

Exercice 1

Que faire avec ChatGPT (et ses amis) ?

8 points

1. La réponse de NotebookLM ou prompt suivant *peux tu écrire le code du lasso en python pour traiter les données du Boston Housing Dataset ?* est donnée ci-dessous.

Commentez cette réponse de NotebookLM en donnant les justifications mathématiques associées au code proposé ci-dessous et si nécessaire, proposez des améliorations.

Réponse de NotebookLM en italique :

Okay, let's craft a Python implementation of the Lasso without relying on scikit-learn, and apply it to the Boston Housing Dataset, while also splitting the data into training and validation sets. First, we will load the Boston Housing Dataset using pandas and numpy. Then we will create a function to split the data. After this, we implement the Lasso using the coordinate descent algorithm and finally, we train and evaluate the model.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_california_housing

def load_and_split_data():
    # Load the California housing dataset
    housing = fetch_california_housing()
    data = pd.DataFrame(housing.data, columns=housing.feature_names)
    target = pd.Series(housing.target, name='target')

    # Split the data into training and validation sets
    X_train, X_val, y_train, y_val = train_test_split(data, target, test_size=0.2,
                                                    random_state=42)

    # Standardize the data
    X_train = (X_train - X_train.mean()) / X_train.std()
    y_train = (y_train - y_train.mean()) / y_train.std()
    X_val = (X_val - X_val.mean()) / X_val.std()
    y_val = (y_val - y_val.mean()) / y_val.std()

    # Convert to numpy arrays
    X_train = np.array(X_train)
    X_val = np.array(X_val)
    y_train = np.array(y_train)
    y_val = np.array(y_val)

    return X_train, X_val, y_train, y_val
```

This function loads the California housing dataset, splits the data into training and validation sets (80/20 split), standardizes the data and returns the data as numpy arrays.

Lasso Implementation (Coordinate Descent) As mentioned in the sources, the Lasso can be efficiently solved using a coordinate descent algorithm. The sources describe a component-wise (CW) algorithm for the Lasso which iteratively updates each coefficient of the parameter vector

```
def compute_cost(X, y, weights, alpha):
    n_samples = X.shape[0]
    predictions = np.dot(X, weights)
    mse = (1 / (2 * n_samples)) * np.sum((predictions - y) ** 2)
    l1_penalty = alpha * np.sum(np.abs(weights[1:])) # Exclure le biais
    return mse + l1_penalty

def soft_threshold(rho, lam):
    """Soft thresholding function."""
    if rho > lam:
        return rho - lam
    elif rho < -lam:
        return rho + lam
    else:
        return 0

def lasso_coordinate_descent(X, y, lam, max_iter=10, tol=1e-4):
    """
    Implementation of the Lasso using coordinate descent.
    """
```

```

Parameters:
X (np.array): Input features (n_samples, n_features)
y (np.array): Target variable (n_samples)
lam (float): Regularization parameter
max_iter (int): Maximum number of iterations
tol (float): Tolerance for convergence

Returns:
beta (np.array): Estimated coefficients (n_features)
"""
    n_samples, n_features = X.shape
    beta = np.zeros(n_features)

    for iteration in range(max_iter):
        beta_old = beta.copy()
        for j in range(n_features):
            # Compute the residual with the jth coefficient set to zero
            r = y - np.dot(X, beta) + np.dot(X[:,j], beta[j])
            # Apply soft thresholding
            rho = np.dot(X[:,j], r)
            beta[j] = soft_threshold(rho, lam)

        if iteration % 1 == 0:
            cost = compute_cost(X, y, beta, lam)
            print(f"Iteration {iteration}: Cost = {cost}")

    # check for convergence
    if np.linalg.norm(beta - beta_old) < tol:
        break

    return beta

```

This code implements the coordinate descent algorithm as described in the sources, using a soft thresholding function. The lasso coordinate descent function returns the estimated coefficients β . *Training and Evaluation* Here's a complete example of how to use the above functions, split the data, train the model, and perform a basic evaluation :

