

Ashley Hale
Due 5/8/2020
Final Project

NOTE: Due to competing priorities for school work, I ran out of time writing up this project. I did complete running the entire tutorial (multiple times, actually). As I was doing so, I wrote section *0. Introduction* and simultaneously did research as I went. As a result, I have a lot of detail in *0. Introduction*, including several citations. In hindsight, I should have written most of this in sections *1. Project Description* and later. Since I ran out of time, sections *1. Project Description* and later are more summary in nature. Please consider *0. Introduction* when grading part 2 of the project. I believe all sections in this document **taken together** fulfill the intentions of both parts 1 and 2 in the final project assignment.

0. Introduction


This report is for the “defined” Final Project, based on the tutorial at <https://k-d-w.org/blog/2019/07/survival-analysis-for-deep-learning/>. As per the assignment, the project has two main parts:

1. Data and Implementation
 - a. This is covered in the remainder of section *0. Introduction*
2. Final Report
 - a. This is covered in sections *1. Project Description* and later

This document and all other materials are available in the GitHub repository <https://github.com/ashhale/survival-analysis>.

Running the Jupyter Notebook

The Jupyter notebook is available in the GitHub repository <https://github.com/ashhale/survival-analysis>. To do anything with it, you must have a Google account (to use Google Colaboratory) and access to the GitHub repository.

To run the notebook, go to https://github.com/ashhale/survival-analysis/blob/master/Ashley_Hale_tutorial.ipynb and click the  button, or click the button here in this document. The notebook will open in “playground mode” using your personal Google account. You can read through, and run code cells in, the file as is. If you want to save it, you’ll need to save a copy of it in your own Google Drive.

IMPORTANT

1. Optional but highly recommended: set up a GPU. See *Using a GPU* below. Do this first since doing it later will essentially “factory reset” your Colab VM.
2. MANDATORY: You must follow the steps in *Installing Python Packages* below before trying to run anything else in the notebook.

Using a GPU

To setup a GPU in the Colab VM, see the notes in Appendix A.

Getting a GPU working was extremely time consuming. TensorFlow 2.x would not work with the latest scikit-survival-0.12.0, and using `!pip install` to get an older version of TensorFlow appeared to work but would never work with a GPU. I spent most of 2 days trying to figure this out. I finally stumbled across information that explicitly states you should not use `!pip` to install versions of TensorFlow since the Google Colab folks have compiled versions of TensorFlow specifically to work with their fleet of GPUs. Instead, you need to use a “magic” command to make it work. For details see

https://colab.research.google.com/notebooks/tensorflow_version.ipynb

Installing Python Packages

1. Due to version mismatches of Python packages, you must execute the first code cell to configure the Colab VM with specific versions. This will take a minute or two at most.
 - a. If you do anything that resets the filesystem in your Colab VM (like disconnecting, or *Runtime > Factory reset runtime*, or changing hardware via *Runtime > Change runtime type*) then you will need to re-run the first code cell.
2. After running the first code cell, and before running any other code cells, you need to do a *Runtime > Restart runtime*. This is needed due to a version mismatch between the already-running Colab VM and `pandas-0.25.3` that was installed in step 1. *Runtime > Restart runtime* essentially reboots the Colab VM, but does not alter its filesystem.
3. After the restart, then you can continue with later code cells as usual.

Run Results

I successfully executed the entire tutorial notebook. Changes I made to the notebook are in the top cell of the notebook under *Implementation Notes*. The following notes describe running the tutorial sections, what they’re doing, and their output.

