Machine learning on machines: building a model to evaluate CPU performance

Sara Robinson Follow
Nov 14, 2018 · 8 min read

Wouldn't it be cool if we could train a machine learning model to predict machine performance? In this post, we'll look at a linear regression model I built using BQML to predict the performance of a machine given hardware and software specifications.

All of the work I do is software related so I rarely think about the hardware running my code — this gave me an opportunity to learn about the hardware side of things.

The dataset: SPEC

To train this model I used data from SPEC*, an organization that builds tools for evaluating computer performance and energy efficiency. They published a series of benchmarking results from 2017, where they used 43 different tests to evaluate the performance of specific hardware. Their benchmarks are divided into 4 categories: integer and floating point tests measuring performance through both *time* (called SPEC speed) and *throughput* (called SPEC rate):

Integer - **Speed**	Integer - **Throughput**
Floating Point - **Speed**	Floating Point - **Throughput**

The four categories of SPEC 2017 benchmarks.

For this post, I'll focus mostly on the Integer Speed tests.

Each test includes specs on the hardware and software of the system where the test was run, along with results for several benchmarks. The benchmarks are intended to simulate different types of applications or workloads.

Here's an example of the hardware and software data for one test:

```
                                    HARDWARE
                                    --------
            CPU Name: Intel Xeon Gold 6150
            Max MHz.: 3700
             Nominal: 2700
             Enabled: 36 cores, 2 chips
           Orderable: 1, 2 chip(s)
            Cache L1: 32 KB I + 32 KB D on chip per core
                  L2: 1 MB I+D on chip per core
                  L3: 24.75 MB I+D on chip per chip
               Other: None
              Memory: 768 GB (24 x 32 GB 2Rx4 PC4-2666V-R)
             Storage: 1 x 240 GB SATA SSD
               Other: None
```

```
                                        SOFTWARE
                                        --------
                OS: Red Hat Enterprise Linux Server release 7.3
                    (x86_64)
                    (Maipo)
                    Kernel 3.10.0-514.el7.x86_64
            Compiler: C/C++: Version 18.0.0.128 of Intel C/C++
                    Compiler for Linux;
                    Fortran: Version 18.0.0.128 of Intel Fortran
                    Compiler for Linux
            Parallel: Yes
            Firmware: Version 0601 released Oct-2017
        Base Pointers: 64-bit
        Peak Pointers: 32/64-bit
```

hardware.txt hosted with ♥ by GitHub view raw

And here's a subset of the SPEC benchmarks for the above system:

	Base	Base	Base		Peak	Peak	Peak	
Benchmarks	Threads	Run Time	Ratio		Threads	Run Time	Ratio	
600.perlbench_s	72	285	6.24	S	72	240	7.41	S
602.gcc_s	72	423	9.42	*	72	412	9.67	S
605.mcf_s	72	426	11.1	*	72	425	11.1	S
620.omnetpp_s	72	257	6.35	*	72	248	6.58	*
623.xalancbmk_s	72	150	9.46	S	72	139	10.2	S
625.x264_s	72	149	11.8	S	72	150	11.8	*
631.deepsjeng_s	72	280	5.11	S	72	282	5.08	S
641.leela_s	72	393	4.34	S	72	392	4.36	S
648.exchange2_s	72	219	13.4	S	72	220	13.4	S
657.xz_s	72	280	22.1	*	72	277	22.3	S

benchmarks.txt hosted with ♥ by GitHub view raw

Why so many benchmarks, and what do they all mean? SPEC's goal is to evaluate machine performance across a variety of common workloads, all outlined here. For example, the 631.deepsjeng_s benchmark evaluates the speed of a machine running the alpha-beta search algorithm for playing a game of Chess. SPEC also has benchmarks for other ML workloads, ocean modeling, video compression, and more.

Is there a relationship between hardware, software, and benchmark results?

The short answer is — yes! But I wanted to confirm this before naively dropping inputs into a model and hoping for the best.

