Exercice de programmation Machine Learning 4 : Régression linéaire régularisée et biais vs variance

adapté au langage Python de Coursera/Andrew Ng 16

octobre 2023

1 Introduction

Dans cet exercice, vous allez implémenter la régression linéaire régularisée et l'utiliser pour étudier des modèles présentant différentes propriétés de biais-variance. Avant de commencer l'exercice de programmation, nous vous recommandons fortement de regarder les conférences vidéo.

Pour commencer l'exercice, vous devrez télécharger le code de démarrage et décompresser son contenu dans le répertoire dans lequel vous souhaitez effectuer l'exercice.

2 Fichiers inclus dans cet exercice

- **ex4.py** - Script Python qui vous aidera à suivre l'exercice
- **ex4data1.mat** - Jeu de données
- **featureNormalize.py** - Fonction de normalisation des caractéristiques
- **plotFit.py** - Tracer un ajustement polynomial
- **trainLinearReg.py** - Entraîne la régression linéaire à l'aide de votre fonction de coût

? **linearRegCostFunction.py** - Fonction de coût de régression linéaire régularisée

? **learningCurve.py** - Génère une courbe d'apprentissage

? **polyFeatures.py** - Mappe les données dans l'espace d'entités polynomiales

? **validationCurve.py** - Génère une courbe de

validation [*?*] indique les fichiers que vous devrez

compléter.

Tout au long de l'exercice, vous utiliserez le script **ex4.py**. Ce script configure le jeu de données pour le
problèmes et faire des appels à des fonctions que vous écrirez. Vous n'avez qu'à modifier les fonctions d'autres fichiers en suivant les instructions de cette affectation.

3 Régression linéaire régularisée

Dans la première section de l'exercice, vous allez mettre en œuvre la régression linéaire régularisée pour prédire la quantité d'eau qui s'écoule d'un barrage à l'aide de la variation du niveau d'eau dans un réservoir. Ensuite, vous passerez en revue quelques diagnostics d'algorithmes d'apprentissage de débogage et examinerez les effets du biais par rapport à la variance.

Le script fourni, **ex4.py**, vous aidera à réaliser cet exercice.

3.1 Visualisation du jeu de données

Nous commencerons par visualiser l'ensemble de données contenant des enregistrements historiques sur la variation du niveau d' *eau, x*, et la quantité d'eau s'écoulant du barrage, *y*.

Ce jeu de données est divisé en trois parties :

- Un ensemble d'apprentissage sur lequel votre modèle apprendra : X, y

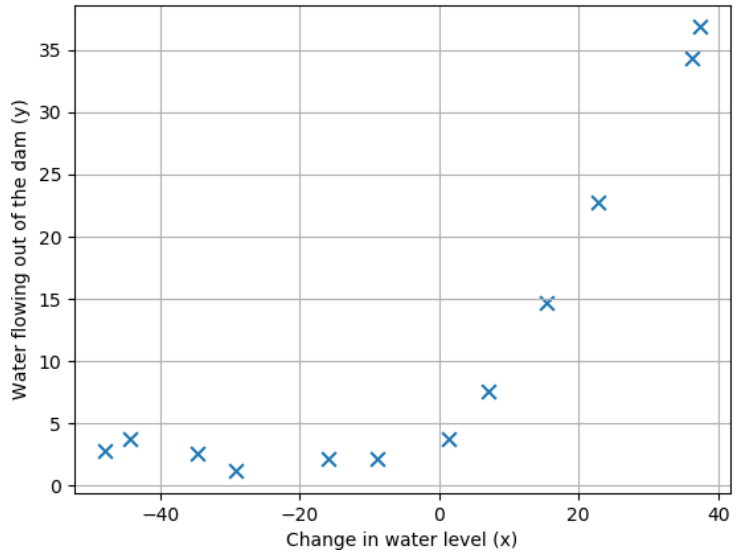

Figure 1 – Données utilisées dans ce devoir.

