hw2

October 7, 2024

1 Homework 2 - Ishaan Sathaye

1.1 Section A: Theory

Consider the following model, where Y represents the price of a house and X represents its size in square feet.

$Y = \beta_0 + \beta_1 X + \epsilon$

1. Ridge penalty: Sq Error + $\lambda(\beta_0^2 + \beta_1^2)$ Derive the Ridge Regression estimators for β_0 and β_1 in terms of λ.

- Simplify Loss Function:
  - $l(\beta_0, \beta_1) = \sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2 + \lambda(\beta_0^2 + \beta_1^2)$
  - $l(\beta_0, \beta_1) = \sum_{i=1}^{n}(y_i^2 - 2y_i\beta_0 - 2y_i\beta_1 x_i + \beta_0^2 + 2\beta_0\beta_1 x_i + \beta_1^2 x_i^2) + \lambda(\beta_0^2 + \beta_1^2)$
- Take Partial Derivate with respect to β_0 first:
  - $\frac{\partial l}{\partial \beta_0} = -2\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i) + 2\lambda\beta_0$
  - $0 = -2\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i) + 2\lambda\beta_0$
  - $\lambda\beta_0 = \sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)$
- Take Partial Derivate with respect to β_1 next:
  - $\frac{\partial l}{\partial \beta_1} = -2\sum_{i=1}^{n} x_i(y_i - \beta_0 - \beta_1 x_i) + 2\lambda\beta_1$
  - $0 = -2\sum_{i=1}^{n} x_i(y_i - \beta_0 - \beta_1 x_i) + 2\lambda\beta_1$
  - $\lambda\beta_1 = \sum_{i=1}^{n} x_i(y_i - \beta_0 - \beta_1 x_i)$
- Solve this system of equations:
  - $\lambda\beta_0 = \sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)$
  - $\lambda\beta_1 = \sum_{i=1}^{n} x_i(y_i - \beta_0 - \beta_1 x_i)$
- Simplify first equation:
  - $n\beta_0 + \beta_1\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i - \lambda\beta_0$
  - $(n+\lambda)\beta_0 + \beta_1\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$
- Simplify second equation:
  - $\beta_0\sum_{i=1}^{n} x_i + \beta_1\sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} x_i y_i - \lambda\beta_1$
  - $\beta_0\sum_{i=1}^{n} x_i + (\sum_{i=1}^{n} x_i^2 + \lambda)\beta_1 = \sum_{i=1}^{n} x_i y_i$
- Again, solve this system of equations:
  - $(n+\lambda)\beta_0 + \beta_1\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$
  - $\beta_0\sum_{i=1}^{n} x_i + (\sum_{i=1}^{n} x_i^2 + \lambda)\beta_1 = \sum_{i=1}^{n} x_i y_i$
- Solve for β_0:
  - $\beta_0 = \frac{\sum_{i=1}^{n} y_i - \beta_1\sum_{i=1}^{n} x_i}{n+\lambda}$
- Solve for β_1 by substituting equation for β_0:

- $\frac{\sum_{i=1}^n y_i - \beta_1 \sum_{i=1}^n x_i}{n+\lambda} \sum_{i=1}^n x_i + (\sum_{i=1}^n x_i^2 + \lambda)\beta_1 = \sum_{i=1}^n x_i y_i$

- Multiply by $n+\lambda$ and expand:
  - $\sum_{i=1}^n y_i \sum_{i=1}^n x_i - \beta_1(\sum_{i=1}^n x_i)^2 + (\sum_{i=1}^n x_i^2 + \lambda)\beta_1(n+\lambda) = \sum_{i=1}^n x_i y_i (n+\lambda)$
- Combine:
  - $\sum_{i=1}^n y_i \sum_{i=1}^n x_i - \beta_1(\sum_{i=1}^n x_i)^2 + (n+\lambda)(\sum_{i=1}^n x_i^2 + \lambda)\beta_1 = \sum_{i=1}^n x_i y_i (n+\lambda)$
  - $\beta_1(\sum_{i=1}^n x_i)^2 + (n+\lambda)(\sum_{i=1}^n x_i^2 + \lambda)\beta_1 = \sum_{i=1}^n x_i y_i (n+\lambda) - \sum_{i=1}^n y_i \sum_{i=1}^n x_i$
- Finally, solve for β_1:
  - $\beta_1 = \frac{\sum_{i=1}^n x_i y_i (n+\lambda) - \sum_{i=1}^n y_i \sum_{i=1}^n x_i}{(\sum_{i=1}^n x_i)^2 + (n+\lambda)(\sum_{i=1}^n x_i^2 + \lambda)}$
- Finally, substitute equation for β_1 into equation for β_0:
  - $\beta_0 = \frac{\sum_{i=1}^n y_i - \frac{\sum_{i=1}^n x_i y_i (n+\lambda) - \sum_{i=1}^n y_i \sum_{i=1}^n x_i}{(\sum_{i=1}^n x_i)^2 + (n+\lambda)(\sum_{i=1}^n x_i^2 + \lambda)} \sum_{i=1}^n x_i}{n+\lambda}$

