ENGINEERING

Would You Survive the Titanic? A Guide to Machine Learning in Python 1

July 4, 2016 11 Comments Patrick Triest 15 min read

What if machines could learn?

This has been one of the most intriguing questions in science fiction and philosophy since the advent of machines. With modern technology, such questions are no longer bound to creative conjecture. Machine learning is all around us. From deciding which movie you might want to watch next on Netflix to predicting stock market trends, machine learning has a profound impact on how data is understood in the modern era.

This tutorial aims to give you an accessible introduction on how to use machine learning techniques for your projects and data sets. In just 20 minutes, you will learn how to use Python to apply different machine learning techniques — from decision trees to deep neural networks — to a sample data set. This is a practical, not a conceptual, introduction; to fully understand the capabilities of machine learning, I highly recommend that you seek out resources that explain the low-level implementations of these techniques.

Our sample dataset: passengers of the RMS Titanic. We will use an open data set with data on the passengers aboard the infamous doomed sea voyage of 1912. By examining factors such as class, sex, and age, we will experiment with different machine learning algorithms and build a program that can predict whether a given passenger would have survived this disaster.

Setting Up Your Machine Learning Laboratory

The best way to learn about machine learning is to follow along with this tutorial on your computer. To do this, you will need to install a few software packages if you do not have them yet:

- Python (version 3.4.2 was used for this tutorial): https://www.python.org
- SciPy Ecosystem (NumPy, SciPy, Pandas, IPython, matplotlib): https://www.scipy.org
- SciKit-Learn: http://scikit-learn.org/stable/
- TensorFlow: https://www.tensorflow.org

There are multiple ways to install each of these packages. I recommend using the “pip” Python package manager, which will allow you to simply run “pip3 install <packagename>” to install each of the dependencies: https://pip.pypa.io/en/stable/.

For actually writing and running the code, I recommend using IPython (which will allow you to run modular blocks of code and immediately view the output values and data visualizations) along with the Jupyter Notebook as a graphical interface: https://jupyter.org.

You will also need the Titanic dataset that we will be analyzing. You can find it here: http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic3.xls .

With all of the dependencies installed, simply run “jupyter notebook” on the command line, from the same directory as the titanic3.xls file, and you will be ready to get started.

The Data at First Glance: Who Survived the Titanic and Why?

First, import the required Python dependencies.

```
In [1]: import matplotlib.pyplot as plt
        %matplotlib inline
        import random
        import numpy as np
        import pandas as pd
        from sklearn import datasets, svm, cross_validation, tree, preproce
        ssing, metrics
        import sklearn.ensemble as ske
        import tensorflow as tf
        from tensorflow.contrib import skflow
```

import.ipynb hosted with ❤ by **GitHub** **view raw**

Once we have read the spreadsheet file into a Pandas dataframe (imagine a hyperpowered Excel table), we can peek at the first five rows of data using the head() command.

```
In [2]: titanic_df = pd.read_excel('titanic3.xls', 'titanic3', index_col=No
        ne, na_values=['NA'])

In [3]: titanic_df.head()
```

Out[3]:

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare
0	1	1	Allen, Miss. Elisabeth Walton	female	29.0000	0	0	24160	211.337
1	1	1	Allison, Master.	male	0.9167	1	2	113781	151.550

datainput.ipynb hosted with ❤ by **GitHub** **view raw**

The column heading variables have the following meanings:

- **survival:** Survival (0 = no; 1 = yes)
- **class:** Passenger class (1 = first; 2 = second; 3 = third)

- **name:** Name
- **sex:** Sex
- **age:** Age
- **sibsp:** Number of siblings/spouses aboard
- **parch:** Number of parents/children aboard
- **ticket:** Ticket number
- **fare:** Passenger fare
- **cabin:** Cabin
- **embarked:** Port of embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)
- **boat:** Lifeboat (if survived)
- **body:** Body number (if did not survive and body was recovered)

Now that we have the data in a dataframe, we can begin an advanced analysis of the data using powerful single-line Pandas functions. First, let's examine the overall chance of survival for a Titanic passenger.

```
In [3]:  titanic_df['survived'].mean()

Out[3]:  0.3819709702062643
```

survival_avg.ipynb hosted with ❤ by **GitHub** **view raw**

The calculation shows that only 38% of the passengers survived. Not the best odds. The reason for this massive loss of life is that the Titanic was only carrying 20 lifeboats, which was not nearly enough for the 1,317 passengers and 885 crew members aboard. It seems unlikely that all of the passengers would have had equal chances at survival, so we will continue breaking down the data to examine the social dynamics that determined who got a place on a lifeboat and who did not.

