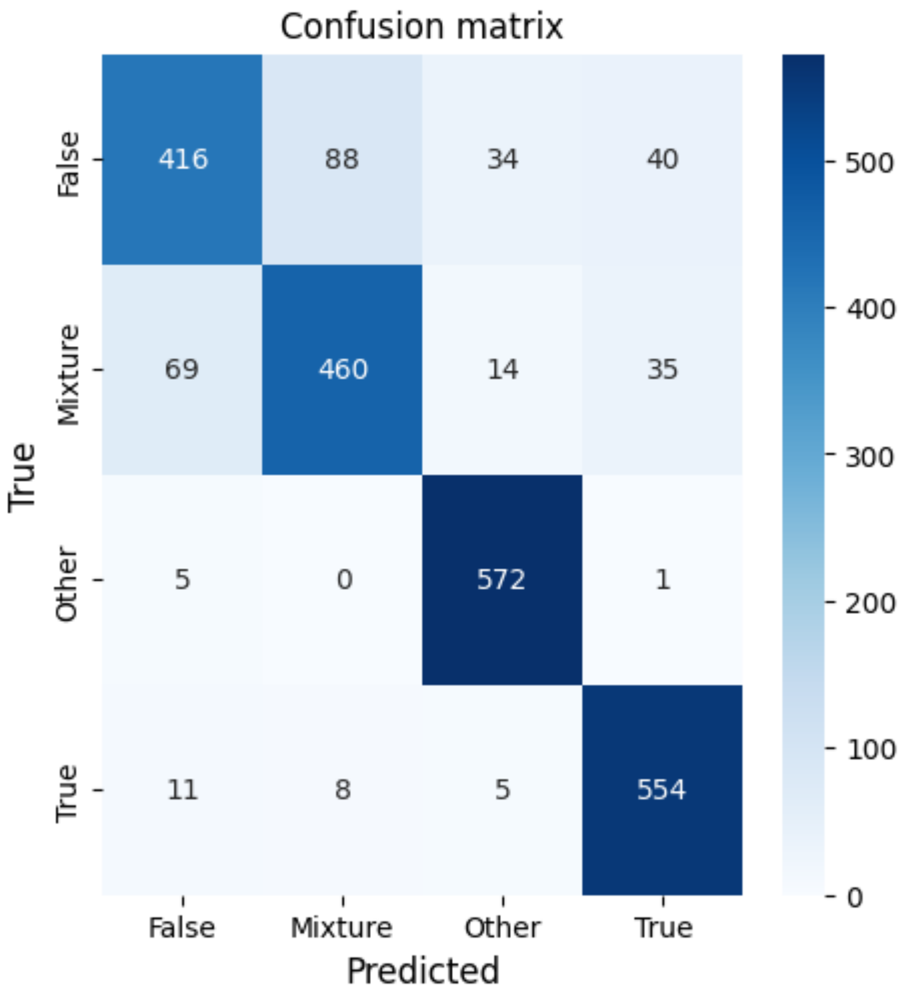
In [ ]:

```
# classifieur SVC
clf_SVC = SVC(kernel='linear')

kfold = KFold(n_splits=10, shuffle=True, random_state=42)
svc_scores = cross_val_score(clf_SVC, X_train_resampled, Y_train_resampled, cv=kfol

# Prédiction avec validation croisée
y_pred_cv = cross_val_predict(clf_SVC, X_train_resampled, Y_train_resampled, cv=kfo

# Calcul de la matrice de confusion
conf_matrix = confusion_matrix(Y_train_resampled, y_pred_cv)

print("Scores de validation croisée :", svc_scores)
print("Moyenne des scores de validation croisée :", svc_scores.mean())

plot_curves(svc_scores)
plot_curves_confusion(conf_matrix, ['False', 'Mixture', 'Other', 'True'])
```

```
Scores de validation croisée : [0.87068966 0.84913793 0.87012987 0.85281385 0.87012
987 0.85714286
 0.87012987 0.86147186 0.87445887 0.88311688]
Moyenne des scores de validation croisée : 0.8659221525600836
```





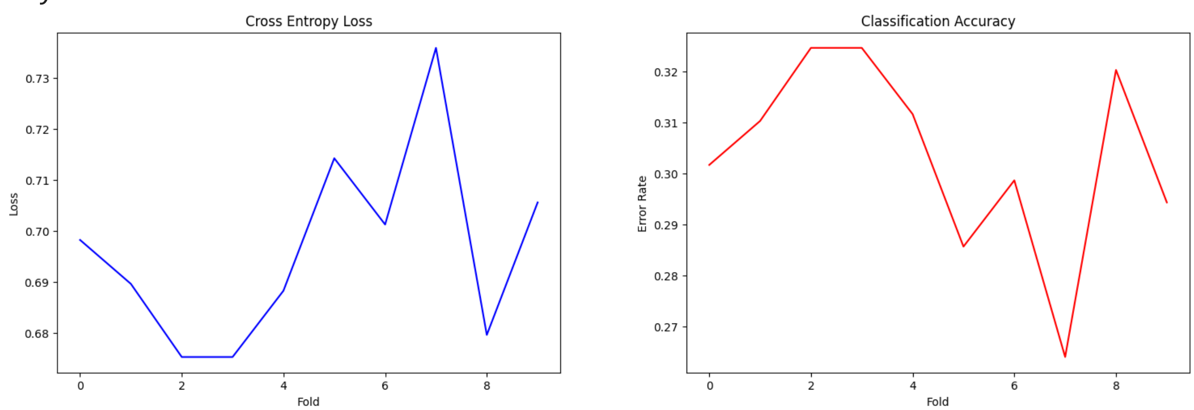


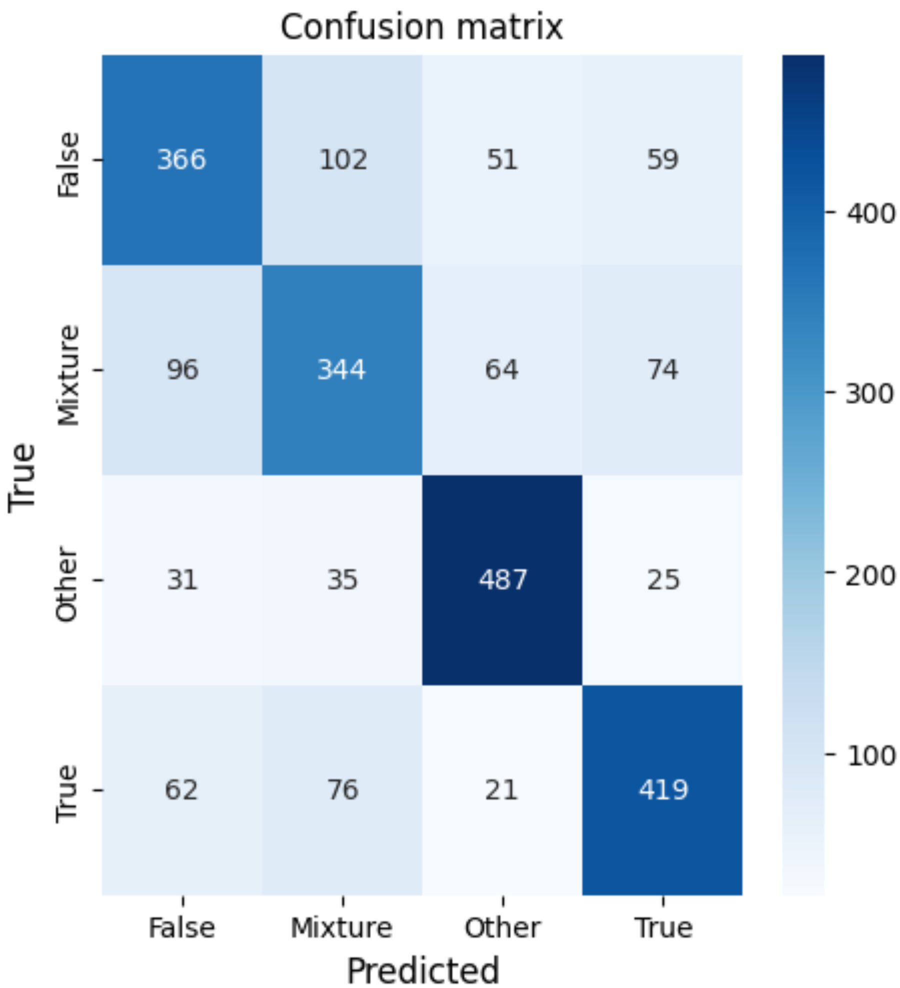
Decision Tree

In [ ]:

```
# Division des données en ensembles d'entraînement et de test
#X_train, X_test, y_train, y_test = train_test_split(X_train_resampled, Y_train_res

# Entraînement du modèle
clf_Tree = DecisionTreeClassifier()
#clf_Tree.fit(X_train, y_train)

# Prédiction
y_pred_cv = cross_val_predict(clf_Tree, X_train_resampled, Y_train_resampled, cv=kf

# Calcul de la matrice de confusion
conf_matrix = confusion_matrix(Y_train_resampled, y_pred_cv)

kfold = KFold(n_splits=10, shuffle=True, random_state=42)
decision_scores = cross_val_score(clf_Tree, X_train_resampled, Y_train_resampled, c

print("Scores de validation croisée :", decision_scores)
print("Moyenne des scores de validation croisée :", decision_scores.mean())

plot_curves(decision_scores)
plot_curves_confusion(conf_matrix, ['False', 'Mixture', 'Other', 'True'])
```

```
Scores de validation croisée : [0.69827586 0.68965517 0.67532468 0.67532468 0.68831
169 0.71428571
 0.7012987  0.73593074 0.67965368 0.70562771]
Moyenne des scores de validation croisée : 0.6963688610240334
```




Confusion matrix
366
102
51
59
96
344
64
74
31
35
487
25
62
76
21
419
False
Mixture
Other
True
True
False
Mixture
Other
True
Predicted
400
300
200
100