Tutorial: Setup

First, I figured out the Python packages to use, and running with the GPU. These are covered in the first two code cells. The end of the output from the first code cell is:

```
Successfully built scikit-survival
ERROR: google-colab 1.0.0 has requirement pandas~=1.0.0; python_version >= "3.0", but you'll
have pandas 0.25.3 which is incompatible.
Installing collected packages: osqp, pandas, scikit-learn, scikit-survival
  Found existing installation: osqp 0.6.1
    Uninstalling osqp-0.6.1:
      Successfully uninstalled osqp-0.6.1
  Found existing installation: pandas 1.0.3
    Uninstalling pandas-1.0.3:
      Successfully uninstalled pandas-1.0.3
  Found existing installation: scikit-learn 0.22.2.post1
    Uninstalling scikit-learn-0.22.2.post1:
      Successfully uninstalled scikit-learn-0.22.2.post1
Successfully installed osqp-0.5.0 pandas-0.25.3 scikit-learn-0.21.3 scikit-survival-0.11

WARNING: The following packages were previously imported in this runtime:
[pandas]
You must restart the runtime in order to use newly installed versions.
```

The output from the second code cell confirms we're working with TensorFlow 1.15.2 and that we will be using a GPU:

```
Start: 2020-05-09 21:34:30
Currently selected TF version: 2.x
Available versions:
* 1.x
* 2.x
TensorFlow 1.x selected.
Using Tensorflow: 1.15.2
Num GPUs Available: 1
End: 2020-05-09 21:34:36
```

Tutorial: Generating Synthetic Survival Data from MNIST

MNIST is a database of handwritten digits, described at <http://yann.lecun.com/exdb/mnist/>. TensorFlow has a built in command to download this dataset, as per https://www.tensorflow.org/api_docs/python/tf/keras/datasets/mnist/load_data. This dataset is being used to generate some fake survival times, based on the digits written in each image. The output of the data load is:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [=====] - 0s 0us/step
```

There are 10 class labels, corresponding to the ten digits 0 through 9. We map each of those 10 class labels into one of 4 groupings of risk, numbered 0 to 3, where group 0 is low risk and

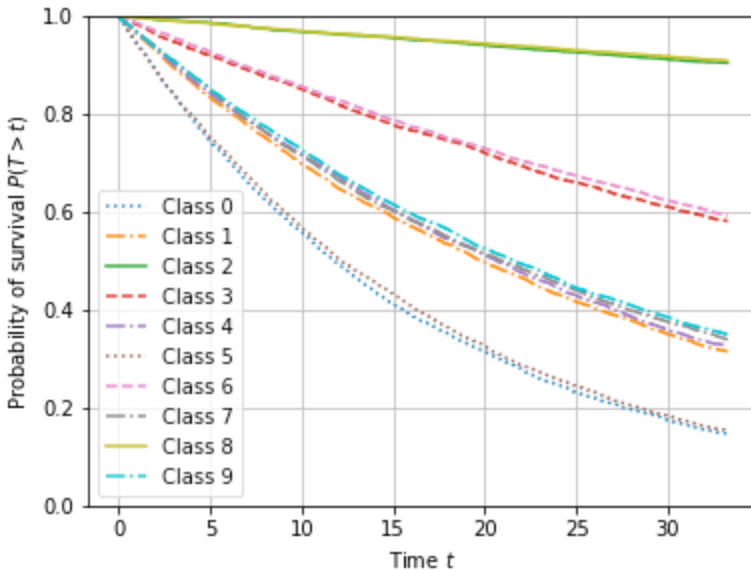
group 3 is high risk. The mapping of class labels (or digits) to risk groups is random. To make the results repeatable for this tutorial, the random number generator is seeded with a specific integer (89). The output of the step that makes these random assignments is:

class_label	risk_score	risk_group
0	3.071	3
1	2.555	2
2	0.058	0
3	1.790	1
4	2.515	2
5	3.031	3
6	1.750	1
7	2.475	2
8	0.018	0
9	2.435	2

Next we generate some survival times using a mathematical approach described in a paper by Bender et al. [1]. We generate these with a goal of a mean survival time of 365 days. We also have a goal of 45% censoring. Right censored data is where the sample has not yet “failed,” or the sample has not yet experienced the “event,” by the end of the study [2]. So we’re shooting for a goal of 45% of samples either surviving through the end of the study or with unknown status (such as dropped out of the study early). Our function gets close to that, for both the training and test data:

```
46.19% samples are right censored in training data.  
46.33% samples are right censored in test data.
```

Finally, we create a survival function, which calculates the probability that a subject will survive beyond a particular time. The probability of survival at time 0 is 1 (everyone survives at the start of the study), and at time infinity is 0 (everyone eventually dies). We create the survival function by stratifying (i.e., classifying, or grouping) the training data by class label (digit 0 to 9) and estimating the survival function using another mathematical approach, the Kaplan-Meier estimator, which, “can take into account some types of censored data, particularly *right-censoring*” [3]. We then graph the results:



This graph makes it easy to see how each class label (digit) survives relative to its risk group. Recall that digits 2 and 8 were in risk group 0, the lowest risk. It makes sense, as the graph shows, that they have the best chance of survival (the two overlapped green lines at the top of the graph). Digits 3 and 6 in risk group 1 have a slightly lower probability of survival, as shown in the second cluster of lines from the top. Digits 1, 4, 7 and 9 in risk group 2 are in the next cluster down, and digits 0 and 5 in risk group 3, the highest risk, have the worst odds of survival and appear in the bottom cluster of lines on the graph.

Tutorial: Evaluating Predictions

We want to see how good our predictions are, or how much our risk scores are “concordant” with (consistent with, in agreement with) actual survival time. We use Harrell's concordance index, a good, understandable explanation of which is given by Tay [4]. Despite the tightness of our graph above, our concordance is nowhere near 1.0. This is because our generated survival times aren't actually deterministic functions of the risk score. Our survival scores are just randomly distributed based on risk scores.

Concordance index on test data with actual risk scores: 0.705

Tutorial: Cox's Proportional Hazards Model

Assuming a subject is alive (has not failed, has had no event occur) at a particular time, the hazard function gives an approximate probability that that subject will die (will fail, will have an event occur) in the following small time interval. Cox's proportional hazards model [5] creates a hazard function that takes into account a set of covariates, or conditions. If trying to predict deaths, covariates might include things like age, gender, weight, health, financial status, etc.

Tutorial: Non-linear Survival Analysis with Neural Networks

This section describes a method to use a neural network to extend the linear Cox's proportional hazards model to handle non-linear cases.

Tutorial: Computing the Loss Function

I didn't follow the math for this section, but in the prior homeworks we learned a loss function is used to measure how well a model is doing. Brownlee also provides a good description of loss functions [6]. In this tutorial, we have an implementation of a Cox's proportional hazards (PH) loss function, `coxph_loss` (starting line 182 in the code cell).

The next code cell implements a class to provide a concordance measurement so we can see how closely the CNN gets to the concordance index we predicted earlier, of 0.705. The class `EvalCindexHook`, starting at line 11, calculates the concordance index for a single epoch of training, and the set of these values can be graphed on the TensorBoard in a later cell.

Tutorial: Creating a Convolutional Neural Network for Survival Analysis on MNIST

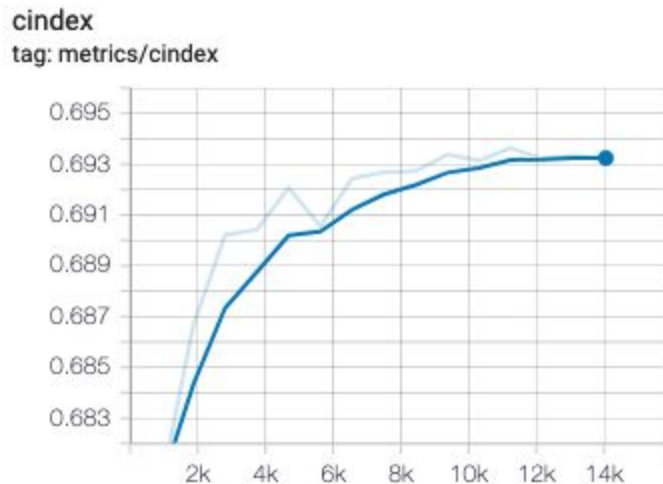
This section is where we implement the CNN. As per the comments, the CNN uses the LeNet architecture. Here's the snippet of code that defines the layers in the CNN, and this is similar to the LeNet architecture we had in homework 7. The last layer has a single node for the predicted risk score.