2. LASSO penalty: Sq Error + $\lambda(|\beta_0| + |\beta_1|)$ Derive the LASSO Regression estimators for β_0 and β_1 in terms of λ, assuming that both β_0 and β_1 are positive.

- Simplify Loss Function:
  - $l(\beta_0, \beta_1) = \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i)^2 + \lambda(|\beta_0| + |\beta_1|)$
  - $l(\beta_0, \beta_1) = \sum_{i=1}^n (y_i^2 - 2y_i\beta_0 - 2y_i\beta_1 x_i + \beta_0^2 + 2\beta_0\beta_1 x_i + \beta_1^2 x_i^2) + \lambda(|\beta_0| + |\beta_1|)$
- Take Partial Derivative with respect to β_0:
  - $\frac{\partial l}{\partial \beta_0} = -2\sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i) + \lambda$
  - $0 = -2\sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i) + \lambda$
  - $\frac{\lambda}{2} = \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i)$
- Take Partial Derivative with respect to β_1:
  - $\frac{\partial l}{\partial \beta_1} = -2\sum_{i=1}^n x_i(y_i - \beta_0 - \beta_1 x_i) + \lambda$
  - $0 = -2\sum_{i=1}^n x_i(y_i - \beta_0 - \beta_1 x_i) + \lambda$
  - $\frac{\lambda}{2} = \sum_{i=1}^n x_i(y_i - \beta_0 - \beta_1 x_i)$
- Solve this system of equations:
  - $\frac{\lambda}{2} = \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i)$
  - $\frac{\lambda}{2} = \sum_{i=1}^n x_i(y_i - \beta_0 - \beta_1 x_i)$
- Solve for β_0:
  - $\sum_{i=1}^n y_i - \beta_0 n - \beta_1 \sum_{i=1}^n x_i = \frac{\lambda}{2}$
  - $\beta_0 = \frac{\sum_{i=1}^n y_i - \beta_1 \sum_{i=1}^n x_i - \frac{\lambda}{2}}{n}$
- Solve for β_1 by substituting equation for β_0:
  - $\sum_{i=1}^n x_i y_i - \beta_0 \sum_{i=1}^n x_i - \beta_1 \sum_{i=1}^n x_i^2 = \frac{\lambda}{2}$
  - $\sum_{i=1}^n x_i y_i - \frac{\sum_{i=1}^n y_i - \beta_1 \sum_{i=1}^n x_i - \frac{\lambda}{2}}{n} \sum_{i=1}^n x_i - \beta_1 \sum_{i=1}^n x_i^2 = \frac{\lambda}{2}$
- Multiply by n and expand:
  - $n\sum_{i=1}^n x_i y_i - \sum_{i=1}^n y_i \sum_{i=1}^n x_i - \beta_1 \sum_{i=1}^n x_i \sum_{i=1}^n x_i - \frac{\lambda}{2}\sum_{i=1}^n x_i - \beta_1 \sum_{i=1}^n x_i^2 n = \frac{\lambda}{2} n$
- Combine:
  - $n\sum_{i=1}^n x_i y_i - \sum_{i=1}^n y_i \sum_{i=1}^n x_i - \beta_1 \sum_{i=1}^n x_i \sum_{i=1}^n x_i - \frac{\lambda}{2}\sum_{i=1}^n x_i - \beta_1 \sum_{i=1}^n x_i^2 n = \frac{\lambda}{2} n$
  - $\beta_1 \sum_{i=1}^n x_i \sum_{i=1}^n x_i + \beta_1 \sum_{i=1}^n x_i^2 n = n\sum_{i=1}^n x_i y_i - \sum_{i=1}^n y_i \sum_{i=1}^n x_i - \frac{\lambda}{2}\sum_{i=1}^n x_i - \frac{\lambda}{2} n$
- Finally, solve for β_1:
  - $\beta_1 = \frac{n\sum_{i=1}^n x_i y_i - \sum_{i=1}^n y_i \sum_{i=1}^n x_i - \frac{\lambda}{2}\sum_{i=1}^n x_i - \frac{\lambda}{2} n}{\sum_{i=1}^n x_i \sum_{i=1}^n x_i + \sum_{i=1}^n x_i^2 n}$
- Finally, substitute equation for β_1 into equation for β_0:
  - $\beta_0 = \frac{\sum_{i=1}^n y_i - \frac{n\sum_{i=1}^n x_i y_i - \sum_{i=1}^n y_i \sum_{i=1}^n x_i - \frac{\lambda}{2}\sum_{i=1}^n x_i - \frac{\lambda}{2} n}{\sum_{i=1}^n x_i \sum_{i=1}^n x_i + \sum_{i=1}^n x_i^2 n} \sum_{i=1}^n x_i - \frac{\lambda}{2}}{n}$

