Regression

Contents

In this chapter, you will be introduced to regression, and build models to predict sales values using a dataset on advertising expenditure. You will learn about the mechanics of linear regression and common performance metrics such as R-squared and root mean squared error. You will perform k-fold cross-validation, and apply regularization to regression models to reduce the risk of overfitting.

Introduction to regression

Creating features

In this chapter, you will work with a dataset called `sales_df`, which contains information on advertising campaign expenditure across different media types, and the number of dollars generated in sales for the respective campaign. The dataset has been preloaded for you. Here are the first two rows:

```
     tv        radio      social_media    sales
1    13000.0   9237.76    2409.57         46677.90
2    41000.0   15886.45   2913.41         150177.83
```

You will use the advertising expenditure as features to predict sales values, initially working with the `"radio"` column. However, before you make any predictions you will need to create the feature and target arrays, reshaping them to the correct format for scikit-learn.

- Create `X`, an array of the values from the `sales_df` DataFrame's `"radio"` column.
- Create `y`, an array of the values from the `sales_df` DataFrame's `"sales"` column.
- Reshape `X` into a two-dimensional NumPy array.
- Print the shape of `X` and `y`.

```
# edited/added
sales_df = pd.read_csv("archive/Supervised-Learning-with-scikit-
learn/datasets/advertising_and_sales_clean.csv").drop(['influencer'], axis = 1)

import numpy as np

# Create X from the radio column's values
X = sales_df["radio"].values

# Create y from the sales column's values
y = sales_df["sales"].values

# Reshape X
X = X.reshape(-1, 1)

# Check the shape of the features and targets
print(X.shape, y.shape)
```

```
## (4546, 1) (4546,)
```

Excellent! See that there are 4546 values in both arrays? Now let's build a linear regression model!

Building a linear regression model

Now you have created your feature and target arrays, you will train a linear regression model on all feature and target values.

As the goal is to assess the relationship between the feature and target values there is no need to split the data into training and test sets.

`X` and `y` have been preloaded for you as follows:

```
y = sales_df["sales"].values
X = sales_df["radio"].values.reshape(-1, 1)
```

- Import `LinearRegression`.
- Instantiate a linear regression model.
- Predict sales values using `X`, storing as `predictions`.

```
# Import LinearRegression
from sklearn.linear_model import LinearRegression

# Create the model
reg = LinearRegression()

# Fit the model to the data
reg.fit(X, y)

# Make predictions
```

```
## LinearRegression()
```

```
predictions = reg.predict(X)

print(predictions[:5])
```

```
## [ 95491.17119147 117829.51038393 173423.38071499 291603.11444202
##  111137.28167129]
```

Great model building! See how sales values for the first five predictions range from $95,000 to over $290,000. Let's visualize the model's fit.

Visualizing a linear regression model

Now you have built your linear regression model and trained it using all available observations, you can visualize how well the model fits the data. This allows you to interpret the relationship between `radio` advertising expenditure and `sales` values.

The variables `X`, an array of `radio` values, `y`, an array of `sales` values, and `predictions`, an array of the model's predicted values for `y` given `X`, have all been preloaded for you from the previous exercise.