- Un jeu de validation pour déterminer le paramètre de régularisation : *Xval*, *yval*
- Un ensemble de tests pour évaluer les performances. Il s'agit d'exemples « inédits » que votre modèle n'a pas vus lors de l'entraînement : *Xtest*, *ytest*

L'étape suivante de **ex4.py** tracera les données d'entraînement (figure 1). Dans les parties suivantes, vous allez implémenter la régression linéaire et l'utiliser pour ajuster une ligne droite aux données et tracer des courbes d'apprentissage. Ensuite, vous allez implémenter la régression polynomiale pour trouver un meilleur ajustement aux données.

3.2 Fonction de coût de régression linéaire régularisée

Rappelons que la régression linéaire régularisée a la fonction de coût suivante :

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

où λ est un paramètre de régularisation qui contrôle le degré de régularisation (ce qui permet d'éviter le surapprentissage). Le terme de régularisation impose une pénalité sur le coût total J. Au fur et à mesure que les magnitudes des paramètres du modèle θj augmentent, la pénalité augmente également. Notez que vous ne devez pas régulariser le *terme θ0*.

Vous devez maintenant compléter le code dans le fichier **linearRegCostFunction.py**. Votre tâche consiste à écrire une fonction pour calculer la fonction de coût de régression linéaire régularisée. Si possible, essayez de vectoriser votre code et évitez d'écrire des boucles. Lorsque vous avez terminé, la partie suivante **de ex4.py** exécutera votre fonction de coût à l'aide de thêta initialisé à $[1\ 1]^T$. Vous devriez vous attendre à voir une sortie de 303,993.

3.3 Gradient de régression linéaire régularisé

En conséquence, la dérivée partielle du coût de la régression linéaire régularisée pour θj est définie comme suit :

$$\frac{\partial J(\theta)}{\partial \theta j} = \frac{1}{m} \sum_{i=1}^{n} h\theta\left(x^{(i)}\right) - y^{(i)} x_j^{(i)} \qquad \text{pour } j = 0$$

$$\partial J(\theta) = 1 \sum_{n} h\theta\left(x^{(i)}\right) - y^{(i)} x_j^{(i)} + \lambda\theta_j \qquad \text{vers}$$

er

$$\partial\theta j \qquad m \qquad j \geq 1$$

$$i=1$$

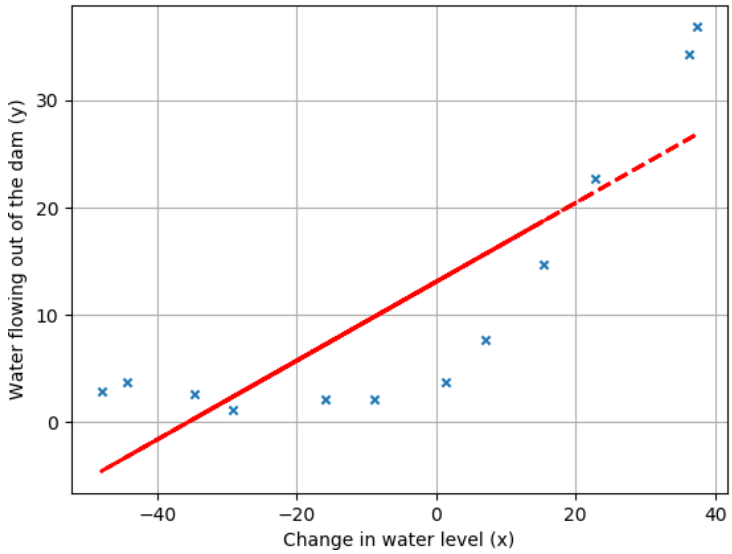

Figure 2 – Ajustement linéaire sur les données.

Dans **linearRegCostFunction.py,** ajoutez du code pour calculer le gradient, en le renvoyant dans la variable grad. Lorsque vous avez terminé, la partie suivante **de ex4.py** exécutera votre fonction de dégradé à l'aide de thêta initialisé à $\begin{matrix}1\\1\end{matrix}$ et $\lambda = 1$. Vous devez vous attendre à voir un gradient de $[-15.30\ 598.250]^T$.

