Lab 5. Linear Regression. Cross Validation.

5.1 Linear Regression

Remember: Types of supervised learning

- Classification: Predict a categorical response
- Regression: Predict a continuous response

Reading data using pandas

Pandas: popular Python library for data exploration, manipulation, and analysis

- Anaconda users: pandas is already installed
- Other users: need installation

```
# conventional way to import pandas
import pandas as pd
```

```
# read CSV file from the 'data' subdirectory using a relative path
data = pd.read_csv('data/Advertising.csv', index_col=0)

# display the first 5 rows
data.head()
```

Primary object types:

- DataFrame: rows and columns (like a spreadsheet)
- Series: a single column

```
# display the last 5 rows
data.tail()
```

```
# check the shape of the DataFrame (rows, columns)
data.shape
```

New dataset:

- “Advertising dataset”

What are the features?

- TV: advertising dollars spent on TV for a single product in a given market (in thousands of dollars)
- Radio: advertising dollars spent on Radio
- Newspaper: advertising dollars spent on Newspaper

What is the response?

- Sales: sales of a single product in a given market (in thousands of items)

What else do we know?
Because the response variable is continuous, this is a regression problem.
There are 200 observations (represented by the rows), and each observation is a single market.

Visualizing data using seaborn

Seaborn: Python library for statistical data visualization built on top of Matplotlib

- Anaconda users: run conda install seaborn from the command line
- Other users: need installation

```
# conventional way to import seaborn
import seaborn as sns

# allow plots to appear within the notebook
%matplotlib inline
```

```
# visualize the relationship between the features and the response using scatterplots
sns.pairplot(data, x_vars=['TV','Radio','Newspaper'], y_vars='Sales', height=7, aspect=0.7, kind='reg')
```

Linear regression

Linear regression is a type of supervised machine learning algorithm that computes the linear relationship between a dependent variable and one or more independent features. When the number of the independent feature, is 1 then it is known as Univariate Linear regression, and in the case of more than one feature, it is known as multivariate linear regression. The goal of the algorithm is to find the best linear equation that can predict the value of the dependent variable based on the independent variables. The equation provides a straight line that represents the relationship between the dependent and independent variables. The slope of the line indicates how much the dependent variable changes for a unit change in the independent variable(s).

Linear regression is used in many different fields, including finance, economics, and psychology, to understand and predict the behavior of a particular variable. For example, in finance, linear regression might be used to understand the relationship between a company's stock price and its earnings or to predict the future value of a currency based on its past performance.

Pros: fast, no tuning required, highly interpretable, well-understood
Cons: unlikely to produce the best predictive accuracy (presumes a linear relationship between the features and response)

Linear regression expression:

$$y=\beta_0+\beta_1x_1+\beta_2x_2+...+\beta_nx_n$$

- y is the response
- β_0 is the intercept
- β_1 is the coefficient for x_1 (the first feature)
- β_n is the coefficient for x_n (the nth feature)

In this case for the considered dataset:

$$y=\beta_0 + \beta_1\times TV + \beta_2\times Radio + \beta_3\times Newspaper$$

The β values are called the model coefficients. These values are "learned" during the model fitting step using the "least squares" criterion. Then, the fitted model can be used to make predictions!

Preparing X and y using pandas

- scikit-learn expects X (feature matrix) and y (response vector) to be NumPy arrays.
- However, pandas is built on top of NumPy.
- Thus, X can be a pandas DataFrame and y can be a pandas Series!

```python
# create a Python list of feature names
feature_cols = ['TV', 'Radio', 'Newspaper']

# use the list to select a subset of the original DataFrame
X = data[feature_cols]

# equivalent command to do this in one line
X = data[['TV', 'Radio', 'Newspaper']]

# print the first 5 rows
X.head()
```

```python
# check the type and shape of X
print(type(X))
print(X.shape)
```

```python
# select a Series from the DataFrame
y = data['Sales']

# equivalent command that works if there are no spaces in the column name
y = data.Sales

# print the first 5 values
y.head()
```

```python
# check the type and shape of y
print(type(y))
print(y.shape)
```