Social classes were heavily stratified in the early twentieth century. This was especially true on the Titanic, where the luxurious first-class areas were completely off limits to the middle-class passengers in second class, and especially to those who carried a third class "economy price" ticket. To get a view into the composition of each class, we can group data by class, and view the averages for each column:

```
In [4]: titanic_df.groupby('pclass').mean()
```

Out[4]:

	survived	age	sibsp	parch	fare	body
pclass						
1	0.619195	39.159918	0.436533	0.365325	87.508992	162.828571
2	0.429603	29.506705	0.393502	0.368231	21.179196	167.387097
3	0.255289	24.816367	0.568406	0.400564	13.302889	155.818182

classagg.ipynb hosted with ❤ by **GitHub** **view raw**

We can start drawing some interesting insights from this data. For instance, passengers in first class had a 62% chance of survival, compared to a 25.5% chance for those in 3rd class. Additionally, the lower classes generally consisted of younger people, and the ticket prices for first class were predictably much higher than those for second and third class. The average ticket price for first class (£87.5) is equivalent to $13,487 in 2016.

We can extend our statistical breakdown using the grouping function for both class and sex:

```
In [5]: class_sex_grouping = titanic_df.groupby(['pclass','sex']).mean()
        class_sex_grouping
```

Out[5]:

		survived	age	sibsp	parch	fare	body
pclass	**sex**						
1	**female**	0.965278	37.037594	0.555556	0.472222	109.412385	NaN
	male	0.340782	41.029250	0.340782	0.279330	69.888385	162.828
2	**female**	0.886792	27.499191	0.500000	0.650943	23.234827	52.000
	male	0.146199	30.815401	0.327485	0.192982	19.904946	171.233
3	**female**	0.490741	22.185307	0.791667	0.731481	15.324250	183.000

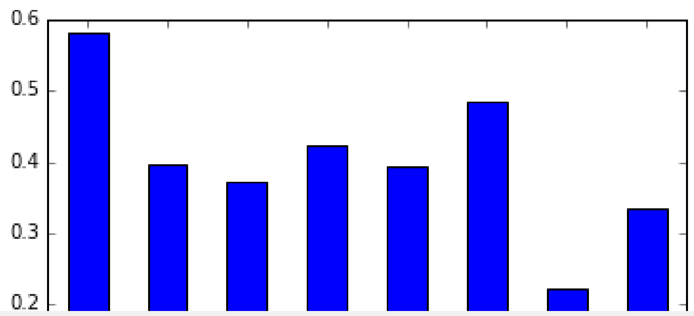
classsexagg.ipynb hosted with ❤ by **GitHub** **view raw**

While the Titanic was sinking, the officers famously prioritized who was allowed in a lifeboat with the strict maritime tradition of evacuating women and children first. Our statistical results clearly reflect the first part of this policy as, across all classes, women were much more likely to survive than the men. We can also see that the women were younger than the men on average, were more likely to be traveling with family, and paid slightly more for their tickets.

The effectiveness of the second part of this "Women and children first" policy can be deduced by breaking down the survival rate by age.

```
In [7]: group_by_age = pd.cut(titanic_df["age"], np.arange(0, 90, 10))
        age_grouping = titanic_df.groupby(group_by_age).mean()
        age_grouping['survived'].plot.bar()

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x1115a58d0>
```



ageagg.ipynb hosted with ❤ by GitHub view raw

Here we can see that children were indeed the most likely age group to survive, although this percentage was still tragically below 60%.

Why Machine Learning?

With analysis, we can draw some fairly straightforward conclusions from this data — being a woman, being in 1st class, and being a child were all factors that could boost your chances of survival during this disaster.

