hw4

October 21, 2024

1 HW 4 - Ishaan Sathaye

1.1 Section A: Derivations

In the following assume we have p predictors in our model, where p may be larger than 1, and n observations.

Hint: You may want to use the notation: $sign(a) = 1 if a > 0, -1 if a < 0$

1. Give the *gradient* equation for Ordinary Least Squares Regression.
   - loss function: $L(\beta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^T x_i)^2$
   - $\nabla L(\beta) = \frac{d}{d\beta}\frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^T x_i)^2$
   - $\nabla L(\beta) = \frac{-2}{n}\sum_{i=1}^{n}(y_i - \beta^T x_i)x_i$
   - In matrix form: $\nabla L(\beta) = \frac{-2}{n}X^T(Y - \hat{Y})$
2. Give the *gradient* equation for Ridge Regression.
   - loss function: $L(\beta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^T x_i)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$
   - $\nabla L(\beta) = \frac{d}{d\beta}\frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^T x_i)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$
   - $\nabla L(\beta) = \frac{-2}{n}\sum_{i=1}^{n}(y_i - \beta^T x_i)x_i + 2\lambda\beta$
   - In matrix form: $\nabla L(\beta) = \frac{-2}{n}X^T(Y - \hat{Y}) + 2\lambda\beta$
3. Give the *gradient* equation for Lasso Regression.
   - loss function: $L(\beta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^T x_i)^2 + \lambda\sum_{j=1}^{p}|\beta_j|$
   - $\nabla L(\beta) = \frac{d}{d\beta}\frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^T x_i)^2 + \lambda\sum_{j=1}^{p}|\beta_j|$
   - $\nabla L(\beta) = \frac{-2}{n}\sum_{i=1}^{n}(y_i - \beta^T x_i)x_i + \lambda sign(\beta)$
   - In matrix form: $\nabla L(\beta) = \frac{-2}{n}X^T(Y - \hat{Y}) + \lambda sign(\beta)$
4. Give the *gradient* equation for Linear Regression with a loss function of $L(\beta) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^4 + \lambda\sum_{j=1}^{p}\beta_j^4$
   - $\nabla L(\beta) = -4\sum_{i=1}^{n}(y_i - \hat{y}_i)^3 x_i + 4\lambda\beta^3$
   - In matrix form: $\nabla L(\beta) = -4X^T(Y - \hat{Y})^3 + 4\lambda\beta^3$
5. Give the *gradient* equation for Linear Regression with a loss function of $L(\beta) = \sum_{i=1}^{n}|y_i - \hat{y}_i| + \lambda\sum_{j=1}^{p}|\beta_j|$
   - $\nabla L(\beta) = -\sum_{i=1}^{n} sign(y_i - \hat{y}_i)x_i + \lambda sign(\beta)$
   - In matrix form: $\nabla L(\beta) = -X^T sign(Y - \hat{Y}) + \lambda sign(\beta)$

1.2 Section B: Coding

1. Write a function to implement gradient descent for LASSO estimation on the cannabis data from last week.

[102]:
```python
import numpy as np
import pandas as pd

def fit_lasso(Y, X, lambda_, eta, stop_condition):
    beta = np.zeros(X.shape[1])
    current_beta = beta
    while True:
        # compute gradient
        gradient = compute_gradient(Y, X, lambda_, beta)
        # update beta
        beta = beta - eta*gradient
        # check stopping condition
        if check_stopping_condition_beta(beta, current_beta, stop_condition):
            break
        current_beta = beta
    return beta

def compute_gradient(Y, X, lambda_, beta):
    n = X.shape[0]
    return -2/n * X.T @ (Y - (X @ beta)) + lambda_ * np.sign(beta)

def check_stopping_condition_beta(beta, previous_beta, stop_condition):
    return np.linalg.norm(beta - previous_beta) < stop_condition
```