```

# Load and split the data
X_train, X_val, y_train, y_val = load_and_split_data()

# Set the regularization parameter
lam = .1

# Train the Lasso model
beta = lasso_coordinate_descent(X_train, y_train, lam)
print("Estimated coefficients (beta):", beta)

# Make predictions on the validation set
y_pred = np.dot(X_val, beta)

# Evaluate the model (e.g., using Mean Squared Error)
mse = np.mean((y_val - y_pred)**2)
print("Validation Mean Squared Error:", mse)

```

Explanation and Key Points

- *Data Preprocessing* : The Boston Housing data is loaded and split into training and validation sets, and it is then standardised to ensure that all the features have a mean of 0 and standard deviation of 1 . This helps the optimization algorithm converge more quickly.
- *Coordinate Descent* : The core of this implementation is the `lasso_coordinate_descent` function. It iteratively updates each β_j while keeping the others fixed.
- *Soft Thresholding* : The `soft_threshold` function applies the shrinkage that is characteristic of the Lasso.
- *No Intercept Term* : The implementation does not explicitly include an intercept term, the intercept is assumed to be 0 given the standardisation of the data.
- *Convergence Check* : The algorithm includes a check for convergence by comparing the norm of the difference of the parameter vectors.
- *Evaluation* : The code computes the Mean Squared Error (MSE) on the validation set to assess the model's performance.

- *Choice of Regularization Parameter lam* : The value of lam (0.1 in this example) needs to be tuned using a technique such as cross-validation. A higher lam value will lead to more shrinkage (more coefficients set to zero), and vice versa.

Additional Considerations

- *Parameter Tuning* : The value for lam is set arbitrarily. In practice you need to tune this using a cross-validation strategy.
- *More Sophisticated Evaluation* : This implementation uses only the mean squared error. More sophisticated evaluation metrics may be appropriate for a real world project.
- *Alternative Algorithms* : Although this implementation uses coordinate descent, the sources also discuss proximal gradient descent and other optimization algorithms .
- *Adaptive Lasso* : The sources also describe other algorithms such as the Adaptive Lasso . This code provides a functional implementation of the Lasso algorithm without using scikit-learn, offering a solid starting point for more advanced projects.

Fin de la réponse de NotebookLM

L'exécution du code à donné les résultats suivants :

```
Iteration 0: Cost = 1.5502127664269435e+54
Iteration 1: Cost = 2.9450053382945687e+105
Iteration 2: Cost = 5.594752301372232e+156
Iteration 3: Cost = 1.0628589668986857e+208
Iteration 4: Cost = 2.0191585304677056e+259
Iteration 5: Cost = inf
Iteration 6: Cost = inf
Iteration 7: Cost = inf
Iteration 8: Cost = inf
Iteration 9: Cost = inf
Estimated coefficients (beta): [-6.45496553e+234 -1.30102735e+238 -3.45257484e+241
4.76639603e+245
5.60813130e+248 -6.69693895e+251 6.54947530e+253 9.99715914e+257]
Validation Mean Squared Error: inf
```

Exercice 2

Revue de code

6 points

1. Si vous aviez UNE recommandation à faire à cet autre auteur de ces lignes de code concernant sa méthodologie en apprentissage automatique (machine learning), quelle serait-elle ?

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV

n_samples, n_features, n_classes = 200, 10000, 2
rng = np.random.RandomState(42)
X = rng.standard_normal((n_samples, n_features))
y = rng.choice(n_classes, n_samples)
X_selected = SelectKBest(k=25).fit_transform(X, y)
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, random_state=42)
parameters = { "learning_rate": [0.1, 0.2],
               "max_depth": [3, 8],
               "subsample": [0.5, 1.0],
               }
BoostingCV_clf = GridSearchCV(ensemble.GradientBoostingClassifier(), parameters, cv
                               =10, n_jobs=-1)
BoostingCV_clf.fit(X_train, y_train)
y_pred = BoostingCV_clf.predict(X_test)
accuracy_score(y_test, y_pred)
```

