Training Analysis

In the preceding output, we're printing an estimate of weights and the calculated cost at each iteration.

The final line in the output shows that the model stopped training because it reached its maximum allowed number of iterations, but the cost could still be lower if we had let it run longer.

Let's plot the model at the end of this training:

```
# Plot the data and trendline after training
graphing.scatter_2D(data, "month_old_when_trained", "mean_rescues_per_year", trendline=model.p
```

The preceding plot tells us that the younger a dog begins training, the more rescues it be perform in a year.

Notice that it doesn't fit the data very well (most points are above the line). That's due to training being cut off early, before the model could find the optimal weights.

Standardizing data

Let's use *standardization* as the form of *feature scaling* for this model, applying it to the `month_old_when_trained` feature:

```
# Add the standardized verions of "age_when_trained" to the dataset.
# Notice that it "centers" the mean age around 0
data["standardized_age_when_trained"] = (data.month_old_when_trained - numpy.mean(data.month_o

# Print a sample of the new dataset
data[:5]
```

Notice the the values `standardized_age_when_trained` column above are distributed in a much smaller range (between -2 and 2) and have their mean centered around `0`.

Visualizing Scaled Features

Let's use a box plot to compare the original feature values to their standardized versions:

```
from plotly.subplots import make_subplots
import plotly.graph_objects as go
import plotly.express as px

fig = px.box(data,y=["month_old_when_trained", "standardized_age_when_trained"])
fig.show()
```

Now, compare the two features by hovering your mouse over the graph. You'll note that:

- `month_old_when_trained` ranges from 1 to 71 and has its median centered around 35.
- `standardized_age_when_trained` ranges from -1.6381 to 1.6798, and is centered exactly at 0.

Training with standardized features

We can now retrain our model using the standardized feature in our dataset:

```
# Let's retrain our model, this time using the standardized feature
model_norm = gradient_descent(data.standardized_age_when_trained, data.mean_rescues_per_year,
```

Let's take a look at that output again.

Despite still being allowed a maximum of 8000 iterations, the model stopped at the 5700 mark.

Why? Because this time, using the standardized feature, it was quickly able to reach a point where the cost could no longer be improved.

In other words, it "converged" much faster than the previous version.

Plotting the standardized model

We can now plot the new model and see the results of standardization:

```
# Plot the data and trendline again, after training with standardized feature
graphing.scatter_2D(data, "standardized_age_when_trained", "mean_rescues_per_year", trendline=
```

It looks like this model fits the data much better that the first one!

The standardized model shows a larger slope and data now centered on `0` on the X-axis, both factors which should allow the model to converge faster.

But how much faster?

Let's plot a comparison between models to visualize the improvements.

```
cost1 = model.cost_history
cost2 = model_norm.cost_history

# Creates dataframes with the cost history for each model
df1 = pandas.DataFrame({"cost": cost1, "Model":"No feature scaling"})
```

```
    df1["number of iterations"] = df1.index + 1
    df2 = pandas.DataFrame({"cost": cost2, "Model":"With feature scaling"})
    df2["number of iterations"] = df2.index + 1
```

```
    # Concatenate dataframes into a single one that we can use in our plot
    df = pandas.concat([df1, df2])

    # Plot cost history for both models
    fig = graphing.scatter_2D(df, label_x="number of iterations", label_y="cost", title="Training
    fig.update_traces(mode='lines')
    fig.show()
```

This plot clearly shows that using a standardized dataset allowed our model to converge much faster. Reaching the lowest cost and finding the optimal weights required a much smaller number of iterations.

This is very important when you are developing a new model, because it allows you to iterate quicker; but also when your model is deployed to a production environment, because it requires less compute time for training and costs less than a "slow" model.

Summary

In this exercise, we covered the following concepts:

- *Feature scaling* techniques are used to improve the efficiency of training models
- How to add a standardized feature to a dataset
- How to visualize standardized features and compare them to their original values

Finally, we compared the performance of models before and after using standardized features, using plots to visualize the improvements.

No compute Compute not connected Viewing Kernel not connected

Next unit: Test and training datasets

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