TP 5 : SVM linéaires

Selladurai Gowshigan

Introduction:

Les machines à vecteurs de support (SVM : Support Vector Machines) sont une classe de méthodes d'apprentissage statistique basées sur le principe de la maximisation de la marge (séparation des classes). Il existe plusieurs formulations (linéaires, versions à noyaux) qui peuvent s'appliquer sur des données séparables (linéairement) mais aussi sur des données non séparables.

Avantages	Désavantages
Ils sont aussi efficaces dans le cas où la dimension de l'espace est plus grande que le nombre d'échantillons d'apprentissage.	Comme il s'agit de méthodes de discrimination entre les classes, elles ne fournissent pas d'estimations de probabilités.
Pour la décision, n'utilisent pas tous les échantillons d'apprentissage, mais seulement une partie (les vecteurs de support). En conséquence, ces algorithmes demandent moins de mémoire.	Si le nombre d'attributs est beaucoup plus grand que le nombre d'échantillons, les performances sont moins bonnes.
Très efficaces en dimension élevée.	

Jeu de données Iris

Pour ce tp nous allons travailler sur le jeu de données Iris

et pour implémenter le SVM nous allons utiliser le module sklearn, plus spécifiquement sklearn.svm.

Pour commencer on va utiliser la version linéaire de SVM : **libLinear** et **libSVM**
et avec les deux premiers attributs (longueur et largeur des sépales).

On charge les données et on entraîne notre SVM linéaire.

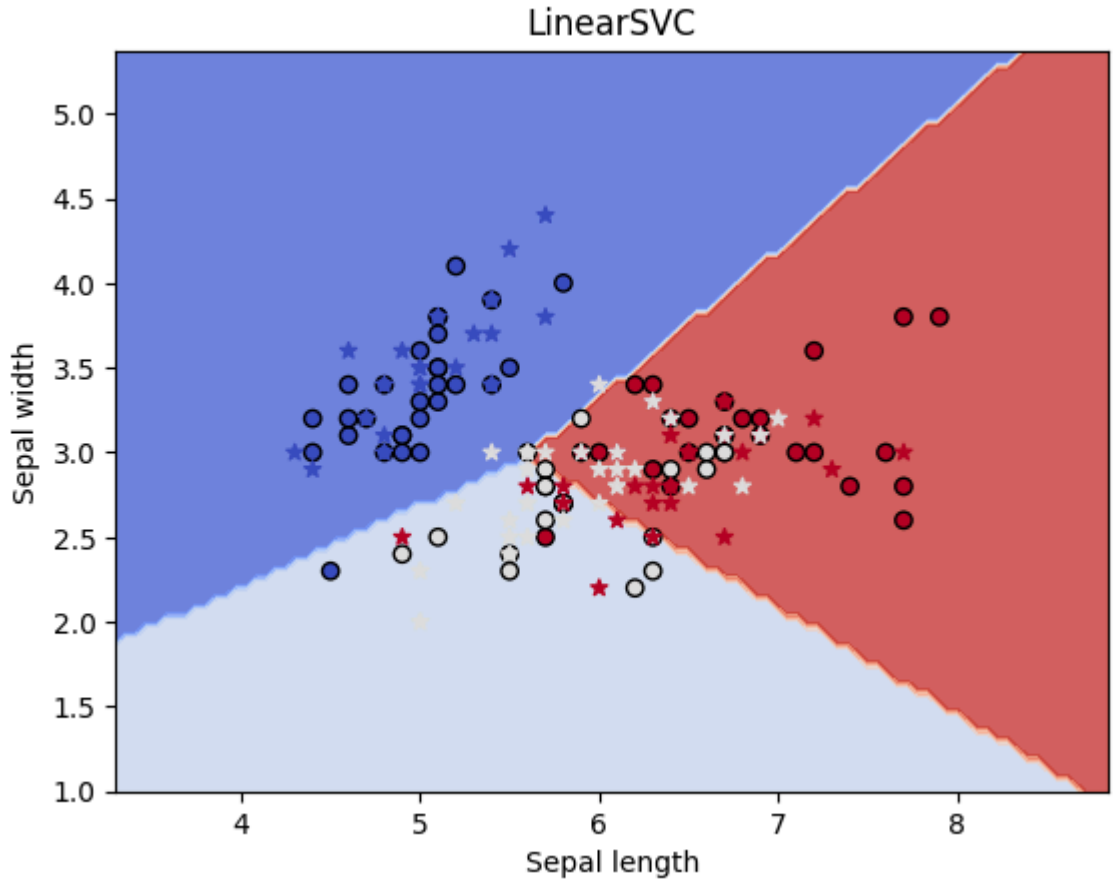
et si on calcule le score d'échantillons bien classifiés sur le jeu de données de test avec les lignes de commandes :

```
Z = lin_svc.predict(X_test)
accuracy = lin_svc.score(X_test,y_test)
print(accuracy)
```

on obtient :

0.6666666666666666 = (~0,67) de justesse.
donc 67% de d'échantillons sont bien classifiés sur le jeu de données de test.