1.2 Section B: Coding

[39]:
```python
import pandas as pd
import numpy as np

# load data
data = pd.read_csv('../hw1/AmesHousing.csv')
data = data[['SalePrice', 'Lot Area']]

data.head()
```

[39]:

	SalePrice	Lot Area
0	215000	31770
1	105000	11622
2	172000	14267
3	244000	11160
4	189900	13830

1. Create a function to compute the Ridge estimator from scratch. λ should be a user-input option.

[40]:
```python
# Ridge regression in matrix form:
# beta = (X^T X + lambda I)^-1 X^T y

def ridge_estimator(X, y, lam):
    n, p = X.shape
    # add 1s column for intercept
    X = np.hstack([np.ones((n, 1)), X])
    I = np.eye(p + 1)
    betas = np.linalg.inv(X.T @ X + lam * I) @ X.T @ y
    return betas
```

2. Write a function called `try_many_lambdas()` that takes in a vector or data frame of possible values, and then returns the estimators corresponding to each value.

[41]:
```python
def try_many_lambdas(lam_values):
    X = data[['Lot Area']].values
    y = data['SalePrice'].values
    betas = {}
    for lam in lam_values:
        beta = ridge_estimator(X, y, lam)
        betas[lam] = beta
    return betas
```

3. Write a function to compute classical validation metrics for all λ's.

[42]:
```python
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
```

```python
from math import log

def tune_lambda_classic(data, lam_values, metric):
    X = data[['Lot Area']].values
    y = data['SalePrice'].values
    metrics = []
    for lam in lam_values:
        beta = ridge_estimator(X, y, lam)
        y_pred = beta[0] + beta[1] * X
        if metric == 'r-sq-adj':
            r2 = r2_score(y, y_pred)
            n, p = X.shape
            r2_adj = 1 - (1 - r2) * (n - 1) / (n - p - 1)
            metrics.append(r2_adj)
        elif metric == 'AIC':
            n, p = X.shape
            mse = mean_squared_error(y, y_pred)
            aic = n * log(mse) + 2 * (p + 1)
            metrics.append(aic)
        elif metric == 'BIC':
            n, p = X.shape
            mse = mean_squared_error(y, y_pred)
            bic = n * log(mse) + (p + 1) * log(n)
            metrics.append(bic)
    return pd.DataFrame({'lambda': lam_values, metric: metrics})
```

4. Run `tune_lambda_classic()` on the AMES data. What was the best choice of λ, and the corresponding estimators?

[43]:
```python
lam_values = np.linspace(0, 1000, 200) # 200 values from 0 to 1000

df_rsq = tune_lambda_classic(data, lam_values, 'r-sq-adj')
df_aic = tune_lambda_classic(data, lam_values, 'AIC')
df_bic = tune_lambda_classic(data, lam_values, 'BIC')
```

[44]:
```python
# get the highest r-sq-adj
df_rsq.loc[df_rsq['r-sq-adj'].idxmax()]
```