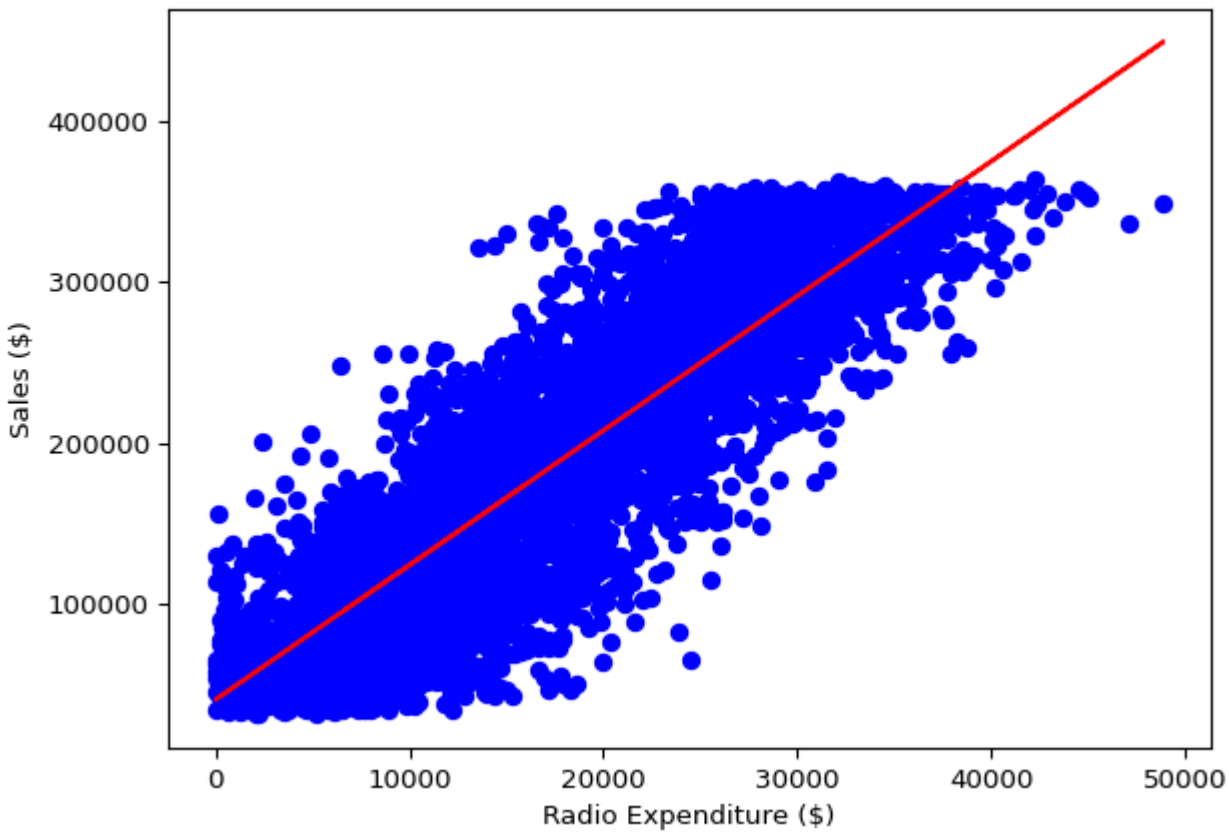
- Import `matplotlib.pyplot` as `plt`.
- Create a scatter plot visualizing `y` against `X`, with observations in blue.
- Draw a red line plot displaying the predictions against `X`.
- Display the plot.

```
# Import matplotlib.pyplot
import matplotlib.pyplot as plt

# Create scatter plot
plt.scatter(X, y, color="blue")

# Create line plot
plt.plot(X, predictions, color="red")
plt.xlabel("Radio Expenditure ($)")
plt.ylabel("Sales ($)")

# Display the plot
plt.show()
```



The model nicely captures a near-perfect linear correlation between radio advertising expenditure and sales! Now let's take a look at what is going on under the hood to calculate this relationship.

The basics of linear regression

Fit and predict for regression

Now you have seen how linear regression works, your task is to create a multiple linear regression model using all of the features in the `sales_df` dataset, which has been preloaded for you. As a reminder, here are the first two rows:

```
     tv        radio      social_media    sales
1    13000.0   9237.76    2409.57         46677.90
2    41000.0   15886.45   2913.41         150177.83
```

You will then use this model to predict sales based on the values of the test features.

`LinearRegression` and `train_test_split` have been preloaded for you from their respective modules.

- Create `X`, an array containing values of all features in `sales_df`, and `y`, containing all values from the `"sales"` column.
- Instantiate a linear regression model.
- Fit the model to the training data.
- Create `y_pred`, making predictions for `sales` using the test features.

```
# Create X and y arrays
X = sales_df.drop("sales", axis=1).values
y = sales_df["sales"].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Instantiate the model
reg = LinearRegression()

# Fit the model to the data
reg.fit(X_train, y_train)

# Make predictions
```

```
## LinearRegression()
```

```
y_pred = reg.predict(X_test)
print("Predictions: {}, Actual Values: {}".format(y_pred[:2], y_test[:2]))
```

```
## Predictions: [53176.66154234 70996.19873235], Actual Values: [55261.28 67574.9 ]
```

Great work! The first two predictions appear to be within around 5% of the actual values from the test set!

Regression performance

Now you have fit a model, `reg`, using all features from `sales_df`, and made predictions of sales values, you can evaluate performance using some common regression metrics.

The variables `X_train`, `X_test`, `y_train`, `y_test`, and `y_pred`, along with the fitted model, `reg`, all from the last exercise, have been preloaded for you.

Your task is to find out how well the features can explain the variance in the target values, along with assessing the model's ability to make predictions on unseen data.

- Import `mean_squared_error`.
- Calculate the model's R-squared score by passing the test feature values and the test target values to an appropriate method.
- Calculate the model's root mean squared error using `y_test` and `y_pred`.
- Print `r_squared` and `rmse`.

```
# Import mean_squared_error
from sklearn.metrics import mean_squared_error

# Compute R-squared
r_squared = reg.score(X_test, y_test)

# Compute RMSE
rmse = mean_squared_error(y_test, y_pred, squared=False)

# Print the metrics
print("R^2: {}".format(r_squared))
```

```
## R^2: 0.9990152104759368
```

```
print("RMSE: {}".format(rmse))
```

```
## RMSE: 2944.4331996001
```

Wow, the features explain 99.9% of the variance in sales values! Looks like this company's advertising strategy is working well!

