TP Classification

Youna Froger, Alice Gydé, Mathis Quais

6 Decembre 2024

1 Questions

1. What is classification in machine learning?
La classification est une tâche de machine learning supervisé et entraîné qui consiste à donner une catégorie à une donnée. On prend souvent l'exemple de l'e-mail spam ou non-spam pour illustrer la classification.

2. Why and when should classification be applied?
La classification permet d'organiser des données de manière automatique, ou bien de prédire les catégories de données qui ne sont pas connues.

3. What are commonly used algorithms for text classification?
Les algorithmes de classification les plus utilisés sont : Naive Bayes, basé sur le théorème Bayes, qui décrit la probabilité d'un événement basée sur la connaissance a priori de conditions liées à l'événement. Support Vector Machine qui est un algorithme essayant de trouver la surface de séparation entre des données la plus optimale. K-Nearest Neighbors qui est un algorithme utilisant la proximité entre les données pour les classifier.

2 Explication du code

Voici le lien vers le GitHub : https://github.com/younafroger/TAL$_H NM2$

L'objectif de ce TP est alors d'expérimenter la classification en utilisant différents modèles proposés par la librairie scikit-learn.
Nous avons dans un premier temps traité comme d'habitude notre jeu de données issu de la librairie scikit-learn, fetch 20newsgroups, en tokenisant et réduisant la dimension de notre jeu de données en une matrice à 5000 dimensions (5000 mots les plus significatifs) via TF-IDF.

La partie suivante est une première classification des documents en utilisant trois modèles (3 algorithmes) différents : Gaussian Naive Bayes, SVM (Support Vector Machine) et Random Forest. Le choix du modèle Gaussian Naive Bayes pour illustrer l'algorithme Naive Bayes vient du fait que, lors de nos premiers tests, en utilisant MultinomialNB (autre modèle utilisant ce même algorithme), nous avons rencontré des erreurs après traitement LSA, car ce modèle ne gère pas les données négatives.

```
X = tfidf_features.toarray()

X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, random_state=42)

models = {
    "Gaussian Naive Bayes": GaussianNB(),
    "SVM": SVC(kernel='linear', random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42)
}

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='weighted')
    print(f"{name} - Exactitude: {acc}, Score F1: {f1}")
```

```
Gaussian Naive Bayes - Exactitude: 0.8085676037483266, Score F1: 0.8088797323999751
SVM - Exactitude: 0.8728246318607764, Score F1: 0.873092705904294
Random Forest - Exactitude: 0.8232931726907631, Score F1: 0.8231292625091707
```

La première étape consiste à diviser l'ensemble des documents en deux ensembles d'entraînements (80 %) et de test (20 %) (cela correspond au paramètre "test_size=0.2". C'est le ratio le plus couramment utilisé. Pour calculer l'efficacité de nos modèles de classification, on utilise les fonctions accuracy_score et f1_score fournis par scikit-learn. En l'occurrence ici, les valeurs sont quasiment identiques peu importe si l'on prend l'exactitude ou le f1-score :
Pour Gaussian Naive Bayes : Ex = 80.85% et F1 = 80.89%
Pour SVM : Ex = 87.82% et F1 = 87.31%
Pour Random Forest : Ex = 82.33% et F1 = 82.31%

On en conclut que le modèle de classification le plus efficace est SVM.

La partie d'après consiste en la réduction de notre matrice TF-IDF avec la méthode LSA.

```
X_train_lsa, X_test_lsa, y_train, y_test = train_test_split(lsa_features, labels, test_size=0.2, random_state=42)

for name, model in models.items():
    model.fit(X_train_lsa, y_train)
    y_pred_lsa = model.predict(X_test_lsa)
    acc = accuracy_score(y_test, y_pred_lsa)
    f1 = f1_score(y_test, y_pred_lsa, average='weighted')
    print(f"{name} après LSA - Accuracy: {acc:.4f}, F1-score: {f1:.4f}")
```

```
Gaussian Naive Bayes après LSA - Accuracy: 0.7323, F1-score: 0.7355
SVM après LSA - Accuracy: 0.8581, F1-score: 0.8587
Random Forest après LSA - Accuracy: 0.8447, F1-score: 0.8454
```

Avec le paramètre "n_component" égal à 100, on réduit la dimension de la matrice TF-IDF à 100 dimensions. Cela simplifie davantage les données. Voici un tableau récapitulatif des résultats obtenus en comparant l'avant-après réduction avec LSA.