[44]:
```
lambda      0.000000
r-sq-adj    0.070731
Name: 0, dtype: float64
```

[45]:
```python
# get the lowest AIC
df_aic.loc[df_aic['AIC'].idxmin()]
```

[45]:
```
lambda         0.000000
AIC        65936.878884
```

```
Name: 0, dtype: float64
```

[46]:
```
# get the lowest BIC
df_bic.loc[df_bic['BIC'].idxmin()]
```

[46]:
```
lambda        0.0000
BIC       65948.8444
Name: 0, dtype: float64
```

[47]:
```
best_lam = df_rsq.loc[df_rsq['r-sq-adj'].idxmax()]['lambda']
betas = try_many_lambdas([best_lam])
print(best_lam, betas[best_lam])
```

```
0.0 [1.53373893e+05 2.70224462e+00]
```

- The best choice of λ was 0, and the corresponding estimators were $\beta_0 = 153373$ and $\beta_1 = 2.702$.

5. Write a function to tune your lambda values.

[48]:
```
def tune_lambda_split(train, test, lam_values, metric):
    X_train = train[['Lot Area']].values
    y_train = train['SalePrice'].values
    X_test = test[['Lot Area']].values
    y_test = test['SalePrice'].values
    metrics = []
    for lam in lam_values:
        beta = ridge_estimator(X_train, y_train, lam)
        y_pred = beta[0] + beta[1] * X_test
        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
            metrics.append(r2)
        elif metric == 'mse':
            mse = np.mean((y_test - y_pred) ** 2)
            metrics.append(mse)
        elif metric == 'mae':
            mae = np.mean(np.abs(y_test - y_pred))
            metrics.append(mae)
    return pd.DataFrame({'lambda': lam_values, metric: metrics})
```

6. Run your `tune_lambda_split` function on the AMES housing data, for a random 80/20 test/training split, using R-squared as your metric. What was the best choice of λ, and the corresponding estimators?

[49]:
```
from sklearn.model_selection import train_test_split

train, test = train_test_split(data, test_size=0.2)
df_rsq = tune_lambda_split(train, test, lam_values, 'r-sq')
best_lam = df_rsq.loc[df_rsq['r-sq'].idxmax()]['lambda']
betas = try_many_lambdas([best_lam])
print(best_lam, betas[best_lam])
```

```
0.0 [1.53373893e+05 2.70224462e+00]
```

[50]:
```
# get the best lambda using mse
df_mse = tune_lambda_split(train, test, lam_values, 'mse')
best_lam_mse = df_mse.loc[df_mse['mse'].idxmin()]['lambda']

# get the best lambda using mae
df_mae = tune_lambda_split(train, test, lam_values, 'mae')
best_lam_mae = df_mae.loc[df_mae['mae'].idxmin()]['lambda']

print(best_lam_mse, best_lam_mae)
```

```
0.0 100.50251256281408
```

The best choice of λ was 0, and the corresponding estimators were $\beta_0 = 153373$ and $\beta_1 = 2.702$.

7. Write a function called `tune_lambda_cv`, which performs v-fold cross-validation.

[51]:
```
def tune_lambda_cv(data, lam_values, metric, v):
    n = len(data)
    splits = np.array_split(data, v)
    metric_vals = []
    for lam in lam_values:
        metric_sum = 0
        for i in range(v):
            # get test and train data based on split
            test = splits[i]
            train = pd.concat([x for j, x in enumerate(splits) if j != i])
            df = tune_lambda_split(train, test, [lam], metric)
            metric_sum += df[metric].values[0]
        metric_vals.append(metric_sum / v)
    return pd.DataFrame({'lambda': lam_values, metric: metric_vals})
```

8. Run your `tune_lambda_cv` function on the AMES housing data with 5 folds using R-squared as your metric. What was the best choice of λ, and the corresponding estimators?

[52]:
```
df_rsq = tune_lambda_cv(data, lam_values, 'r-sq', 5)
best_lam = df_rsq.loc[df_rsq['r-sq'].idxmax()]['lambda']
betas = try_many_lambdas([best_lam])
print(best_lam, betas[best_lam])
```

```
0.0 [1.53373893e+05 2.70224462e+00]
```

The best choice of λ was 0, and the corresponding estimators were $\beta_0 = 153373$ and $\beta_1 = 2.702$.