3.4 Ajustement de la régression linéaire

Une fois que votre fonction de coût et votre gradient fonctionnent correctement, la partie suivante **de ex4.py** exécutera le code dans **trainLinearReg.py** pour calculer les valeurs optimales de θ. Cette fonction **d'apprentissage utilise fmin_cg** pour optimiser la fonction de coût.

Dans cette partie, nous mettons le paramètre de régularisation λ à zéro. Parce que notre implémentation actuelle de la régression linéaire essaie d'ajuster un θ à 2 dimensions, la régularisation ne sera pas incroyablement utile pour un θ de si faible dimension. Dans les dernières parties de l'exercice, vous utiliserez la régression polynomiale avec régularisation. Enfin, le **script ex4.py** doit également tracer la ligne la mieux ajustée, ce qui donne une image similaire à celle de la figure 2.

La ligne d'ajustement optimale nous indique que le modèle n'est pas bien ajusté aux données, car celles-ci ont un modèle non linéaire. Bien que la visualisation du meilleur ajustement comme indiqué soit un moyen possible de déboguer votre algorithme d'apprentissage, il n'est pas toujours facile de visualiser les données et le modèle. Dans la section suivante, vous allez implémenter une fonction pour générer des courbes d'apprentissage qui peuvent vous aider à déboguer votre algorithme d'apprentissage même s'il n'est pas facile de visualiser les données.

4 Biais-variance

Commençons par cette vidéo !

Un concept important de l'apprentissage automatique est le compromis biais-variance. Les modèles avec un biais élevé ne sont pas assez complexes pour les données et ont tendance à sous-ajuster, tandis que les modèles avec une variance élevée surajustent les données d'apprentissage.

Dans cette partie de l'exercice, vous allez tracer les erreurs d'entraînement et de test sur une courbe d'apprentissage afin de diagnostiquer les problèmes de biais-variance.

4.1 Courbes d'apprentissage

Commençons par cette vidéo !

Vous allez maintenant implémenter du code pour générer les courbes d'apprentissage qui seront utiles dans le débogage des algorithmes d'apprentissage. Rappelez-vous qu'une courbe d'apprentissage trace

l'erreur d'apprentissage et de validation en fonction de l'ensemble d'apprentissage

taille. Votre travail consiste à renseigner **learningCurve.py** afin qu'il renvoie un vecteur d'erreurs pour l'ensemble d'apprentissage et le jeu de validation.

Pour tracer la courbe d'apprentissage, nous avons besoin d'une erreur d'ensemble d'apprentissage et de validation pour différentes tailles d'ensemble d'apprentissage. Pour obtenir différentes tailles *d'ensemble d'apprentissage, vous devez utiliser différents sous-ensembles de l'ensemble d'apprentissage d'origine X.*

Plus précisément, pour une taille d'ensemble d'apprentissage de n, vous devez utiliser les n premiers exemples (c'est-à-dire X[:n+1, :] et y[:n+1]).

Vous pouvez utiliser la **fonction trainLinearReg** pour trouver les paramètres θ. Notez que Lambda est passé en paramètre à la fonction **learningCurve**.

Après avoir appris *les paramètres* θ, vous devez calculer l'erreur sur les ensembles d'apprentissage et de validation.

Rappelez-vous que l'erreur d'apprentissage d'un jeu de données est définie comme suit :

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

En particulier, notez que l'erreur d'apprentissage n'inclut pas le terme de régularisation. Une façon de calculer l'erreur d'apprentissage consiste à utiliser votre fonction de coût existante et à définir λ sur 0 uniquement lorsque vous l'utilisez pour calculer l'erreur d'apprentissage.

Lorsque vous calculez l'erreur de l'ensemble d'apprentissage, assurez-vous de la calculer sur le sous-ensemble d'apprentissage (c'est-à-dire X[:n+1, :] et y[:n+1]) (au lieu de l'ensemble d'apprentissage). Toutefois, pour l'erreur de validation, vous devez la calculer sur l'ensemble du jeu de validation. Vous devez stocker les erreurs calculées dans le train d'erreurs vectorielles et la valeur d'erreur.