Let's say we wanted to write a program to predict whether a given passenger would survive the disaster. This could be done through an elaborate system of nested if-else statements with some sort of weighted scoring system, but such a program would be long, tedious to write, difficult to generalize, and would require extensive fine tuning.

This is where machine learning comes in: we will build a program that learns from the sample data to predict whether a given passenger would survive.

Preparing The Data

Before we can feed our data set into a machine learning algorithm, we have to remove missing values and split it into training and test sets.

If we perform a count of each column, we will see that much of the data on certain fields is missing. Most machine learning algorithms will have a difficult time handling missing values, so we will need to make sure that each row has a value for each column.

```
In [8]:  titanic_df.count()

Out[8]:  pclass       1309
         survived     1309
         name         1309
         sex          1309
         age          1046
         sibsp        1309
         parch        1309
         ticket       1309
         fare         1308
         cabin         295
         embarked     1307
         boat          486
         body          121
```

dataclean.ipynb hosted with ❤ by **GitHub** **view raw**

Most of the rows are missing values for “boat” and “cabin”, so we will remove these columns from the data frame. A large number of rows are also missing the “home.dest” field; here we fill the missing values with “NA”. A significant number of rows are also missing an age value. We have seen above that age could have a significant effect on survival chances, so we will have to drop all of rows that are missing an age value. When we run the count command again, we can see that all remaining columns now contain the same number of values.

Now we need to format the remaining data in a way that our machine learning algorithms will accept.

```
In [13]: def preprocess_titanic_df(df):
             processed_df = df.copy()
             le = preprocessing.LabelEncoder()
             processed_df.sex = le.fit_transform(processed_df.sex)
             processed_df.embarked = le.fit_transform(processed_df.embarked)
             processed_df =
         processed_df.drop(['name','ticket','home.dest'],axis=1)
             return processed_df

In [14]: processed_df = preprocess_titanic_df(titanic_df)
```

346 Shares

mlsetup.ipynb hosted with ❤ by **GitHub** **view raw**

103

The “sex” and “embarked” fields are both string values that correspond to categories (i.e “Male” and “Female”) so we will ru each through a preprocessor. This preprocessor will convert these strings into integer keys, making it easier for the classification algorithms to find patterns. For instance, “Female” and “Male” will be converted to 0 and 1 respectively. The “name”, “ticket”, and “home.dest” columns consist of non-categorical string values. These are difficult to use in a classification algorithm, so we will drop them from the data set.

93

17

```
In [15]: X = processed_df.drop(['survived'], axis=1).values
         y = processed_df['survived'].values

In [16]: X_train, X_test, y_train, y_test = cross_validation.train_test_spli
         t(X,y,test_size=0.2)
```

xysplit.ipynb hosted with ❤ by **GitHub** **view raw**

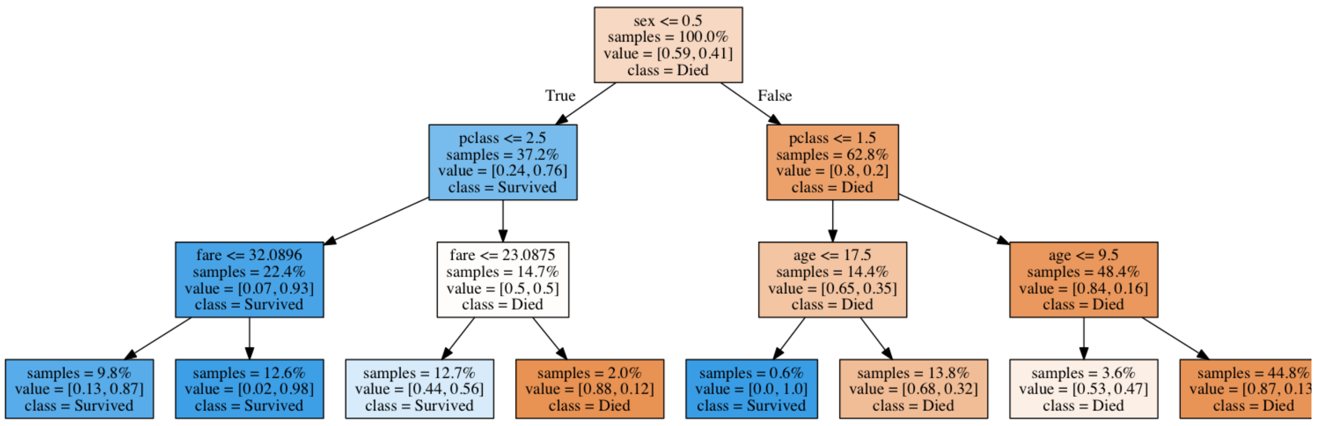
Next, we separate the data set into two arrays: “X” containing all of the values for each row besides “survived”, and “y” containing only the “survived” value for that row. The classification algorithms will compare the attribute values of “X” to the corresponding values of “y” to detect patterns in how different attributes values tend to affect the survival of a passenger.