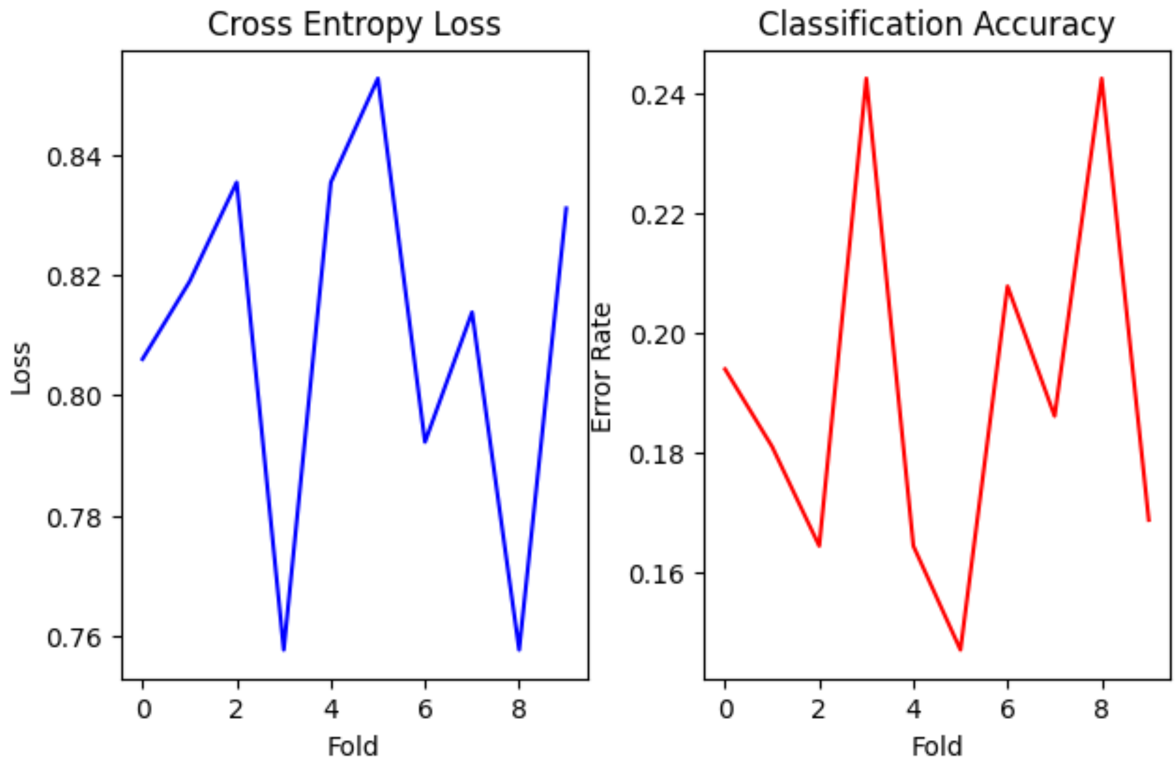
KNN

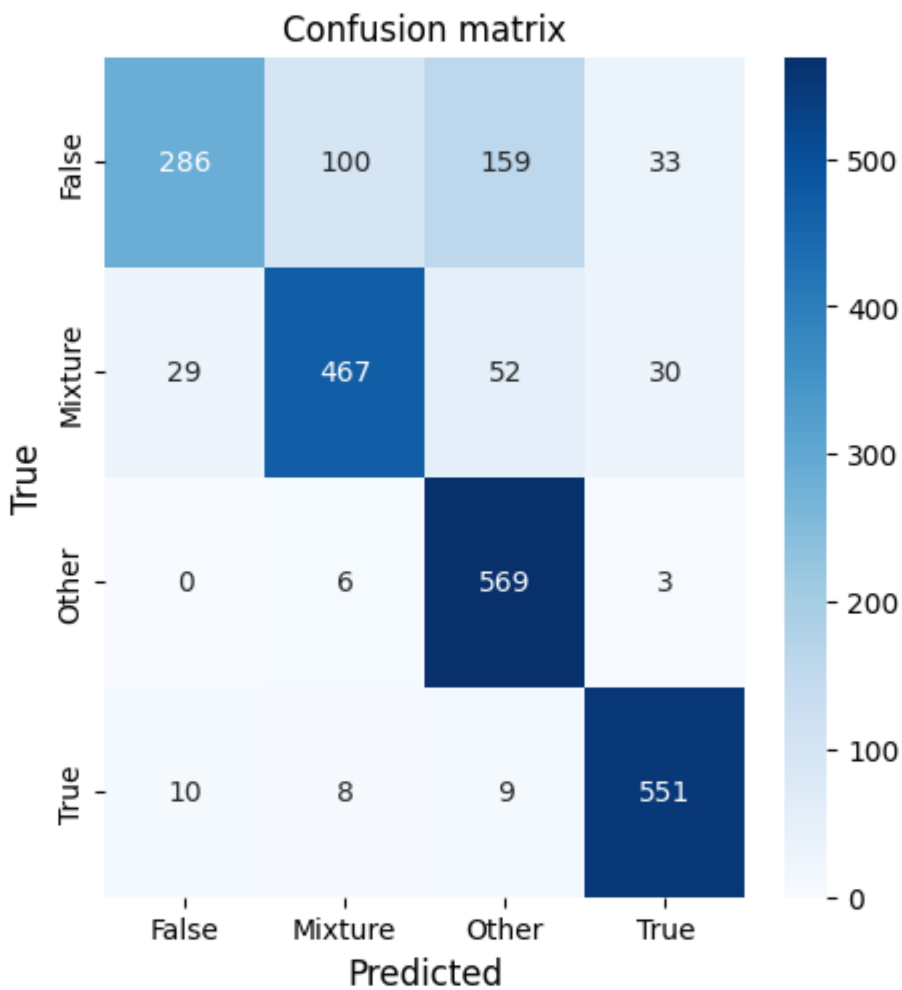
In [ ]:

```
# Normaliser les données
scaler = StandardScaler()
scores, abcisse = list(), list()


for i in range(1, 10):
  # Initialiser le classifieur KNN
  k = i  # Nombre de voisins
  knn_classifier = KNeighborsClassifier(n_neighbors=k)

  kfold = KFold(n_splits=10, shuffle=True, random_state=42)
  knn_scores = cross_val_score(knn_classifier, X_train_resampled, Y_train_resampled
  scores.append(knn_scores.mean())
  abcisse.append(i)

# Trouver le meilleur nombre de voisins
best_k = abcisse[scores.index(max(scores))]
print("Meilleur nombre de voisins :", best_k)
knn_classifier = KNeighborsClassifier(n_neighbors=best_k)

kfold = KFold(n_splits=10, shuffle=True, random_state=42)
knn_score = cross_val_score(knn_classifier, X_train_resampled, Y_train_resampled, c

# Entraîner le modèle KNN avec le meilleur nombre de voisins
best_knn_classifier = KNeighborsClassifier(n_neighbors=best_k)
#best_knn_classifier.fit(X_train_resampled, Y_train_resampled)

# Prédiction
#y_pred_cv = best_knn_classifier.predict(X_train_resampled)
y_pred_cv = cross_val_predict(best_knn_classifier, X_train_resampled, Y_train_resam

# Calculer la matrice de confusion
conf_matrix = confusion_matrix(Y_train_resampled, y_pred_cv)

# Calculer la précision
#accuracy_KNN = accuracy_score(y_test, y_pred)
#print(f"Accuracy KNN : {accuracy_KNN * 100:.2f}")
plt.title('KNN en fonction des k voisins')
plt.plot(abcisse, scores)
plt.ylabel('Score de validation')
plt.xlabel('Nombre de voisins')
#plt.xlim(1,10)
print("Scores de validation croisée :", knn_score)
print("Moyenne des scores de validation croisée :", knn_score.mean())

plot_curves(knn_score)
plot_curves_confusion(conf_matrix, ['False', 'Mixture', 'Other', 'True'])
```

```
Meilleur nombre de voisins : 1
Scores de validation croisée : [0.80603448 0.81896552 0.83549784 0.75757576 0.83549
784 0.85281385
 0.79220779 0.81385281 0.75757576 0.83116883]
Moyenne des scores de validation croisée : 0.8101190476190476
```

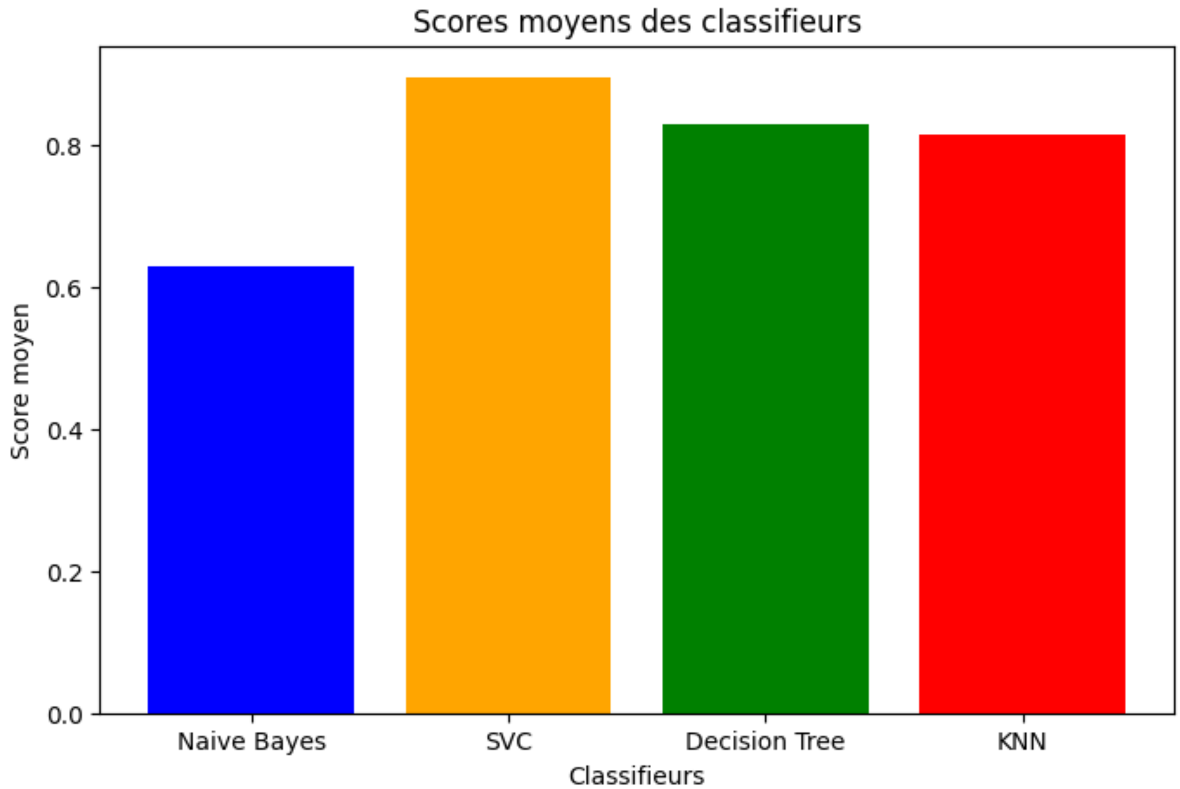
```
<ipython-input-81-1de4a8dbf2fd>:19: MatplotlibDeprecationWarning: Auto-removal of o
verlapping axes is deprecated since 3.6 and will be removed two minor releases late
r; explicitly call ax.remove() as needed.
  plt.subplot(121)
```

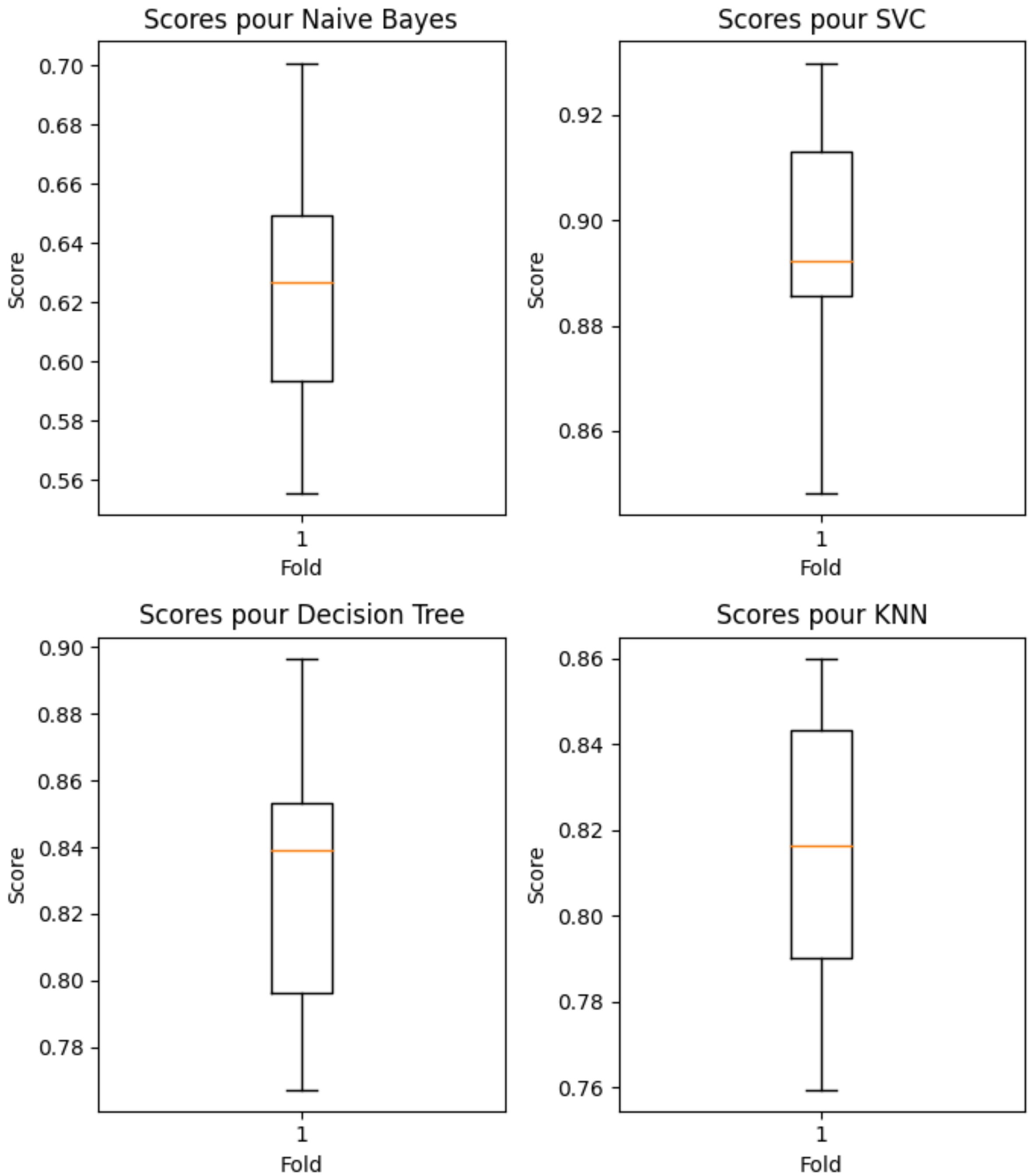

Cross Entropy Loss
Loss
Fold
Classification Accuracy
Error Rate
Fold


Confusion matrix
True
Predicted
False
Mixture
Other
True
286
100
159
33
29
467
52
30
0
6
569
3
10
8
9
551

Evaluation

In [ ]:
```
plot_curves_results(naive_scores, svc_scores, decision_scores, knn_score)
```





{VRAI} vs. {FAUX}

Restructuration

On restructure nos donneés pour garder uniquement vrai ou faux.

In [ ]:

```python
# Indices des valeurs où Y_train_resampled est égal à 0 ou 3 (faux et vrai)
indices = np.where((Y_train_resampled == 0) | (Y_train_resampled == 3))

X_train_vraifaux = X_train_resampled[indices]
Y_train_vraifaux = Y_train_resampled[indices]

# Remplace les occurrences de 3 par 1
Y_train_vraifaux[Y_train_vraifaux == 3] = 1

print("Y:")
print(Y_train_vraifaux.tolist())
print("\n 0 = faux, 1 = vrai")
```

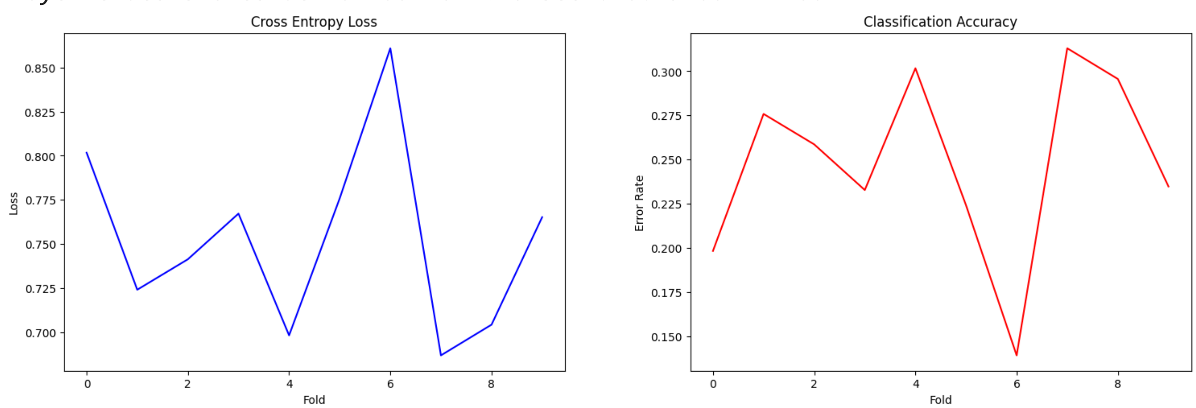
```
Y:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 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, 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,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0,
1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0,
0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0,
1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0,
0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1,
1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0,
1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0,
0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0,
0, 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, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 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 = faux, 1 = vrai
```

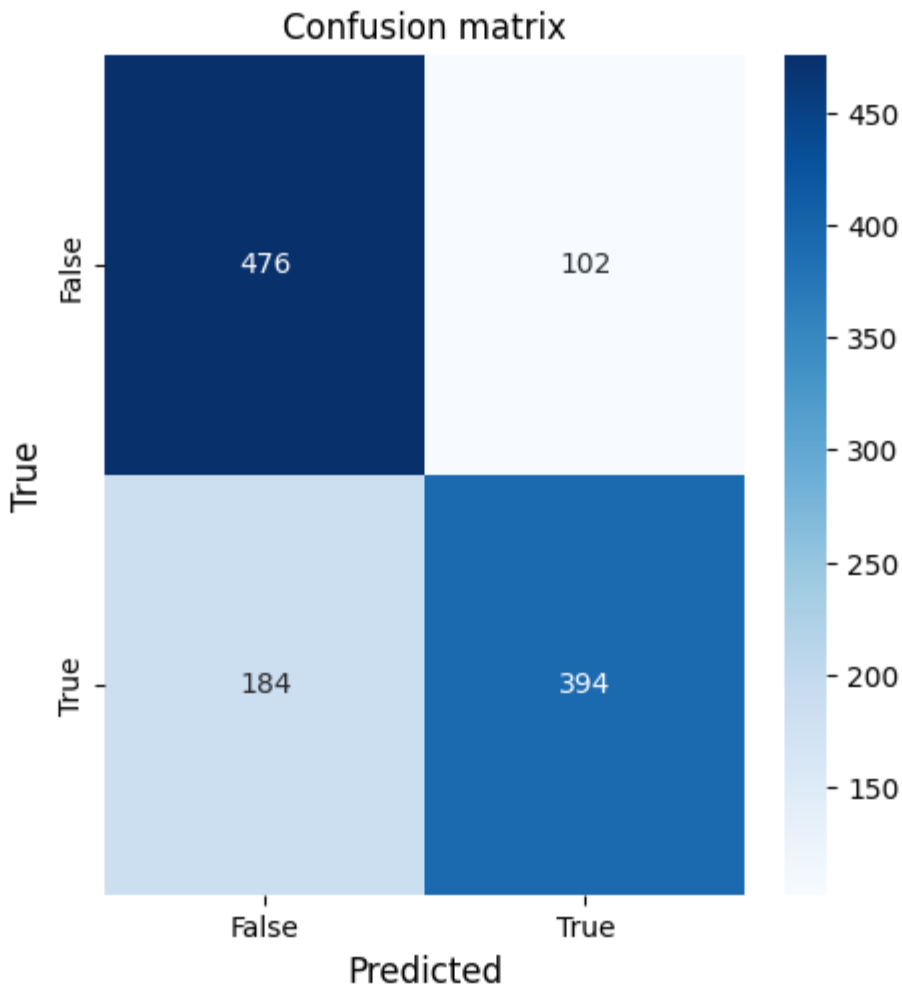
Vrai et faux ayant déja reçu un Upsampling, il n'y a pas besoin de retraitement.