Lorsque vous aurez terminé, **ex4.py** imprimerons les courbes d'apprentissage et produirez un tracé similaire à celui de la figure 3.

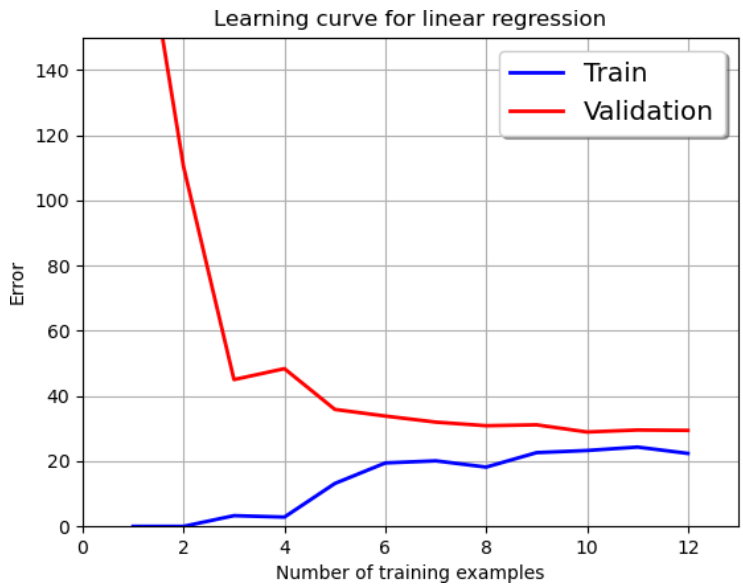

Figure 3 – Courbe d'apprentissage de la régression linéaire

Dans la figure 3, vous pouvez observer que l'erreur d'apprentissage et l'erreur de validation sont élevées lorsque le nombre d'exemples d'apprentissage continue d'augmenter. Cela reflète un problème de biais élevé dans le modèle, le modèle de régression linéaire est trop simple et ne peut pas bien s'adapter à notre jeu de données. Dans la section suivante, vous allez implémenter la régression polynomiale afin d'ajuster un meilleur modèle pour ce jeu de données.

5 Régression polynomiale

Le problème avec notre modèle linéaire était qu'il était trop simple pour les données et qu'il entraînait un sous-ajustement (biais élevé). Dans cette partie de l'exercice, vous allez résoudre ce problème en ajoutant d'autres fonctionnalités.

Pour utiliser la régression polynomiale, notre hypothèse se présente sous la forme :

$$h\theta(\mathrm{x}) = \theta 0 + \theta 1 * (\mathrm{waterLevel}) + \theta 2 * (waterLevel)2 + \ldots \theta p * (waterLevel)^p$$
$$= \theta 0 + \theta 1 * \mathrm{x}1 + \theta 2 * \mathrm{x}2 + \ldots \theta p * XP.$$

Notez qu'en définissant *x1* = (*waterLevel*) ; *x2* = (*niveaud'eau*)^2 ; . . . ; *xp* = (waterLevel)p, *on obtient un modèle de régression linéaire où les caractéristiques sont les différentes puissances de la valeur d'origine (waterLevel).*

Vous allez maintenant ajouter d'autres entités à l'aide des puissances plus élevées de l'entité x existante dans le jeu de données. Votre tâche dans cette partie consiste à comprendre le code **dans polyFeatures.py dans** lequel la fonction mappe l'ensemble d'apprentissage d'origine *X* de taille *mx1* dans ses puissances supérieures. Plus précisément, lorsqu'un jeu d'apprentissage X de taille mx1 est passé à la fonction, celle-ci doit renvoyer une *matrice mxp* Xpoly, *où la colonne 1 contient les valeurs d'* origine de X, la colonne 2 contient les valeurs de *X2*, la colonne *3 contient les valeurs* de X3, et ainsi de suite. Notez que vous n'avez pas besoin de tenir compte de la puissance zéro dans cette fonction.