To take a closer look at the data, I wrote a Python script to parse the CSV files for each test using the CPU2017 Integer Speed results. The script extracts data on the hardware and software used in each test, along with the score for each of the benchmarks run on that machine. In the script, I first initialize a `dict()` with the data I want to capture from each test:

```python
data_row = {
    'vendor': '',
    'model_name': '',
    'nominal_mhz': '',
    'max_mhz': '',
    'cores': '',
    'chips': '',
    'channels': '',
    'l1_cache_mem_kb': '',
    'l2_cache_mem_kb': '',
    'l3_cache_mem_mb': '',
    'mem_gb': '',
    'mem_speed': '',
    'os': '',
    'compiler': '',
    'sponsor': '',
    '600.perlbench_s': '',
    '602.gcc_s': '',
    '605.mcf_s': '',
    '620.omnetpp_s': '',
    '623.xalancbmk_s': '',
    '625.x264_s': '',
    '631.deepsjeng_s': '',
    '641.leela_s': '',
    '648.exchange2_s': '',
    '657.xz_s': ''
}
```

data.py hosted with ♥ by GitHub view raw

Then I write the header row to the CSV using the keys from the dict above:

```
with open('training-data.csv', 'a') as csvfile:
    csvfile.write(','.join(data_row.keys()) + '\n')
```

write-header.py hosted with ❤ by GitHub view raw

Here's an example CSV that I'll be iterating over in the script. The next step is to iterate over the local directory where I've saved all the CSV files for the integer speed tests, and use the Python `csv` module to read them:

```
for f in os.listdir('path/to/csvfiles/'):
    reader = csv.reader(open('path/to/csvfiles/' + f))
```

read-csvs.py hosted with ❤ by GitHub view raw

The files aren't typical CSV format, but they all follow the same pattern so I can grab the benchmark data with the following:

```
benchmark_start = 0
benchmark_end = 0
seen_start = False
seen_end = False

# Iterate over each row in the CSV
for i,row in enumerate(reader):
  if row[0] == 'Benchmark' and not seen_start:
      seen_start = True
      benchmark_start = i + 1
  if row[0] == 'Selected Results Table':
      benchmark_end = i - 2
      seen_end = True
```

get-benchmark-indices.py hosted with ❤ by GitHub view raw

Using the benchmark start and end indices, I can iterate over the rows of the CSV that contain the benchmarks. Then I look for the specific data points I'm collecting within a CSV. Here's how I grab the vendor and machine model name:

```
if row[0] == 'CPU Name':
    full_name = row[1].split(' ')
    data_row['vendor'] = full_name[0]
    data_row['model_name'] = ' '.join(full_name[1:])
```

get-cpu-name.py hosted with ❤ by GitHub view raw

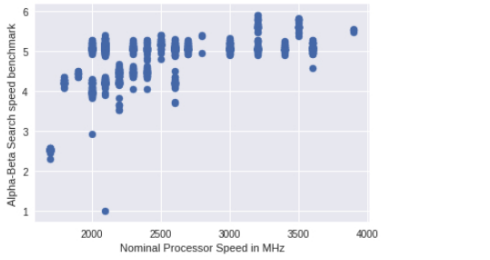
When I'm done collecting all of the machine specs, I create a CSV string of the data and write it to my file:

```
csv_string = (','.join(data_row.values()))
with open('training-data.csv', 'a') as csvfile:
    csvfile.write(csv_string + '\n')
```

data-to-csv.py hosted with ❤ by GitHub view raw

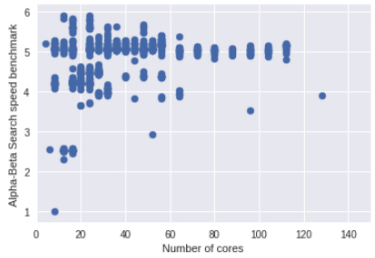
With all of the machine data and benchmark scores in a CSV file, I used a Python notebook and matplotlib to explore relationships between machine specs and benchmark results to see if there was a linear relationship. Here I've plotted the nominal speed of a machine and the `631.deepsjeng_s` (alpha-beta search) benchmark:

```
<matplotlib.collections.PathCollection at 0x7f8281221320>
```



Looks like I could draw a line or curve that roughly estimates the relationship above. Let's take a look at another input. Here's a plot of a machine's number of cores and the `631.deepsjeng_s` benchmark:

```
<matplotlib.collections.PathCollection at 0x7f82813793c8>
```



