Bachelet Julie
Jordan Theodore

Projet : Introduction au Machine Learning

2.1 Implémentation LDA et régression logistique

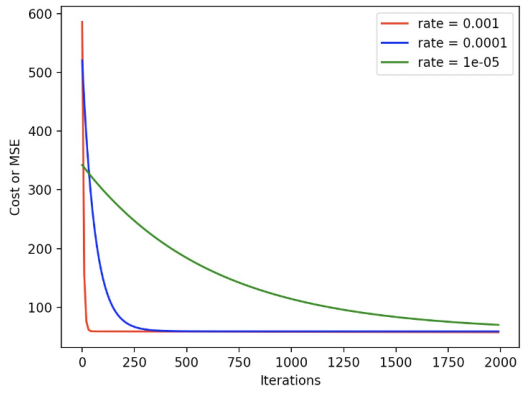
1) DESCENTE DE GRADIANT

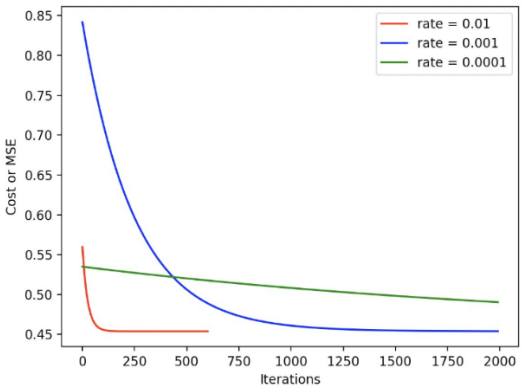
L'algorithme du gradient descent est un algorithme populaire pour la régression linéaire car il permet de trouver les valeurs optimales des coefficients pour un modèle de régression linéaire en minimisant la fonction de coût (mean squared error). Il s'agit d'un algorithme de descente stochastique qui itère sur les coefficients en les mettant à jour en utilisant la dérivée de la fonction de coût pour chaque itération, en utilisant le taux de mise à jour spécifié pour minimiser le coût. Cette méthode peut généralement trouver une solution optimale pour les jeux de données simples à complexe, ce qui la rend très utile pour la régression linéaire.

En utilisant des jeux de données différents, nous pouvons observer des différences dans les tendances et les formes des graphiques produits. Le choix des données peut également affecter la performance de l'algorithme.

Nous avons utilisé deux jeux de données pour tester notre algorithme de descente de gradient : l'un généré manuellement et l'autre provenant de la bibliothèque scikit-learn (jeu de données Boston qui comprend des informations sur des propriétés immobilières dans la ville de Boston).

On obtient des résultats différents en fonction des données, mais aussi en fonction des paramètres. Par exemple, voici deux graphiques provenant des deux jeux de données expliqués précédemment.





Le premier graphique représente le coût, ou l'erreur moyenne au carré en fonction des itérations pour le jeu de données Boston, et le second pour le jeu de données construit

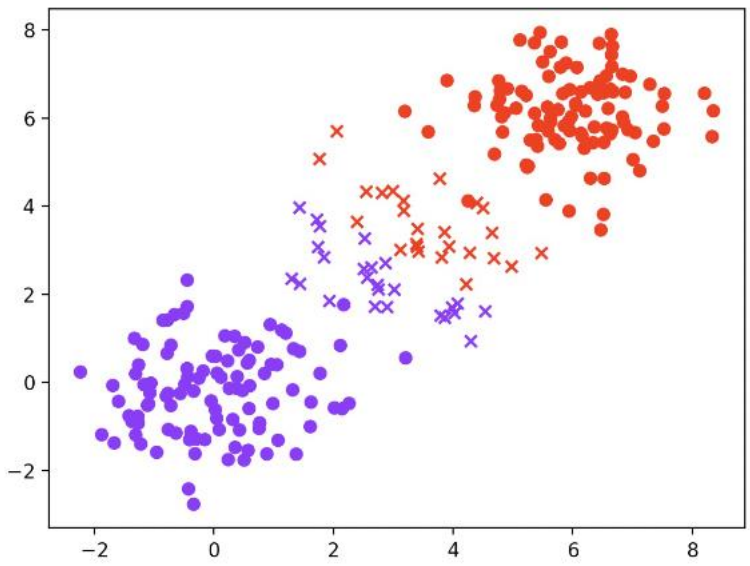
manuellement. Nous pouvons voir des courbes différentes puisque la vitesse à laquelle le MSE s'approche de zéro (et donc la convergence de l'algorithme) peut varier en fonction du jeu de données. On obtient 3 courbes dans chaques graphiques puisque nous avons fait tourner notre algorithme avec des rates différent (taux d'apprentissage). C'est un paramètre qui contrôle la vitesse à laquelle les poids sont mis à jour à chaque itération. Si le taux d'apprentissage est trop grand, le modèle peut manquer l'optimum global et diverger, tandis que si le taux d'apprentissage est trop petit, le modèle peut être très lent à converger. Il est donc important de trouver un taux d'apprentissage approprié pour un jeu de données et un modèle donné.