Vous disposez maintenant d'une fonction qui mappe les entités à une dimension supérieure**, et la partie 6 de ex4.py** l'appliquera à l'ensemble d'apprentissage, à l'ensemble de validation et à l'ensemble de test (que vous n'avez pas encore utilisé).

5.1 Apprentissage de la régression polynomiale

Une fois que vous avez terminé **polyFeatures.py**, le script **ex4.py** procédera à l'entraînement de la régression polynomiale à l'aide de votre fonction de coût de régression linéaire.

Gardez à l'esprit que même si nous avons des termes polynomiales dans notre vecteur de caractéristiques, nous résolvons toujours un problème d'optimisation de régression linéaire. Les termes polynomiaux se sont simplement transformés en caractéristiques que nous pouvons utiliser pour la régression linéaire. Nous utilisons la même fonction de coût et le même gradient que ceux que vous avez écrits pour la partie précédente de cet exercice.

Pour cette partie de l'exercice, vous utiliserez un polynôme de degré 8. Il s'avère que si nous exécutons l'entraînement directement sur les données projetées, cela ne fonctionnera pas bien car les entités seront mal mises à l'échelle (par exemple, un exemple avec x = 40 aura maintenant une caractéristique *x8* = 408 = 6.5x1012). Par conséquent, vous devrez utiliser la normalisation des fonctionnalités.

Avant d'apprendre les paramètres θ pour la régression polynomiale, **ex4.py** allons d'abord appeler **featureNor- malize** et normaliser les caractéristiques de l'ensemble d'apprentissage, en stockant les paramètres mu, sigma séparément. Nous avons déjà implémenté cette fonction pour vous et c'est la même fonction que lors de la première mission.

Après avoir appris les paramètres θ, vous devriez voir deux diagrammes (Figure 4) générés pour la régression polynomiale avec $\lambda = 0$.

D'après la figure 4 (à gauche), vous devriez voir que l'ajustement polynomial est capable de suivre très bien les points de données

- ainsi, l'obtention d'une faible erreur d'apprentissage. Cependant, l'ajustement polynomial est très complexe et tombe même aux extrêmes. Il s'agit d'un indicateur que le modèle de régression polynomiale surajuste les données d'apprentissage et ne généralise pas bien.

Pour mieux comprendre les problèmes avec le modèle non régularisé *(λ= 0), vous pouvez voir que la courbe d'apprentissage* (Figure 4 à droite) montre le même effet lorsque l'erreur d'apprentissage est faible, mais que l'erreur de validation est élevée. Il y a un écart entre les erreurs d'apprentissage et de validation, ce qui indique un problème de variance élevée.

Une façon de lutter contre le problème de surapprentissage (variance élevée) est d'ajouter la régularisation au modèle. Dans la section suivante, vous aurez l'occasion d'essayer différents *paramètres* λ pour voir comment la régularisation peut conduire à un meilleur modèle.

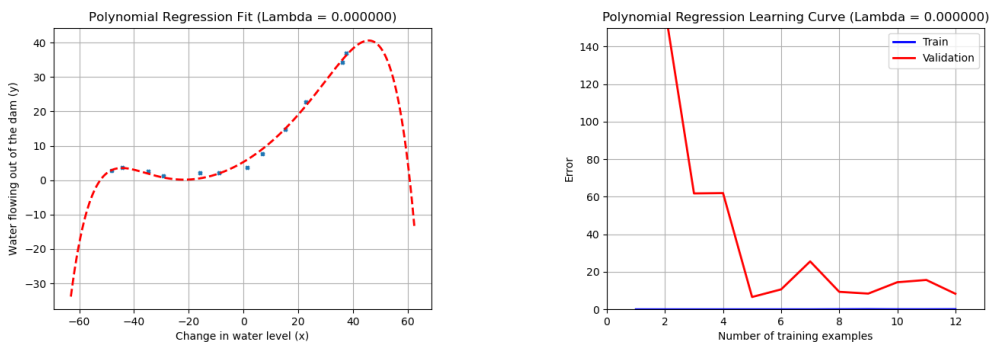

Figure 4 – Polynôme (a) ajusté pour λ = 0, (b) courbe d'apprentissage.