Cross-validation

Cross-validation for R-squared

Cross-validation is a vital approach to evaluating a model. It maximizes the amount of data that is available to the model, as the model is not only trained but also tested on all of the available data.

In this exercise, you will build a linear regression model, then use 6-fold cross-validation to assess its accuracy for predicting sales using social media advertising expenditure. You will display the individual score for each of the six-folds.

The `sales_df` dataset has been split into `y` for the target variable, and `X` for the features, and preloaded for you. `LinearRegression` has been imported from `sklearn.linear_model`.

- Import `KFold` and `cross_val_score`.
- Create `kf` by calling `KFold()`, setting the number of splits to six, `shuffle` to `True`, and setting a seed of `5`.
- Perform cross-validation using `reg` on `X` and `y`, passing `kf` to `cv`.
- Print the `cv_scores`.

```
# Import the necessary modules
from sklearn.model_selection import KFold, cross_val_score

# Create a KFold object
kf = KFold(n_splits=6, shuffle=True, random_state=5)

reg = LinearRegression()

# Compute 6-fold cross-validation scores
cv_scores = cross_val_score(reg, X, y, cv=kf)

# Print scores
print(cv_scores)
```

```
## [0.99894062 0.99909245 0.9990103  0.99896344 0.99889153 0.99903953]
```

Notice how R-squared for each fold ranged between `0.74` and `0.77`? By using cross-validation, we can see how performance varies depending on how the data is split!

Analyzing cross-validation metrics

Now you have performed cross-validation, it's time to analyze the results.

You will display the mean, standard deviation, and 95% confidence interval for `cv_results`, which has been preloaded for you from the previous exercise.

`numpy` has been imported for you as `np`.

- Calculate and print the mean of the results.
- Calculate and print the standard deviation of `cv_results`.
- Display the 95% confidence interval for your results using `np.quantile()`.

```
# edited/added
cv_results = cv_scores

# Print the mean
print(np.mean(cv_results))

# Print the standard deviation
```

```
## 0.9989896443678249
```

```
print(np.std(cv_results))

# Print the 95% confidence interval
```

```
## 6.608118371529651e-05
```

```
print(np.quantile(cv_results, [0.025, 0.975]))
```

```
## [0.99889767 0.99908583]
```

An average score of `0.75` with a low standard deviation is pretty good for a model out of the box! Now let's learn how to apply regularization to our regression models.

Regularized regression

Regularized regression: Ridge

Ridge regression performs regularization by computing the *squared* values of the model parameters multiplied by alpha and adding them to the loss function.

In this exercise, you will fit ridge regression models over a range of different alpha values, and print their $\\(R^2\\)$ scores. You will use all of the features in the `sales_df` dataset to predict `"sales"`. The data has been split into `X_train`, `X_test`, `y_train`, `y_test` for you.

A variable called `alphas` has been provided as a list containing different alpha values, which you will loop through to generate scores.

- Import `Ridge`.
- Instantiate `Ridge`, setting alpha equal to `alpha`.
- Fit the model to the training data.
- Calculate the (*R*^{2}) score for each iteration of `ridge`.

```
# Import Ridge
from sklearn.linear_model import Ridge
alphas = [0.1, 1.0, 10.0, 100.0, 1000.0, 10000.0]
ridge_scores = []

for alpha in alphas:

  # Create a Ridge regression model
  ridge = Ridge(alpha=alpha)

  # Fit the data
  ridge.fit(X_train, y_train)

  # Obtain R-squared
  score = ridge.score(X_test, y_test)
  ridge_scores.append(score)
```

```
## Ridge(alpha=0.1)
## Ridge()
## Ridge(alpha=10.0)
## Ridge(alpha=100.0)
## Ridge(alpha=1000.0)
## Ridge(alpha=10000.0)
```

```
print(ridge_scores)
```

```
## [0.9990152104759369, 0.9990152104759373, 0.9990152104759419, 0.999015210475987,
## 0.9990152104764387, 0.9990152104809561]
```

Well done! The scores don't appear to change much as `alpha` increases, which is indicative of how well the features explain the variance in the target—even by heavily penalizing large coefficients, underfitting does not occur!

Lasso regression for feature importance

In the video, you saw how lasso regression can be used to identify important features in a dataset.

In this exercise, you will fit a lasso regression model to the `sales_df` data and plot the model's coefficients.