2. On a demandé à un data scientist d'écrire un programme python pour estimer les paramètres d'un modèle linéaire à l'aide de la régression du lasso pondéré avec un paramètre $\lambda = 0, 1$,

$$\widehat{\beta}_w = \arg \min_{\beta \in \mathbb{R}^p} \frac{1}{2} \|y - X\beta\|^2 + \lambda \sum_{j=1}^p w_j |\beta_j|$$

et sans utiliser sklearn. Lea réponse suivante à été produit (par Gemini). Qu'en pensez vous ?

```

import numpy as np

def soft_thresholding(x, lambda_val): """Opérateur de seuillage doux."""
    return np.sign(x) * np.maximum(np.abs(x) - lambda_val, 0)

def weighted_lasso_proximal_gradient(X, y, weights, lam=0.1, lr=0.01, max_iter
=1000, tolerance=1e-4):
    """
    X: Matrice des donnees (n_samples, n_features).
    y: Vecteur cible (n_samples,).
    weights: Poids pour chaque caracteristique (n_features,).

    Returns: Vecteur des coefficients estimates.
    """
    beta = np.zeros(n_features) # Initialisation des coefficients
    X = (X - X.mean()) / np.sqrt(np.sum((X - X.mean())**2,axis=0))
    y = (y - y.mean()) / np.sqrt(np.sum((y - y.mean())**2))

    for _ in range(max_iter):
        g = X.T @ (X @ beta - y) #Gradient de la partie differentiable
        beta_old = beta.copy()

        for j in range(n_features):
            beta[j] = soft_thresholding(beta[j] - lr * g[j], lam * weights[j])

        if np.linalg.norm(beta - beta_old) < tolerance:
            break

    return beta

# Exemple d'utilisation
n_samples = 100
n_features = 10
X = np.random.rand(n_samples, n_features)
X_test = np.random.rand(n_samples, n_features)
y = X @ np.random.rand(n_features) + np.random.normal(0, 0.1, n_samples)
y_test = X_test @ np.random.rand(n_features) + np.random.normal(0, 0.1, n_samples)

w = np.random.rand(n_features)
beta_estimated = weighted_lasso_proximal_gradient(X, y, w, lam=0.1)
y_pred = X_test @ beta_estimated
error = np.sum((y_pred - y_test)**2)

```

Exercice 3

L'exam selon NoteboookLM

6 points

Comme NoteboookLM, répondez aux questions suivantes en 5 à 15 lignes.

1. **Sélection de variables et Régression linéaire** : Expliquez les objectifs de la sélection de variables dans le contexte de la régression linéaire.
2. **Implémentation du Lasso** : Des sources discutent de l'implémentation du Lasso

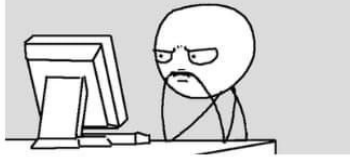
$$\min_{\beta} = \frac{1}{2} \|X\beta - y\|_2^2 + \lambda \|\beta\|_1,$$

comme un problème de Programmation Quadratique (QP). Expliquez pourquoi le Lasso peut être formulé comme un QP et décrivez la formulation QP correspondante. Quel est l'intérêt de formuler le lasso comme un QP ?

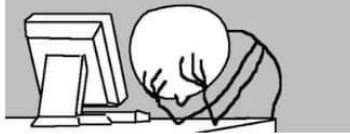
3. **Chemins de Régularisation** :
 - a) Expliquez ce que l'on entend par chemin de régularisation dans le contexte du Lasso.
 - b) Qu'est-ce que λ_{min} et λ_{max} pour le chemin de régularisation du Lasso et comment sont-ils calculés ? À quoi correspondent-ils en termes de paramètres estimés ?
4. **Algorithme du Gradient Proximal pour le Lasso** : Comment le pas d'apprentissage est-il choisi dans l'algorithme du gradient proximal pour le Lasso et pourquoi ?
5. **MCP (Minimax Concave Penalty)** : Décrivez la pénalité MCP (Minimax Concave Penalty) et expliquez pourquoi elle est non-convexe ?

Before Chat GPT

* Developer coding - 2 hours



* Developer debugging - 6 hours



After Chat GPT

* ChatGPT generating code - 5 min



* Developer debugging - 24 hours

