UNIVERSITE SULTAN MOULAY SLIMANE
ECOLE NATIONALE DES SCIENCES APPLIQUEES
KHOURIBGA

Master Big Data et Aide à la Décision

Realisé par

MARZAQ KHALID

- Recherche Opérationnelle 2 -
TP 2 : Support Vector Machine (SVM)

Année Universitaire : 2023-2024

Partie 1

Mise en oeuvre - SVM -

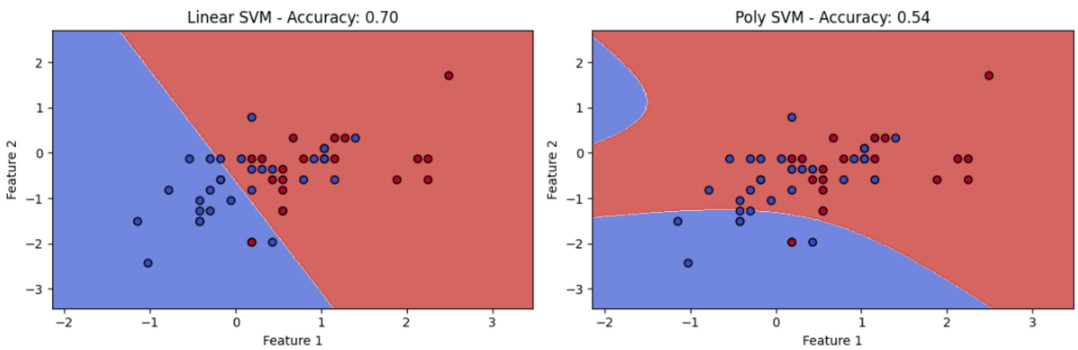
1.1 SVM Linéaire vs SVM Polynomial [1]

Nous avons utilise scikit-learn pour entraîner et visualiser deux modèles SVM (linéaire et polynomial) sur le jeu de données Iris. Les kernels utilisés sont linéaire et polynomial de degré 3. Le dataset Iris est chargé, divisé en ensembles de formation et de test (50% - 50%), et les caractéristiques sont standardisées(Standardisation est un processus qui vise à mettre à l'échelle les caractéristiques de manière à ce qu'elles aient une moyenne nulle et un écart type de 1).

Linear kernel vs Poly kernel

```
[9] import numpy as np
    import matplotlib.pyplot as plt
    from sklearn.svm import SVC
    from sklearn import datasets
    from sklearn.preprocessing import StandardScaler
    from sklearn.model_selection import train_test_split

    # Function to visualize decision boundaries
    def plot_decision_boundary(X, y, clf, title, ax):
        h = .02
        x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
        y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
        xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
        Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
        Z = Z.reshape(xx.shape)

        ax.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
        ax.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm, edgecolors='k', marker='o')
        ax.set_title(title)
        ax.set_xlabel('Feature 1')
        ax.set_ylabel('Feature 2')
```

```
[12] scaler = StandardScaler()
     iris = datasets.load_iris()
     X = iris.data
     X = scaler.fit_transform(X)
     y = iris.target
     X = X[y != 0, :2]
     y = y[y != 0]

     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=42)

     # Linear Kernel SVM
     linear_svm = SVC(kernel='linear')
     linear_svm.fit(X_train, y_train)
     linear_accuracy = linear_svm.score(X_test, y_test)

     # Polynomial Kernel SVM
     poly_svm = SVC(kernel='poly', degree=3)
     poly_svm.fit(X_train, y_train)
     poly_accuracy = poly_svm.score(X_test, y_test)

     # Create subplots
     fig, axes = plt.subplots(1, 2, figsize=(12, 4))

     # Visualize decision boundaries
     plot_decision_boundary(X_test, y_test, linear_svm, f'Linear SVM - Accuracy: {linear_accuracy:.2f}', axes[0])
     plot_decision_boundary(X_test, y_test, poly_svm, f'Poly SVM - Accuracy: {poly_accuracy:.2f}', axes[1])

     # Adjust layout and show plots
     plt.tight_layout()
     plt.show()
```



D'après les résultats obtenus, l'accuracy (précision) du modèle SVM avec un kernel linéaire est de 0.7, tandis que celle avec un kernel polynomial est de 0.54. Ces scores indiquent que le modèle avec kernel linéaire a une performance globalement meilleure que celui avec kernel polynomial sur l'ensemble de test. Il est possible que, dans ce cas spécifique, la relation linéaire entre les caractéristiques soit mieux capturée par le SVM avec un kernel linéaire, conduisant à de meilleures prédictions.