The feature and target variable arrays have been pre-loaded as `X` and `y`, along with `sales_columns`, which contains the dataset's feature names.

- Import `Lasso` from `sklearn.linear_model`.
- Instantiate a Lasso regressor with an alpha of `0.3`.
- Fit the model to the data.
- Compute the model's coefficients, storing as `lasso_coef`.

```
# edited/added
sales_columns = sales_df.drop(['sales'], axis = 1).columns

# Import Lasso
from sklearn.linear_model import Lasso

# Instantiate a lasso regression model
lasso = Lasso(alpha=0.3)

# Fit the model to the data
lasso.fit(X, y)

# Compute and print the coefficients
```

```
## Lasso(alpha=0.3)
```

```
lasso_coef = lasso.coef_
print(lasso_coef)
```

```
## [ 3.56256962 -0.00397035  0.00496385]
```

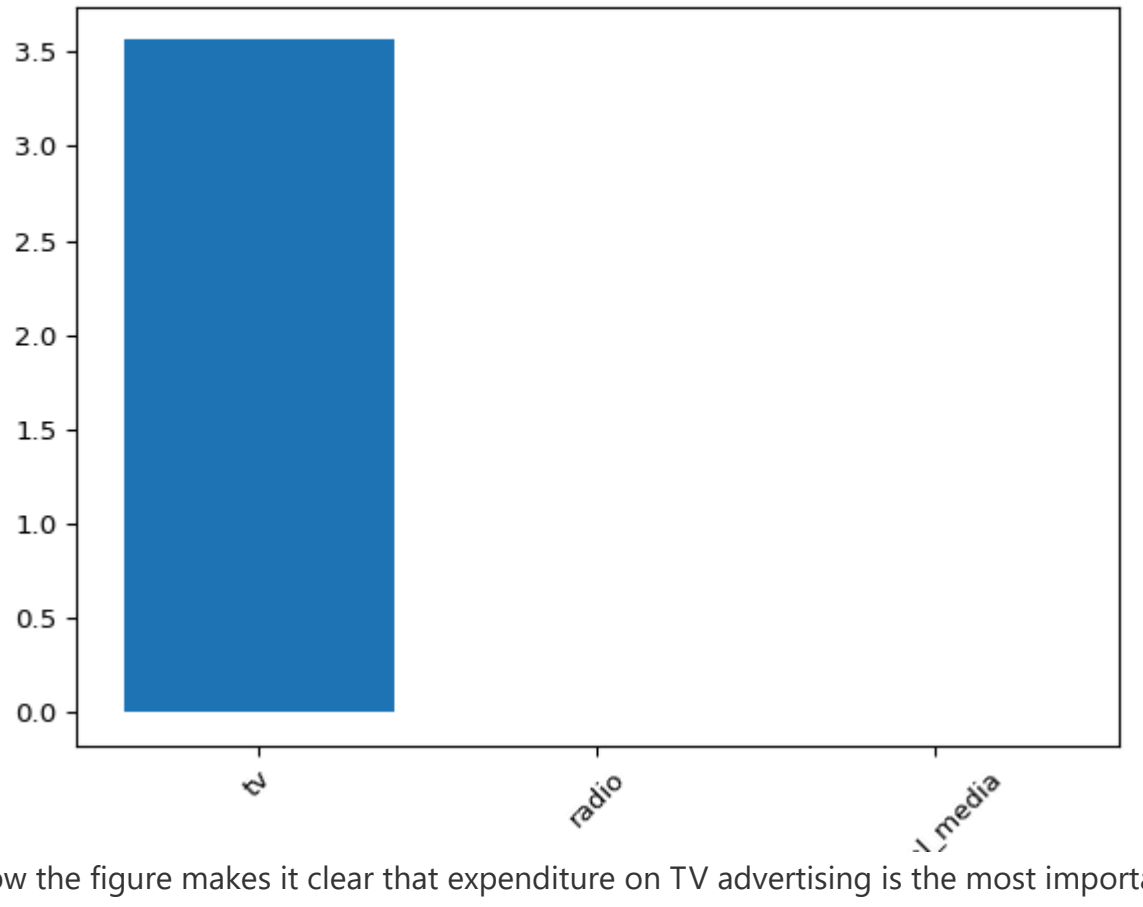
```
plt.bar(sales_columns, lasso_coef)
```

```
## <BarContainer object of 3 artists>
```

```
plt.xticks(rotation=45)
```

```
## ([0, 1, 2], [Text(0, 0, ''), Text(0, 0, ''), Text(0, 0, '')])
```

```
plt.show()
```



See how the figure makes it clear that expenditure on TV advertising is the most important feature in the dataset to predict sales values! In the next chapter, we will learn how to further assess and improve our model's performance!