Finally, we break the “X” and “y” array into two parts each — a training set and a testing set. We will feed the training set into the classification algorithm to form a trained model. Once the model is formed, we will use it to classify the testing set, allowing us to determine the accuracy of the model. Here we have have made a 20/80 split, such that 80% of the dataset will be used for training and 20% will be used for testing.

Classification – The Fun Part

We will start off with a simple decision tree classifier. A decision tree examines one variable at a time, and splits into one of two branches based on the result of that value, at which point it does the same for the next variable. A fantastic visual explanation of how decision trees work can be found here: http://www.r2d3.us/visual-intro-to-machine-learning-part-1/.

This is what a trained decision tree for the Titanic dataset looks like, if we set the maximum number of levels to 3:



346
Shares
103
93
17

The tree first splits by sex, and then by class, since it has learned during the training phase that these are the two most important features for determining survival. The dark blue boxes indicate passengers who are likely to survive, and the da orange boxes represent passengers who are almost certainly doomed. Interestingly, after splitting by class, the main deciding factor determining the survival of women is the ticket fare that they paid, while the deciding factor for men is their age (with children being much more likely to survive).

To create this tree, we first initialize an instance of an untrained decision tree classifier. (Here we will set the maximum depth of the tree to 10). Next we “fit” this classifier to our training set, enabling it to learn about how different factors affec the survivability of a passenger. Now that the decision tree is ready, we can “score” it using our test data to determine how accurate it is.

```
In [17]: clf_dt = tree.DecisionTreeClassifier(max_depth=10)

In [18]: clf_dt.fit (X_train, y_train)
         clf_dt.score (X_test, y_test)

Out[18]: 0.78947368421052633
```

mltest1.ipynb hosted with ❤ by **GitHub** **view raw**

The resulting reading, 0.7703, means that the model correctly predicted the survival of 77% of the test set. Not bad for our first model!

If you are being an attentive, skeptical reader (as you should be), you might be thinking that the accuracy of the model could vary depending on which rows were selected for the training and test sets. We will get around this problem by using a shuffle validator.

```
In [19]: shuffle_validator = cross_validation.ShuffleSplit(len(X),
         n_iter=20, test_size=0.2, random_state=0)
         def test_classifier(clf):
             scores = cross_validation.cross_val_score(clf, X, y, cv=shuffle
         _validator)
             print("Accuracy: %0.4f (+/- %0.2f)" % (scores.mean(), scores.st
         d()))

In [20]: test_classifier(clf_dt)

         Accuracy: 0.7742 (+/- 0.02)
```

cvs.ipynb hosted with ❤ by **GitHub** **view raw**

346
Shares
103
93
17

This shuffle validator applies the same random 20:80 split as before, but this time it generates 20 unique permutations of this split. By passing this shuffle validator as a parameter to the “cross_val_score” function, we can score our classifier against each of the different splits, and compute the average accuracy and standard deviation from the results.

The result shows that our decision tree classifier has an overall accuracy of 77.34%, although it can go up to 80% and down to 75% depending on the training/test split. Using scikit-learn, we can easily test other machine learning algorithms using the exact same syntax.

```
In [21]: clf_rf = ske.RandomForestClassifier(n_estimators=50)
         test_classifier(clf_rf)

         Accuracy: 0.7837 (+/- 0.02)

In [22]: clf_gb = ske.GradientBoostingClassifier(n_estimators=50)
         test_classifier(clf_gb)

         Accuracy: 0.8201 (+/- 0.02)

In [23]: eclf = ske.VotingClassifier([('dt', clf_dt), ('rf', clf_rf), ('gb',
          clf_gb)])
         test_classifier(eclf)

         Accuracy: 0.8036 (+/- 0.02)
```

oca.ipynb hosted with ❤ by GitHub view raw

The “Random Forest” classification algorithm will create a multitude of (generally very poor) trees for the data set using different random subsets of the input variables, and will return whichever prediction was returned by the most trees. This helps to avoid “overfitting”, a problem that occurs when a model is so tightly fitted to arbitrary correlations in the training data that it performs poorly on test data.