9. Write a function called `tune_lambda_loo`, which performs leave-one-out cross-validation.

[53]:
```
def tune_lambda_loo(data, lam_values, metric):
    n = len(data)
    metric_vals = []

    X = data[['Lot Area']].values
    y = data['SalePrice'].values

    for lam in lam_values:
        pred_vals = np.zeros(n)
        for i in range(n):
            X_train = np.delete(X, i, axis=0)
            y_train = np.delete(y, i)

            betas = ridge_estimator(X_train, y_train, lam)

            pred_vals[i] = betas[0] + betas[1] * X[i]

        # Calculate the chosen metric
        if metric == 'r-sq':
            y_mean = np.mean(y)
            ss_tot = np.sum((y - y_mean) ** 2)
            ss_res = np.sum((y - pred_vals) ** 2)
            r2 = 1 - ss_res / ss_tot
            metric_vals.append(r2)
        elif metric == 'mse':
            mse = np.mean((y - pred_vals) ** 2)
            metric_vals.append(mse)
        elif metric == 'mae':
            mae = np.mean(np.abs(y - pred_vals))
            metric_vals.append(mae)

    return pd.DataFrame({'lambda': lam_values, metric: metric_vals})
```

10. Run your `tune_lambda_loo` function on the AMES housing data, using R-squared as your metric. What was the best choice of λ, and the corresponding estimators?

[54]:
```
df_rsq = tune_lambda_loo(data, lam_values, 'r-sq')
best_lam = df_rsq.loc[df_rsq['r-sq'].idxmax()]['lambda']
betas = try_many_lambdas([best_lam])
print(best_lam, betas[best_lam])
```

```
0.0 [1.53373893e+05 2.70224462e+00]
```

The best choice of λ was 0, and the corresponding estimators were $\beta_0 = 153373$ and $\beta_1 = 2.702$.

1.3 Section C: Concepts

1. Under what circumstances would your Ridge estimators be equal to 0?

- The Ridge estimators would be equal to 0 when the penalty term is greater than the sum of the squared errors. This means when λ becomes really large, the penalty term will dominate the loss function which involves the sum of squared coeffs. This will force the coefficients to start to shrink towards 0.

2. Under what circumstances would your LASSO estimators be equal to 0?

- The LASSO estimators can be exactly 0 when λ is large enough. This is because the LASSO penalty term is the sum of the absolute values of the coefficients. At a certain point, the betas would become 0 with a certain value of λ, but greater than that the values of betas become negative.

3. Consider the simple toy dataset

sq footage (in thousands) | 1 | 2 | 3 | 4 |

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

Compute the Ridge and LASSO estimators for $\lambda = 0.1$, $\lambda = 10$, and $\lambda = 100$.

[55]:

```
def lasso_estimator(X, y, lam):
    n, p = X.shape
    # add 1s column for intercept
    X = np.hstack([np.ones((n, 1)), X])
    betas = np.zeros(p + 1)
    beta_1_numerator = n * np.sum(X[:, 1] * y) - np.sum(y) * np.sum(X[:, 1]) - lam / 2 * np.sum(X[:, 1]) - lam / 2 * n
    beta_1_denominator = np.sum(X[:, 1] ** 2) * n + np.sum(X[:, 1]) ** 2
    betas[1] = beta_1_numerator / beta_1_denominator
    beta_0_numerator = np.sum(y) - beta_1_numerator * np.sum(X[:, 1]) / n - lam / 2
    betas[0] = beta_0_numerator / n
    return betas
```

[56]:

```
X = np.array([[1], [2], [3], [4]])
y = np.array([1, 2, 3, 14])

# for lambda = 0.1
ridge_betas = ridge_estimator(X, y, 0.1)
lasso_betas = lasso_estimator(X, y, 0.1)
print('Lambda:', 0.1)
print('Ridge:', ridge_betas)
print('Lasso:', lasso_betas)
print()

# for lambda = 10
ridge_betas = ridge_estimator(X, y, 10)
```

```
lasso_betas = lasso_estimator(X, y, 10)
print('Lambda:', 10)
print('Ridge:', ridge_betas)
print('Lasso:', lasso_betas)
print()

# for lambda = 100
ridge_betas = ridge_estimator(X, y, 100)
lasso_betas = lasso_estimator(X, y, 100)
print('Lambda:', 100)
print('Ridge:', ridge_betas)
print('Lasso:', lasso_betas)
```

```
Lambda: 0.1
Ridge: [-4.18624519  3.71636053]
Lasso: [-44.575        0.36045455]

Lambda: 10
Ridge: [0.2173913  1.69565217]
Lasso: [-2.5         0.04545455]

Lambda: 100
Ridge: [0.14157973 0.52757079]
Lasso: [380.          -2.81818182]
```