While there's some relationship between the inputs and benchmark measured above, it's clear that we couldn't use a single feature (like number of cores) to accurately predict a machine's speed. There are many features of a machine that affect its performance, and to make use of all of them we'll build a linear regression model to evaluate the speed of a machine. Here's what we'll use as inputs:

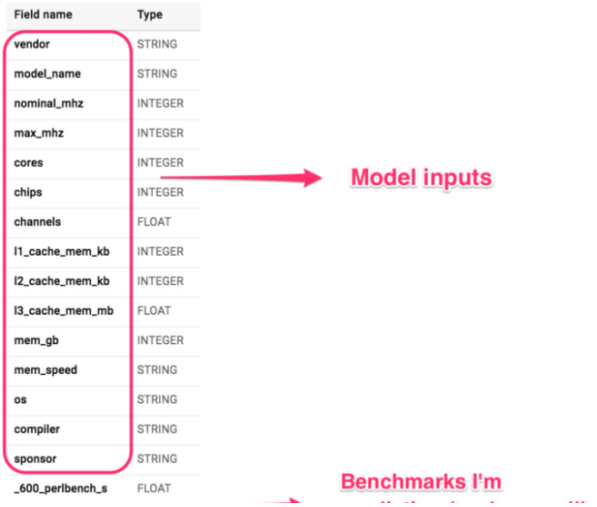
```
- The hardware vendor
- The machine model name
- Nominal and max speed of the microprocessor in MHz
- Number of cores
- Number of memory channels
- Size of L1, L2, and L3 caches
- Memory in GB
- Memory speed
- The OS running on the system
- The compiler being used
- The company running the test (Lenovo, Huawei, Dell, etc.)
```

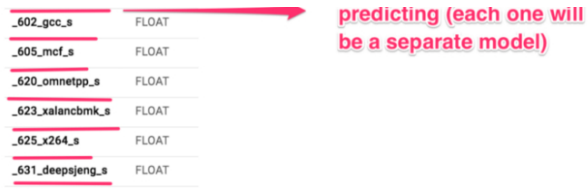
A linear regression model typically outputs a single numeric value, so we'll need to create separate models to predict each benchmark score. It's important to note that not all of our inputs are numerical — vendor name, model name, and OS name are categorical, string inputs. Since a linear regression model expects numerical inputs, we'll need a way to encode these as integers before feeding them into our model.

Building a linear regression model with BigQuery Machine Learning

We could write the code for our linear regression model by hand, but I'd like to focus on the predictions generated by my model rather than the nitty gritty of the model code. BigQuery Machine Learning (BQML) is a great tool for this job. If I create a BigQuery table with my input data and benchmarks, I can write a SQL query to train a linear regression model. BQML will handle the underlying model code, hyperparameter tuning, splitting my data into training and test sets, and converting categorical data to integers.

Since I've already got the data for my model as a CSV, I can use the BigQuery web UI to upload this data into a table. Here's the schema for the table:





And here's a preview of the data:

To get a sense of one of the categorical features we're working with, let's take a look at the `sponsor` field in our table, which refers to the company that ran an individual test. Here's the query we'll run to get a breakdown of our data by test sponsor:

```sql
SELECT
  sponsor,
  count(*) num_tests
FROM `sara-bqml.cpu_performance.spec_int_speed`
GROUP BY 1
ORDER BY num_tests DESC
```

test-sponsor.sql hosted with ❤ by GitHub view raw

The results:

Row	sponsor	num_tests
1	Lenovo Global Technology	549
2	Huawei	312
3	Dell Inc.	184
4	HPE	172
5	Cisco Systems	85
6	NEC Corporation	76
7	ASUSTeK Computer Inc.	21
8	Supermicro	10
9	New H3C Technologies Co. Ltd.	5
10	Sugon	2
11	Oracle Corporation	1
12	Fujitsu	1

And here are the top operating systems used in the tests:

Row	os	c
1	SUSE Linux Enterprise Server 12	916
2	Red Hat Enterprise Linux Server release 7.4	243
3	Red Hat Enterprise Linux Server release 7.3	228
4	CentOS Linux release 7.4.1708	20