5.2 Réglage du paramètre de régularisation

Commençons par cette vidéo !

Dans cette section, vous allez observer comment le paramètre de régularisation affecte le biais-variance de la régression polynomiale régularisée. Vous devez maintenant modifier le paramètre lambda dans le **ex4.py** et essayer

$\lambda \in \{1, 100\}$. Pour chacune de ces valeurs, le script doit générer un ajustement polynomial aux données ainsi qu'une courbe d'apprentissage.

Pour $\lambda = 1$, vous devriez voir un ajustement polynomial (Figure 5) qui suit bien la tendance des données et une courbe d'apprentissage montrant que l'erreur de validation et d'apprentissage converge vers une valeur relativement faible. Cela montre que le modèle de régression polynomiale régularisée $\lambda = 1$ n'a pas les problèmes de biais élevé ou de variance élevée. En effet, il permet d'obtenir un bon compromis entre le biais et la variance.

Pour $\lambda = 100$, vous devriez voir un ajustement polynomial (Figure 6) qui ne suit pas bien les données. Dans ce cas, il y a trop de régularisation et le modèle n'est pas en mesure d'ajuster les données d'entraînement.

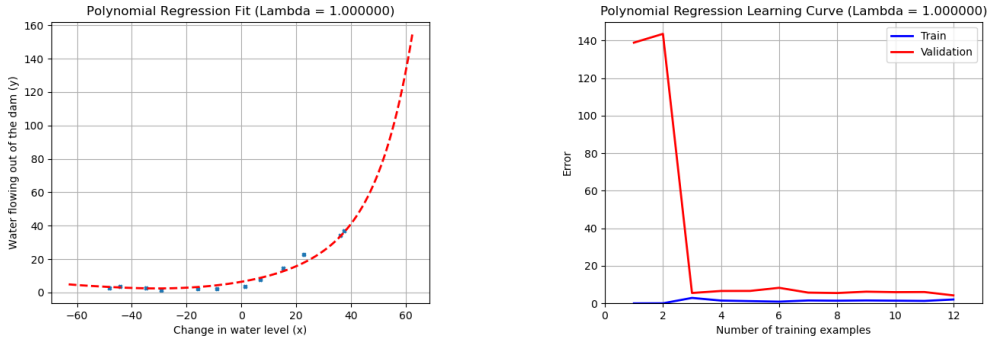

Figure 5 – Polynôme (a) ajusté pour $\lambda = 1$, (b) courbe d'apprentissage.

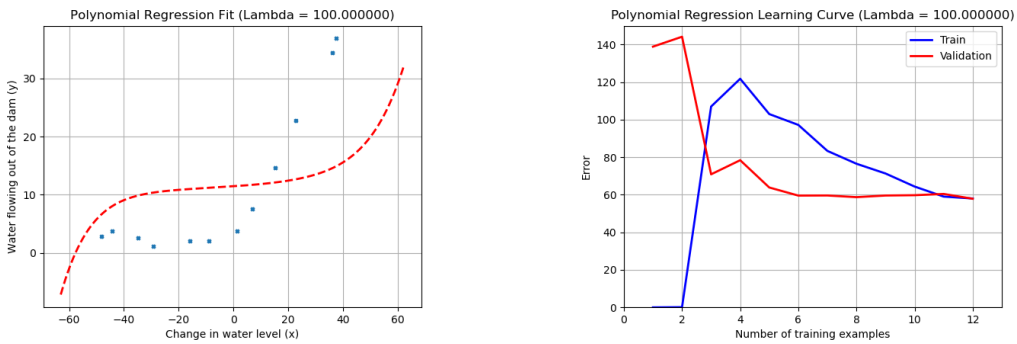

Figure 6 – Polynôme (a) ajusté pour $\lambda = 100$, (b) courbe d'apprentissage.