Naives Bayes

In [ ]:
```
kfold = KFold(n_splits=10, shuffle=True, random_state=42)  # Définir le nombre de p
naive_bayes_classifier = MultinomialNB()

# Effectuer la validation croisée k-fold
naive_scores = cross_val_score(naive_bayes_classifier, X_train_vraifaux, Y_train_vr

# prédictions
y_pred_cv = cross_val_predict(naive_bayes_classifier, X_train_vraifaux, Y_train_vra

# Calculer la matrice de confusion
conf_matrix = confusion_matrix(Y_train_vraifaux, y_pred_cv)

# Afficher les scores de validation croisée
print("Scores de validation croisée :", naive_scores)
print("Moyenne des scores de validation croisée :", naive_scores.mean())

plot_curves(naive_scores)
plot_curves_confusion(conf_matrix, ['False', 'True'])
```

```
Scores de validation croisée : [0.80172414 0.72413793 0.74137931 0.76724138 0.69827
586 0.77586207
 0.86086957 0.68695652 0.70434783 0.76521739]
Moyenne des scores de validation croisée : 0.7526011994002999
```

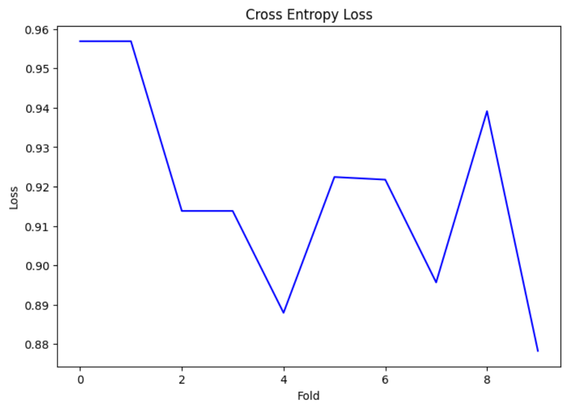


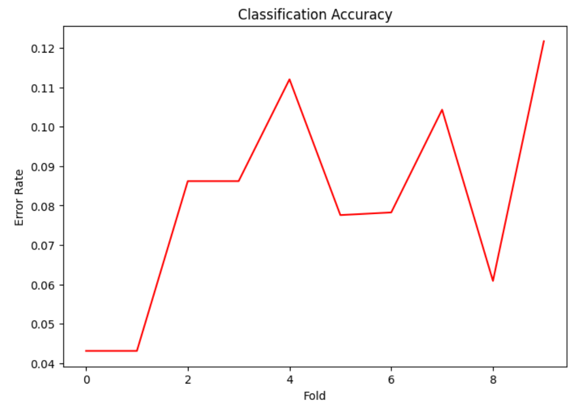


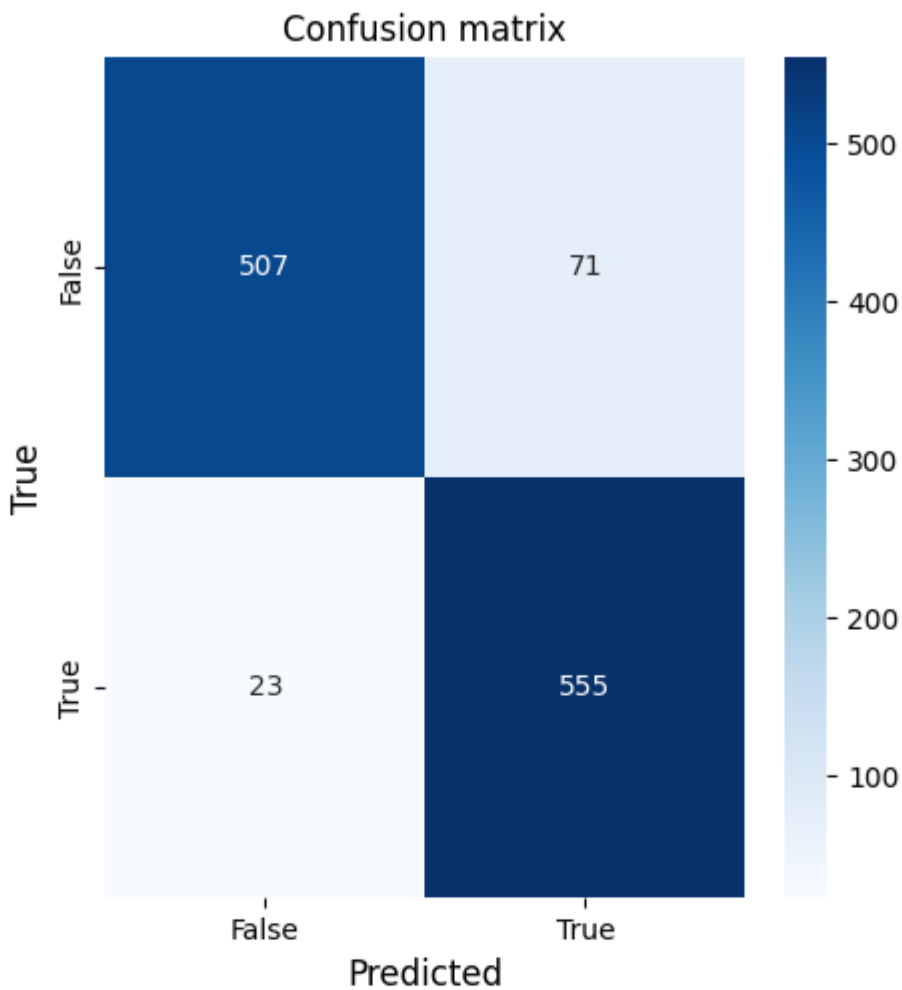
SVC

In [ ]:
```
# Création du classifieur SVC
clf_SVC = SVC(kernel='linear')

kfold = KFold(n_splits=10, shuffle=True, random_state=42)
svc_scores = cross_val_score(clf_SVC, X_train_vraifaux, Y_train_vraifaux, cv=kfold)

# Prédiction avec validation croisée
y_pred_cv = cross_val_predict(clf_SVC, X_train_vraifaux, Y_train_vraifaux, cv=kfold

# Calculer la matrice de confusion
conf_matrix = confusion_matrix(Y_train_vraifaux, y_pred_cv)

print("Scores de validation croisée :", svc_scores)
print("Moyenne des scores de validation croisée :", svc_scores.mean())

plot_curves(svc_scores)
plot_curves_confusion(conf_matrix, ['False','True'])
```

```
Scores de validation croisée : [0.95689655 0.95689655 0.9137931  0.9137931  0.88793
103 0.92241379
 0.92173913 0.89565217 0.93913043 0.87826087]
Moyenne des scores de validation croisée : 0.9186506746626687
```



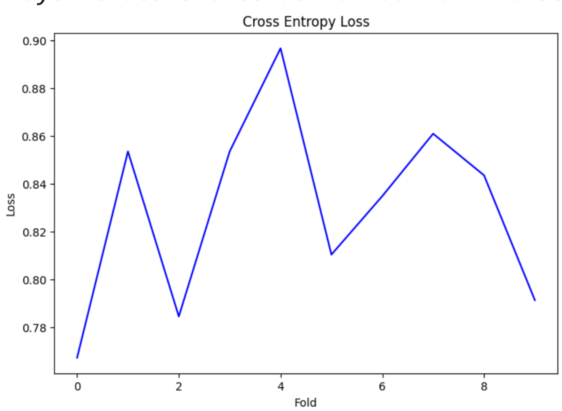


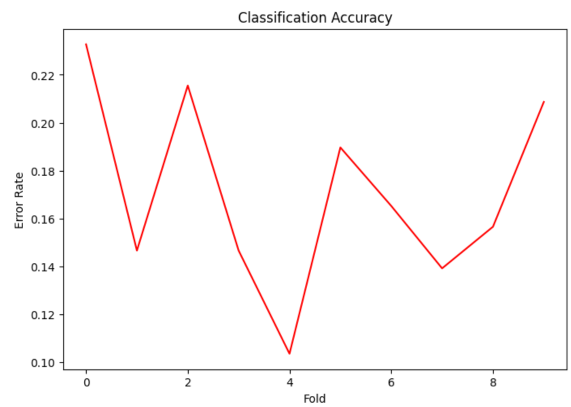


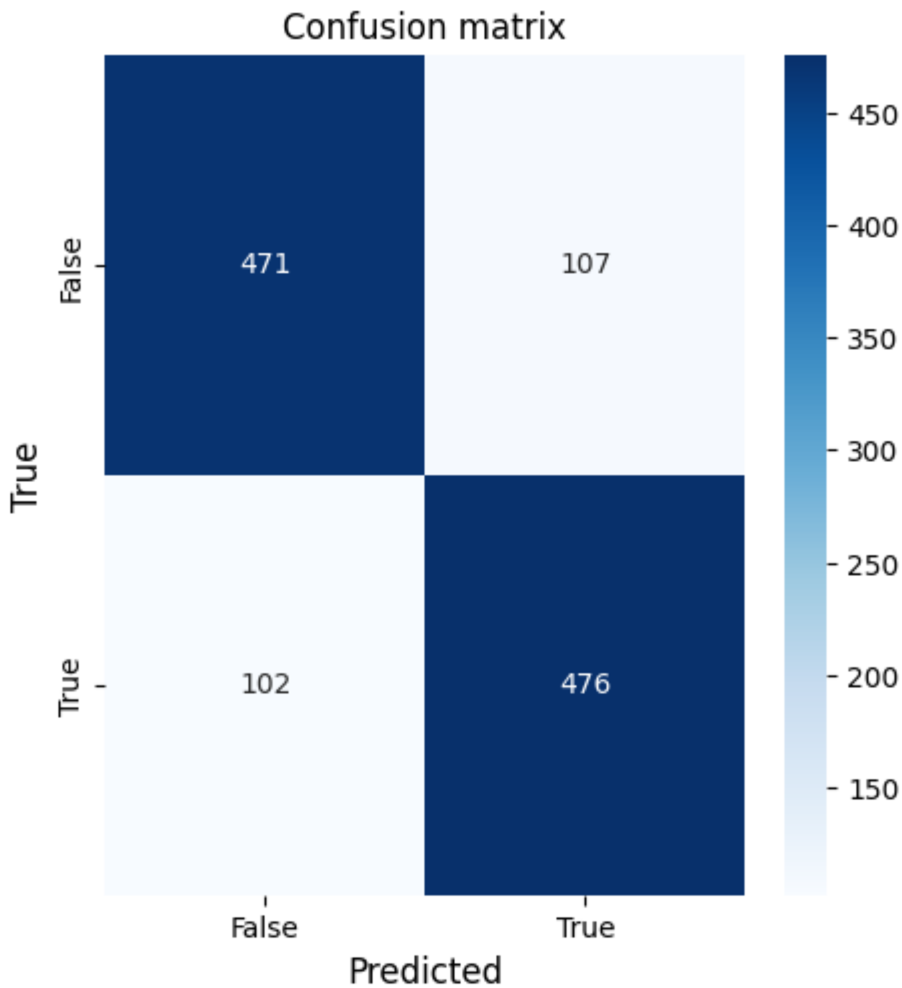
Decision Tree

In [ ]:
```python
# Initialisation et entraînement du modèle d'arbre de décision
clf_Tree = DecisionTreeClassifier()


# Prédiction sur l'ensemble de test
y_pred_cv = cross_val_predict(clf_Tree, X_train_vraifaux, Y_train_vraifaux, cv=kfol

# Calcul de la matrice de confusion
conf_matrix = confusion_matrix(Y_train_vraifaux, y_pred_cv)

kfold = KFold(n_splits=10, shuffle=True, random_state=42)
decision_scores = cross_val_score(clf_Tree, X_train_vraifaux, Y_train_vraifaux, cv=

print("Scores de validation croisée :", decision_scores)
print("Moyenne des scores de validation croisée :", decision_scores.mean())

plot_curves(decision_scores)
plot_curves_confusion(conf_matrix, ['False', 'True'])
```

```
Scores de validation croisée : [0.76724138 0.85344828 0.78448276 0.85344828 0.89655
172 0.81034483
 0.83478261 0.86086957 0.84347826 0.79130435]
Moyenne des scores de validation croisée : 0.8295952023988006
```





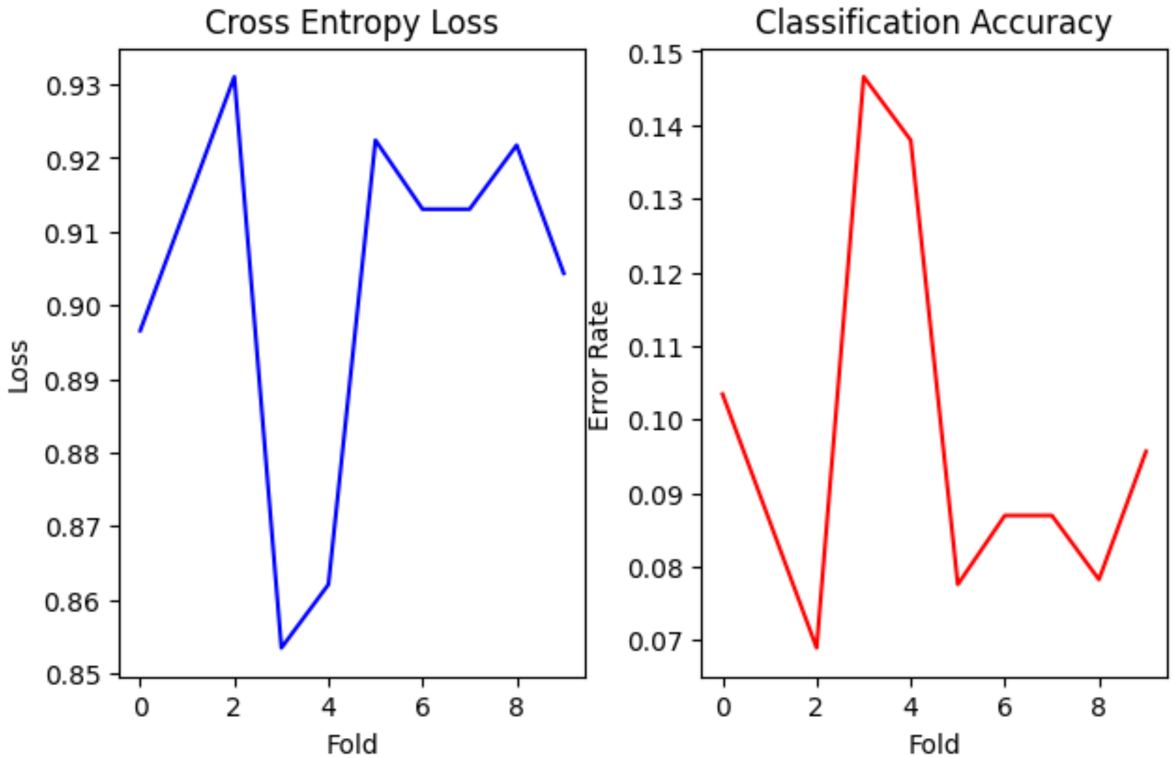

Confusion matrix
471
107
102
476
False
True
True
False
True
Predicted
450
400
350
300
250
200
150