2) LDA

D'un autre côté, la méthode LDA nécessite une distribution normale des données et des hypothèses supplémentaires sur les relations linéaires entre les variables, ce qui peut rendre cette méthode applicable uniquement à certaines situations. Il sert à séparer les données en groupes distincts en maximisant la variance entre les groupes et minimisant la variance à l'intérieur de chaque groupe. Cet algorithme peut être utilisé pour prédire la classe d'appartenance d'un nouvel individu en se basant sur ses caractéristiques quantitatives.

Pour tester notre implémentation pour cette méthode, nous avons créé un jeu de données d'entrainement et un jeu de données test, qui se génèrent aléatoirement en suivant une distribution gaussienne multivariée avec une moyenne et une matrice de covariance spécifiées en argument.

Au lancement de notre algorithme avec nos données, nous obtenons un graphique qui ressemble à celui-ci :



L’algorithme à classé nos données en deux classes : une violette et une rouge. De plus nous observons des formes différentes pour nos données d’entrainement (rond) et nos données test (croix). Notre algorithme à donc permis d’attribuer une classe à nos données de test, en fonction des classes des données d’entrainement.

En résumé, la descente de gradient est utilisée pour minimiser une fonction de coût pour optimiser les paramètres d'un modèle, tandis que LDA est un algorithme de classification qui utilise des techniques statistiques pour prédire la classe d'une observation.

2.2 Evaluation et comparaison des modèles avec scikit-learn et données réelles.

Nous avons choisi pour cet exercice le jeu de données Heart Failure Prediction. Il comprend des données démographiques, des antécédents médicaux et des données cliniques pour des patients présentant une insuffisance cardiaque, et d’autres qui n’en souffrent pas. Le but de ce jeu de données est de prédire le risque d'insuffisance cardiaque pour un patient en fonction de ces données. C’est un jeu de données de taille raisonnable, ce qui le rend facile à utiliser pour l’entrainement et le test des algorithmes de classification. Cela pourra aussi nous permettre de comparer les performances des algorithmes différents sur ce même jeu de données.

Nous avons décidé d’utiliser les méthodes LDA et K-PPV sur ce jeu de données.

La classification LDA est une méthode simple qui n’exige pas beaucoup de réglages. Elle nous fourni des résultats qui sont facilement interprétables en termes de directions de séparations discriminantes entre les classes, ce qui peut être utile pour comprendre les facteurs importants qui influencent la prédiction de la maladie cardiaque. De plus, elle a une bonne performance pour les problèmes de classification binaire, dans notre cas nous avons en effet deux classes à séparer, les patients atteints d’une insuffisance cardiaque ou non.

La méthode K-PPV est aussi un algorithme facile à comprendre et il peut être implémenté rapidement. C’est un algorithme de classification supervisé basé sur l'apprentissage par les plus proches voisins. Il peut être utilisé pour prédire la maladie cardiaque en comparant les données d'un patient à celles des patients les plus proches dans le jeu de données pour déterminer la classe à laquelle il appartient. K-PPV est souvent robuste aux bruits dans les données et peut donner de bons résultats même pour les petits ensembles de données, ce qui peut être utile pour le jeu de données Heart Failure Prediction qui peut ne pas comporter beaucoup de données pour chaque patient.

Maintenant, nous allons voir les résultats que nous avons obtenus avec ces méthodes afin de trouver la meilleure méthode.