5.3 Sélection de λ à l'aide d'un jeu de validation

Dans les parties précédentes de l'exercice, vous avez observé que la valeur *de* λ peut affecter de manière significative les résultats de la régression polynomiale régularisée sur l'ensemble d'apprentissage et de validation. En particulier, un modèle sans régularisation ($\lambda = 0$) s'adapte bien à l'ensemble d'apprentissage, mais ne généralise pas. À l'inverse, un modèle avec trop de régularisation ($\lambda = 100$) ne s'adapte pas bien à l'ensemble d'apprentissage et à l'ensemble de test. Un bon choix de λ (par exemple, $\lambda = 1$) peut fournir un bon ajustement aux données.

Dans cette section, vous allez implémenter une méthode automatisée pour sélectionner le *paramètre* λ. Concrètement, vous allez utiliser un jeu de validation pour évaluer la qualité de chaque *valeur de* λ. Après avoir sélectionné la meilleure *valeur* λ à l'aide de l'ensemble de validation, nous pouvons ensuite évaluer le modèle sur l'ensemble de test pour estimer les performances du modèle sur des données réelles non vues.

Votre tâche consiste à compléter le code dans **validationCurve.py**. Plus précisément, vous devez utiliser la **fonction train- LinearReg** pour entraîner le modèle à l'aide de différentes valeurs *de* λ et calculer l'erreur d'apprentissage et l'erreur de validation.

Vous devriez essayer λ dans l'intervalle suivant : *0,* 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, *1,* 3, *10.*

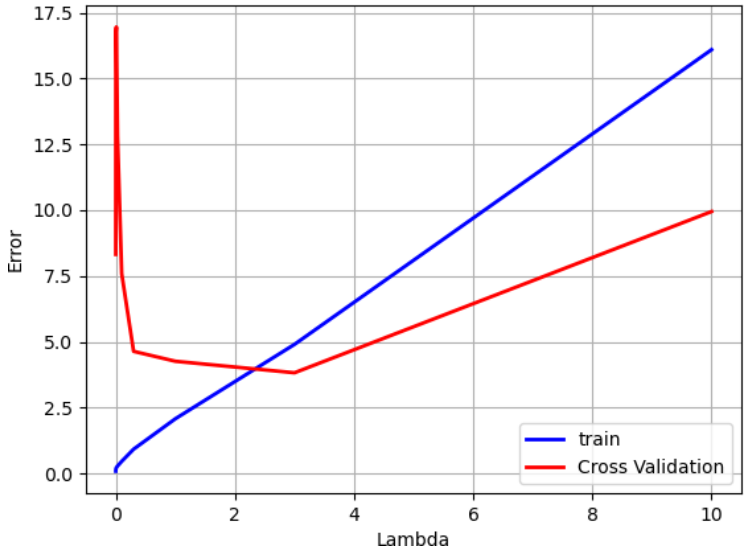

Figure 7 – Sélection λ à l'aide d'un jeu de validation.

Une fois que vous avez terminé le code, la partie suivante **de ex4.py** exécutera votre fonction et tracera une courbe de validation de l'erreur v.s. λ *qui vous permet de sélectionner le* paramètre λ à utiliser. Vous devriez voir un graphique similaire à la figure 7. Dans cette figure, nous pouvons voir que la meilleure valeur de λ est d'environ 3. En raison du caractère aléatoire des fractionnements d'apprentissage et de validation du jeu de données, l'erreur de validation peut parfois être inférieure à l'erreur d'apprentissage.

5.4 Erreur d'ensemble de test de calcul

Dans la partie précédente de l'exercice, vous avez implémenté du code pour calculer l'erreur de validation pour différentes valeurs du paramètre de régularisation λ. Cependant, pour obtenir une meilleure indication des performances du modèle dans le monde réel, il est important d'évaluer le modèle « final » sur un ensemble de test qui n'a été utilisé dans aucune partie de l'apprentissage (c'est-à-dire qu'il n'a été utilisé *ni pour sélectionner les paramètres* λ, ni pour apprendre les paramètres du modèle θ). Pour cet exercice, vous devez calculer l'erreur de test à l'aide de la meilleure valeur de λ que vous avez trouvée. Dans notre validation, nous avons obtenu une erreur de test de 3,8599 pour $\lambda = 3$ avec maxiter= 100 en entrée de la fonction **trainLinearReg**.