1.2 Démonstration

Montrez que le problème primal résolu par le SVM peut se réécrire de la façon suivante :

$$\underset{w\in H, w_0\in\mathbb{R}}{\arg\min}\left(\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}[1 - y_i(h_w, \Phi(x_i) + w_0)]^+\right)$$

`Démonstration`

Pour montrer que le problème primal du SVM peut se réécrire comme indiqué, examinons la formulation standard du problème primal du SVM pour la séparation linéaire.
La formulation standard du problème primal est la suivante :

$$\text{minimize} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i$$

sous les contraintes

$$y_i(w \cdot \Phi(x_i) + w_0) \geq 1 - \xi_i$$

$$\xi_i \geq 0$$

pour $i = 1, 2, \ldots, n$.
Maintenant, introduisons les variables d'écart ξ_i pour chaque échantillon. La fonction de perte (hinge loss) est $\max(0, 1 - y_i(w \cdot \Phi(x_i) + w_0))$, et ξ_i permet de quantifier le dépassement de cette fonction de perte au-delà de 1. En introduisant ces variables d'écart, la somme des pertes devient $\sum_{i=1}^{n} \xi_i$.
La contrainte $y_i(w \cdot \Phi(x_i) + w_0) \geq 1 - \xi_i$ devient $1 - y_i(w \cdot \Phi(x_i) + w_0) \leq \xi_i$.
En combinant tout cela, on obtient la formulation réécrite :

$$\text{minimize} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}[1 - y_i(w \cdot \Phi(x_i) + w_0)]^+$$

où $[1 - y_i(w \cdot \Phi(x_i) + w_0)]^+$ représente le terme de perte positive, et ξ_i est implicitement inclus dans ce terme, reflétant le dépassement de la fonction de perte hinge au-delà de 1.

1.3 Explication

Expliquez la phrase : `« Un SVM minimise l'erreur de classification à l'aide d'un majorant convexe de la fonction qui vaut 1 lorsque la marge est négative et 0 sinon ». La fonction` $x \to [1 - x]_+ = \max(0, 1 - x)$ `est appelée Hinge (charnière en français).`

La phrase "un SVM minimise l'erreur de classification à l'aide d'un majorant convexe de la fonction qui vaut 1 quand la marge est négative et 0 sinon" signifie que le Support Vector Machine (SVM) cherche à minimiser l'erreur de classification en utilisant une fonction convexe qui attribue une pénalité lorsque la marge entre les classes est négative.

La fonction $x \to [1 - x]_+ = \max(0, 1 - x)$, est appelée Hinge (charnière en français). Elle mesure la perte associée à la classification incorrecte. Lorsque la marge est positive (i.e., le point est correctement classé avec une marge positive), la fonction renvoie zéro. Cependant, lorsque la marge est négative, la fonction renvoie une valeur positive qui augmente à mesure que la marge devient plus négative. Ainsi, elle introduit une pénalité pour les erreurs de classification et encourage la recherche d'une marge maximale pour améliorer la généralisation du modèle SVM.

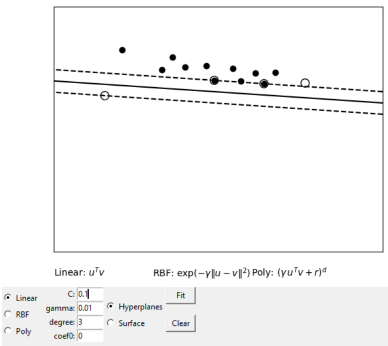
Partie 2

SVM GUI

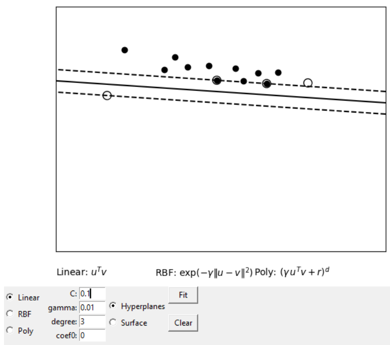
SVM GUI (Graphical User Interface) [2] est une interface graphique pour la librairie scikit-learn en Python. Elle permet de créer des jeux de données, de choisir différents noyaux et paramètres de régularisation pour les machines à vecteurs de support (SVM), d'afficher les résultats en temps réel, et d'explorer l'impact de ces choix sur la classification.