Splitting X and y into training and testing sets

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
```

```python
# default split is 75% for training and 25% for testing
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

Linear regression in scikit-learn

```python
# import model
from sklearn.linear_model import LinearRegression

# instantiate
linreg = LinearRegression()

# fit the model to the training data (learn the coefficients)
linreg.fit(X_train, y_train)
```

Interpreting model coefficients

```python
# print the intercept and coefficients
print(linreg.intercept_)
print(linreg.coef_)
```

```python
# pair the feature names with the coefficients
list(zip(feature_cols, linreg.coef_))
```

$$y = 2.88 + 0.0466 \times \ + 0.179 \times Radio + 0.00345 \times Newspaper$$

How do we interpret the TV coefficient (0.0466)?

For a given amount of Radio and Newspaper ad spending, a "unit" increase in TV ad spending is associated with a 0.0466 "unit" increase in Sales.
Or more clearly: For a given amount of Radio and Newspaper ad spending, an additional $1,000 spent on TV ads is associated with an increase in sales of 46.6 items.

<u>Important notes:</u>

- This is a statement of association, not causation.
- If an increase in TV ad spending was associated with a decrease in sales, $_1$ would be negative.

Making predictions

```
# make predictions on the testing set
y_pred = linreg.predict(X_test)
```

We need an **evaluation metric** in order to compare our predictions with the actual values!

Model evaluation metrics for regression

Evaluation metrics for classification problems, such as accuracy, are not useful for regression problems. Instead, we need evaluation metrics designed for comparing continuous values. Let's create some example numeric predictions, and calculate three common evaluation metrics for regression problems:

```
# define true and predicted response values
true = [100, 50, 30, 20]
pred = [90, 50, 50, 30]
```

Mean Absolute Error (MAE) is the mean of the absolute value of the errors:

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

```
# calculate MAE by hand
print((10 + 0 + 20 + 10)/4.)

# calculate MAE using scikit-learn
from sklearn import metrics
print(metrics.mean_absolute_error(true, pred))
```

Mean Squared Error (MSE) is the mean of the squared errors:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

```
# calculate MSE by hand
print((10**2 + 0**2 + 20**2 + 10**2)/4.)

# calculate MSE using scikit-learn
print(metrics.mean_squared_error(true, pred))
```

Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

```
# calculate RMSE by hand
import numpy as np
print(np.sqrt((10**2 + 0**2 + 20**2 + 10**2)/4.))

# calculate RMSE using scikit-learn
print(np.sqrt(metrics.mean_squared_error(true, pred)))
```

Comparing these metrics:

- **MAE** is the easiest to understand, because it's the average error.
- **MSE** is more popular than MAE, because MSE "punishes" larger errors.
- **RMSE** is even more popular than MSE, because RMSE is interpretable in the "y" units.

Computing the RMSE for our Sales predictions

```
print(np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

Feature selection

Does Newspaper "belong" in our model? In other words, does it improve the quality of our predictions?

Let's remove it from the model and check the RMSE!

```
# create a Python list of feature names
feature_cols = ['TV', 'Radio']

# use the list to select a subset of the original DataFrame
X = data[feature_cols]

# select a Series from the DataFrame
y = data.Sales

# split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)

# fit the model to the training data (learn the coefficients)
linreg.fit(X_train, y_train)

# make predictions on the testing set
y_pred = linreg.predict(X_test)

# compute the RMSE of our predictions
print(np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

The RMSE decreased when **we removed Newspaper from the model**. (Error is something we want to minimize, so a lower number for RMSE is better.) Thus, it is unlikely that this feature is useful for predicting Sales, and should be removed from the model.