```
4.     model = tf.keras.Sequential([
5.         tf.keras.layers.Conv2D(6, kernel_size=(5, 5), activation='relu', name='conv_1'),
6.         tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
7.         tf.keras.layers.Conv2D(16, (5, 5), activation='relu', name='conv_2'),
8.         tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
9.         tf.keras.layers.Flatten(),
10.        tf.keras.layers.Dense(120, activation='relu', name='dense_1'),
11.        tf.keras.layers.Dense(84, activation='relu', name='dense_2'),
12.        tf.keras.layers.Dense(1, activation='linear', name='dense_3')
13.    ])
```

After this, we setup our TensorBoard, and then train the model. Running the code cells individually makes the TensorBoard flaky, but if you use the menus to batch-run cells in sequence, the TensorBoard runs and updates dynamically while the CNN trains. Though when using the GPU it only updates once or twice since the training is pretty quick.

As I was studying this tutorial, I ended up running the training a few times. Without a GPU, the training ranged from a bit more than 9 minutes to a bit over 10 minutes. With a GPU, the training consistently took about 1.25 minutes.

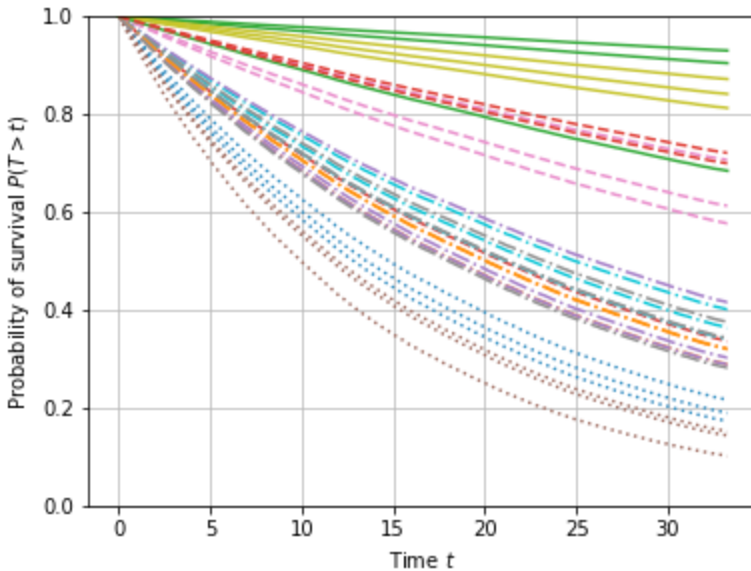
The TensorBoard was interesting to explore. There were several graphs available. Of particular note was the graph here, which shows the concordance value as training progresses. We noted earlier that we would not be able to exceed 0.705, and this graph indicates we got close, a bit under 0.694.



Additional TensorBoard screenshots, and a short video of the Projector views, are in the GitHub repository.

Tutorial: Predicting Survival Functions

Finally, we randomly select 3 images for each class (digit) and plot the predicted survival function for it. This graph uses the same line styles used in the earlier graph. Solid lines are the digits with the lowest risk, and those lines are at the top of the graph, indicating they live the longest. The dotted lines are the digits with the highest risk and they appear at the bottom of the graph, indicating they die the earliest. To match the line coloring and line styles to particular digits, see the legend in the earlier graph.



1. Project Description

With this tutorial, we learned about how to do survivor analysis, which is a different type of analysis than we've done with CNNs in past assignments. Survivor analysis is used to estimate the time to a an event, such as a death in a medical trial, or the failure of an object like a light bulb, or the time for a steel beam to rust through.

2. Description of Data Used

In this tutorial, we used the MNIST database of handwritten digits, described at <http://yann.lecun.com/exdb/mnist/>. Clearly, this data set is not medical or clinical in nature. But it was used to generate some fake survival times, based on the digits written in each image. There are 10 class labels, corresponding to the ten digits 0 through 9. We map each of those 10 class labels into one of 4 groupings of risk, numbered 0 to 3, where group 0 is low risk and group 3 is high risk. The mapping of class labels (or digits) to risk groups is random. To make the results repeatable for this tutorial, the random number generator is seeded with a specific integer (89).

3. Related Work

Related work is referenced in several places in section 0. *Introduction*.

4. Methods

The methods used are described throughout section *0. Introduction*.

5. Results

The results included a trained CNN based on a dataset of handwritten images used to generate simulated survival data including 10 classes mapped to 4 risk groupings. For more details, see section *0. Introduction*.

Also, note there are images and videos of the TensorBoard in the GitHub repository.

6. Discussion

The tutorial served its purpose and achieved the results it set out to do. Most things worked out pretty well, though the concordance of 0.705 meant our success was limited from early on. It would be good to try this with a more realistic dataset where we could achieve higher concordance and thus have a chance for better overall results.

7. Future Work