2.1 Influence du paramètre C

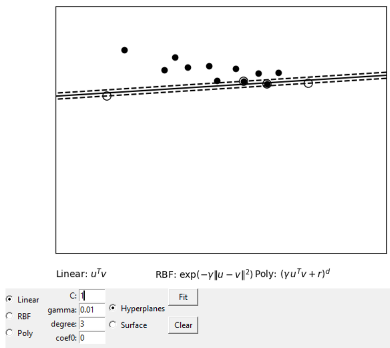
Le paramètre C dans les machines à vecteurs de support (SVM) est un hyperparamètre qui contrôle la pénalité attribuée aux erreurs d'entraînement. Il détermine la marge d'erreur acceptable lors de la création de la frontière de décision.



(a) C = 0.01



(b) C = 0.1



(c) C = 1.0

Un C plus élevé indique une tolérance plus faible aux erreurs, ce qui conduit à une frontière de décision plus précise mais potentiellement à un surajustement. À l'inverse, un C plus faible favorise une marge plus large avec une tolérance plus élevée aux erreurs, ce qui peut améliorer la généralisation du modèle mais au détriment de la précision sur les données d'entraînement.

Partie 3

Classification de visages

Nous entreprenons une tâche de classification des visages en utilisant la célèbre base de données `« Labeled Faces in the Wild » (LFW)`. Pour simplifier notre problème, nous avons choisi de travailler avec deux classes spécifiques, à savoir '`Tony Blair`' et '`Colin Powell`', chacune représentant une personnalité distincte.

Notre ensemble de données est configuré comme suit :

- Nombre de classes : 2 `('Tony Blair' et 'Colin Powell')`
- Nombre d'échantillons d'entraînement : 285
- Nombre d'échantillons de test : 95

L'objectif de cette classification est d'entraîner un modèle capable de discerner et de classifier correctement les visages de Tony Blair et Colin Powell.

Importation des donnees

```
[15] from time import time

     import matplotlib.pyplot as plt
     from scipy.stats import loguniform

     from sklearn.datasets import fetch_lfw_people
     from sklearn.decomposition import PCA
     from sklearn.metrics import ConfusionMatrixDisplay, classification_report
     from sklearn.model_selection import RandomizedSearchCV, train_test_split
     from sklearn.preprocessing import StandardScaler
     from sklearn.svm import SVC
```

```
from sklearn.datasets import fetch_lfw_people
from sklearn.model_selection import train_test_split

# Load the Labeled Faces in the Wild (LFW) dataset
lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4)

# Extract information about the dataset
n_samples, h, w = lfw_people.images.shape
X = lfw_people.data
n_features = X.shape[1]
y = lfw_people.target
target_names1 = lfw_people.target_names
selected_classes = ['Tony Blair', 'Colin Powell']

# Filter target_names to include only the selected classes
target_names = [name for name in target_names1 if name in selected_classes]

# Filter the dataset to include only Tony Blair and Colin Powell
selected_indices = [i for i in range(n_samples) if target_names1[y[i]] in selected_classes]

# Filter y to include only the labels corresponding to selected classes
y_selected = y[selected_indices]

# Use the filtered y to create the selected_indices
selected_indices = [i for i in range(n_samples) if y[i] in y_selected]

X_selected = X[selected_indices]
y_selected = y[selected_indices]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_selected, y_selected, test_size=0.25, random_state=42)

print("Selected dataset size: %s"%target_names)
print("n_classes: %d" % len(selected_classes))
print("n_train : %d"%len(X_train))
print("n_test : %d"%len(X_test))
```

3.1 Données centrées et réduites

La normalisation des données, en les centrant autour de zéro et en réduisant leur échelle, offre une stabilité numérique accrue et favorise une convergence plus rapide des algorithmes d'optimisation. Cette pratique assure également une équité entre les caractéristiques, permettant à toutes de contribuer de manière équilibrée à l'apprentissage du modèle, indépendamment de leur échelle d'origine. En résumé, la normalisation améliore la performance et la robustesse des modèles d'apprentissage automatique.