5.2. Cross-Validation

Motivation: Need a way to choose between Machine Learning models

- Goal is to estimate likely performance of a model on out-of-sample data

Initial idea: Train and test on the same data

- But, maximizing training accuracy rewards overly complex models which overfit the training data.
-

Alternative idea: Train/test split

- Split the dataset into two pieces, so that the model can be trained and tested on different data
- Testing accuracy is a better estimate than training accuracy of out-of-sample performance
- But, it provides a high variance estimate since changing which observations happen to be in the testing set can significantly change testing accuracy.

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
```

```python
# read in the iris data
iris = load_iris()

# create X (features) and y (response)
X = iris.data
y = iris.target
```

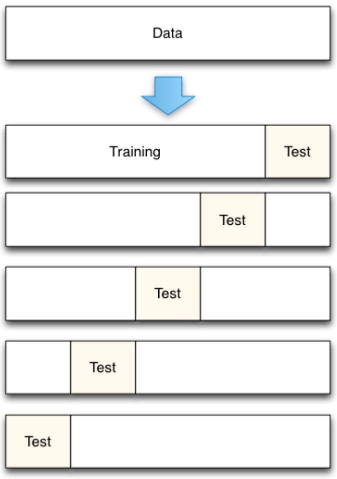
```python
# use train/test split with different random_state values
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=4)

# check classification accuracy of KNN with K=5
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print(metrics.accuracy_score(y_test, y_pred))
```

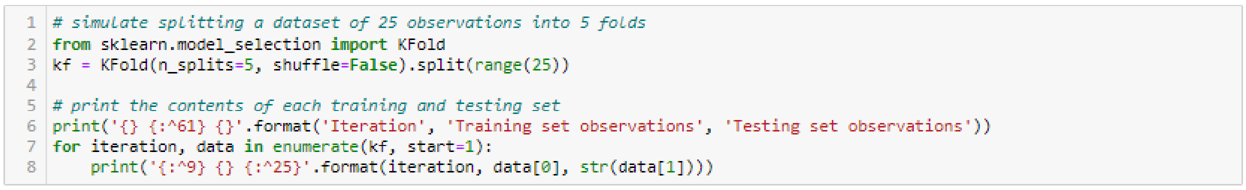
Question: What if we created a bunch of train/test splits, calculated the testing accuracy for each, and averaged the results together?

Answer: That's the essence of cross-validation!

Steps for K-fold cross-validation

- Split the dataset into K equal partitions (or "folds").
- Use fold 1 as the testing set and the union of the other folds as the training set.
- Calculate testing accuracy.
- Repeat steps 2 and 3, K times, using a different fold as the testing set each time.
- Use the average testing accuracy as the estimate of out-of-sample accuracy.





```python
# simulate splitting a dataset of 25 observations into 5 folds
from sklearn.model_selection import KFold
kf = KFold(n_splits=5, shuffle=False).split(range(25))

# print the contents of each training and testing set
print('{} {:^61} {}'.format('Iteration', 'Training set observations', 'Testing set observations'))
for iteration, data in enumerate(kf, start=1):
    print('{:^9} {} {:^25}'.format(iteration, data[0], str(data[1])))
```

- Dataset contains 25 observations (numbered 0 through 24)
- 5-fold cross-validation, thus it runs for 5 iterations
- For each iteration, every observation is either in the training set or the testing set, but not both
- Every observation is in the testing set exactly once

Comparing cross-validation to train/test split

Advantages of cross-validation:

- More accurate estimate of out-of-sample accuracy
- More "efficient" use of data (every observation is used for both training and testing)

Advantages of train/test split:

- Runs K times faster than K-fold cross-validation
- Simpler to examine the detailed results of the testing process

Cross-validation recommendations

1. K can be any number, but K=10 is generally recommended
2. For classification problems, stratified sampling is recommended for creating the folds
    - Each response class should be represented with equal proportions in each of the K folds
    - scikit-learn's cross_val_score function does this by default

Cross-validation example: parameter tuning

Goal: Select the best tuning parameters (aka "hyperparameters") for KNN on the iris dataset