First, I'd like more time to dig in and understand some of the underlying math. I'd also like to try other CNN architectures to see how well they do. This seems like a much more achievable exercise now that I've had my first taste of training CNNs with a GPU available. (I'm very impressed with Google Colab overall. I'm pretty sure this project could have been done with a low-end Chromebook!)

Beyond that, I think the most interesting thing would be to try the techniques in this tutorial with more interesting clinical data. I'm sure there are, and will be more, extensive datasets regarding COVID-19 survivability. And while our class is focused on health and medicine, I would find it interesting to apply these techniques to some examples of technology failures, if such datasets exist.

8. Conclusion

Overall, I think this tutorial rates fair to good as a learning exercise. I might rate it higher if I'd had more time to work on it (my fault, not the professor's). I dug into math as needed in earlier sections, but as the submission deadline loomed, I was forced to accept later math at face

value. Even so, I think I came away with a decent understanding of what each step does, just at a higher level in some sections than others.

9. Citations

While not typically provided in formal papers, when a citation is for an academic paper, and I was able to find a PDF of that paper that I could read, I append a URL for the PDF file after the citation.

[1] Bender, Ralf, Thomas Augustin, and Maria Blettner. "Generating survival times to simulate Cox proportional hazards models." *Statistics in medicine* 24, no. 11 (2005): 1713-1723. (PDF found at

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.330.215&rep=rep1&type=pdf>)

[2] Wikipedia contributors, "Censoring (statistics)," *Wikipedia, The Free Encyclopedia*, [https://en.wikipedia.org/w/index.php?title=Censoring_\(statistics\)&oldid=950498717](https://en.wikipedia.org/w/index.php?title=Censoring_(statistics)&oldid=950498717) (accessed May 9, 2020).

[3] Wikipedia contributors, "Kaplan–Meier estimator," *Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/w/index.php?title=Kaplan%E2%80%93Meier_estimator&oldid=948523181 (accessed May 9, 2020).

[4] Tay, Kenneth. "What is Harrell's C-index?" Statistical Odds & Ends. <https://statisticaloddsandends.wordpress.com/2019/10/26/what-is-harrells-c-index/> (accessed May 9, 2020)

[5] Cox, David R. "Regression models and life-tables." *Journal of the Royal Statistical Society: Series B (Methodological)* 34, no. 2 (1972): 187-202. (PDF found at <https://web.stanford.edu/~lutian/coursepdf/cox1972paper.pdf>)

[6] Brownlee, Jason. "Loss and Loss Functions for Training Deep Learning Neural Networks." Machine Learning Mastery. <https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/> (accessed May 9, 2020)

Appendix A - Setup and Configuration Info

Setup on Google Colaboratory for GPU Access

The tutorial says, “The notebook to reproduce the results is available on [GitHub](#), or you can run it directly using [Google Colaboratory](#).” Clicking the [Google Colaboratory](#) link opens a new browser tab with the tutorial Jupyter notebook already opened in Google Colab.

Exploring the Colab menus, I found *Edit > Notebook Settings* which allows you to select a Hardware Accelerator (GPU or TPU) and has a link (“?” in a circle) to a FAQ:

<https://research.google.com/colaboratory/faq.html#gpu-availability>

There are a few questions and answers that describe more about the availability of GPUs, what kind you might get, advice to turn them off when not needed, etc., including, “For examples of how to utilize GPU and TPU runtimes in Colab, see the [Tensorflow With GPU](#) and [TPUs In Colab](#) example notebooks.” Clicking on the GPU example opens a notebook that is worth exploring to learn how the GPU helps performance and also how to disable it when not using it, and enable it when needed. On my run, the result was: GPU speedup over CPU: 28x

I also found code to see info about your current GPU and memory at this notebook:

<https://colab.research.google.com/notebooks/pro.ipynb>

By copying the two code cells from that notebook to the GPU example notebook above, I was able to see the GPU and memory available in the example notebook:

```
1.  gpu_info = !nvidia-smi
2.  gpu_info = '\n'.join(gpu_info)
3.  if gpu_info.find('failed') >= 0:
4.      print('Select the Runtime > "Change runtime type" menu to enable a GPU accelerator, ')
5.      print('and then re-execute this cell.')
6.  else:
7.      print(gpu_info)
```

```
+-----+
| NVIDIA-SMI 440.82                Driver Version: 418.67                CUDA Version: 10.1                |
+-----+-----+-----+-----+-----+-----+
| GPU  Name            Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|====+=====+====+=====+=====+=====+=====+
|   0   Tesla T4             Off   | 00000000:00:04:0 Off |                    0 |
| N/A   66C    P0      27W /  70W |  1695MiB / 15079MiB |      0%      Default |
+-----+-----+-----+-----+-----+-----+

+-----+
| Processes:                                     GPU Memory |
|  GPU       PID    Type    Process name                     Usage      |
+=====+
|
```

+-----+

```
1. from psutil import virtual_memory
2. ram_gb = virtual_memory().total / 1e9
3. print('Your runtime has {:.1f} gigabytes of available RAM\n'.format(ram_gb))
4.
5. if ram_gb < 20:
6.     print('To enable a high-RAM runtime, select the Runtime > "Change runtime type"')
7.     print('menu, and then select High-RAM in the Runtime shape dropdown. Then, ')
8.     print('re-execute this cell.')
9. else:
10.    print('You are using a high-RAM runtime!')
```

Your runtime has 13.7 gigabytes of available RAM

To enable a high-RAM runtime, select the Runtime > "Change runtime type" menu, and then select High-RAM in the Runtime shape dropdown. Then, re-execute this cell.

Random Notes About Google Colaboratory

These are random notes and bookmarks that proved of interest while using Google Colaboratory as a first-time user.

- To Save to a Private GitHub Repository
 - *[I later made my repository public, but these are still good notes for the future.]*
 - Setup a private repository in GitHub; I don't know if it's necessary, but I populated it with a README.md to start
 - You must save the tutorial into your Google Drive
 - This creates a Colab Notebooks folder in your Google Drive
 - You can open the notebook in the future by going to that folder, right click, and *Open with > Google Colaboratory*
 - Then in Colab, *File > Save a copy in GitHub*
 - You'll be prompted to give GitHub authorization, go through that process
 - I don't think you need to save at the end of the process, you're just setting up permissions in GitHub to allow Colab to write to your public repositories
 - Back in Colab, go to *File > Open notebook*
 - Select the GitHub tab and check the box for Include private repos
 - You will be sent to GitHub again to authorize Colab for access to your private repos
 - If the pulldown Repository menu does not populate with your private repo, try canceling that dialog and repeat *File > Open notebook*
 - Once you can see private repos, in Colab, go to *File > Save a copy in GitHub*
 - To remove Colab's authorization in GitHub, go to <https://github.com/settings/applications>
- Snippets: Importing libraries - Colaboratory
 - https://colab.research.google.com/notebooks/snippets/importing_libraries.ipynb
 - Importing libs, using files, etc.

- Mounting Google Drive in your VM (reading and writing to Google Drive)
 - <https://colab.research.google.com/notebooks/snippets/drive.ipynb>

Google Colab and Related References

- Getting Started with TensorFlow in Google Colaboratory
 - <https://towardsdatascience.com/getting-started-with-tensorflow-in-google-colaboratory-9a97458e1014>
- Google Colab Tips and Tricks
 - <https://www.google-colab.com/google-colab-tips-and-tricks/>