Machine Learning From Scratch

Collin Prather

April 25th, 2018

Machine Learning From Scratch

Collin Prather

April 25th, 2018

Make sure that everyone launches binder or colab.google first!!

Also, give them a walkthrough of the whole presentation and tell them what we'll be building and what we'll be predicting!

* we'll be talking about how to apply machine learning to problems!

Machine Learning Overview

Pull a graph of google search trends indicating how terms like "Data Science" and "Machine Learning" have blown up.
Try to form talk around hitting on the theoretical mathematical side of ML as well as the difficluties/complexities faced in Applied ML
data + algorthms = predicting the future (it's really a lot more than this – understanding context and how to frame the question (usually) from a business perspective is huge)
classification v. regression
supervised/unsupervised/reinforcement learning
when talking on reinforcement learning, mention and recommend AlphaGo documentary (it's on netflix!)
considerations/complexities in building ML models

What is Machine Learning?

Arthur Samuel:

Machine learning is "Field of study that gives computers the ability to learn without being explicitly programmed".

What is Machine Learning?

Machine Learning

Arthur Samuel:
Machine learning is “Field of study that gives computers the ability to learn without being explicitly programmed”.

Even according to the experts, the exact definifion of the field of machine learning is a bit fuzzy, but As early as 1959, Arthur Samuel quote. .

also, include Ng’s explanation from MLYearning!

What is Machine Learning

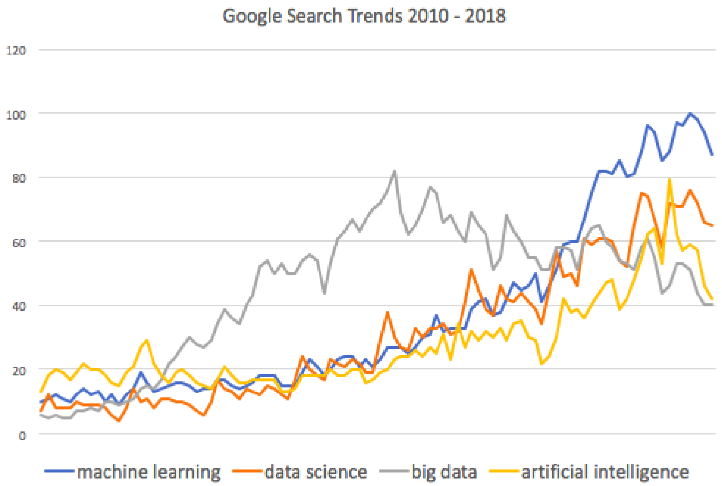
data + algorithms = predicting the future
combo of statistics, calculus, etc...

What is Machine Learning

data + algorithms = predicting the future
combo of statistics, calculus, etc...

ML techniqes can be applied to a wide range of problems in diverse industries. In fact, ML has become ubiquitous in our everyday lives * Siri/ Amazon Alexa
* Recommendation systems (amazon, netflix)
* Fraud Detection
* Disease diagnosis
* Supply Chain Optimization

According to Google...



What has caused this spike?

1. Data Availability
   - digital data
   - Iot (sensor data)
2. Computational Scale
   - Moore's Law

What has caused this spike?

1. Data Availability
   - digital data
   - Iot (sensor data)
2. Computational Scale
   - Moore's Law

The math that powers machine learning algorithms has been around for quite a few years... so what's changed?
1. Data Availability (here we should just a few examples)
2. Computational Scale (NG MLY 01 pg 10)

The rise of the big data era has given us access to astounding amounts of data. That phenomenon paired with with the exponential growth we've experienced in computational advances, has created the perfect storm for the emergence of the field of machine learning.

CITY OF
GRAND
RAPIDS

CITY OF GRAND RAPIDS

Setp 0 is to identify a problem that can be framed in such a way that it can be solved using machine learning.

Let's say that you work for the city of Grand Rapids, and you find that there are an increased number of hit and runs when the driver 1 was drinkin.

Do some research, maybe find a way to graph this?? You can do it!

Get the Data

This may look like:

- SQL query
- CSV download
- Web-scraping
- Designing experiments/surveys and collecting data yourself

Get the Data

This may look like:

- SQL query
- CSV download
- Web-scraping
- Designing experiments/surveys and collecting data yourself

In our case, we'll head to GRData.

Get the Data

This may look like:

- SQL query
- CSV download
- Web-scraping
- Designing experiments/surveys and collecting data yourself

In our case, we'll head to GRData.

Obtaining the data you'll need looks very different depending on what domain you're working in. In some instances, it can be fairly simple and straightforward, for example, In a business context, most often it will require querying some sort of internal database. Could also be downloading a csv file. In other instances, it may require a bit more creativity – For particular social research, you may need to scrape the web. In some cases, you may even need to collect some data yourself! Here are two examples:
1. You're developing a new data product at your company and are collecting data to fuel it
2. You're in public health and are working to make healthcare accessible to all residents of the greater GR area. You may need to conduct your own research to identify what may be inhibitting people from reaching healthcare.

In our case, we're lucky enough to have access to a meticulously maintained public database on the city of GR: GRData. Scroll through,

Get the Data

In our case, we'll head to GRData

In [33]:
```
crash = pd.read_csv('Data/CGR_Crash_Data.csv')
crash.head()
```

Out[33]:

	X	Y	OBJECTID	ROADSOFTID	BIKE	CITY	CRASHDATE	CRASHSEVER	CRASHTYPE	WORKZNEACT	...
0	-85.639647	42.927216	6001	929923	No	Grand Rapids	2007-02-16	Property Damage Only	Side-Swipe Same	Uncoded & Errors	...
1	-85.639487	42.927213	6002	935745	No	Grand Rapids	2007-06-22	Property Damage Only	Side-Swipe Same	Uncoded & Errors	...
2	-85.639387	42.927212	6003	926813	No	Grand Rapids	2007-01-08	Property Damage Only	Head-on	Work on Shoulder / Median	...
3	-85.639288	42.927210	6004	943813	No	Grand Rapids	2007-11-12	Property Damage Only	Side-Swipe Same	Uncoded & Errors	...
4	-85.639288	42.927210	6005	943791	No	Grand Rapids	2007-11-09	Property Damage Only	Parking	Uncoded & Errors	...