Normalisation des donnees

```
[21] scaler = StandardScaler()
     X_train = scaler.fit_transform(X_train)
     X_test = scaler.transform(X_test)
```

3.2 Analyse en composantes principales (PCA)

Nous avons utilise l'analyse en composantes principales (PCA) pour extraire les caractéristiques principales (eigenfaces) des visages dans le jeu de données. Il réduit la dimensionnalité des données en projetant les images sur la base orthonormale des eigenfaces, ce qui peut améliorer la prédiction en conservant les informations les plus discriminantes. La réduction de dimension est effectuée avec la classe PCA de scikit-learn, en maintenant seulement les 150 premières composantes principales.

Normalisation des donnees

```
n_components = 150

print(
    "Extracting the top %d eigenfaces from %d faces" % (n_components, X_train.shape[0])
)
t0 = time()
pca = PCA(n_components=n_components, svd_solver="randomized", whiten=True).fit(X_train)
print("done in %0.3fs" % (time() - t0))

eigenfaces = pca.components_.reshape((n_components, h, w))

print("Projecting the input data on the eigenfaces orthonormal basis")
t0 = time()
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
print("done in %0.3fs" % (time() - t0))
```

```
Extracting the top 150 eigenfaces from 285 faces
done in 0.108s
Projecting the input data on the eigenfaces orthonormal basis
done in 0.008s
```

3.3 Entrainemnet du modéle

On a utilise une recherche aléatoire (RandomizedSearchCV) pour trouver les meilleurs hyperparamètres (C et gamma) d'un modèle SVM avec noyau RBF, en utilisant des données d'entraînement réduites en dimension par PCA.

Entrainement

```
print("Fitting the classifier to the training set")
t0 = time()
param_grid = {
    "C": loguniform(1e3, 1e5),
    "gamma": loguniform(1e-4, 1e-1),
}
clf = RandomizedSearchCV(
    SVC(kernel="rbf", class_weight="balanced"), param_grid, n_iter=10
)
clf = clf.fit(X_train_pca, y_train)
print("done in %0.3fs" % (time() - t0))
print("Best estimator found by grid search:")
print(clf.best_estimator_)
```

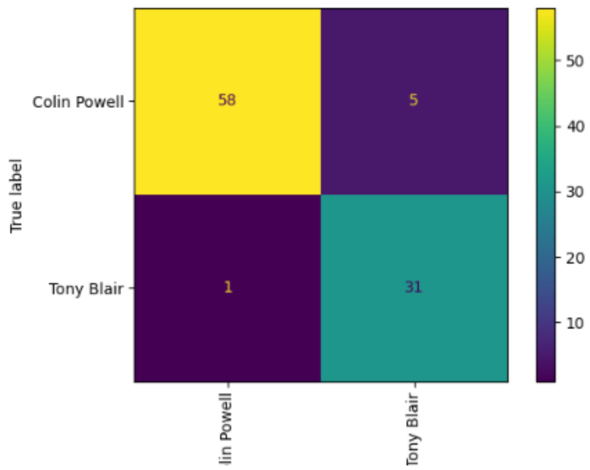
```
Fitting the classifier to the training set
done in 0.343s
Best estimator found by grid search:
SVC(C=16147.53185200393, class_weight='balanced', gamma=0.0016538957006737675)
```