1) LDA

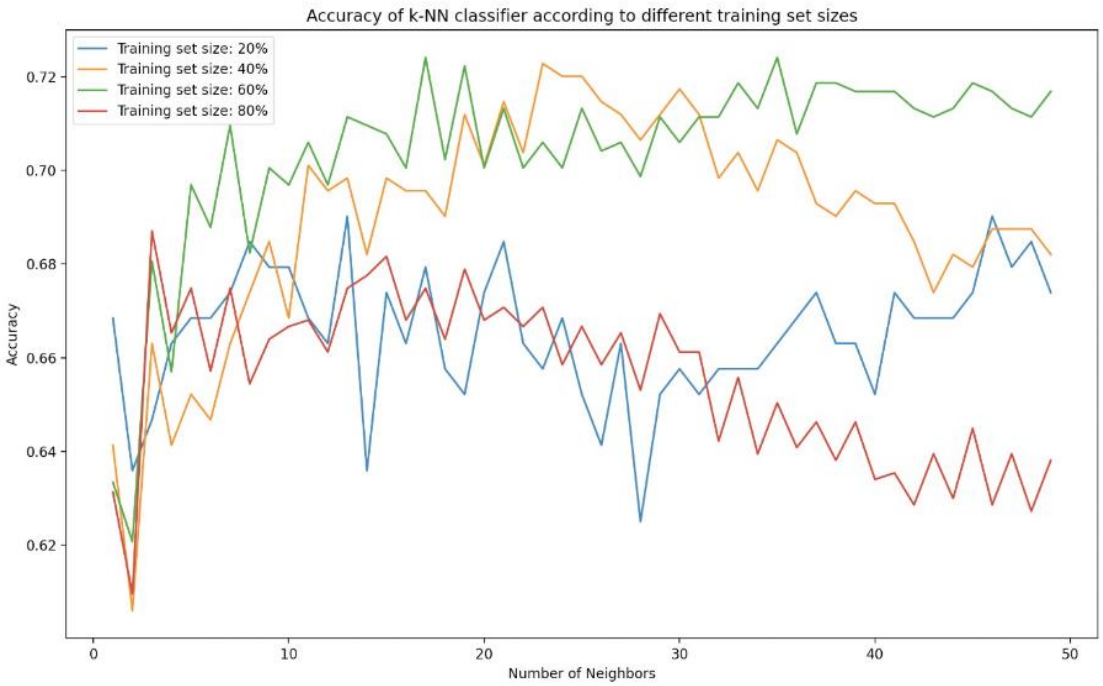
Avec la méthode LDA, nous avons obtenu une accuracy de 0,85.

2) K-PPV

Avec la méthode K-PPV, nous avons obtenu des accuracy différentes en fonction du nombre de “plus proches voisins” et de la taille de l’ensemble d'entraînement.

Nous observons qu’avec un training set de 20% (courbe bleue) nous obtenons des résultats très moyens et que plus la taille augmente, plus l’accuracy augmente. En effet, la meilleure accuracy correspond à la courbe rouge (50%) et augmente en fonction du nombre de voisins entre 0 et 15 voisins environ (de 0,6 à 0,72 d’accuracy) et oscille ensuite entre 0,71 et 0,73 d’accuracy. On a un pic à 30 voisins où l’on frôle les 0,74 d’accuracy, mais cela ne permet toujours pas d’atteindre les 0,85 que l’on a obtenu avec LDA.

Nous observons que plus la taille du training set augmente, plus l’accuracy augmente. Nous avons donc essayé d’augmenter encore la taille du training set pour avoir de meilleurs résultats. Voilà ce que nous avons obtenu :



Le training set size de 60% est pas mal mais équivaut à celui de 50%, en revanche celui de 80% est très mauvais. En effet plus l'ensemble d'entraînement est grand, plus le classifieur a d'informations pour apprendre et donc mieux se généraliser à de nouvelles données. Cependant, si l'ensemble est trop grand (comme dans notre cas pour le training set de 80%), cela peut entraîner un sur-apprentissage, le modèle peut apprendre les bruits dans les données d'entraînement au lieu de généraliser les tendances.

3) Random Forest

Pour le moment, LDA est la meilleure méthode. Maintenant, implémentons une nouvelle méthode permettant de comparer une nouvelle fois les résultats obtenus avec notre jeu de données.