2. Write a function to perform cross-validation on a set of lambdas.

[140]:
```python
def tune_lambda_split(train, test, lam, metric):
    X_train = train.drop(columns=['Rating']).values
    X_train = np.hstack((np.ones((X_train.shape[0], 1)), X_train))
    y_train = train['Rating'].values
    X_test = test.drop(columns=['Rating']).values
    X_test = np.hstack((np.ones((X_test.shape[0], 1)), X_test))
    y_test = test['Rating'].values
    betas = fit_lasso(y_train, X_train, lam, eta=0.1, stop_condition=1e-3)
    y_pred = X_test @ betas  # Test set predictions

    if metric == 'r-sq':
        y_bar = np.mean(y_test)
        ss_tot = np.sum((y_test - y_bar) ** 2)
        ss_res = np.sum((y_test - y_pred) ** 2)
        r2 = 1 - ss_res / ss_tot
        return r2
    elif metric == 'mse':
        mse = np.mean((y_test - y_pred) ** 2)
        return mse
    elif metric == 'mae':
        mae = np.mean(np.abs(y_test - y_pred))
```

```
        return mae

def tune_lambda(df, lam_values, metric, k):
    n = df.shape[0]
    fold_size = n // k
    remainder = n % k
    metrics = []
    for lam in lam_values:
        metric_values = []
        for i in range(k):
            start_idx = i * fold_size
            if i == k - 1:
                end_idx = (i + 1) * fold_size + remainder
            else:
                end_idx = (i + 1) * fold_size
            test = df.iloc[start_idx:end_idx]
            train = df.drop(test.index)
            fold_metric = tune_lambda_split(train, test, lam, metric)
            metric_values.append(fold_metric)
        metrics.append(np.mean(metric_values))
    return pd.DataFrame({'lambda': lam_values, metric: metrics})
```

3. Apply your cross-validation function to the cannabis dataset to find the "best" lambda.

[141]:
```
df = pd.read_csv("../hw3/cannabis_full.csv")
predictors = df.drop(columns=['Strain', 'Type', 'Effects', 'Flavor', 'Rating'])
df_clean = df.dropna(subset=predictors.columns)
df_clean = pd.get_dummies(df_clean, columns=['Type'], drop_first=True)
# standardize the data
predictors = df_clean.drop(columns=['Strain', 'Effects', 'Flavor', 'Rating'])
predictors = (predictors - predictors.mean()) / predictors.std()
predictors['Rating'] = df_clean['Rating']
```

[142]:
```
lam_values = [0.001, 0.0001, 0.00001, 0.000001]
folds = 5
df_tune_rsq = tune_lambda(predictors, lam_values, 'r-sq', folds)
best_lambda_rsq = df_tune_rsq.loc[df_tune_rsq['r-sq'].idxmax()]['lambda']
print(best_lambda_rsq)

df_tune_mse = tune_lambda(predictors, lam_values, 'mse', folds)
best_lambda_mse = df_tune_mse.loc[df_tune_mse['mse'].idxmin()]['lambda']
print(best_lambda_mse)

df_tune_mae = tune_lambda(predictors, lam_values, 'mae', folds)
best_lambda_mae = df_tune_mae.loc[df_tune_mae['mae'].idxmin()]['lambda']
print(best_lambda_mae)
```

```
0.001
```

```
0.001
0.001
```

It seems that the best lambda is 0.001.

4. Fit your final LASSO model, using the "best" lambda, on the cannabis dataset. Interpret the results.

[147]:
```
# 4. Fit your final LASSO model, using the "best" lambda, on the cannabis␣
 ↪dataset. Interpret the results.

X = predictors.drop(columns=['Rating']).values
X = np.hstack((np.ones((X.shape[0], 1)), X))
y = predictors['Rating'].values
betas = fit_lasso(y, X, best_lambda_rsq, eta=0.1, stop_condition=1e-3)

predictor_names = predictors.columns
coefficients_with_names = list(zip(betas, ['Intercept'] +␣
 ↪list(predictor_names)))
largest_coefficients = sorted(coefficients_with_names, key=lambda x: abs(x[0]),␣
 ↪reverse=True)[:3]
print("Largest 3 coefficients:")
for coef, name in largest_coefficients:
    print(f"{name}: {coef}")
```

```
Largest 3 coefficients:
Intercept: 4.3183280649557405
Relaxed: 0.23746128314044956
Creative: 0.20452049803384287
```

These are the 3 largest coefficients and the interpretation of the results of the model being fit on the dataset.