3.4 Résultats

Résultat

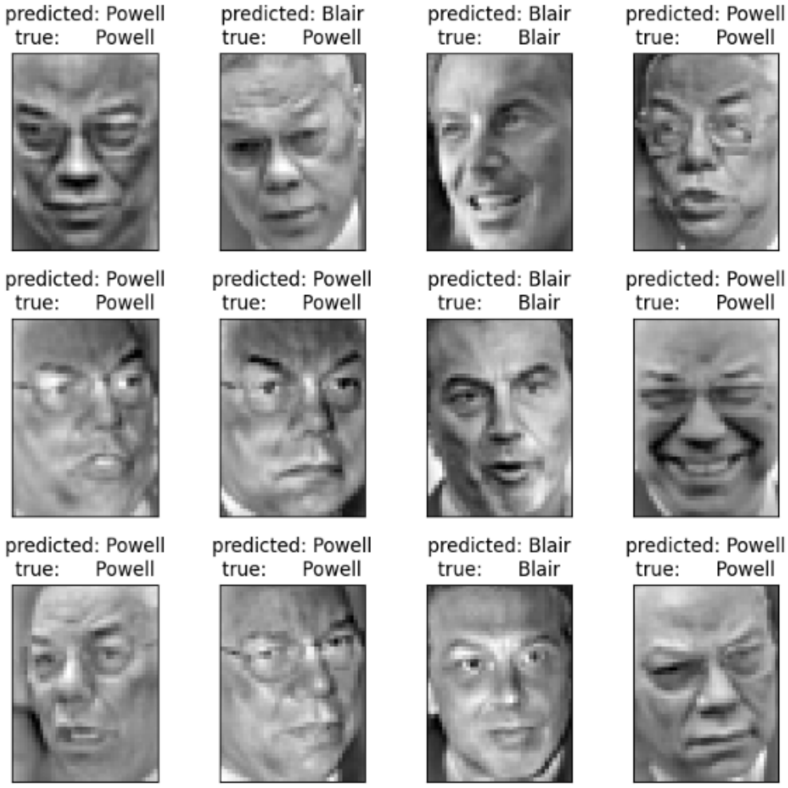
```
[24] print("Predicting people's names on the test set")
t0 = time()
y_pred = clf.predict(X_test_pca)
print("done in %0.3fs" % (time() - t0))

print(classification_report(y_test, y_pred, target_names=target_names))
ConfusionMatrixDisplay.from_estimator(
    clf, X_test_pca, y_test, display_labels=target_names, xticks_rotation="vertical"
)
plt.tight_layout()
plt.show()
```

```
Predicting people's names on the test set
done in 0.010s
              precision    recall  f1-score   support

Colin Powell       0.98      0.92      0.95        63
  Tony Blair       0.86      0.97      0.91        32

    accuracy                           0.94        95
   macro avg       0.92      0.94      0.93        95
weighted avg       0.94      0.94      0.94        95
```



```
[25] def plot_gallery(images, titles, h, w, n_row=3, n_col=4):
         """Helper function to plot a gallery of portraits"""
         plt.figure(figsize=(1.8 * n_col, 2.4 * n_row))
         plt.subplots_adjust(bottom=0, left=0.01, right=0.99, top=0.90, hspace=0.35)
         for i in range(n_row * n_col):
             plt.subplot(n_row, n_col, i + 1)
             plt.imshow(images[i].reshape((h, w)), cmap=plt.cm.gray)
             plt.title(titles[i], size=12)
             plt.xticks(())
             plt.yticks(())
```