```
from sklearn.model_selection import cross_val_score
```

```
# 10-fold cross-validation with K=5 for KNN (the n_neighbors parameter)
knn = KNeighborsClassifier(n_neighbors=5)
scores = cross_val_score(knn, X, y, cv=10, scoring='accuracy')
print(scores)
```

```
# use average accuracy as an estimate of out-of-sample accuracy
print(scores.mean())
```

```
# search for an optimal value of K for KNN
k_range = list(range(1, 31))
k_scores = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X, y, cv=10, scoring='accuracy')
    k_scores.append(scores.mean())
print(k_scores)
```

```
import matplotlib.pyplot as plt
%matplotlib inline

# plot the value of K for KNN (x-axis) versus the cross-validated accuracy (y-axis)
plt.plot(k_range, k_scores)
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross-Validated Accuracy')
```

Cross-validation example: model selection

Goal: Compare the best KNN model with logistic regression on the iris dataset

```
# 10-fold cross-validation with the best KNN model
knn = KNeighborsClassifier(n_neighbors=20)
print(cross_val_score(knn, X, y, cv=10, scoring='accuracy').mean())
```

```
# 10-fold cross-validation with logistic regression
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(solver='liblinear')
print(cross_val_score(logreg, X, y, cv=10, scoring='accuracy').mean())
```

Cross-validation example: feature selection

Goal: Select whether the Newspaper feature should be included in the linear regression model on the advertising dataset

```python
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
```

```python
# read in the advertising dataset
data = pd.read_csv('data/Advertising.csv', index_col=0)
```

```python
# create a Python list of three feature names
feature_cols = ['TV', 'Radio', 'Newspaper']

# use the list to select a subset of the DataFrame (X)
X = data[feature_cols]

# select the Sales column as the response (y)
y = data.Sales
```

```python
# 10-fold cross-validation with all three features
lm = LinearRegression()
scores = cross_val_score(lm, X, y, cv=10, scoring='neg_mean_squared_error')
print(scores)
```

```python
# fix the sign of MSE scores
mse_scores = -scores
print(mse_scores)
```

```python
# convert from MSE to RMSE
rmse_scores = np.sqrt(mse_scores)
print(rmse_scores)
```

```python
# calculate the average RMSE
print(rmse_scores.mean())
```

```python
# 10-fold cross-validation with two features (excluding Newspaper)
feature_cols = ['TV', 'Radio']
X = data[feature_cols]
print(np.sqrt(-cross_val_score(lm, X, y, cv=10, scoring='neg_mean_squared_error')).mean())
```

Improvements to cross-validation

Repeated cross-validation

- Repeat cross-validation multiple times (with different random splits of the data) and average the results
- More reliable estimate of out-of-sample performance by reducing the variance associated with a single trial of cross-validation

Creating a hold-out set

- "Hold out" a portion of the data before beginning the model building process
- Locate the best model using cross-validation on the remaining data, and test it using the hold-out set
- More reliable estimate of out-of-sample performance since hold-out set is truly out-of-sample

Exercises

1. For the "Iris" dataset, implement a KNN classification algorithm and find the optimal parameter K (K takes values in the range (1,15)) using 5-fold cross-validation. Compare the accuracy of KNN using this optimal K, with a logistic regression algorithm (not linear regression!).

2. For the "Advertising" dataset, check the accuracy of linear regression using 10-fold cross-validation, using only one feature at a time ("Newspaper", "TV", or "Radio"). Which of these features generates the lowest testing error for the linear regression algorithm?

3. Repeat the procedure for linear regression using two features "Newspaper-TV", "Newspaper-Radio", and "TV-Radio". Which of these feature combinations generates the lowest testing error for the linear regression algorithm?