Visualisons maintenant la surface de décision apprise par notre modèle :

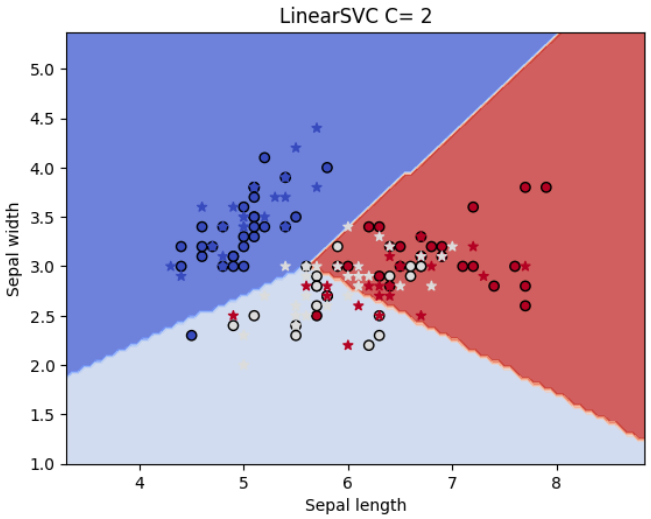


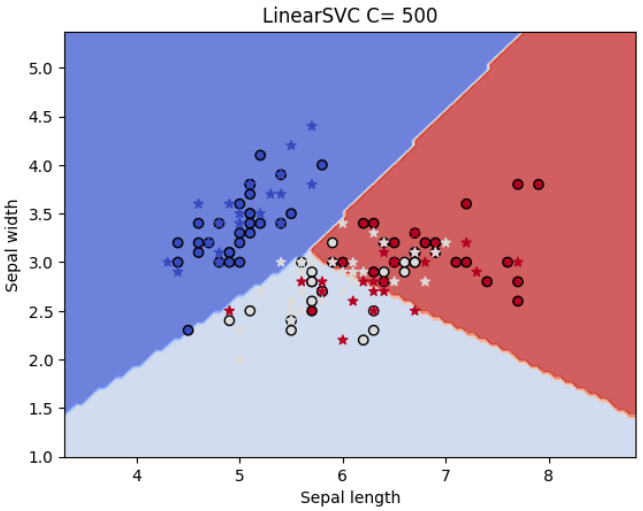
Testons différentes valeurs pour le paramètre C et observons comment la frontière de décision évolue en fonction de C :

Tout d'abord, "C" le paramètre de régularisation est un coefficient positif qui pèse l'importance des points mal classés en les pénalisant, ces points sont mesurés par rapport à la largeur de la marge.

C élevé -> pénalise fortement les erreurs, on essaie de réduire au maximum le nombre de points mal classés, cela a pour conséquence d'avoir une marge plus étroite. Risque de surapprentissage.

C petit -> on accepte quelques erreurs pour avoir une marge plus flexible (plus large), cela a pour conséquence de généraliser davantage le modèle. Risque de sous-apprentissage.





Ici quand on teste différentes valeurs pour le paramètres C, on observe quasi pas de d'évolution dans la frontière de décision. Cela est dû au fait qu'il n'y a pas beaucoup de points aux frontières et les données sont bien séparées et la distance entre les clusters de données et la frontière significative.

On observe également que le score de accuracy n'augmente que très peu:

C=1 -> accuracy ≈ 0,67
C=2 -> accuracy ≈ 0,68
C=500 -> accuracy ≈ 0,68

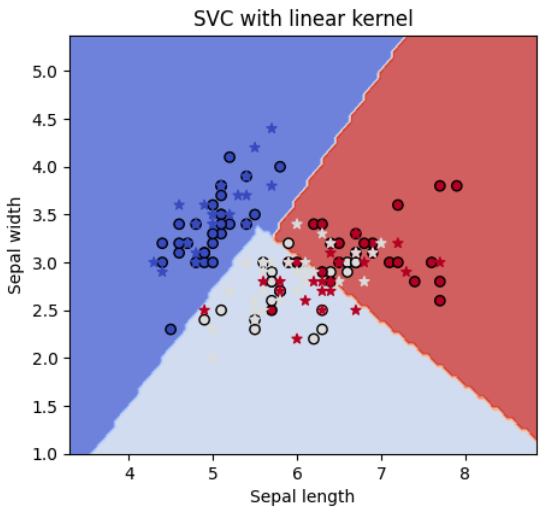
D’après la visualisation ci-dessus, ce modèle ne paraît pas adapté au problème, en effet on arrive pas à totalement séparer les 3 classes dans l’espace 2d formé par les deux attributs(longueur et largeur des sépales).