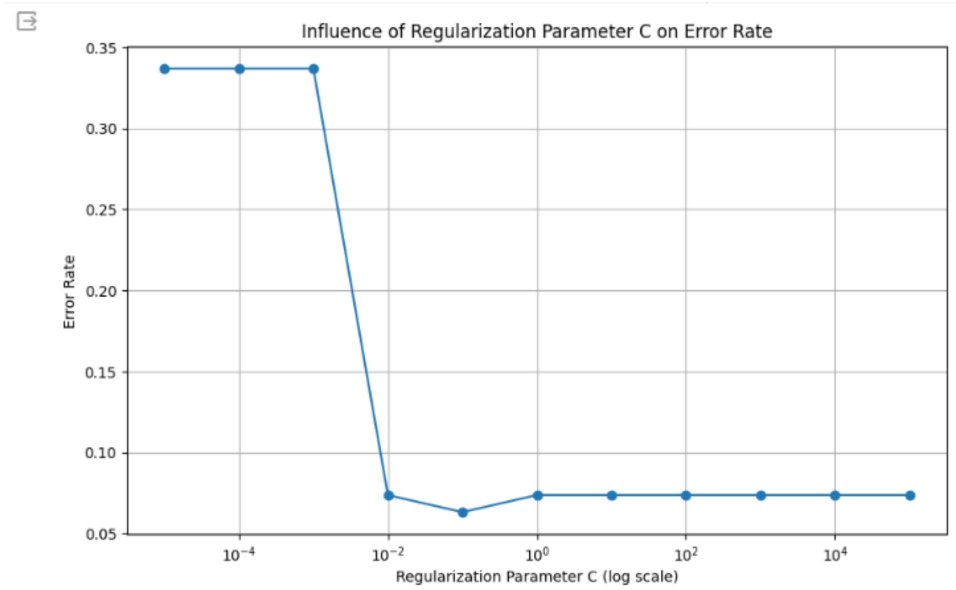
```
[26] def title(y_pred, y_test, target_names, i):
         pred_name = target_names[y_pred[i]].rsplit(" ", 1)[-1]
         true_name = target_names[y_test[i]].rsplit(" ", 1)[-1]
         return "predicted: %s\ntrue:      %s" % (pred_name, true_name)

     prediction_titles = [
         title(y_pred, y_test, target_names1, i) for i in range(y_pred.shape[0])
     ]

     plot_gallery(X_test, prediction_titles, h, w)
```



3.5 Influence du paramètre C

Influence du paramètre C

```
[34] import numpy as np
     import matplotlib.pyplot as plt
     from sklearn.datasets import fetch_lfw_people
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler
     from sklearn.svm import SVC
     from sklearn.decomposition import PCA
     from sklearn.metrics import classification_report, ConfusionMatrixDisplay
     from time import time

     # Load the Labeled Faces in the Wild (LFW) dataset
     lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4)

     # Extract information about the dataset
     n_samples, h, w = lfw_people.images.shape
     X = lfw_people.data
     n_features = X.shape[1]
     y = lfw_people.target
     target_names1 = lfw_people.target_names
     selected_classes = ['Tony Blair', 'Colin Powell']

     # Filter target_names to include only the selected classes
     target_names = [name for name in target_names1 if name in selected_classes]

     # Filter the dataset to include only Tony Blair and Colin Powell
     selected_indices = [i for i in range(n_samples) if target_names1[y[i]] in selected_classes]

     # Filter y to include only the labels corresponding to selected classes
     y_selected = y[selected_indices]

     # Use the filtered y to create the selected_indices
     selected_indices = [i for i in range(n_samples) if y[i] in y_selected]

     X_selected = X[selected_indices]
     y_selected = y[selected_indices]

     # Split the data into training and testing sets
     X_train, X_test, y_train, y_test = train_test_split(X_selected, y_selected, test_size=0.25, random_state=42)

     # Normalize the data (features are already centered and scaled)
     scaler = StandardScaler()
     X_train = scaler.fit_transform(X_train)
     X_test = scaler.transform(X_test)
     # Apply PCA
     n_components = 150  # Number of principal components
     pca = PCA(n_components=n_components, whiten=True).fit(X_train)
     X_train_pca = pca.transform(X_train)
     X_test_pca = pca.transform(X_test)

     # Range of values for the regularization parameter C
     C_values = np.logspace(-5, 5, num=11)

     # Lists to store results
     error_rates = []

     # Iterate over different values of C
     for C_val in C_values:
         # Train an SVM classifier with the current C value
         clf = SVC(kernel='linear', C=C_val)
         clf.fit(X_train_pca, y_train)

         # Predict labels on the test set
         y_pred = clf.predict(X_test_pca)

         # Calculate error rate
         error_rate = 1 - clf.score(X_test_pca, y_test)
         error_rates.append(error_rate)

     # Plot the influence of the regularization parameter C on the error rate
     plt.figure(figsize=(10, 6))
     plt.semilogx(C_values, error_rates, marker='o')
     plt.title('Influence of Regularization Parameter C on Error Rate')
     plt.xlabel('Regularization Parameter C (log scale)')
     plt.ylabel('Error Rate')
     plt.grid(True)
     plt.show()
```