The “Gradient Boosting” classifier will generate many weak, shallow prediction trees and will combine, or “boost”, them into a strong model. This model performs very well on our data set, but has the drawback of being relatively slow and difficult to optimize, as the model construction happens sequentially so it cannot be parallelized.

A “Voting” classifier can be used to apply multiple conceptually divergent classification models to the same data set and will return the majority vote from all of the classifiers. For instance, if the gradient boosting classifier predicts that a passenger will not survive, but the decision tree and random forest classifiers predict that they will live, the voting classifier will chose the latter.

This has been a very brief and non-technical overview of each technique, so I encourage you to learn more about the mathematical implementations of all of these algorithms to obtain a deeper understanding of their relative strengths and weaknesses. Many more classification algorithms are available “out-of-the-box” in scikit-learn and can be explored here: http://scikit-learn.org/stable/modules/ensemble .

Computational Brains — An Introduction to Deep Neural Networks

Neural networks are a rapidly developing paradigm for information processing based loosely on how neurons in the brain processes information. A neural network consists of multiple layers of nodes, where each node performs a unit of computation and passes the result onto the next node. Multiple nodes can pass inputs to a single node and vice versa.

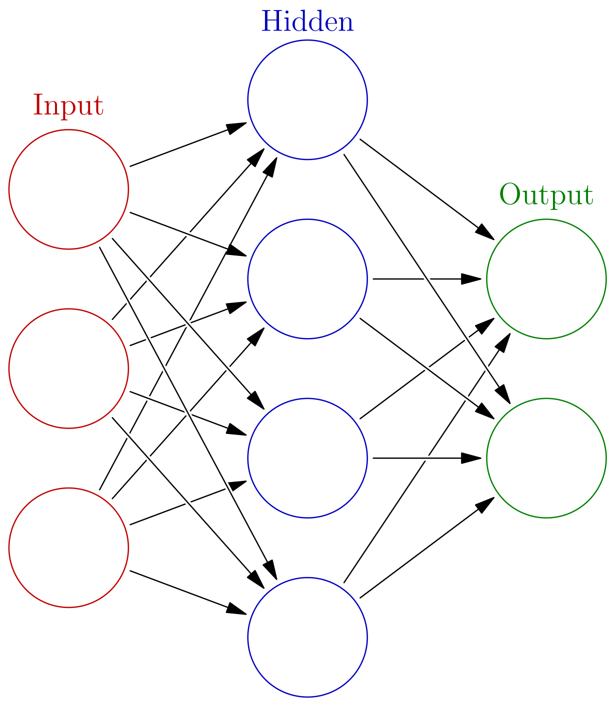
The neural network also contains a set of weights, which can be refined over time as the network learns from sample data The weights are used to describe and refine the connection strengths between nodes. For instance, in our Titanic data set, node connections transmitting the passenger sex and class will likely be weighted very heavily, since these are important for determining the survival of a passenger.

346
Shares

103

93

17



A Deep Neural Network (DNN) is a neural network that works not just by passing data between nodes, but by passing data between *layers* of nodes. Each layer of nodes is able to aggregate and recombine the outputs from the previous layer, allowing the network to gradually piece together and make sense of unstructured data (such as an image). Such networks can also be heavily optimized due to their modular nature, allowing the operations of each node layer to be parallelized en masse across multiple CPUs and even GPUs.

We have barely begun to skim the surface of explaining neural networks. For a more in depth explanation of the inner workings of DNNs, this is a good resource: http://deeplearning4j.org/neuralnet-overview.html.

This awesome tool allows you to visualize and modify an active deep neural network: http://playground.tensorflow.org.