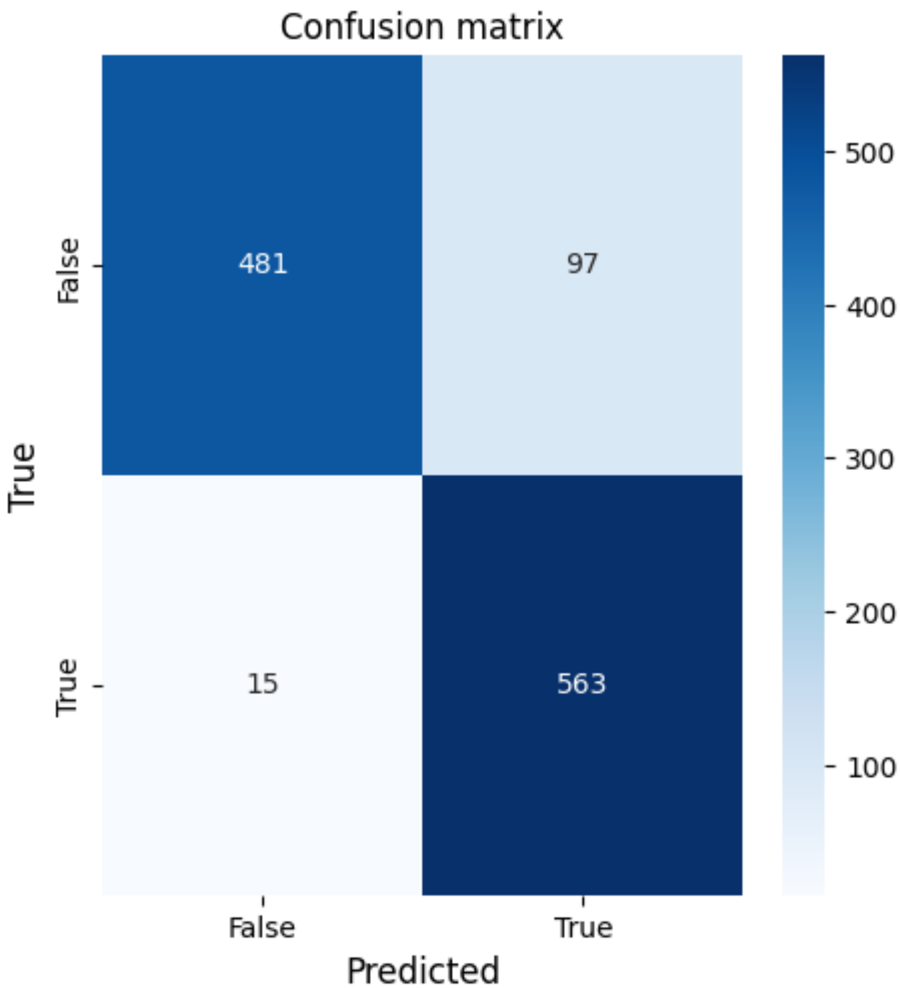
KNN

```
In [ ]: # Diviser les données en ensembles d'entraînement et de test
#X_train, X_test, y_train, y_test = train_test_split(X_train_vraifaux, Y_train_vrai

# Normaliser les données
scaler = StandardScaler()
#X_train = scaler.fit_transform(X_train)
#X_test = scaler.transform(X_test)
scores, abcisse = list(), list()


for i in range(1, 10):
  # Initialiser le classifieur KNN
  k = i  # Nombre de voisins à considérer
  knn_classifier = KNeighborsClassifier(n_neighbors=k)

  kfold = KFold(n_splits=10, shuffle=True, random_state=42)
  knn_scores = cross_val_score(knn_classifier, X_train_vraifaux, Y_train_vraifaux,
  scores.append(knn_scores.mean())
  abcisse.append(i)

# Trouver le meilleur nombre de voisins
best_k = abcisse[scores.index(max(scores))]
print("Meilleur nombre de voisins :", best_k)
knn_classifier = KNeighborsClassifier(n_neighbors=best_k)
kfold = KFold(n_splits=10, shuffle=True, random_state=42)
knn_score = cross_val_score(knn_classifier, X_train_vraifaux, Y_train_vraifaux, cv=

# Entraîner le modèle KNN avec le meilleur nombre de voisins
best_knn_classifier = KNeighborsClassifier(n_neighbors=best_k)
best_knn_classifier.fit(X_train_vraifaux, Y_train_vraifaux)

# Prédiction sur l'ensemble de test
#y_pred_cv = best_knn_classifier.predict(X_train_vraifaux)
y_pred_cv = cross_val_predict(best_knn_classifier, X_train_vraifaux, Y_train_vraifa

# Calculer la matrice de confusion
conf_matrix = confusion_matrix(Y_train_vraifaux, y_pred_cv)

# Calculer la précision
#accuracy_KNN = accuracy_score(y_test, y_pred)
#print(f"Accuracy KNN : {accuracy_KNN * 100:.2f}")
plt.title('KNN en fonction des k voisins')
plt.plot(abcisse, scores)
plt.ylabel('Score de validation')
plt.xlabel('Nombre de voisins')
#plt.xlim(1,10)
print("Scores de validation croisée :", knn_score)
print("Moyenne des scores de validation croisée :", knn_score.mean())

plot_curves(knn_score)
plot_curves_confusion(conf_matrix, ['False', 'True'])
```

```
Meilleur nombre de voisins : 1
Scores de validation croisée : [0.89655172 0.9137931  0.93103448 0.85344828 0.86206
897 0.92241379
 0.91304348 0.91304348 0.92173913 0.90434783]
Moyenne des scores de validation croisée : 0.9031484257871065
```

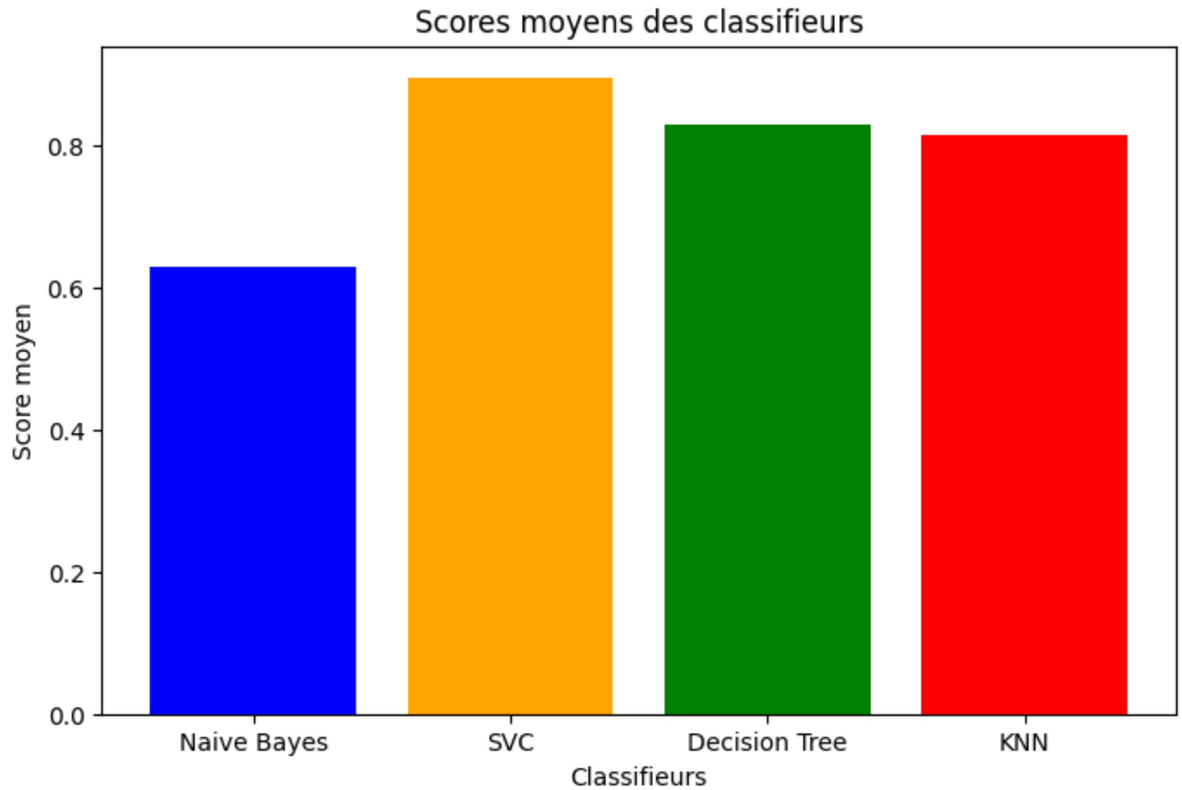
```
<ipython-input-81-1de4a8dbf2fd>:19: MatplotlibDeprecationWarning: Auto-removal of o
verlapping axes is deprecated since 3.6 and will be removed two minor releases late
r; explicitly call ax.remove() as needed.
  plt.subplot(121)
```

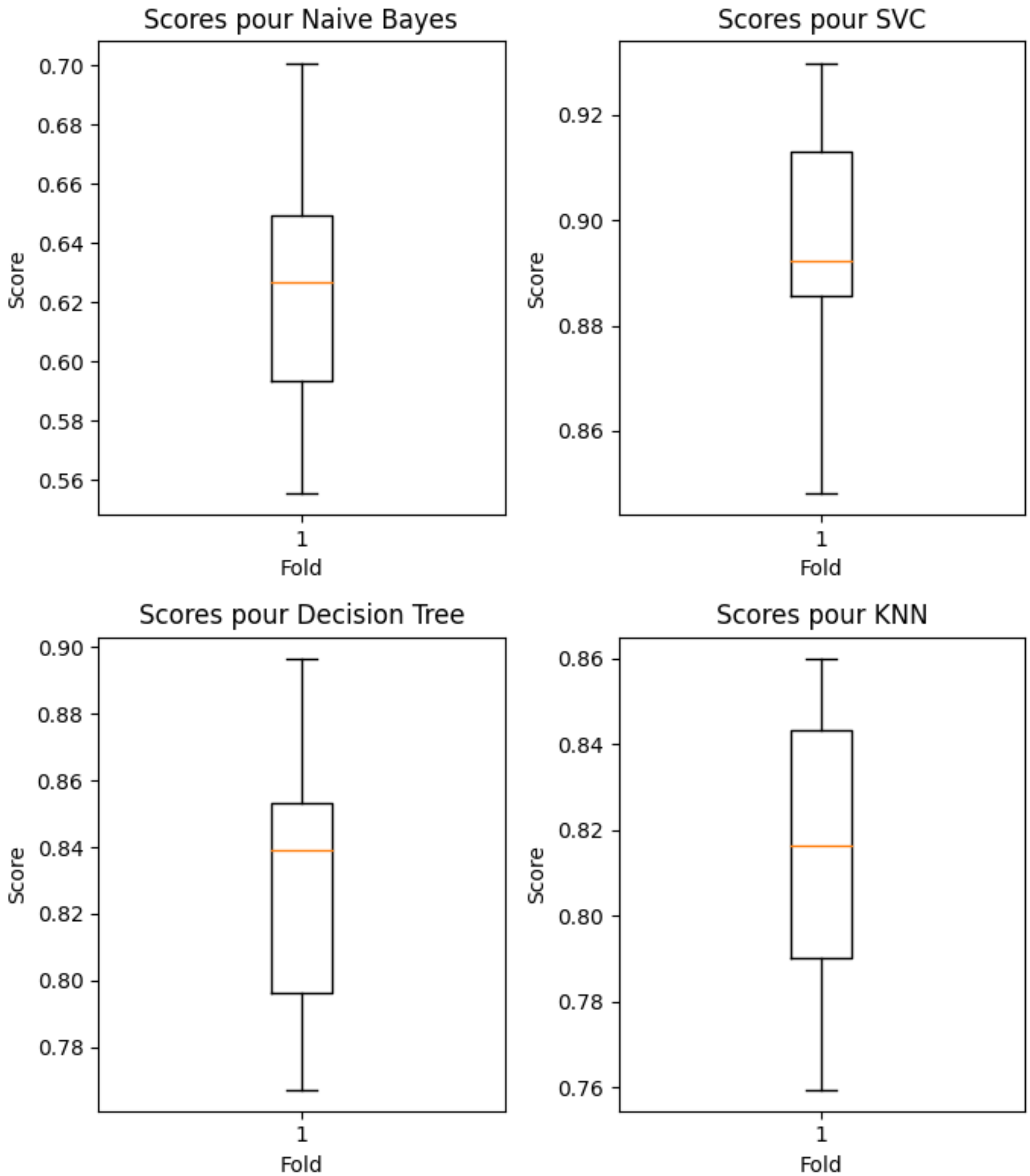




Evaluation

In [ ]:
```
plot_curves_results(naive_scores, svc_scores, decision_scores, knn_score)
```


Scores moyens des classifieurs
Score moyen
0.0
0.2
0.4
0.6
0.8
Naive Bayes
SVC
Decision Tree
KNN
Classifieurs



{VRAI ou FAUX} vs. {AUTRE}

Restructuration

On restructure une nouvelle fois pour fusionner vrai, faux et garder autre. Pour éviter de traiter une grosse structure, nous récupérons les valeurs avant Upsampling. Ensuite nous réalisons l'Upsampling

```
In [ ]: # Indices des valeurs où Y_train_resampled est égal à 0, 2 ou 3
indices = np.where((Y_train == 0) | (Y_train == 2) | (Y_train == 3))

X_train_autre = X[indices]
Y_train_autre = Y_train[indices]

# Fusionne le faux et vrai
Y_train_autre[Y_train_autre == 3] = 0

# Remplace les occurences de 2 par 1
Y_train_autre[Y_train_autre == 2] = 1

print("Y:")
print(Y_train_autre.tolist())
print("Faux ou vrai = 0, autre = 1")
```

```
Y:
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 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, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 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, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1,
1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1,
0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,
0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
Faux ou vrai = 0, autre = 1
```

Upsampling

```
In [ ]: X_train_autre_resampled, Y_train_autre_resampled = oversampler.fit_resample(X_train

        # Comptage avant/après sampling
        unique_classes_before, counts_before = np.unique(Y_train_autre, return_counts=True)
        print("\nAvant l'upsampling:")
        for cls, count in zip(unique_classes_before, counts_before):
            print(f"\tClasse {cls}: {count} exemples")

        unique_classes_after, counts_after = np.unique(Y_train_autre_resampled, return_coun
        print("\nAprès l'upsampling:")
        for cls, count in zip(unique_classes_after, counts_after):
            print(f"\tClasse {cls}: {count} exemples")
```

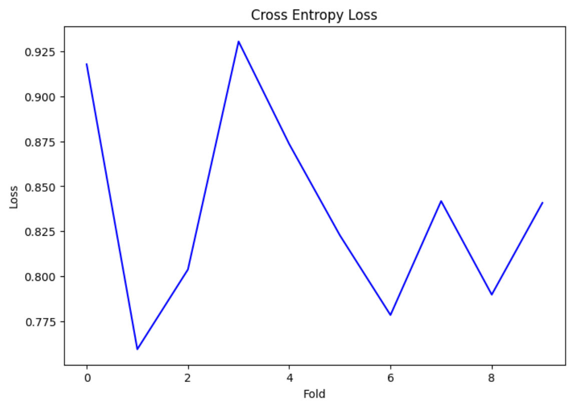
```
Avant l'upsampling:
        Classe 0: 789 exemples
        Classe 1: 117 exemples

Après l'upsampling:
        Classe 0: 789 exemples
        Classe 1: 789 exemples
```