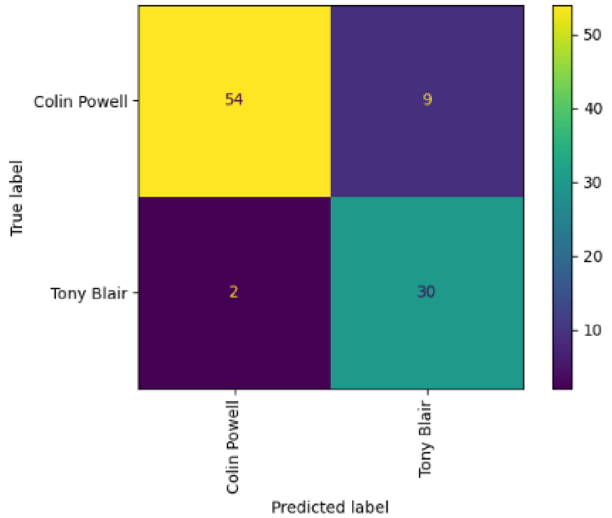
Le graphique illustre que lorsque le paramètre de régularisation C est très faible (10^{-4}), le taux d'erreur est élevé. En augmentant C, il y a une chute rapide et marquée du taux d'erreur jusqu'à un certain point (autour de 10^0 à 10^1), après quoi le taux d'erreur se stabilise, indiquant qu'une augmentation supplémentaire de C a peu ou pas d'effet sur la réduction de l'erreur. Cela suggère qu'il existe une valeur optimale de C au-delà de laquelle il n'y a pas d'amélioration significative de la performance du modèle.

3.6 Variables de nuisances

Variables de nuisances

```
[38] # Load the Labeled Faces in the Wild (LFW) dataset
     lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4)

     # Extract information about the dataset
     n_samples, h, w = lfw_people.images.shape
     X = lfw_people.data
     n_features = X.shape[1]
     y = lfw_people.target
     target_names1 = lfw_people.target_names
     selected_classes = ['Tony Blair', 'Colin Powell']

     # Filter target_names to include only the selected classes
     target_names = [name for name in target_names1 if name in selected_classes]

     # Filter the dataset to include only Tony Blair and Colin Powell
     selected_indices = [i for i in range(n_samples) if target_names1[y[i]] in selected_classes]

     # Filter y to include only the labels corresponding to selected classes
     y_selected = y[selected_indices]

     # Use the filtered y to create the selected_indices
     selected_indices = [i for i in range(n_samples) if y[i] in y_selected]

     X_selected = X[selected_indices]
     y_selected = y[selected_indices]

     # Split the data into training and testing sets
     X_train, X_test, y_train, y_test = train_test_split(X_selected, y_selected, test_size=0.25, random_state=42)

     # Add nuisance variables (randomly generated normal variables)
     num_nuisance_variables = 300
     nuisance_variables = np.random.normal(0, 1, size=(X_train.shape[0], num_nuisance_variables))
     X_train_with_nuisance = np.hstack((X_train, nuisance_variables))

     nuisance_variables_test = np.random.normal(0, 1, size=(X_test.shape[0], num_nuisance_variables))
     X_test_with_nuisance = np.hstack((X_test, nuisance_variables_test))

     # Normalize the data (features are already centered and scaled)
     scaler = StandardScaler()
     X_train_with_nuisance = scaler.fit_transform(X_train_with_nuisance)
     X_test_with_nuisance = scaler.transform(X_test_with_nuisance)
     # Apply PCA
     n_components = 150  # Number of principal components
     pca = PCA(n_components=n_components, whiten=True).fit(X_train_with_nuisance)
     X_train_pca = pca.transform(X_train_with_nuisance)
     X_test_pca = pca.transform(X_test_with_nuisance)

     # Train an SVM classifier
     clf = SVC(kernel='linear')
     clf.fit(X_train_pca, y_train)

     # Predict labels on the test set
     y_pred = clf.predict(X_test_pca)

     # Evaluate and visualize the results
     print("Predicting people's names on the test set")
     t0 = time()
     print(classification_report(y_test, y_pred, target_names=target_names))
     ConfusionMatrixDisplay.from_estimator(
         clf, X_test_pca, y_test, display_labels=target_names, xticks_rotation="vertical"
     )
     plt.tight_layout()
     plt.show()
     print("done in %0.3fs" % (time() - t0))
```

```
Predicting people's names on the test set
              precision    recall  f1-score   support

Colin Powell       0.96      0.86      0.91        63
  Tony Blair       0.77      0.94      0.85        32

    accuracy                           0.88        95
   macro avg       0.87      0.90      0.88        95
weighted avg       0.90      0.88      0.89        95
```