5.5 Exercice facultatif : Tracer des courbes d'apprentissage à l'aide d'exemples choisis au hasard

En pratique, en particulier pour les petits ensembles d'apprentissage, lorsque vous tracez des courbes d'apprentissage pour déboguer vos algorithmes, il est souvent utile de faire la moyenne sur plusieurs ensembles d'exemples sélectionnés au hasard pour déterminer l'erreur d'apprentissage et l'erreur de validation.

Concrètement, pour déterminer l'erreur d'apprentissage et l'erreur de validation pour i *exemples, vous devez d'abord sélectionner* au hasard i exemples dans l'*ensemble d'apprentissage et* i exemples dans l'ensemble de validation. Vous apprendrez ensuite les paramètres θ à l'aide de l'ensemble d'apprentissage choisi au hasard et évaluerez les paramètres θ *sur l'*ensemble d'apprentissage et l'ensemble de validation choisis au hasard. Les étapes ci-dessus doivent ensuite être répétées plusieurs fois (disons 50) et l'erreur moyenne doit être utilisée pour déterminer l'erreur d'apprentissage et l'erreur de validation pour *i* exemples.

Pour cet exercice facultatif, vous devez mettre en œuvre la stratégie ci-dessus pour calculer les courbes d'apprentissage. À titre de référence, la figure 8 montre la courbe d'apprentissage que nous avons obtenue pour la régression polynomiale avec $\lambda = 0{,}01$. Votre chiffre peut différer légèrement en raison de la sélection aléatoire d'exemples.

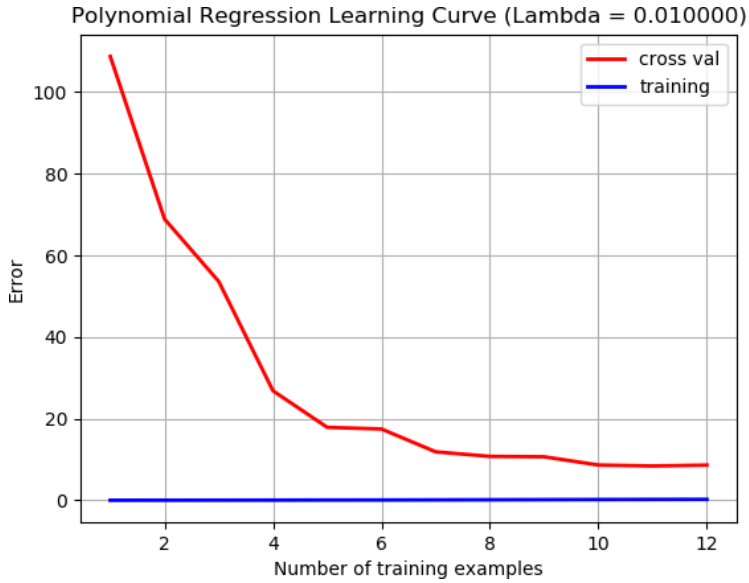

Figure 8 – Exercice facultatif : Courbe d'apprentissage avec des exemples choisis au hasard.

5.6 Il y a des questions auxquelles il faut répondre...

Le codage n'est qu'un prétexte pour comprendre les différents concepts du machine learning. **Ici aucune équation... Exprimez avec vos mots les concepts pour les comprendre.**

L'évaluation des performances est extrêmement importante ; il s'agit de bien comprendre les concepts du machine learning.

- Comment découpe-t-on un jeu de données pour entraîner et évaluer un modèle de prédiction ?
- Définissez ce qu'est la capacité d'un modèle de prédiction (c'est-à-dire sa complexité). Sur quel ensemble l'évalue-t-on ?
- Définissez le concept d'erreur de généralisation ? Sur quel ensemble l'évalue-t-on ?
- A quoi sert l'ensemble de validation ?
- A quoi sert la « courbe d'apprentissage » ?