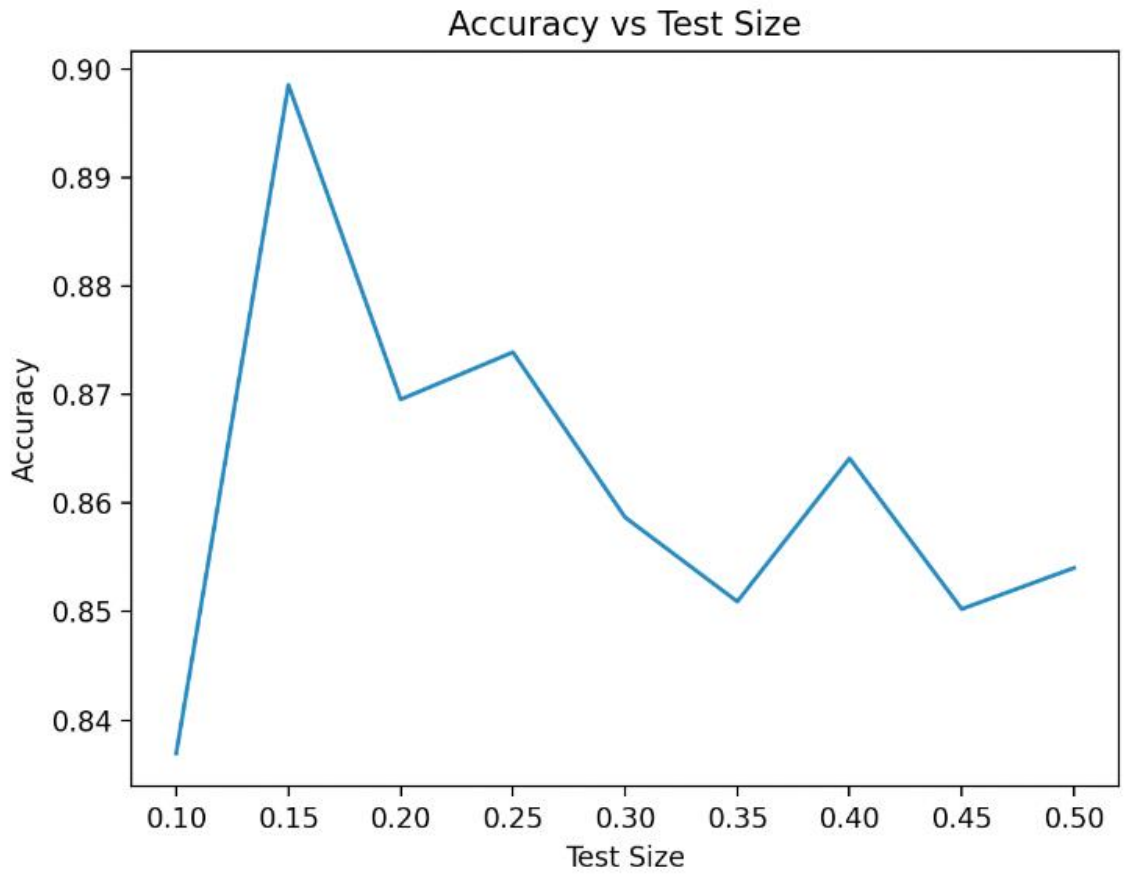
Après avoir fait quelques recherches, nous avons trouvé une méthode qui nous semblait intéressante : Random Forest.

Random Forest est un algorithme d'apprentissage automatique basé sur l'ensemble qui construit plusieurs arbres de décision et les combine pour obtenir une prédiction plus fiable et robuste. Chaque arbre de décision est construit à partir d'un sous-ensemble aléatoire de l'ensemble de données d'entraînement et de sous-ensembles aléatoires de caractéristiques pour la division des nœuds. Les prédictions de chaque arbre sont ensuite combinées pour obtenir une prédiction finale, soit en prenant la moyenne pour les probabilités de prédiction ou en prenant la majorité des votes pour les algorithmes de classification. La combinaison de plusieurs arbres de décision permet à Random Forest de lisser les erreurs individuelles d'un arbre pour obtenir une prédiction plus stable et fiable. De plus, la sélection aléatoire des sous-ensembles de données d'entraînement et des

caractéristiques pour chaque arbre réduit le risque de sur-apprentissage sur les données d'entraînement. (D’après internet)

Random Forest peut gérer efficacement les caractéristiques corrélées sans affecter la performance de prédiction, ce qui peut être un problème pour les algorithmes tels que LDA. De plus, Random Forest peut gérer les ensembles de données complexes avec une bonne performance en termes de précision, ce qui peut être difficile pour les algorithmes tels que LDA ou K-PPV. Random Forest offre la possibilité de mesurer l'importance relative de chaque caractéristique pour la prédiction, ce qui peut être utile pour la sélection de caractéristiques et pour comprendre les facteurs les plus importants associés à la défaillance cardiaque. Cette fonctionnalité n'est pas disponible dans LDA ou K-PPV. Enfin, Random Forest est robuste aux données bruyantes, ce qui était un problème surtout pour K-PPV comme nous l’avons vu précédemment.

Nous avons mesuré la performance de cet algorithme en comparant les prédictions à la valeur réelle des données de test en utilisant le score de l’algorithme. Nous avons illustré la précision de cet algorithme en fonction de la taille de l’ensemble de test dans le graphique suivant :



On observe très clairement le pic de précision pour un échantillon test de 15%, avec une accuracy de quasiment 0,9, ce qui dépasse la précision de l'algorithme LDA. Par la suite, la précision diminue avec la taille de l'échantillon de test.

En conclusion, l'algorithme le plus performant parmi les 3 que nous avons testé pour ce jeu de données est le Random Forest, avec un échantillon de test de taille 15%. Cela nous permet d'obtenir une précision de quasiment 0,9.