On passe d'une classe ayant ~1200 exemples à ~800

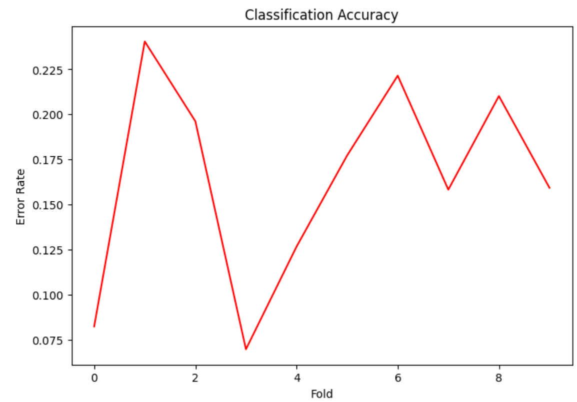
Naives Bayes

```
In [ ]: kfold = KFold(n_splits=10, shuffle=True, random_state=42)  # Définir le nombre de p
        naive_bayes_classifier = MultinomialNB()

        # Effectuer la validation croisée k-fold
        naive_scores = cross_val_score(naive_bayes_classifier, X_train_autre_resampled, Y_t

        # Effectuer la validation croisée k-fold et obtenir les prédictions
        y_pred_cv = cross_val_predict(naive_bayes_classifier, X_train_autre_resampled, Y_tr

        # Calculer la matrice de confusion
        conf_matrix = confusion_matrix(Y_train_autre_resampled, y_pred_cv)

        # Afficher les scores de validation croisée
        print("Scores de validation croisée :", naive_scores)
        print("Moyenne des scores de validation croisée :", naive_scores.mean())

        plot_curves(naive_scores)
        plot_curves_confusion(conf_matrix, ['False or True','Other'])
```

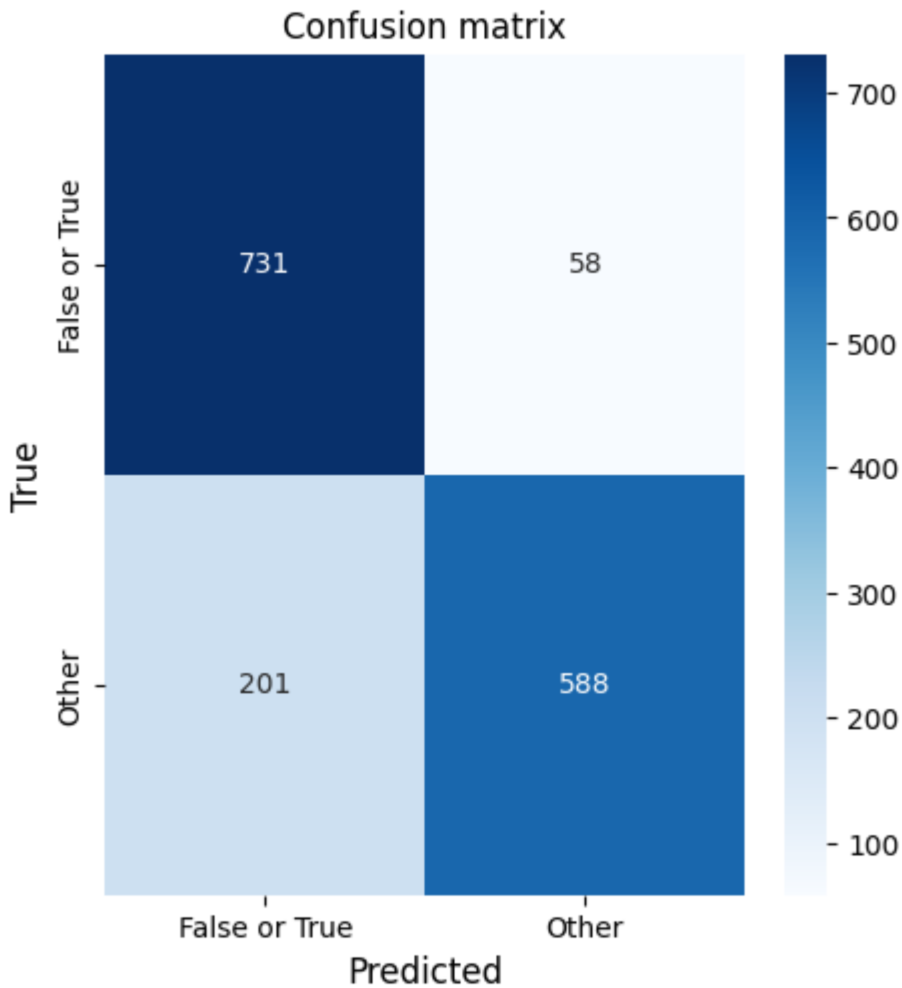
```
Scores de validation croisée : [0.91772152 0.75949367 0.80379747 0.93037975 0.87341
772 0.82278481
 0.77848101 0.84177215 0.78980892 0.84076433]
Moyenne des scores de validation croisée : 0.8358421349673467
```



Cross Entropy Loss
Loss
Fold
0.925
0.900
0.875
0.850
0.825
0.800
0.775
0
2
4
6
8



Classification Accuracy
Error Rate
Fold
0.225
0.200
0.175
0.150
0.125
0.100
0.075
0
2
4
6
8

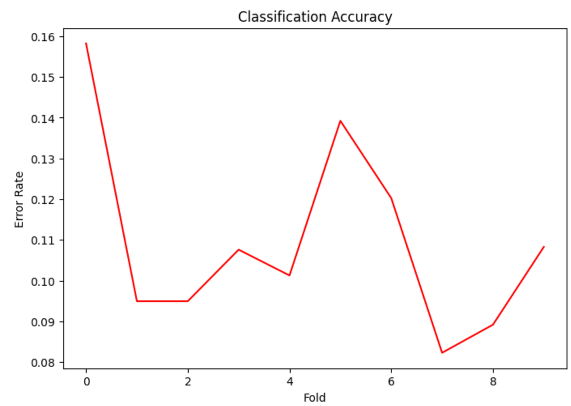


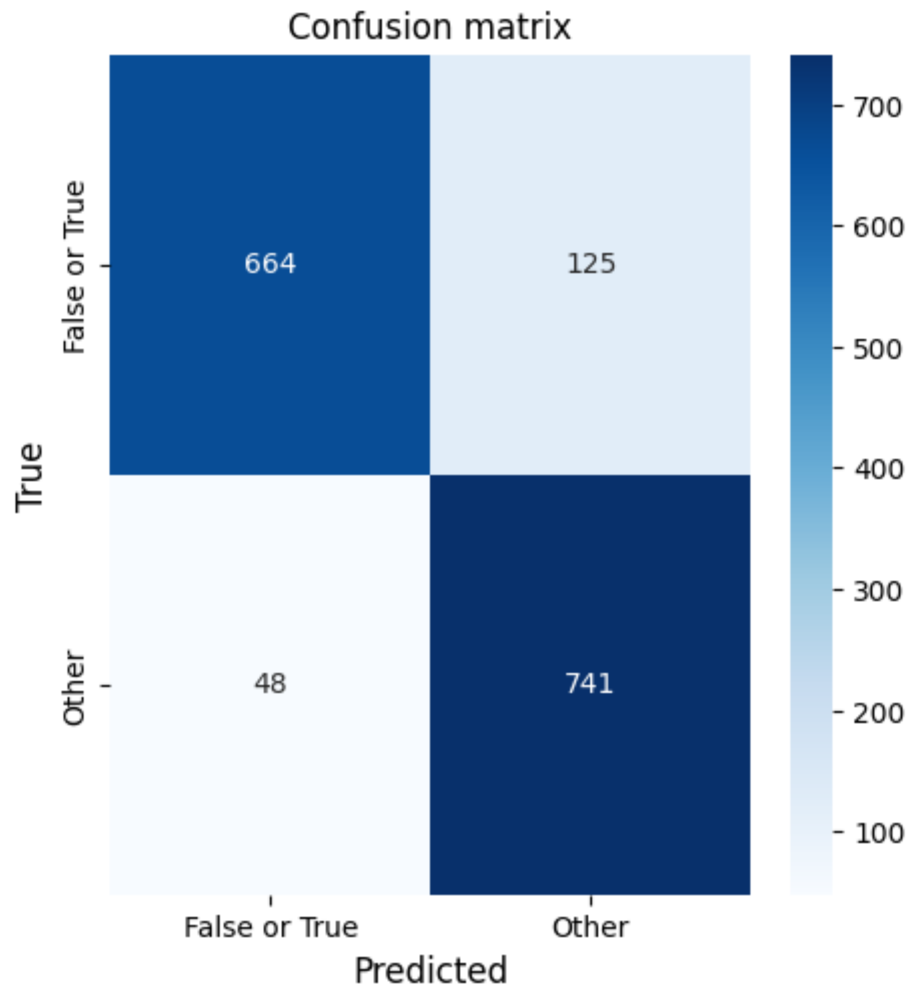
Confusion matrix
True
Predicted
False or True
Other
731
58
201
588
700
600
500
400
300
200
100

SVC

In [ ]:
```
# Création du classifieur SVC
clf_SVC = SVC(kernel='linear')

kfold = KFold(n_splits=10, shuffle=True, random_state=42)
svc_scores = cross_val_score(clf_SVC, X_train_autre_resampled, Y_train_autre_resamp

# Prédiction avec validation croisée
y_pred_cv = cross_val_predict(clf_SVC, X_train_autre_resampled, Y_train_autre_resam

# Calcul de la matrice de confusion
conf_matrix = confusion_matrix(Y_train_autre_resampled, y_pred_cv)

print("Scores de validation croisée :", svc_scores)
print("Moyenne des scores de validation croisée :", svc_scores.mean())

plot_curves(svc_scores)
plot_curves_confusion(conf_matrix, ['False or True','Other'])
```

```
Scores de validation croisée : [0.84177215 0.90506329 0.90506329 0.89240506 0.89873
418 0.86075949
 0.87974684 0.91772152 0.91082803 0.89171975]
Moyenne des scores de validation croisée : 0.8903813593485449
```



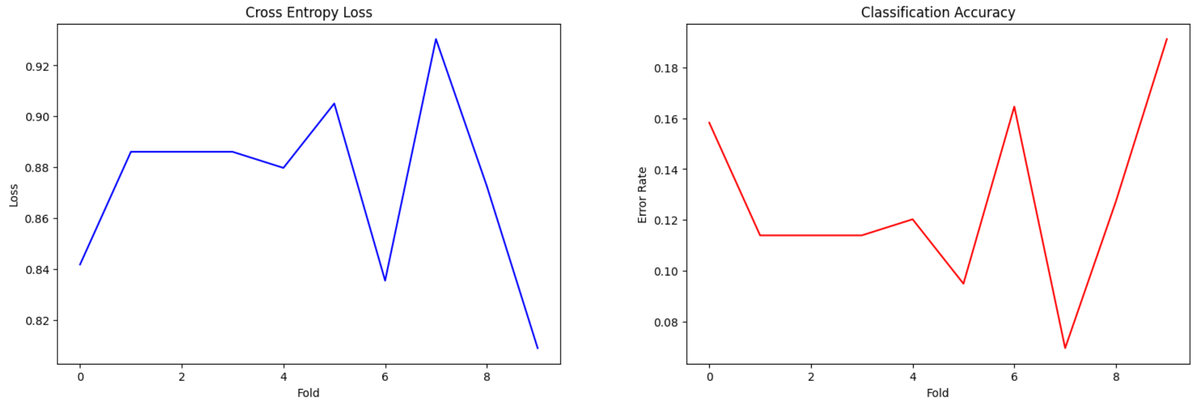


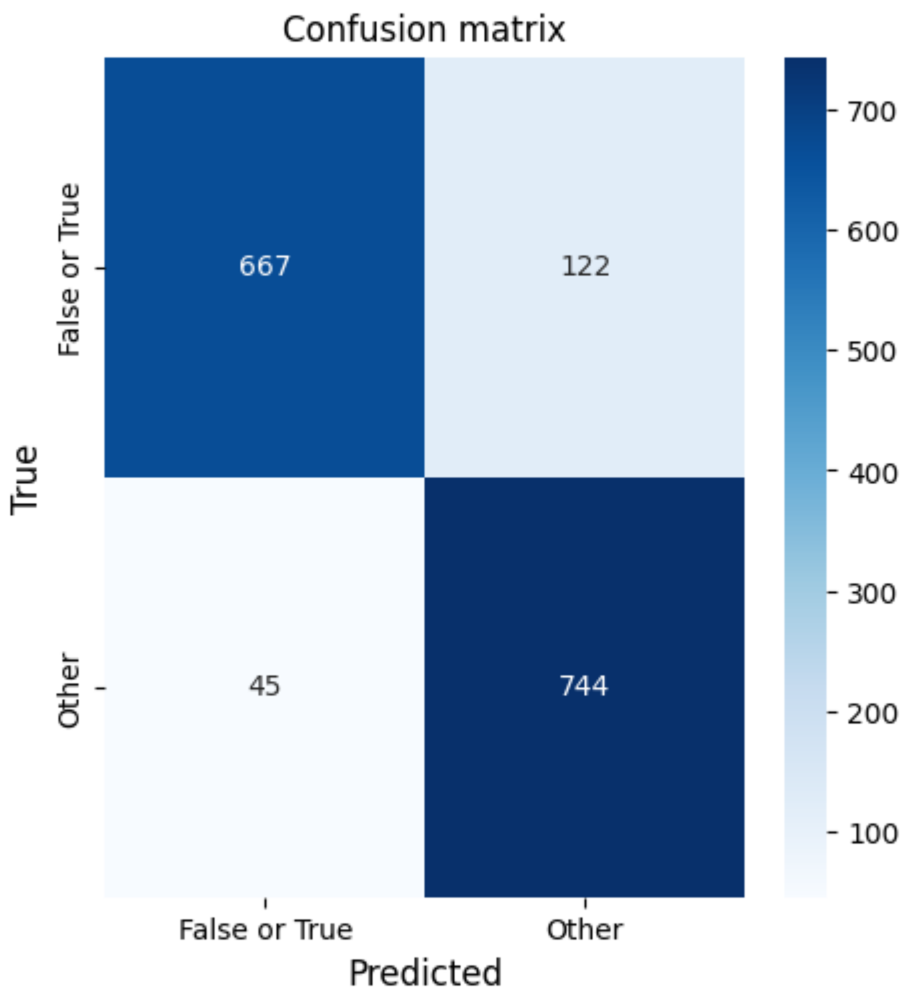
Decision Tree

In [ ]:

```
# Division des données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X_train_autre_resampled, Y_trai

# Initialisation et entraînement du modèle d'arbre de décision
clf_Tree = DecisionTreeClassifier()


# Prédiction sur l'ensemble de test
y_pred_cv = cross_val_predict(clf_SVC, X_train_autre_resampled, Y_train_autre_resam

# Calcul de la matrice de confusion
conf_matrix = confusion_matrix(Y_train_autre_resampled, y_pred_cv)

kfold = KFold(n_splits=10, shuffle=True, random_state=42)
decision_scores = cross_val_score(clf_Tree, X_train_autre_resampled, Y_train_autre_

print("Scores de validation croisée :", decision_scores)
print("Moyenne des scores de validation croisée :", decision_scores.mean())

plot_curves(decision_scores)
plot_curves_confusion(conf_matrix, ['False or True', 'Other'])
```

```
Scores de validation croisée : [0.84177215 0.88607595 0.88607595 0.88607595 0.87974
684 0.90506329
 0.83544304 0.93037975 0.87261146 0.8089172 ]
Moyenne des scores de validation croisée : 0.8732161573812787
```