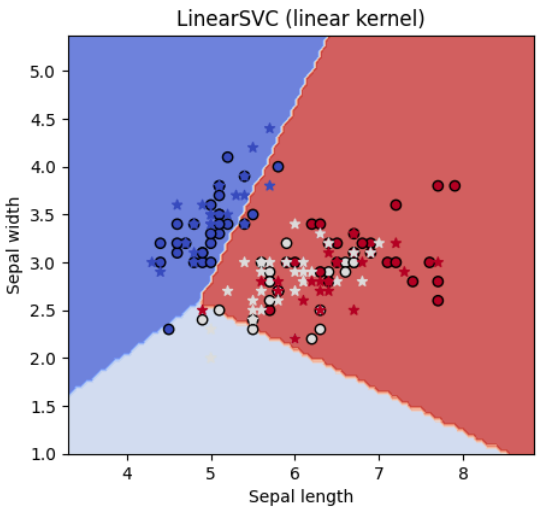
Plusieurs solutions s’offre à nous pour améliorer les résultats :

- On ajoute les d’autres attributs à notre modèles à notre modèle.
- On utilise des SVM à noyaux pour les séparer non linéairement.

Les modèles linéaires LinearSVC() et SVC(kernel='linear'), comme nous l’avons déjà dit, produisent des résultats légèrement différents à cause du fait qu’ils optimisent des fonctions de coût différentes mais aussi à cause du fait qu’ils gèrent les problèmes multi-classe de manière différente:

Critère	One-vs-All	One-vs-One
Nombre de classifieurs	K	K(K−1)/2
Équilibre des jeux d’entraînement	Déséquilibré	Équilibré
Temps d’entraînement	Plus rapide pour petit K	Peut devenir très long si K est grand
Temps de prédiction	K prédictions (une par classifieur)	K(K−1)/2 prédictions (une par paire)
Robustesse entre classes proches	Parfois moins bonne (classifieur apprend à discriminer une classe contre toutes les autres)	Meilleure, car chaque paire est entraînée séparément
Complexité mémoire	Stocke K hyperplans	Stocke K(K−1)/2 hyperplans





Classifieur SVC(C=500, kernel='linear') accuracy = 0.72
Classifieur LinearSVC(C=500) accuracy = 0.6933333333333334

Nous pouvons observer que les frontières sont mieux définies pour le LinearSVC mais notre SVM à noyau performe mieux pour le score justesse (accuracy score).

Réalisons l'optimisation d'une nouvelle machine à vecteur de support linéaire mais en utilisant les quatre attributs du jeu de données Iris.

```
# Charger les données Iris
iris = datasets.load_iris()
X = iris.data  # 4 attributs
y = iris.target  # 3 classes
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.5,random_state=0)

lin_svc = svm.LinearSVC(C=C,dual='auto')
lin_svc.fit(X_train, y_train)
accuracy = lin_svc.score(X_test,y_test)
print(accuracy)
```

on obtient 0.9733333333333334

Le score de classification en test a augmenté : environ 0.97 = 97%
Cela est dû au fait que les attributs qu'on a ajoutés nous aident fortement à classifier les différentes classes.
Plus concrètement, les deux premiers attribut nous aident à classifier seulement bien une seule classe et pas les deux autres présents dans notre modèle de test.
Le fait d'ajouter les deux attributs restants a permis à notre modèle de distinguer les 2 autres plus facilement.

Jeu de données Digits

Nous allons maintenant travailler sur le jeu de données Digits, une collection de chiffres manuscrits et construire un classifieur LinearSVC et l'évaluer.

Pour obtenir de meilleurs résultats nous allons procéder à un GridSearch pour maximiser le score de accuracy, pour cela on va tester différentes valeurs de C.

```
from sklearn.datasets import load_digits
digits = load_digits()
X, y = digits.data, digits.target

param_grid = {
      'C': [1, 2, 3, 4, 5, 10]
}

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.7,random_state=0)

grid_search = GridSearchCV(

estimator=svm.LinearSVC(random_state=0,dual='auto',max_iter=10000),
      param_grid=param_grid,
      scoring='accuracy',   # on optimise l'accuracy
      cv=5,                 # 5-fold cross-validation
      n_jobs=-1,            # parallélisation
      return_train_score=True
)
grid_search.fit(X_train, y_train)

print("Meilleurs paramètres :", grid_search.best_params_)
print(f"Meilleure accuracy CV : {grid_search.best_score_:.3f}")
```

On obtient :

Meilleurs paramètres : {'C': 2}
Meilleure accuracy CV : 0.922

Ainsi on obtient le meilleur résultat de généralisation pour C = 2 , avec une précision d'environ 92% sur les données de test.

Conclusion

Les SVM linéaires sont des classifieurs robustes et efficaces dès lors que les données sont raisonnablement séparables dans une dimension suffisante.

Le choix du paramètre de régularisation CCC est déterminant pour le compromis biais/variance et doit être optimisé via validation croisée.

Lorsque l'espace des attributs est trop réduit (Iris 2D), la linéarité ne suffit pas, et il faut soit augmenter la dimension (Iris 4D), soit recourir à des noyaux non linéaires.

Sur Digits, un SVM linéaire correctement réglé (C = 2) atteint déjà une précision de l'ordre de 92 %, ce qui en fait une solution viable pour des applications de classification d'images de ce type.

On peut noter qu'on a pas prétraité (gestion des valeurs manquantes , standardisé) les données dans ce tp avant d'effectuer les entraînements pour les SVM en général sans cette étape, nous pouvons avoir des performances faible voire incorrectes mais ici c'est suffisant afin de se concentrer que sur les modèles de SVM.
Cela reste néanmoins une bonne pratique.