The major advantage of neural networks over traditional machine learning techniques is their ability to find patterns in unstructured data (such as images or natural language). Training a deep neural network on the Titanic data set is total overkill, but it’s a cool technology to work with, so we’re going to do it anyway.

An emerging powerhouse in programing neural networks is an open source library from Google called TensorFlow. This library is the foundation for many of the most recent advances in machine learning, such as being used to train computer programs to create unique works of music and visual art (https://magenta.tensorflow.org/welcome-to-magenta). The syntax for using TensorFlow is somewhat abstract, but there is a wrappercalled “skflow” in the TensorFlow package that allows us to build deep neural networks using the now-familiar scikit-learn syntax.

346
Shares

103

93

17

```
In [24]: tf_clf_dnn = skflow.TensorFlowDNNClassifier(hidden_units=[20, 40, 2
         0], n_classes=2, batch_size=256, steps=1000, learning_rate=0.05)
         tf_clf_dnn.fit(X_train, y_train)
         tf_clf_dnn.score(X_test, y_test)

         Step #100, epoch #25, avg. train loss: 0.65958
         Step #200, epoch #50, avg. train loss: 0.61890
         Step #300, epoch #75, avg. train loss: 0.60432
         Step #400, epoch #100, avg. train loss: 0.57985
         Step #500, epoch #125, avg. train loss: 0.55417
         Step #600, epoch #150, avg. train loss: 0.52275
         Step #700, epoch #175, avg. train loss: 0.50493
         Step #800, epoch #200, avg. train loss: 0.48512
         Step #900, epoch #225, avg. train loss: 0.47301
         Step #1000, epoch #250, avg. train loss: 0.46723
```

tensorflow.ipynb hosted with ❤ by **GitHub** **view raw**

Above, we have written the code to build a deep neural network classifier. The "hidden units" of the classifier represent the neural layers we described earlier, with the corresponding numbers representing the size of each layer.

```
In [25]: def custom_model(X, y):
             layers = skflow.ops.dnn(X, [20, 40, 20], tf.tanh)
             return skflow.models.logistic_regression(layers, y)

In [26]: tf_clf_c = skflow.TensorFlowEstimator(model_fn=custom_model, n_clas
         ses=2, batch_size=256, steps=1000, learning_rate=0.05)
         tf_clf_c.fit(X_train, y_train)
         metrics.accuracy_score(y_test, tf_clf_c.predict(X_test))

         Step #100, epoch #25, avg. train loss: 0.61585
         Step #200, epoch #50, avg. train loss: 0.59296
         Step #300, epoch #75, avg. train loss: 0.56212
         Step #400, epoch #100, avg. train loss: 0.53000
         Step #500, epoch #125, avg. train loss: 0.51094
         Step #600, epoch #150, avg. train loss: 0.48331
```

tf2.ipynb hosted with ❤ by **GitHub** **view raw**

346
Shares

We can also define our own training model to pass to the TensorFlow estimator function (as seen above). Our defined mod
is very basic. For more advanced examples of how to work within this syntax, see the skflow documentation here: https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/skflow.

103

Despite the increased power and lengthier runtime of these neural network models, you will notice that the accuracy is sti
about the same as what we achieved using more traditional tree-based methods. The main advantage of neural networks
unsupervised learning of unstructured data — doesn't necessarily lend itself well to our Titanic dataset, so this is not too surprising.

93

I still, however, think that running the passenger data of a 104-year-old shipwreck through a cutting-edge deep neural network is pretty cool.

17

These Are Not Just Data Points. They're People.

Given that the accuracy for all of our models is maxing out around 80%, it will be interesting to look at specific passengers for whom these classification algorithms are incorrect.

```
In [27]: passengers_set_1 = titanic_df[titanic_df.pclass == 1].iloc[:20,:].copy()
         passengers_set_2 = titanic_df[titanic_df.pclass == 2].iloc[:20,:].copy()
         passengers_set_3 = titanic_df[titanic_df.pclass == 3].iloc[:20,:].copy()
         passenger_set = pd.concat([passengers_set_1,passengers_set_2,passengers_set_3])
         testing_set = preprocess_titanic_df(passenger_set)

In [28]: training_set = pd.concat([titanic_df, passenger_set]).drop_duplicates(keep=False)
         training_set = preprocess_titanic_df(training_set)
```

demonstration_setup.ipynb hosted with ❤ by **GitHub** **view raw**

The above code forms a test data set of the first 20 listed passengers for each class, and trains a deep neural network against the remaining data.