L'ajout de variables de nuisance, telles que des variables aléatoires générées, a entraîné une dégradation des performances de classification. Avant l'ajout de ces variables, le modèle présentait une précision élevée, tant pour la classe "Colin Powell" que pour la classe "Tony Blair", avec un score global d'exactitude de 94%. Cependant, après l'ajout des variables de nuisance, la précision, le rappel et le score F1 ont tous diminué, conduisant à une baisse significative de l'exactitude globale à 88%.

Cette diminution des performances peut être expliquée par le fait que les variables de nuisance introduisent du bruit supplémentaire et de la complexité inutile dans le modèle.

3.7 Choix d'un noyau non-linéaire RBF

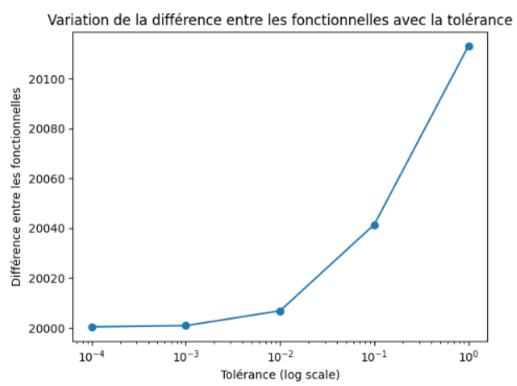
L'utilisation d'un noyau RBF dans un modèle SVM offre la possibilité de modéliser des relations non linéaires, améliorant ainsi la capacité du modèle à capturer des frontières de décision complexes. Le noyau RBF peut conduire à des performances améliorées lorsque les relations entre les caractéristiques et les étiquettes sont non linéaires.

Partie 4

Calcul du saut de dualité

Calcul du saut de dualité

```
[40] import numpy as np
     import matplotlib.pyplot as plt
     from sklearn import svm
     from sklearn.datasets import make_blobs

     # Fonction pour calculer les fonctionnelles primales et duales
     def calculate_functionals(X, y, tol):
         clf = svm.SVC(kernel="linear", C=1000, tol=tol)
         clf.fit(X, y)
         # Calcul des fonctionnelles primales
         primal_functional = 0.5 * np.dot(clf.coef_, clf.coef_.T) + clf.C * np.sum(np.maximum(0, 1 - y * clf.decision_function(X)))
         # Calcul des fonctionnelles duales
         dual_functional = np.sum(clf.dual_coef_ * clf.dual_coef_) / 2
         return primal_functional, dual_functional

     # Génération de données
     X, y = make_blobs(n_samples=40, centers=2, random_state=6)

     # Valeurs de tolérance à tester
     tolerances = np.logspace(-4, 0, 5)

     # Calcul des différences entre les fonctionnelles
     differences = np.zeros_like(tolerances)
     for i, tol in enumerate(tolerances):
         primal, dual = calculate_functionals(X, y, tol)
         differences[i] = np.abs(primal - dual)

     # Affichage des résultats
     plt.plot(tolerances, differences, marker='o')
     plt.xscale('log')
     plt.xlabel('Tolérance (log scale)')
     plt.ylabel('Différence entre les fonctionnelles')
     plt.title('Variation de la différence entre les fonctionnelles avec la tolérance')
     plt.show()
```



le graphique montre que lorsque la tolérance de l'optimisation est faible, les valeurs des fonctionnelles primal et dual sont très proches, mais à mesure que la tolérance augmente, la différence entre ces deux valeurs s'accroît rapidement.

Bibliographie

[1] B. Schölkopf and A. J. Smola. *Learning with kernels : support vector machines, regularization, optimization, and beyond.* MIT press, 2002.

[2] Scikit-learn. Support vector machines gui example. https://scikit-learn.org/stable/auto_examples/applications/svm_gui.html, 2024.