I can train my model with a single SQL query in BQML:

```sql
CREATE OR REPLACE MODEL
  `cpu_performance.spec_int_speed_631`
OPTIONS
  (model_type='linear_reg',input_label_cols=['_631_deepsjeng_s']) AS
SELECT * FROM (
  SELECT
    vendor,
    model_name,
    max_mhz,
    nominal_mhz,
    cores,chips,
    channels,
    mem_gb,
    mem_speed,
    l1_cache_mem_kb ,
    l2_cache_mem_kb ,
    l3_cache_mem_mb ,
    os,
    compiler,
    sponsor,
    _631_deepsjeng_s  # This is what we're predicting
  FROM
    `sara-bqml.cpu_performance.spec_int_speed`
  )
```

create-model.sql hosted with ❤ by GitHub view raw

This took 1 minute and 14 seconds to train. When it completes, I can look at the stats for each iteration of training:

Iteration	Training Data Loss	Evaluation Data Loss	Learn Rate	Completion Time (seconds)
8	0.0106	0.0181	0.2000	4.33
7	0.0107	0.0181	0.4000	5.06
6	0.0108	0.0184	0.2000	6.57
5	0.0112	0.0188	0.4000	3.61
4	0.0119	0.0198	0.2000	12.56
3	0.0155	0.0235	0.4000	5.88
2	0.0227	0.0341	0.2000	4.78
1	0.0737	0.0874	0.2000	4.10
0	1.0094	1.0645	0.2000	3.71

Loss metrics for each epoch of training in BQML

The number I want to focus on here is the *Evaluation Data Loss* — this is the loss metric calculated after each iteration of training. We can see that it has steadily decreased from the first to the last iteration. To get additional evaluation metrics, I can run an `ML.EVALUATE` query. Here are the results:

Row	mean_absolute_error	mean_squared_error	mean_squared_log_error	median_absolute_error	r2_score	explained_variance
1	0.047075499932594696	0.012111808200206346 1	4.3008960888392503E-4	0.029743702375679 25	0.9695759591590659	0.969577473050911

Evaluation metrics on my BQML model

Mean squared error (MSE) measures the difference between the values our model predicted using the test set and the actual values. You can also think of it as the distance between your regression (best fit) line and the predicted values. A smaller value is better, and our model's `.0121` MSE is very good.

Since this is a regression model (predicting a continuous numerical value), the best way to see how it performed is to evaluate the difference between the value predicted by the model and the ground truth benchmark score. We can do this with an `ML.PREDICT` query.

When this query runs it'll create a new field prefixed with `predicted_` and the name of our label column (in this case `_631_deepsjeng_s`). Let's run `ML.PREDICT` across the original dataset and output a few features of the system being tested, the actual speed test result, and predicted benchmark:

```
SELECT
  model_name,
  cores,
  nominal_mhz,
  compiler,
  predicted__631_deepsjeng_s as predicted_score,
  _631_deepsjeng_s as actual_score
FROM ML.PREDICT (
  MODEL `cpu_performance.spec_int_speed_631` , (
    SELECT * FROM `sara-bqml.cpu_performance.spec_int_speed`
  )
)
```

model-predict.sql hosted with ❤ by GitHub view raw

Our predictions are very close to the actual score values!

Row	model_name	cores	nominal_mhz	compiler	predicted_score	actual_score
1	Xeon E5-2699 v4	44	2200	C/C++: Version 17.0.0.098 of Intel C/C++	3.856425990738549	3.840884
2	Xeon E7-8890 v4	96	2200	C/C++: Version 17.0.0.098 of Intel C/C++	3.6253341606751515	3.528768
3	Xeon E7-8890 v4	384	2200	C/C++: Version 17.0.0.098 of Intel C/C++	3.3985257841657623	3.518977
4	Xeon Gold 5115	20	2400	C/C++: Version 18.0.0.128 of Intel C/C++	4.44681494602961	4.487496
5	Xeon Gold 5115	20	2400	C/C++: Version 18.0.0.128 of Intel C/C++	4.426730918719588	4.495226
6	Xeon Gold 5115	40	2400	C/C++: Version 18.0.0.128 of Intel C/C++	4.387249756362787	4.388471