Once the model is trained we can use it to predict the survival of passengers in the test data set, and compare these to the known survival of each passenger using the original dataset.

```
In [31]: prediction = tf_clf_dnn.predict(X_test)
         passenger_set[passenger_set.survived != prediction]
```

Out[31]:

	pclass	survived	name	sex	age	sibsp	parch	ticket	f
2	1	0	Allison, Miss. Helen Loraine	female	2.0	1	2	113781	
3	1	0	Allison, Mr. Hudson Joshua Creighton	male	30.0	1	2	113781	
			Allison, Mrs.						

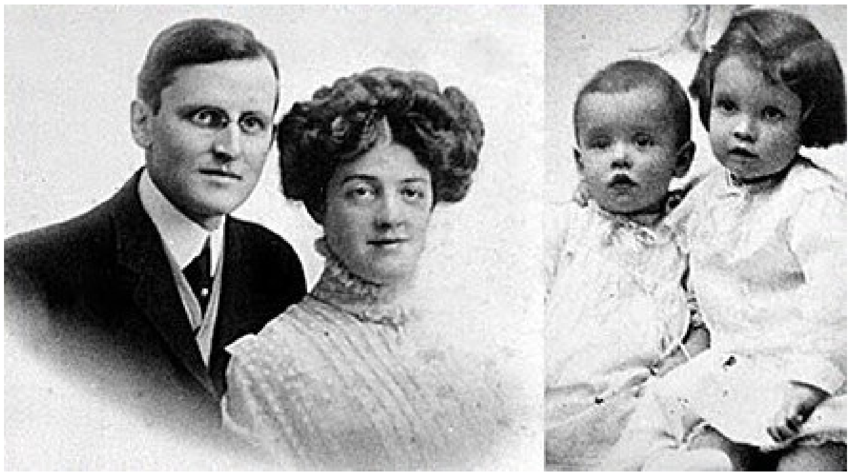
dem_analysis.ipynb hosted with ❤ by **GitHub** **view raw**

346 Shares 103 93 17

The above table shows all of the passengers in our test data set whose survival (or lack thereof) was incorrectly classified b the neural network model.

Sometimes when you are dealing the data sets like this, the human side of the story can get lost beneath the complicated math and statistical analysis. By examining passengers for whom our classification model was incorrect, we can begin to uncover some of the most fascinating, and sometimes tragic, stories of humans defying the odds.

For instance, the first three incorrectly classified passengers are all members of the Allison family, who perished even though the model predicted that they would survive. These first class passengers were very wealthy, as can be evidenced b their far-above-average ticket prices. For Betsy (25) and Loraine (2) in particular, not surviving is very surprising, considerir that we found earlier that over 96% of first class women lived through the disaster.



From left to right: Hudson (30), Bess (25), Trevor (11 months), and Loraine Allison (2)

So what happened? A surprising amount of information on each Titanic passenger is available online; it turns out that the Allison family was unable to find their youngest son Trevor and was unwilling to evacuate the ship without him. Tragically, Trevor was already safe in a lifeboat with his nurse and was the only member of the Allison family to survive the sinking.

Another interesting misclassification is John Jacob Astor, who perished in the disaster even though the model predicted he would survive. Astor was the wealthiest person on the Titanic, an impressive feat on a ship full of multimillionaire industrialists, railroad tycoons, and aristocrats. Given his immense wealth and influence, which the model may have deduced from his ticket fare (valued at over $35,000 in 2016), it seems likely that he would have been among of the 35% of men in first class to survive. However, this was not the case: although his pregnant wife survived, John Jacob Astor's body was recovered a week later, along with a gold watch, a diamond ring with three stones, and no less than $92,481 (2016 value) in cash.