Since the predictors are standardized, the coefficients can be interpreted as the expected change in the response variable for a one standard deviation change in the predictor.

- The intercept is 4.32, which is the expected value of the response variable when all predictors are 0.
- For a one standard deviation increase in the Relaxed predictor, the expected value of the response variable increases by 0.24.
- For a one standard deviation increase in the Creative predictor, the expected value of the response variable increases by 0.20.

1.3 Section C: Concepts

1. In class, when performing gradient descent to find LASSO estimates, we used as our initial values: $\beta = (0, 0, ..., 0)^T$. Suggest three different ways to choose the initial β's. Give some intuition for hjwy each one might be better than using all 0's.

- The first way is to initialize the β's with really small random values. This could lead to a much faster convergence since there are different start points for the algorithm to converge

to the minimum.

- The second way way is to initialize the β's with the OLS estimates. Once the initial estimates are found, the gradient descent can start here are further optimize the values.
- The third way is make educated guesses and this can be done through some initial information or intuition about the problem and its individual predictors. If we have this prior knowledge then better initial estimates can be made.

2. Consider the model:

house price $= \beta_0 + \beta_1 *$ size in square feet \$ + \$

with a squared-error loss and Ridge penalty loss function.

For the small example data:

sq footage | 1500 | 2200 | 3700 | 4100

price (in millions) | 1 | 2 | 3 | 14

Show the first 2 gradient descent updates with: - $\eta = 0.01$ - $\eta = 1.0$

[166]:
```
X = np.array([[1, 1500], [1, 2200], [1, 3700], [1, 4100]])
y = np.array([1, 2, 3, 14])

def fit_lasso_c(X, y, eta, lam, max_iter=2, stop_condition=1e-3):
    beta = np.zeros(X.shape[1])
    current_beta = beta
    for i in range(max_iter):
        # compute gradient
        gradient = compute_gradient_c(y, X, beta, lam)
        print(f"gradient after iteration {i}: {gradient}")
        # update beta
        beta = beta - eta*gradient
        print(f"beta after iteration {i}: {beta}")
        # check stopping condition
        if check_stopping_condition_beta(beta, current_beta, stop_condition):
            break
        current_beta = beta
    return beta

def compute_gradient_c(Y, X, beta, lambda_):
    n = X.shape[0]
    # using squared error loss and ridge penalty term:
    return -2/n * X.T @ (Y - (X @ beta)) + 2 * lambda_ * beta

print("eta = 0.01")
a = fit_lasso_c(X, y, 0.01, 1)
print(a)
print()
print("eta = 1.0")
b = fit_lasso_c(X, y, 1.0, 1)
```

```
print(b)
```

```
eta = 0.01
gradient after iteration 0: [-1.00e+01 -3.72e+04]
beta after iteration 0: [1.00e-01 3.72e+02]
gradient after iteration 1: [2.13899040e+06 6.99170412e+09]
beta after iteration 1: [-2.13898040e+04 -6.99166692e+07]
[-2.13898040e+04 -6.99166692e+07]

eta = 1.0
gradient after iteration 0: [-1.00e+01 -3.72e+04]
beta after iteration 0: [1.00e+01 3.72e+04]
gradient after iteration 1: [2.13900030e+08 6.99174095e+11]
beta after iteration 1: [-2.13900020e+08 -6.99174058e+11]
[-2.13900020e+08 -6.99174058e+11]
```

The first 2 gradient descent updates with $\eta = 0.01$ are: - gradient after iteration 0: [-1.00e+01 -3.72e+04] - new guesses after iteration 0: [1.00e-01 3.72e+02] - gradient after iteration 1: [2.13899040e+06 6.99170412e+09] - new guesses after iteration 1: [-2.13898040e+04 -6.99166692e+07]

The first 2 gradient descent updates with $\eta = 1.0$ are: - gradient after iteration 0: [-1.00e+01 -3.72e+04] - new guesses after iteration 0: [1.00e+01 3.72e+04] - gradient after iteration 1: [2.13900030e+08 6.99174095e+11] - new guesses after iteration 1: [-2.13900020e+08 -6.99174058e+11]