Linear Regression with scikit-learn¶

From the video series: Introduction to machine learning with scikit-learn

Agenda¶

- What is **linear regression**, and how does it work?
- How do I **train and interpret** a linear regression model in scikit-learn?
- What are some **evaluation metrics** for regression problems?
- How do I choose **which features to include** in my model?

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: installation instructions

In [ ]:

```
# conventional way to import pandas

import pandas as pd
```

In [ ]:

```
# read CSV file directly from a URL and save the results

data = pd.read_csv('http://www-bcf.usc.edu/~gareth/ISL/Advertising.csv', index_col=
0)



# display the first 5 rows

data.head()
```

In [ ]:

```
data.shape
```

Primary object types:

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

In [ ]:

```
# display the last 5 rows

data.tail()
```

In [ ]:

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

data.shape
```

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.

In [ ]:

```
# conventional way to import seaborn

import seaborn as sns



# allow plots to appear within the notebook

%matplotlib inline
```

In [ ]:

```
# visualize the relationship between the features and the response using scatterplo
ts

sns.pairplot(data, x_vars=['TV','Radio','Newspaper'], y_vars='Sales', size=7, aspec
t=0.7, kind='reg');
```

Linear regression¶

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)

Form of linear regression¶

- is the response
- is the intercept
- is the coefficient for (the first feature)
- is the coefficient for (the nth feature)

In this case:

The 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!

In [ ]:

```
# 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]



# print the first 5 rows

X.head()
```

In [ ]:

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

In [ ]:

```
# 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()
```

In [ ]:

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

Splitting X and y into training and testing sets¶

In [ ]:

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

In [ ]:

```
# 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¶

In [ ]:

```
# 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¶

In [ ]:

```
# print the intercept and coefficients
print linreg.intercept_
print linreg.coef_
```

In [ ]:

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

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**.

Important notes:

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

Making predictions

In [ ]:

```
X_test.loc[59]
```

In [ ]:

```
# make predictions on the testing set

y_pred = linreg.predict(X_test)
```

In [ ]:

```
y_pred
```

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:

In [ ]:

```
# 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:

In [ ]:

```
# 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:

In [ ]:

```
# 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:

In [ ]:

```
# 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

In [ ]:

```
metrics.mean_squared_error?
```

In [ ]:

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

KNN for regression

In [ ]:

```
# [u'TV', u'Radio', u'Newspaper', u'Sales']


X = data[['TV', 'Radio']]

y = data.Sales
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=5)


## KNN
from sklearn.neighbors import KNeighborsRegressor
knr = KNeighborsRegressor(n_neighbors=6)
knr.fit(X_train, y_train)
y_pred_knr = knr.predict(X_test)
print "RMSE KNN Regressor: ", np.sqrt(metrics.mean_squared_error(y_test, y_pred_knr
))


## Linear Regression
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
y_pred_lr = lin_reg.predict(X_test)
print "RMSE Linear Regression: ", np.sqrt(metrics.mean_squared_error(y_test, y_pred
_lr))
```

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!

In [ ]:

```
# 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)
```

```python
# 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.