John Jacob Astor IV

346
Shares

103

93

17

Olaus Jorgensen Abelseth

On the other end of the spectrum is Olaus Jorgensen Abelseth, a 25-year-old Norwegian sailor. Abelseth, as a man in 3rd class, was not expected to survive by our classifier. Once the ship sank, however, he was able to stay alive by swimming for 20 minutes in the frigid North Atlantic water before joining other survivors on a waterlogged collapsible boat and rowing through the night. Abelseth got married three years later, settled down as a farmer in North Dakota, had 4 kids, and died in 1980 at the age of 94.

Initially I was disappointed by the accuracy of our machine learning models maxing out at about 80% for this data set. It's easy to forget that these data points each represent real people, each of whom found themselves stuck on a sinking ship without enough lifeboats. When we looked into data points for which our model was wrong, we can uncover incredible stories of human nature driving people to defy their logical fate. It is important to never lose sight of the human element when analyzing this type of data set. This principle will be especially important going forward, as machine learning is increasingly applied to human data sets by organizations such as insurance companies, big banks, and law enforcement agencies.

What next?

So there you have it — a primer for data analysis and machine learning in Python. From here, you can fine-tune the machine learning algorithms to achieve better accuracy on this data set, design your own neural networks using TensorFlow, discover more fascinating stories of passengers whose survival does not match the model, and apply all of these techniques to any other data set. (Check out this Game of Thrones dataset: https://www.kaggle.com/mylesoneill/game-of-thrones). When it comes to machine learning, the possibilities ar endless and the opportunities are titanic.

346
Shares

103

93

17

```
>>> print "hello world"
hello world
>>> print "better world"
```

WRITTEN BY

Patrick Triest

346
Shares

103 93 17

NEXT READING

OPEN DATA

Real Numbers: This state has the only open data portal in India

June 27, 2016 0 Comment Christine Garcia 2 min read

11 Comments

346
Shares

103

93

17

CLINICAL STUDIES

amazing tips on how to carry out data analysis and machine learning procedures

REPLY ↓

Pingback: Linkdump #4 | WZB Data Science Blog

NICK

For some reason when i get to decision tree section, i get many errors on string conversion. eg.

clf_dt.fit (X_train, y_train)
ValueError: could not convert string to float: D

I'm using sklearn 0.17 on python 2.7.12

Any ideas?

REPLY ↓

PATRICK TRIEST

The most likely reason for this error is that a string column in your processed dataframe is not being properly encoded, as the fitting algorithm does not work with string values (see the "Preparing The Data" section). Try running "processed_df.dtypes" on your dataframe, each of the columns should be either of the type "int64" or "float64". If any of them are of a different type, double check your pre-processing code to make sure the problem-column is being either encoded or dropped.

REPLY ↓

Pingback: This Week in Big Data Analytics (Jul 10, 2016) - Big Data Analytics Guide

BTO

@NICK there is a chance you missed some lines of code following "titanic_df.count()", make sure to scroll down and run the remaining part of that chunk. If that is not the case, then there must be another reason why you have string instead of numeric values in your training data set named X_train that is causing the problem.

REPLY ↓

Pingback: Would You Survive the Titanic? A Guide to Machine Learning in Python — Dr. Jonathan Jenkins, DBA, CSSBB, MSG

Pingback: Would You Survive the Titanic? A Guide to Machine Learning in Python | Tech News

346
Shares

103

93

17

CHRISTOPHE FAURIE

I've been told about this post by somebody who had read a post I had written on the use of MondoBrain for this same Titanic file. (see near the bottom of this page : http://www.mondobrain.com/business-cases.html)
The first concusion I drew was that rich people had fared better than poor ones and women than men. So I wondered who were the poor men who survived and who were the rich ones who died. For the latter you get : bachelors. That could explain John Jacob Astor and the norwegian : bachelors had to fend for themselves. (For the former : young kids with single parent.)
I didn't know the Titanic crew was so chivalrous… (The conclusion of my post !)
By the way: the movie seems to have been right: men have sacrificed for women.

REPLY ↓

Pingback: Would You Survive the Titanic? A Guide to Machine Learning in Python - CloudJocks

Pingback: Would You Survive the Titanic? A Guide to Machine Learning in Python – WebProfIT Consulting

Leave A Comment

COMMENT

NAME *

EMAIL *

WEBSITE

Post Comment

346
Shares

103

93

17

346
Shares

103

93

17