4. How did your estimators change as λ got bigger for Ridge and LASSO? Why?

- As λ got bigger for Ridge and lasso, the slope estimator β_1 got smaller. This is because a penalty is being applied to the loss function which forces the coefficients to shrink towards 0. Essentially, the model will now underfit with bigger lambda since we are getting close to predicting the mean value of y for everything. This is especially true for LASSO where the coefficients can become exactly 0. However for both models, the intercept β_0 increased as lambda got bigger.

5. You now observe a new observation: A house that is 5000 square feet and costs \$5 million. Which of your six sets of estimators (three lambda values each for Ridge and LASSO) predicted this new value the best?

[57]:
```
y_true = 5
# ridge with lambda = 0.1
ridge_betas = ridge_estimator(X, y, 0.1)
y_pred = ridge_betas[0] + ridge_betas[1] * 5
print('Ridge with lambda = 0.1:', y_pred)
abs_error = np.abs(y_true - y_pred)
print('Absolute error:', abs_error)

# lasso with lambda = 0.1
lasso_betas = lasso_estimator(X, y, 0.1)
y_pred = lasso_betas[0] + lasso_betas[1] * 5
```

```python
print('Lasso with lambda = 0.1:', y_pred)
abs_error = np.abs(y_true - y_pred)
print('Absolute error:', abs_error)
print()

# ridge with lambda = 10
ridge_betas = ridge_estimator(X, y, 10)
y_pred = ridge_betas[0] + ridge_betas[1] * 5
print('Ridge with lambda = 10:', y_pred)
abs_error = np.abs(y_true - y_pred)
print('Absolute error:', abs_error)

# lasso with lambda = 10
lasso_betas = lasso_estimator(X, y, 10)
y_pred = lasso_betas[0] + lasso_betas[1] * 5
print('Lasso with lambda = 10:', y_pred)
abs_error = np.abs(y_true - y_pred)
print('Absolute error:', abs_error)
print()

# lasso with lambda = 100
lasso_betas = lasso_estimator(X, y, 100)
y_pred = lasso_betas[0] + lasso_betas[1] * 5
print('Lasso with lambda = 100:', y_pred)
abs_error = np.abs(y_true - y_pred)
print('Absolute error:', abs_error)

# lasso with lambda = 100
lasso_betas = lasso_estimator(X, y, 100)
y_pred = lasso_betas[0] + lasso_betas[1] * 5
print('Lasso with lambda = 100:', y_pred)
abs_error = np.abs(y_true - y_pred)
print('Absolute error:', abs_error)
```

```
Ridge with lambda = 0.1: 14.395557454079452
Absolute error: 9.395557454079452
Lasso with lambda = 0.1: -42.77272727272727
Absolute error: 47.77272727272727

Ridge with lambda = 10: 8.695652173913043
Absolute error: 3.695652173913043
Lasso with lambda = 10: -2.2727272727272725
Absolute error: 7.2727272727272725

Lasso with lambda = 100: 365.90909090909093
Absolute error: 360.90909090909093
Lasso with lambda = 100: 365.90909090909093
```

```
Absolute error: 360.90909090909093
```

Ridge with $\lambda = 10$ predicted the new value the best with an estimated price of $3.69 million.

6. Run your `tune_lambda_cv()` function setting the number of splits to n (the number of rows in the AMES dataset) and using R-squared as your metric. Why does it break? How is this different from Leave-One-Out cross-validation?

- The below code breaks due to a divide by zero error. This happens because the number of splits is now equal to the number of rows. Due to this, the training set becomes empty and so it breaks. This is different from LOO because LOO only leaves out one observation at a time, so the training set is never empty.

```
[ ]: df_rsq = tune_lambda_cv(data, lam_values, 'r-sq', len(data))
```