To measure this another way, we can subtract the predicted score from the actual score, and then get the average of the absolute values of these differences:

```sql
SELECT
  avg(abs(predicted__631_deepsjeng_s - _631_deepsjeng_s)) as avg_score_diff
FROM ML.PREDICT (
  MODEL `cpu_performance.spec_int_speed_631` , (
    SELECT * FROM `sara-bqml.cpu_performance.spec_int_speed`
  )
)
```

avg-score-diff.sql hosted with ❤ by GitHub view raw

On average, our model's prediction is only .04 off from the original value — pretty impressive.

Generating predictions on new data

Now that I've got a trained model, I can use it to predict the speed benchmarks for a machine that wasn't part of my training set with the following query:

```sql
SELECT * FROM ML.PREDICT (
  MODEL `cpu_performance.spec_int_speed_631` , (
    SELECT
      "Intel" vendor,
      "Xeon Platinum 8180M" model_name,
      3800 max_mhz,
      2500 nominal_mhz,
      112 cores,
      4 chips,
      32 l1_cache_mem_kb,
      1000 l2_cache_mem_kb,
      38.5 l3_cache_mem_mb,
      1536 mem_gb,
      "2666V" mem_speed,
      12.0 channels,
      "Cisco Systems" sponsor,
      "SUSE Linux Enterprise Server 12" os,
      "C/C++: Version 18.0.0.128 of Intel C/C++" compiler
))
```

predict-example.sql hosted with ❤ by GitHub view raw

The SPEC speed score for this machine on the `631.deepsjeng_s` benchmark should be `5.19`. When I run this query my model predicts `5.20` which is pretty close to the actual value. I can now use this model to generate predictions on new combinations of hardware and software specs that weren't included in the test dataset. In addition to the alpha-beta search benchmark, I created models for other Integer Speed benchmarks that had similar accuracy to this one — let me know if you're interested in these results and I can share details.

Because I didn't have to write any of the underlying model code, I was able to focus on feature engineering and finding the right combination of inputs to build a high accuracy model — thanks BQML!

More with SPEC data

If you're interested in experimenting with this data and creating your own models, check out the Integer Speed benchmark results here. It's also worth noting that SPEC has lots of data on previous CPU benchmarks. The 2006 SPEC CPU benchmarks have many more years worth of data (though SPEC has retired this set), so you could train a model using a similar approach to predict the performance of older machines.

This post focused on integer speed benchmarks. I haven't included the other SPEC benchmarks here (floating point speed and throughput) because they didn't perform as well with linear regression models (MSE in the 100s compared to MSE of less than 0.1 in the integer speed models). I'm planning to experiment with these tests using other types of models, stay tuned :)

In addition to SPEC, there are also many other performance benchmarks you could use like LINPACK, which is used by Top 500.

Have feedback?

This is my first foray into hardware performance data, so I'd love any feedback you have on the features I used as inputs, other types of models to

try, or anything else. Leave a comment below or find me on Twitter at @SRobTweets.

If you want to learn more about BQML, my teammates have some great posts: this one covers predicting Stack Overflow response times, and this one walks you through building a model to predict flight delays.

Thank you to these awesome people for their contributions and feedback: Brian Kang, Shane Glass, Abhishek Kashyap, Eric Van Hensbergen, Jon Masters, and a few other SPEC experts at Red Hat 😀

****

*Data source: https://www.spec.org/cpu2017/results/cint2017.html Data retrieved on November 11 2018. SPEC Fair Use Rules: https://www.spec.org/fairuse.html

Machine Learning | Linear Regression | Bigquery | Hardware | Cpu

WRITTEN BY
Sara Robinson
Connoisseur of code, country music, and homemade ice cream.
Helping developers build awesome apps @googlecloud.
Opinions = my own, not that of my company.

HackerNoon.com
Elijah McClain, George Floyd, Eric Garner, Breonna Taylor, Ahmaud Arbery, Michael Brown, Oscar Grant, Atatiana Jefferson, Tamir Rice, Bettie Jones, Botham Jean