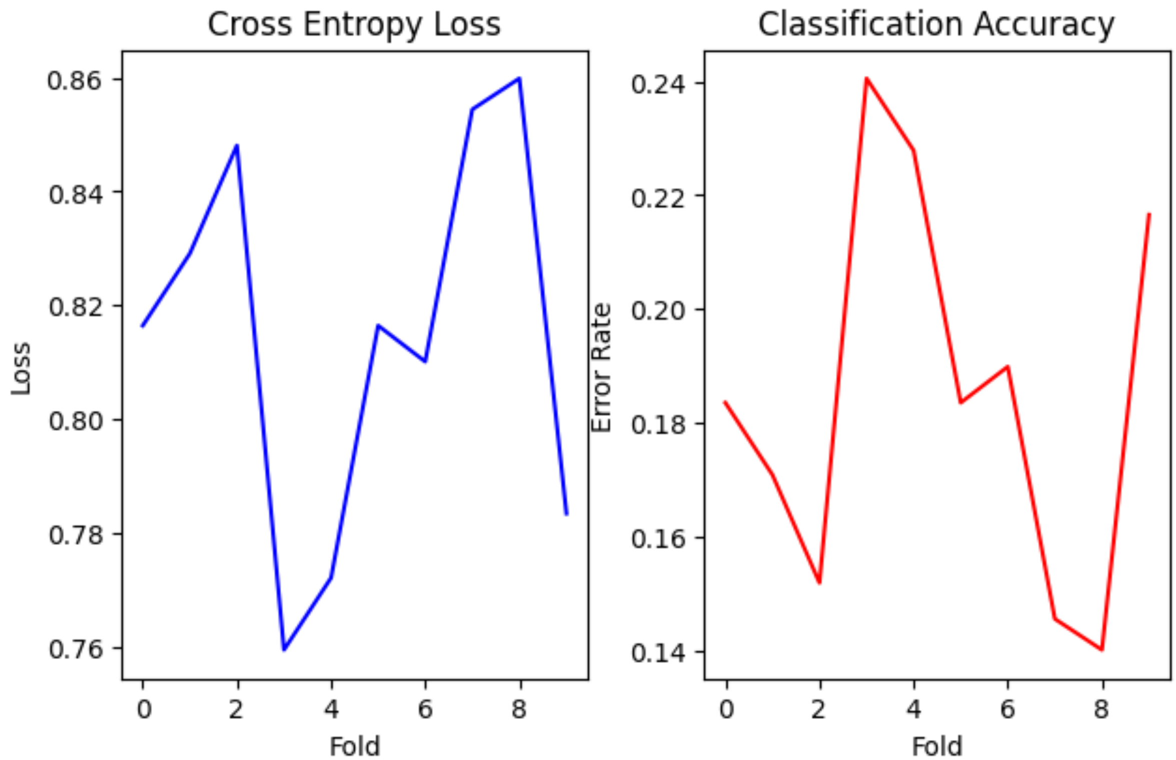
KNN

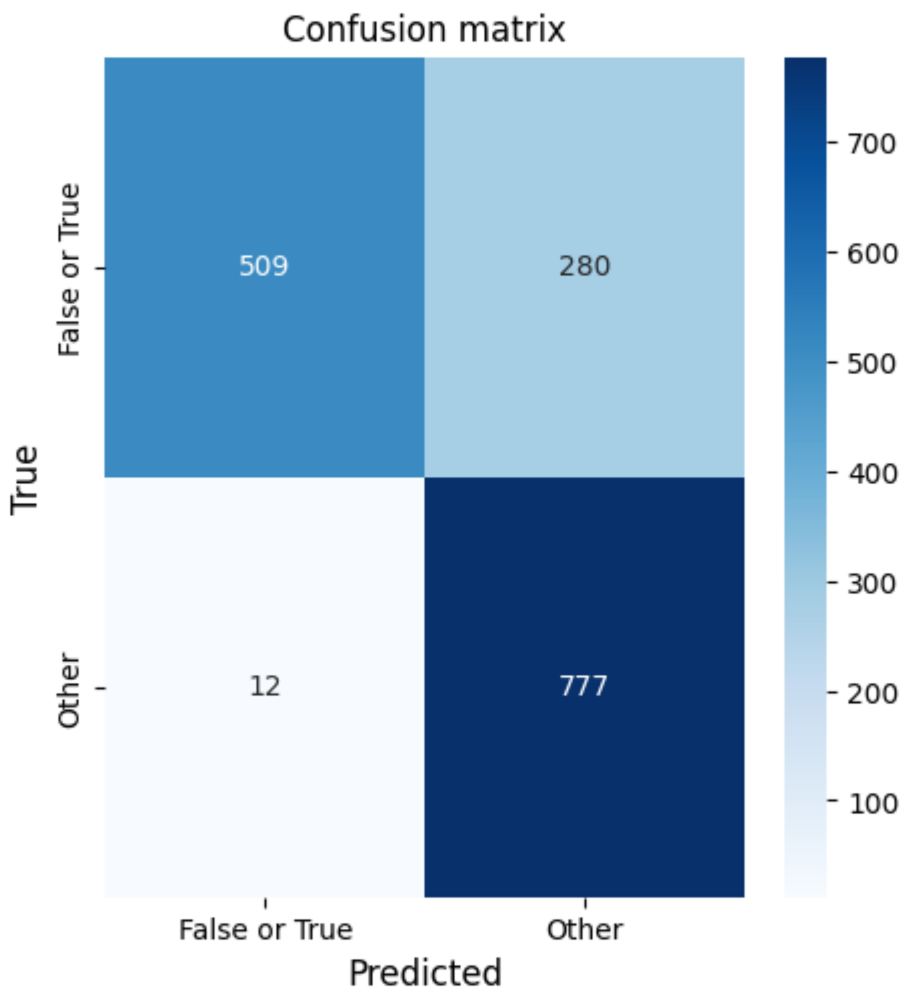
In [ ]:

```python
# Normaliser les données (important pour KNN)
scaler = StandardScaler(with_mean=False)
scores, abcisse = list(), list()


for i in range(1, 10):
  # Initialiser le classifieur KNN
  k = i  # Nombre de voisins à considérer
  knn_classifier = KNeighborsClassifier(n_neighbors=k)

  kfold = KFold(n_splits=10, shuffle=True, random_state=42)
  knn_scores = cross_val_score(knn_classifier, X_train_autre_resampled, Y_train_aut
  scores.append(knn_scores.mean())
  abcisse.append(i)

# Trouver le meilleur nombre de voisins
best_k = abcisse[scores.index(max(scores))]
print("Meilleur nombre de voisins :", best_k)
knn_classifier = KNeighborsClassifier(n_neighbors=best_k)
kfold = KFold(n_splits=10, shuffle=True, random_state=42)
knn_score = cross_val_score(knn_classifier, X_train_autre_resampled, Y_train_autre_

# Entraîner le modèle KNN avec le meilleur nombre de voisins sur toutes les données
best_knn_classifier = KNeighborsClassifier(n_neighbors=best_k)
best_knn_classifier.fit(X_train_autre_resampled, Y_train_autre_resampled)

# Prédiction sur l'ensemble de test
#y_pred_cv = best_knn_classifier.predict(X_train_autre_resampled)
y_pred_cv = cross_val_predict(best_knn_classifier, X_train_autre_resampled, Y_train

# Calculer la matrice de confusion
conf_matrix = confusion_matrix(Y_train_autre_resampled, y_pred_cv)

# Calculer la précision
#accuracy_KNN = accuracy_score(y_test, y_pred)
#print(f"Accuracy KNN : {accuracy_KNN * 100:.2f}")
plt.title('KNN en fonction des k voisins')
plt.plot(abcisse, scores)
plt.ylabel('Score de validation')
plt.xlabel('Nombre de voisins')
#plt.xlim(1,10)
print("Scores de validation croisée :", knn_score)
print("Moyenne des scores de validation croisée :", knn_score.mean())

plot_curves(knn_score)
plot_curves_confusion(conf_matrix, ['False or True', 'Other'])
```

```
Meilleur nombre de voisins : 2
Scores de validation croisée : [0.8164557  0.82911392 0.84810127 0.75949367 0.77215
19  0.8164557
 0.81012658 0.85443038 0.85987261 0.78343949]
Moyenne des scores de validation croisée : 0.8149641215834877
```

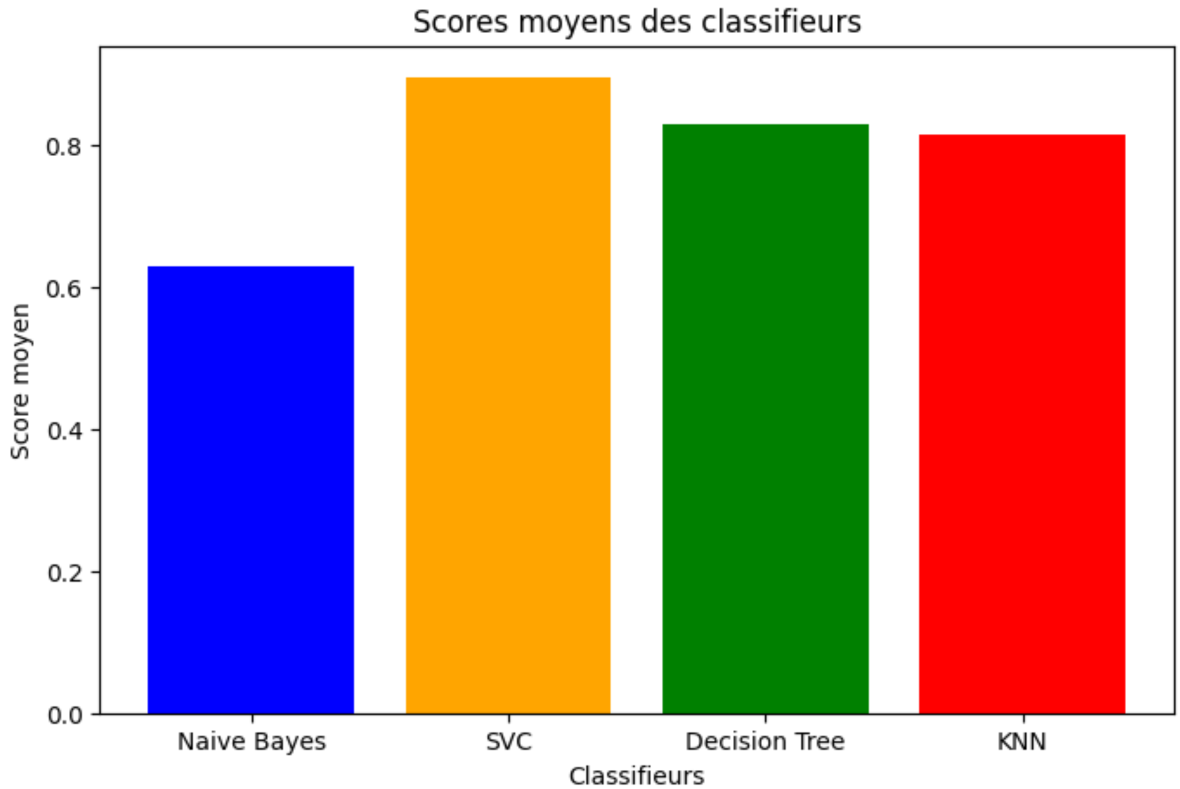
```
<ipython-input-81-1de4a8dbf2fd>:19: MatplotlibDeprecationWarning: Auto-removal of o
verlapping axes is deprecated since 3.6 and will be removed two minor releases late
r; explicitly call ax.remove() as needed.
  plt.subplot(121)
```

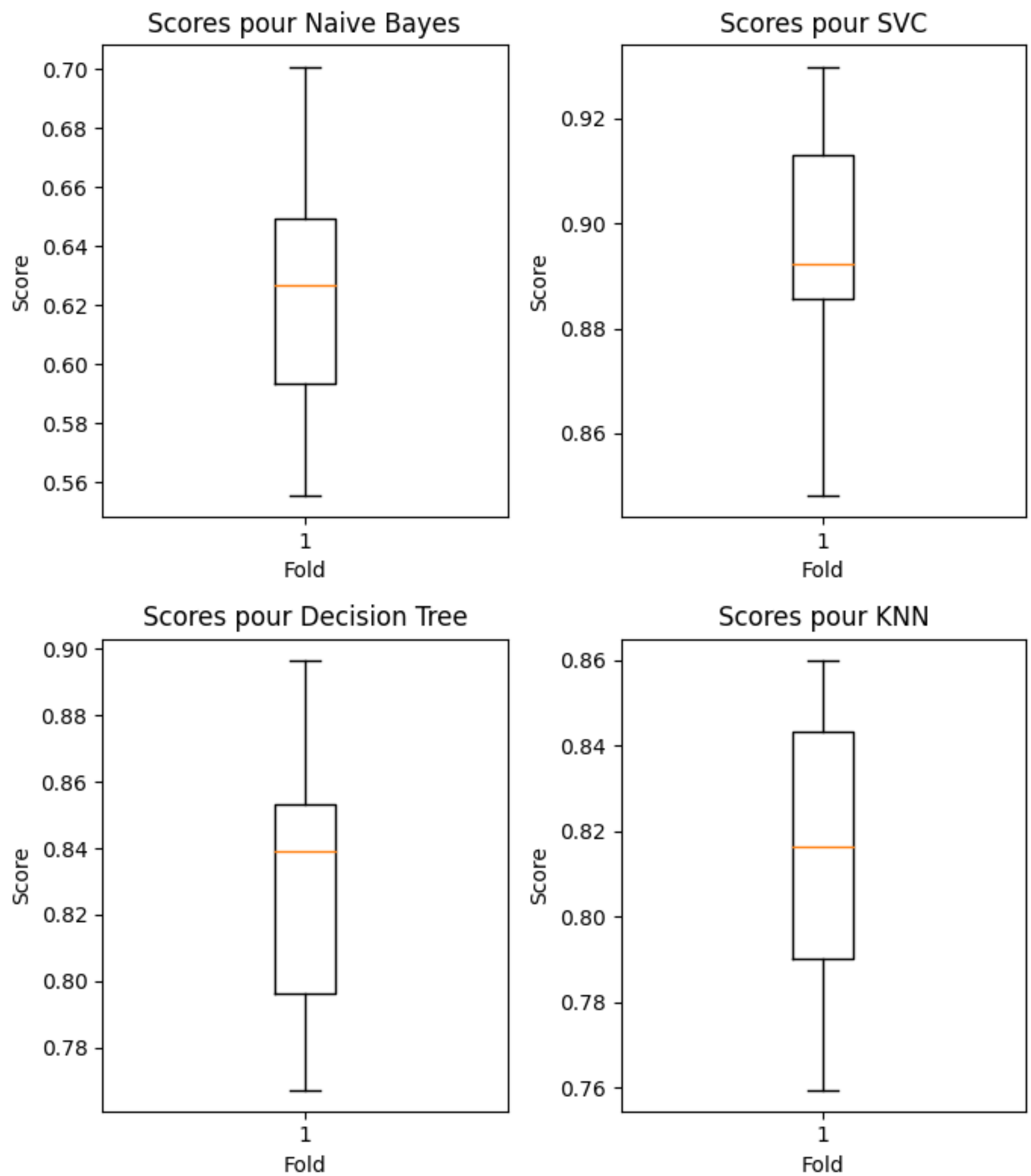

Cross Entropy Loss
Loss
Fold
0.86
0.84
0.82
0.80
0.78
0.76
0
2
4
6
8
Classification Accuracy
Error Rate
Fold
0.24
0.22
0.20
0.18
0.16
0.14
0
2
4
6
8


Confusion matrix
True
Predicted
False or True
Other
False or True
Other
509
280
12
777
700
600
500
400
300
200
100

Evaluation

In [ ]:
```
plot_curves_results(naive_scores, svc_scores, decision_scores, knn_score)
```





Comparaisons des modèles

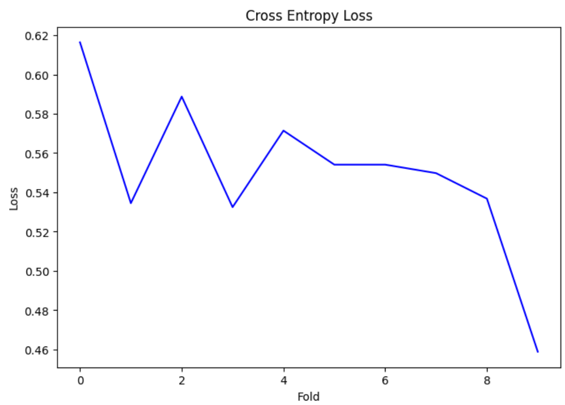
Dans la dernière partie, nous allons utilisé d'autres méthodes afin de voir l'impact de ces dernières sur la qualité de la classification.