Modèles	Exactitude avec TF-IDF	Exactitude avec LSA	Score F1 avec TF-IDF	Score F1 avec LSA
Gaussian Naive Bayes	**80,85 %**	73,23 %	**80,89 %**	73,55 %
SVM	**87,28 %**	85,81 %	**87,31 %**	85,87 %
Random Forest	82,33 %	**84,47 %**	82,31 %	**84,54 %**

La réduction LSA a permis de rendre l'entraînement plus rapide (qui s'est fait en environ 5 secondes) comparé aux 30 secondes avec la matrice TF-IDF. Le modèle Naive Bayes semble moins performant après la réduction LSA, tandis que le modèle Random Forest semble gagné en performance. SVM est un modèle qui reste le plus performant, peu importe la taille de notre matrice.

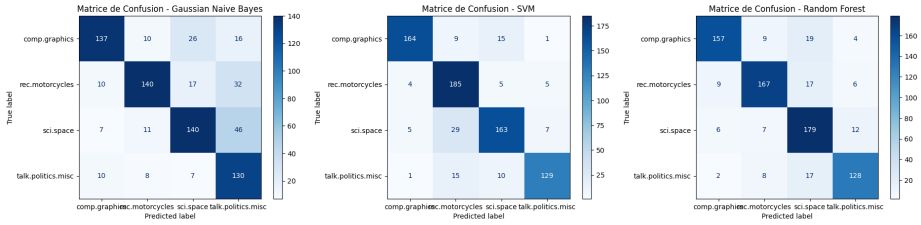
```
cv_results = {}
for model_name, model in models.items():
    scores = cross_val_score(model, X_train_lsa, y_train, cv=5, scoring='accuracy')
    cv_results[model_name] = scores
    print(f"Validation croisée pour {model_name} :")
    print(f"Scores : {scores}")
    print(f"Score moyen : {scores.mean()}\n")

class_names = newsgroups.target_names
#enfin, on utilise la fonction confusion_matrix pour afficher la matrice de confusion pour chaques modèles
for model_name, model in models.items():
    y_pred = model.predict(X_test_lsa)

    cm = confusion_matrix(y_test, y_pred, labels=model.classes_)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)

    print(f"Matrice de confusion pour {model_name}:")
    disp.plot(cmap=plt.cm.Blues)
    plt.title(f"Matrice de Confusion - {model_name}")
    plt.show()
```

Enfin, le dernier bloc de code nous permet de faire deux choses : une validation croisée de nos trois modèles (avec pour paramètre 5 tours de validation puis le calcul d'une moyenne du score des 5 tours) et une matrice de confusion pour les trois modèles.

3 Conclusion

D'après les résultats de la validation croisée, SVM reste le modèle le plus performant, même si la moyenne de score obtenue n'est pas éloignée de celle de Random Forest. Le modèle Gaussian Naive Bayes lui reste en retrait, avec un score vraiment inférieur. Pour la validation croisée, nous avons utilisé nos données réduites avec LSA, car un message d'erreur nous empêchait de le faire avec les données réduites uniquement avec TF-IDF, du fait du nombre de dimension qui était trop élevée pour la validation croisée de Gaussian Naive Bayes.

En ce qui concerne les matrices de confusion, le modèle SVM est celui qui fait le moins d'erreur de classification (quasiment aucune pour la classe rec.motorcycles). La matrice de SVM forme une diagonale bien marquée, avec peu de confusion.



Toutes ces étapes d'entraînements et de test, mêlées au calcul de métriques appropriées (Score F1 notamment) permettent d'identifier le modèle de classification le plus et le moins approprié à notre jeu de données. En effet, si le modèle Gaussian Naive Bayes montre des limites plutôt importantes, SVM quant à lui est le meilleur modèle de classification.