model

In this exercise, we'll train both a simple linear-regression model and a multiple linear-regression model and compare their performance using R-Squared.

Loading data

Let's start by having a look at our data.


```
import pandas
!pip install statsmodels
!wget https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to-machine-learning,
!wget https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to-machine-learning,

#Import the data from the .csv file
dataset = pandas.read_csv('doggy-illness.csv', delimiter="\t")

#Let's have a look at the data
dataset
```

For this exercise, we'll try to predict `core_temperature` from some of the other available features.

Data visualization

Let's quickly eyeball which features seem to have some kind of relationship with `core_temperature` .

```
import graphing # Custom graphing code that uses Plotly. See our GitHub repository for details

graphing.box_and_whisker(dataset, "male", "core_temperature", show=True)
graphing.box_and_whisker(dataset, "attended_training", "core_temperature", show=True)
graphing.box_and_whisker(dataset, "ate_at_tonys_steakhouse", "core_temperature", show=True)
```

```
graphing.scatter_2D(dataset, "body_fat_percentage", "core_temperature", show=True)
graphing.scatter_2D(dataset, "protein_content_of_last_meal", "core_temperature", show=True)
graphing.scatter_2D(dataset, "age", "core_temperature")
```


At a glance, fatter, older, and male dogs seem to more commonly have higher temperatures than thinner, younger, or female dogs. Dogs who ate a lot of protein last night also seem to be more unwell. The other features don't seem particularly useful.

Simple linear regression

Let's try to predict `core_temperature` using simple linear regression, and note the R-Squared for these relationships.

```
import statsmodels.formula.api as smf
import graphing # custom graphing code. See our GitHub repo for details

for feature in ["male", "age", "protein_content_of_last_meal", "body_fat_percentage"]:
    # Perform linear regression. This method takes care of
    # the entire fitting procedure for us.
    formula = "core_temperature ~ " + feature
    simple_model = smf.ols(formula = formula, data = dataset).fit()

    print(feature)
    print("R-squared:", simple_model.rsquared)

    # Show a graph of the result
    graphing.scatter_2D(dataset, label_x=feature,
                                 label_y="core_temperature",
                                 title = feature,
                                 trendline=lambda x: simple_model.params[1] * x + simple_model
                                 show=True)
```

Scrolling through these graphs, we get R-square values of 0.0002 (`body_fat_percentage`), 0.1 (`male`), and 0.26 (`age`).

While `protein_content_of_last_meal` looks very promising too, the relationship looks curved, not linear. We'll leave this feature for now and come back to it in the next exercise.

R-Squared

We've shown the R-Squared value for these models and used it as a measure of "correctness" for our regression, but what is it?

Intuitively, we can think of R-Squared as ratio for how much better our regression line is than a naive regression that just goes straight through the mean of all examples.

Roughly, the R-Squared is calculated by taking the loss/error of the trained model, and dividing by the loss/error of the naive model. That gives a range where `0` is better and `1` is worse, so the whole thing is subtracted from `1` to flip those results.

In the following code, we once again show the scatter plot with `age` and `core_temperature` , but this time, we show two regression lines. The first is the naive line that just goes straight through the mean. This has an R-Squared of `0` (since it's no better than itself). An R-Squared of `1` would be a line that fit each training example perfectly. The second plot shows our trained regression line, and we once again see its R-Squared.

```
formula = "core_temperature ~ age"
age_trained_model = smf.ols(formula = formula, data = dataset).fit()
age_naive_model = smf.ols(formula = formula, data = dataset).fit()
age_naive_model.params[0] = dataset['core_temperature'].mean()
age_naive_model.params[1] = 0

print("naive R-squared:", age_naive_model.rsquared)
print("trained R-squared:", age_trained_model.rsquared)

# Show a graph of the result
graphing.scatter_2D(dataset, label_x="age",
                                label_y="core_temperature"
```

```
                                label_y= core_temperature ,
                                title = "Naive model",
                                trendline=lambda x: dataset['core_temperature'].repeat(len(x))
                                show=True)
# Show a graph of the result
graphing.scatter_2D(dataset, label_x="age",
                                label_y="core_temperature",
                                title = "Trained model",
                                trendline=lambda x: age_tra
```


Multiple Linear Regression

Instead of modeling these separately, lets try to combine these into a single model. Body fat didn't seem to be useful after all, so let's just use `male` and `age` as features.

```
model = smf.ols(formula = "core_temperature ~ age + male", data = dataset).fit()

print("R-squared:", model.rsquared)
```

By using both features at the same time, we got a better result than any of the one-feature (univariate) models.

How can we view this, though? Well, a simple linear regression is drawn in 2D. If we're working with an extra variable, we add one dimension and work in 3D.

```
import numpy as np
# Show a graph of the result
# this needs to be 3D, because we now have three variables in play: two features and one label
```

```
def predict(age, male):
    '''
    This converts given age and male values into a prediction from the model
    '''
    # to make a prediction with statsmodels, we need to provide a dataframe
    # so create a dataframe with just the age and male variables
    df = pandas.DataFrame(dict(age=[age], male=[male]))
    return model.predict(df)
```

```
# Create the surface graph
fig = graphing.surface(
    x_values=np.array([min(dataset.age), max(dataset.age)]),
    y_values=np.array([0, 1]),
    calc_z=predict,
    axis_title_x="Age",
    axis_title_y="Male",
    axis_title_z="Core temperature"
)

# Add our datapoints to it and display
fig.add_scatter3d(x=dataset.age, y=dataset.male, z=dataset.core_temperature, mode='markers')
fig.show()
```

The preceding graph above interactive. Try rotating it to see how the model (shown as a solid plane) would predict core temperature from different combinations of age and sex.

Inspecting our model

When we have more than two features, it becomes very difficult to visualize these models. We usually have to look at the parameters directly. Let's do that now. *Statsmodels*, one of the common machine learning and statistics libraries, provides a `summary()` method that provides information about our model.

```
# Print summary information
```

```
model.summary()
```

If we look at the top right-hand corner, we can see our R-squared statistic that we printed out earlier.

Slightly down and to the left, we can also see information about the data we trained our model on. For example, we can see that we trained it on 98 observations (`No. Observations`).

Under this, we find information about our parameters in a column called `coef` (which stands for *coefficients*, a synonym for parameters in machine learning). Here, we can see the intercept was about `38` , meaning that the model predicts a core temperature of 38 for a dog with `age=0` and `male=0` . Underneath this, we see the parameter for age is 0.14, meaning that for each additional year of age, the predicted temperature would rise 0.14 degrees celsius. For `male` , we can see a parameter of 0.32, meaning that the model estimates all dogs (that is, where `male == 1`) to have temperatures 0.32 degrees celsius higher than female dogs (where `male == 0`).

Although we don't have space here to go into detail, the `P` column is also very useful. This tells us how confident the model is about this parameter value. As a rule of thumb, if the *p-value* is less than 0.05, there is a good chance that this relationship if trustable. For example, here both `age` and `male` are less than 0.05, so we should feel confident using this model in the real world.

As a final exercise, let's do the same thing with our earlier simple linear-regression model relating `age` to `core_temperature`

```
age_trained_model.summary()
```

Summary

We covered the following concepts in this exercise:

- Built simple and multiple linear-regression models.
- Compared the performance of both models by looking at R-Squared values.
- Inspected models to understand how they work

inspected models to understand how they work.

No compute Compute not connected Viewing Kernel not connected

Next unit: Polynomial Regression

Alt + A

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆


Alt + A