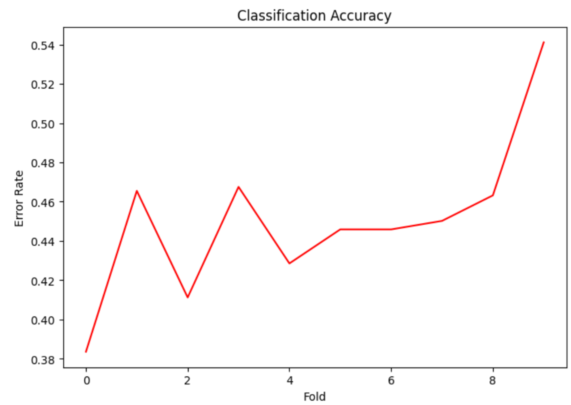
Différents paramètres des classifieurs

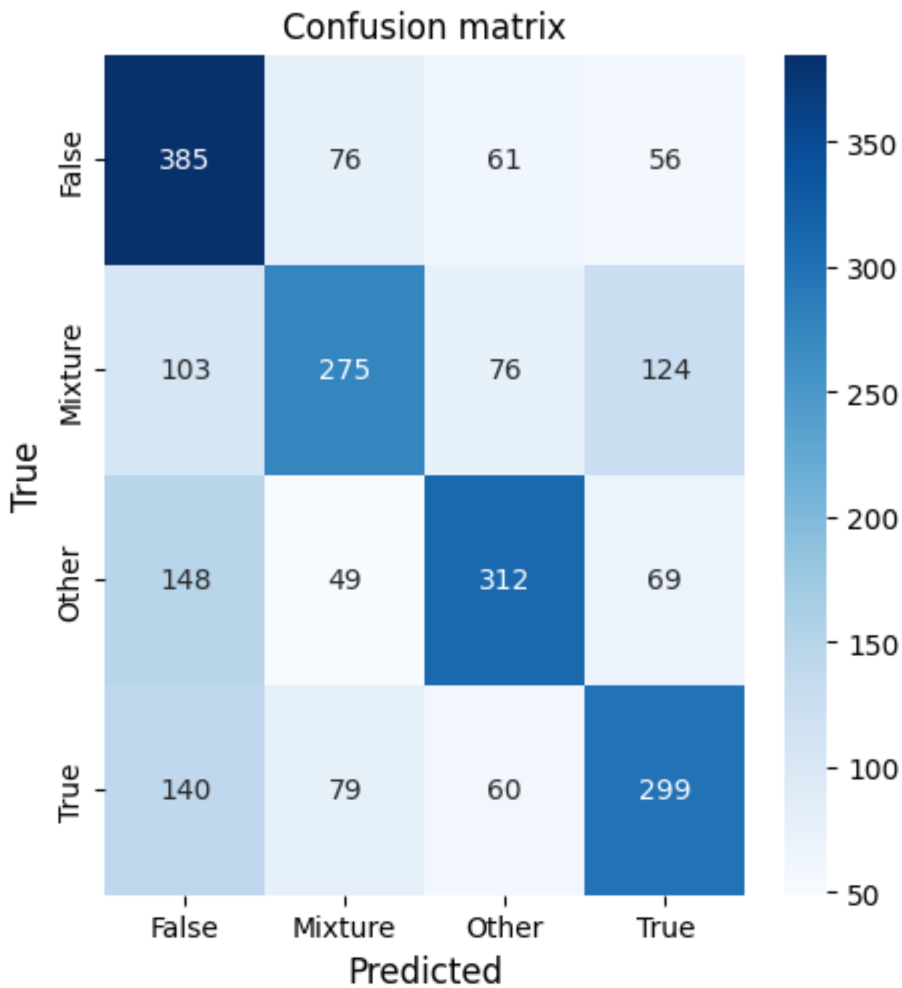
Naive Bayes ComplementNB

In [ ]:
```
kfold = KFold(n_splits=10, shuffle=True, random_state=42)  # Définir le nombre de p
naive_bayes_classifier = ComplementNB()

# Effectuer la validation croisée k-fold
naive_scores = cross_val_score(naive_bayes_classifier, X_train_resampled, Y_train_r

# Effectuer la validation croisée k-fold et obtenir les prédictions
y_pred_cv = cross_val_predict(naive_bayes_classifier, X_train_resampled, Y_train_re

# Calculer la matrice de confusion
conf_matrix = confusion_matrix(Y_train_resampled, y_pred_cv)

# Afficher les scores de validation croisée
print("Scores de validation croisée :", naive_scores)
print("Moyenne des scores de validation croisée :", naive_scores.mean())

plot_curves(naive_scores)
plot_curves_confusion(conf_matrix, ['False', 'Mixture', 'Other', 'True'])
```

```
Scores de validation croisée : [0.61637931 0.53448276 0.58874459 0.53246753 0.57142
857 0.55411255
 0.55411255 0.54978355 0.53679654 0.45887446]
Moyenne des scores de validation croisée : 0.5497182415285864
```



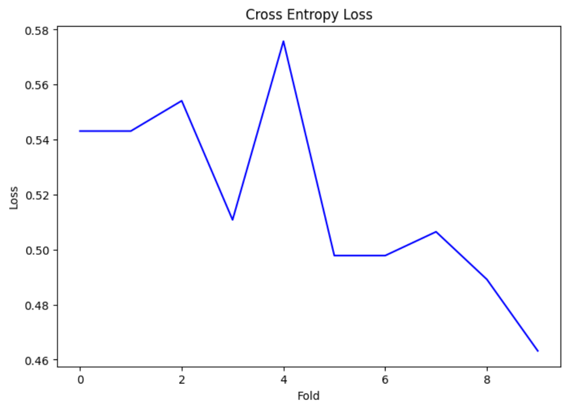


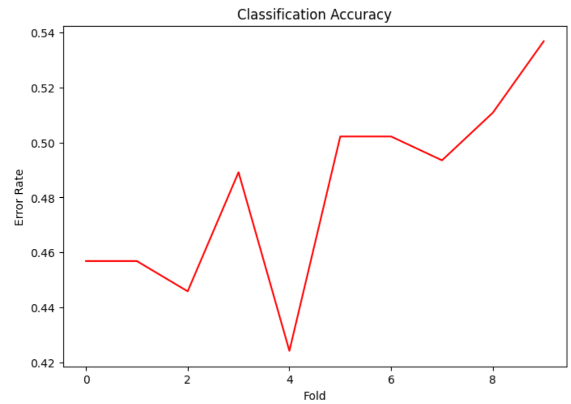


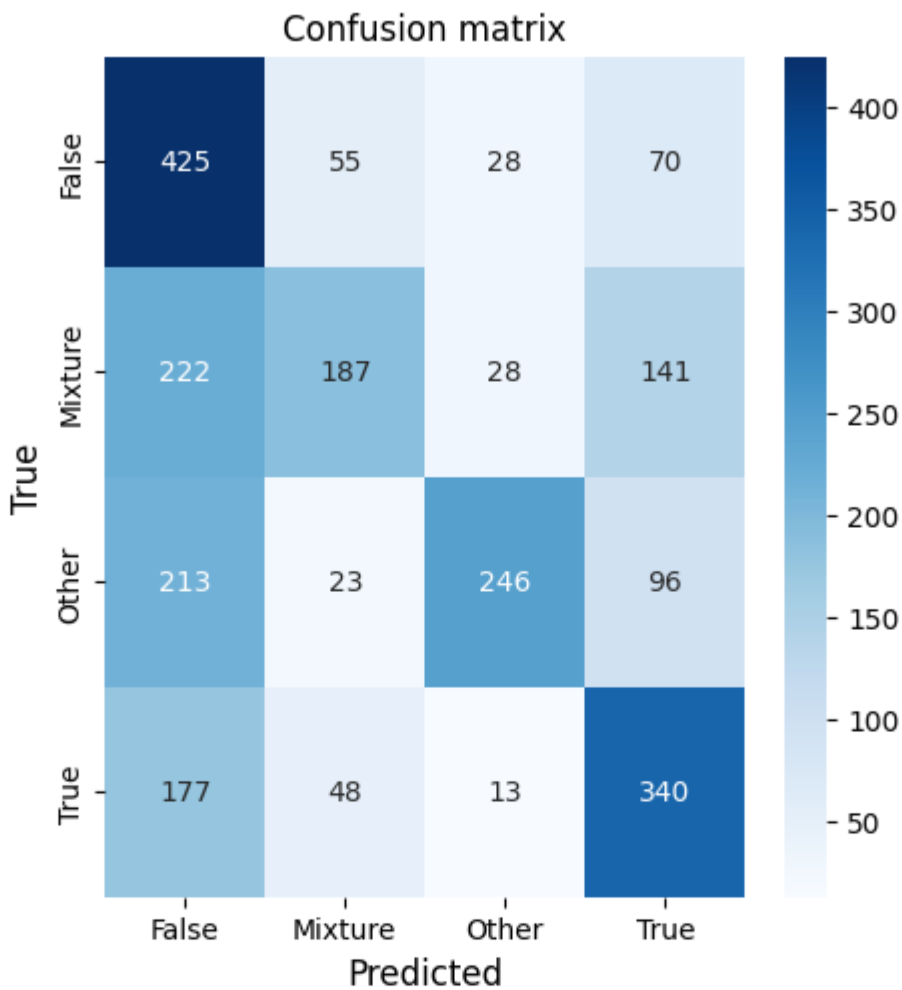
Naive Bayes BernouilliNB

```
In [ ]: kfold = KFold(n_splits=10, shuffle=True, random_state=42)  # Définir le nombre de p
naive_bayes_classifier = BernoulliNB()

# Effectuer la validation croisée k-fold
naive_scores = cross_val_score(naive_bayes_classifier, X_train_resampled, Y_train_r

# Effectuer la validation croisée k-fold et obtenir les prédictions
y_pred_cv = cross_val_predict(naive_bayes_classifier, X_train_resampled, Y_train_re

# Calculer la matrice de confusion
conf_matrix = confusion_matrix(Y_train_resampled, y_pred_cv)

# Afficher les scores de validation croisée
print("Scores de validation croisée :", naive_scores)
print("Moyenne des scores de validation croisée :", naive_scores.mean())

plot_curves(naive_scores)
plot_curves_confusion(conf_matrix, ['False', 'Mixture', 'Other', 'True'])
```

```
Scores de validation croisée : [0.54310345 0.54310345 0.55411255 0.51082251 0.57575
758 0.4978355
 0.4978355  0.50649351 0.48917749 0.46320346]
Moyenne des scores de validation croisée : 0.5181444991789819
```


Cross Entropy Loss
Loss
Fold
0.58
0.56
0.54
0.52
0.50
0.48
0.46
0
2
4
6
8

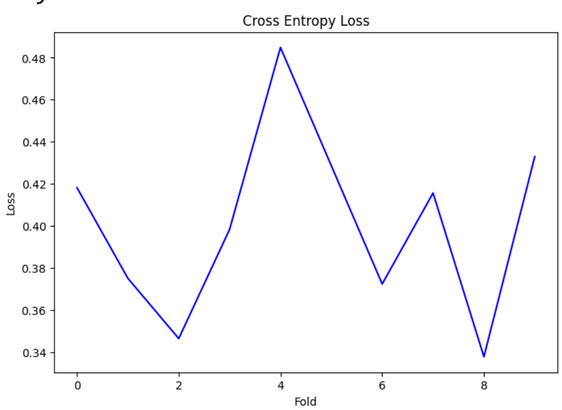

Classification Accuracy
Error Rate
Fold
0.54
0.52
0.50
0.48
0.46
0.44
0.42
0
2
4
6
8

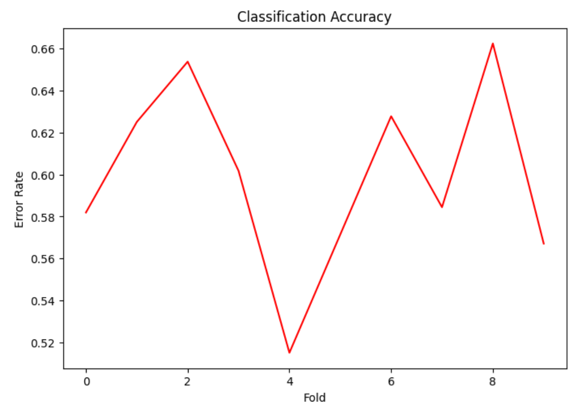

Confusion matrix
True
Predicted
False
Mixture
Other
True
425
55
28
70
222
187
28
141
213
23
246
96
177
48
13
340
400
350
300
250
200
150
100
50

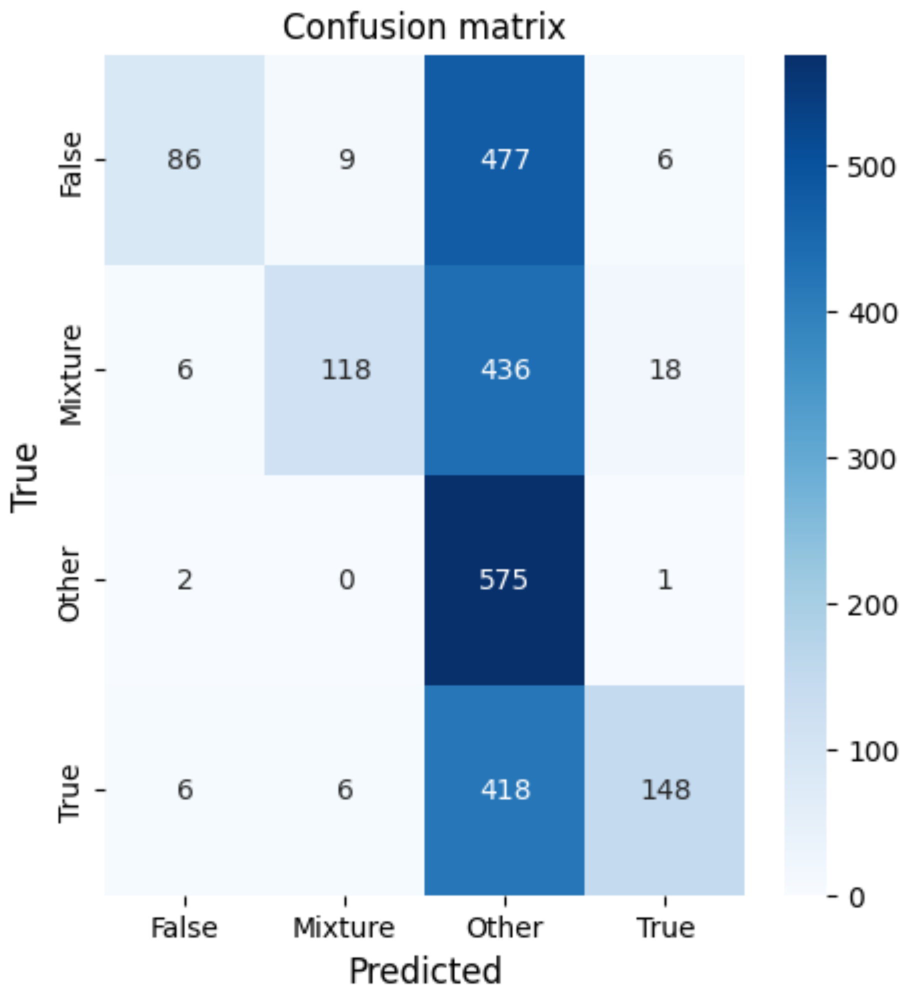
SVC poly

```
In [ ]: # classifieur SVC
clf_SVC = SVC(kernel='poly', degree=3)

kfold = KFold(n_splits=10, shuffle=True, random_state=42)
svc_scores = cross_val_score(clf_SVC, X_train_resampled, Y_train_resampled, cv=kfol

# Prédiction avec validation croisée
y_pred_cv = cross_val_predict(clf_SVC, X_train_resampled, Y_train_resampled, cv=kfo

# Calcul de la matrice de confusion
conf_matrix = confusion_matrix(Y_train_resampled, y_pred_cv)

print("Scores de validation croisée :", svc_scores)
print("Moyenne des scores de validation croisée :", svc_scores.mean())

plot_curves(svc_scores)
plot_curves_confusion(conf_matrix, ['False', 'Mixture', 'Other', 'True'])
```

```
Scores de validation croisée : [0.41810345 0.375      0.34632035 0.3982684  0.48484
848 0.42857143
 0.37229437 0.41558442 0.33766234 0.43290043]
Moyenne des scores de validation croisée : 0.4009553664726079
```



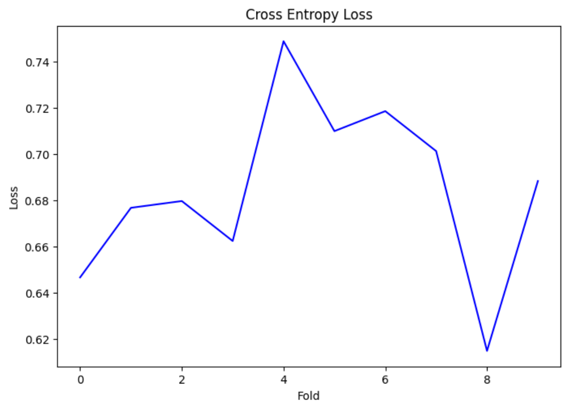


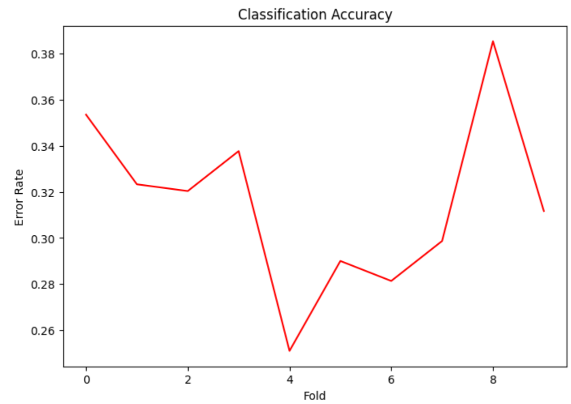


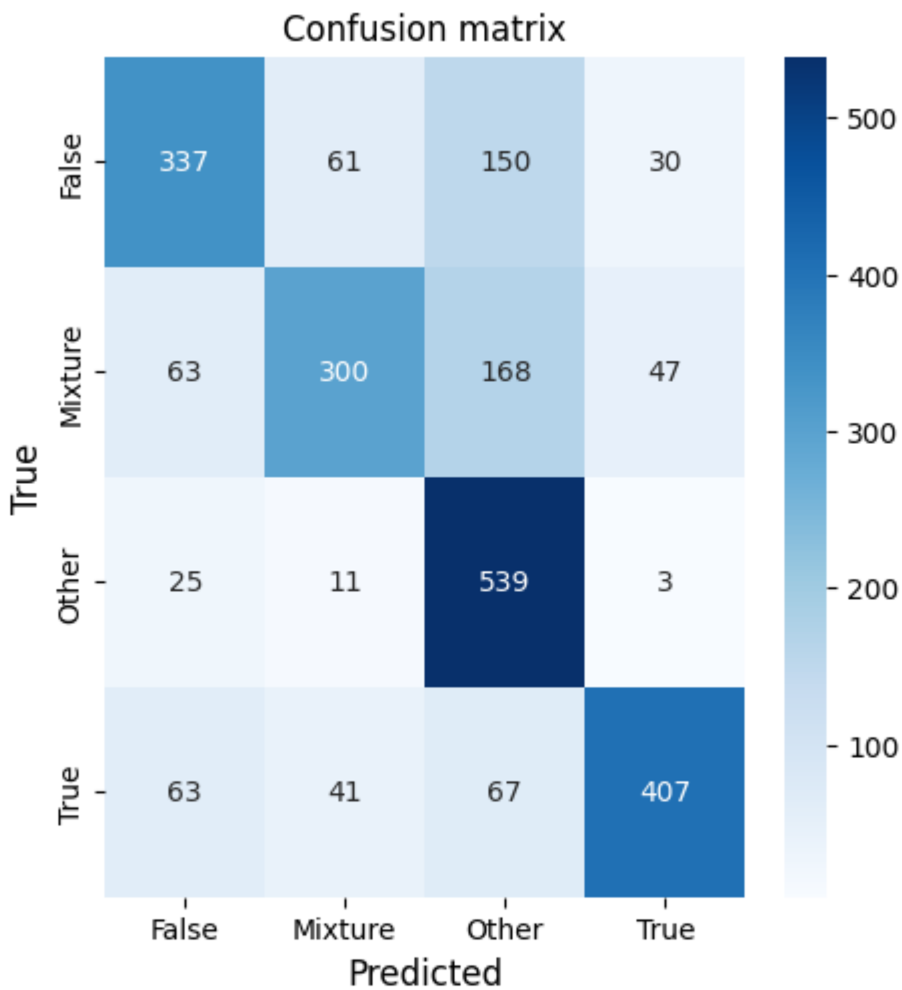
SVC rbf

In [ ]:
```
# classifieur SVC
clf_SVC = SVC(kernel='rbf')

kfold = KFold(n_splits=10, shuffle=True, random_state=42)
svc_scores = cross_val_score(clf_SVC, X_train_resampled, Y_train_resampled, cv=kfol

# Prédiction avec validation croisée
y_pred_cv = cross_val_predict(clf_SVC, X_train_resampled, Y_train_resampled, cv=kfo

# Calcul de la matrice de confusion
conf_matrix = confusion_matrix(Y_train_resampled, y_pred_cv)

print("Scores de validation croisée :", svc_scores)
print("Moyenne des scores de validation croisée :", svc_scores.mean())

plot_curves(svc_scores)
plot_curves_confusion(conf_matrix, ['False', 'Mixture', 'Other', 'True'])
```

```
Scores de validation croisée : [0.64655172 0.67672414 0.67965368 0.66233766 0.74891
775 0.70995671
 0.71861472 0.7012987  0.61471861 0.68831169]
Moyenne des scores de validation croisée : 0.6847085385878489
```



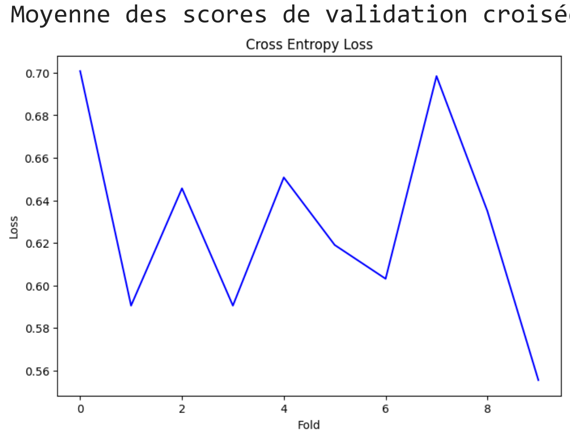


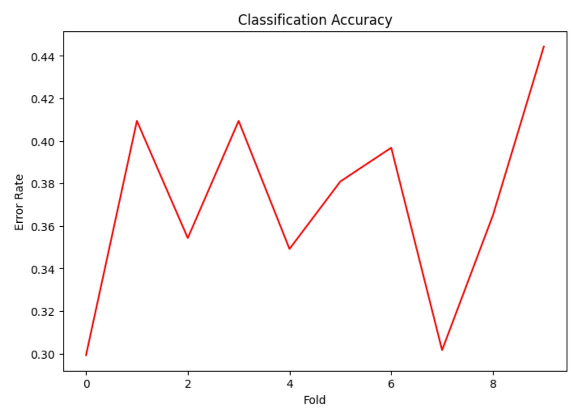


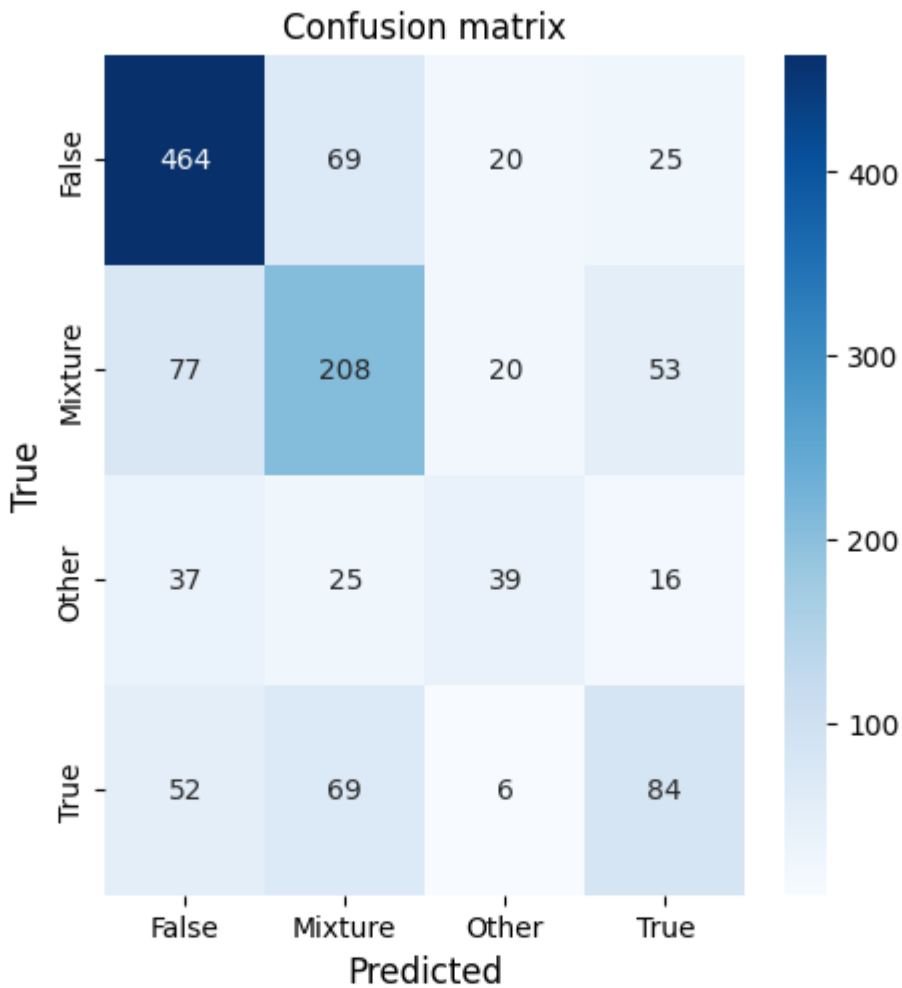
Utilisation des variables vectorisées

In [ ]:
```
ngram_range = (1, 2)  # Utiliser des unigrammes et des bigrammes
vectorizer = TfidfVectorizer(ngram_range=ngram_range, min_df=0.005, max_df=0.9)

# Normaliser le texte
normalizedText = normalized['text'].copy()
Y_normalized = label.transform(normalized["our rating"])

vectorizedText = vectorizer.fit_transform(normalizedText)

scaler = StandardScaler(with_mean=False)  # with_mean=False pour éviter un problème
scaled = scaler.fit_transform(vectorizedText)

vectorized = pd.DataFrame(data=scaled.toarray(), columns=vectorizer.get_feature_nam

# Définir le nombre de plis
kfold = KFold(n_splits=10, shuffle=True, random_state=42)

# Naive Bayes
naive_bayes_classifier = MultinomialNB()

# Effectuer la validation croisée k-fold
naive_scores = cross_val_score(naive_bayes_classifier, vectorized, Y_normalized, cv

# Prédictions par validation croisée
y_pred_cv = cross_val_predict(naive_bayes_classifier, vectorized, Y_normalized, cv=

# Calculer la matrice de confusion
conf_matrix = confusion_matrix(Y_normalized, y_pred_cv)

print("Scores de validation croisée :", naive_scores)
print("Moyenne des scores de validation croisée :", naive_scores.mean())

# Plots
plot_curves(naive_scores)
plot_curves_confusion(conf_matrix, ['False', 'Mixture', 'Other', 'True'])
```

```
Scores de validation croisée : [0.7007874  0.59055118 0.64566929 0.59055118 0.65079
365 0.61904762
 0.6031746  0.6984127  0.63492063 0.55555556]
Moyenne des scores de validation croisée : 0.6289463817022872
```






Confusion matrix
True
False
Mixture
Other
True
464
69
20
25
77
208
20
53
37
25
39
16
52
69
6
84
False
Mixture
Other
True
Predicted
400
300
200
100