5 rows × 77 columns

Get the Data

In our case, we'll head to GRData

In [33]:
```
crash = pd.read_csv('Data/CGR_Crash_Data.csv')
crash.head()
```

Out[33]:

	X	Y	OBJECTID	ROADSOFTID	BIKE	CITY	CRASHDATE	CRASHSEVER	CRASHTYPE	WORKZNEACT	...
0	-85.639647	42.927216	6001	929923	No	Grand Rapids	2007-02-16	Property Damage Only	Side-Swipe Same	Uncoded & Errors	...
1	-85.639487	42.927213	6002	935745	No	Grand Rapids	2007-06-22	Property Damage Only	Side-Swipe Same	Uncoded & Errors	...
2	-85.639387	42.927212	6003	926813	No	Grand Rapids	2007-01-08	Property Damage Only	Head-on	Work on Shoulder / Median	...
3	-85.639288	42.927210	6004	943813	No	Grand Rapids	2007-11-12	Property Damage Only	Side-Swipe Same	Uncoded & Errors	...
4	-85.639288	42.927210	6005	943791	No	Grand Rapids	2007-11-09	Property Damage Only	Parking	Uncoded & Errors	...

5 rows × 77 columns

After downloading the csv file, we can read the file into a pandas dataframe and explore it in our Jupyter notebook.

* note something about how we'll often refer to each row as an observation and each column as a feature

Explore the Data

- Verify data
- Visualize data
- Identify patterns
- Give direction to analysis

Explore the Data

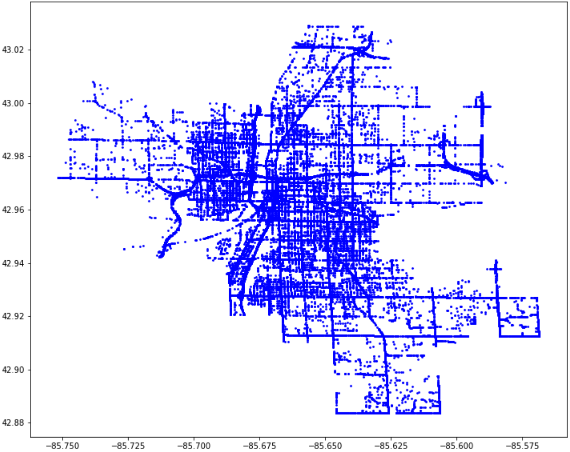
- Verify data
- Visualize data
- Identify patterns
- Give direction to analysis

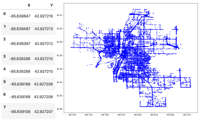
This is kind of an unstructured approach to understanding initial patterns in the data and potentially points of interest.This process isnt meant to reveal every bit of information a dataset holds, but rather give you direction in your analysis and potentially give you clues in how to process/model the data.[1] Now, if you're just emailed a csv file, this step is especially crucial, and it may take you some time to explore the data, get a feeling for what you're dealing with. If you are analyzing data that you work with day in and day out, this "exploration" process may be a bit more implicit.
The main idea hear is to build an understanding of your data. Without an appreciation for the context of the data, it's just numbers. But when you see the data in context, it's fascinating, it's a story.

More often than not, your exploration of the data leads to more questions than answers.

	X	Y
0	-85.639647	42.927216
1	-85.639487	42.927213
2	-85.639387	42.927212
3	-85.639288	42.927210
4	-85.639288	42.927210
5	-85.639188	42.927208
6	-85.639168	42.927208
7	-85.639108	42.927207

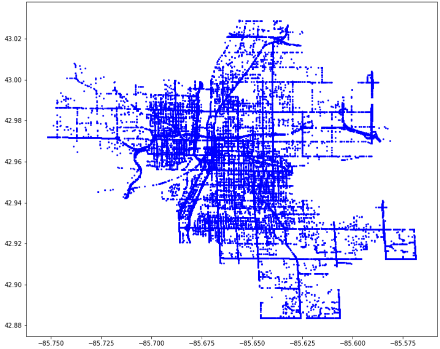
	X	Y
0	-85.639647	42.927216
1	-85.639487	42.927213
2	-85.639387	42.927212
3	-85.639288	42.927210
4	-85.639288	42.927210
5	-85.639188	42.927208
6	-85.639168	42.927208
7	-85.639108	42.927207

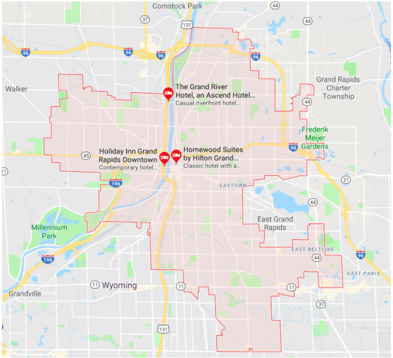


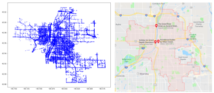


	X	Y
0	-85.639647	42.927216
1	-85.639487	42.927213
2	-85.639387	42.927212
3	-85.639288	42.927210
4	-85.639288	42.927210
5	-85.639188	42.927208
6	-85.639168	42.927208
7	-85.639108	42.927207

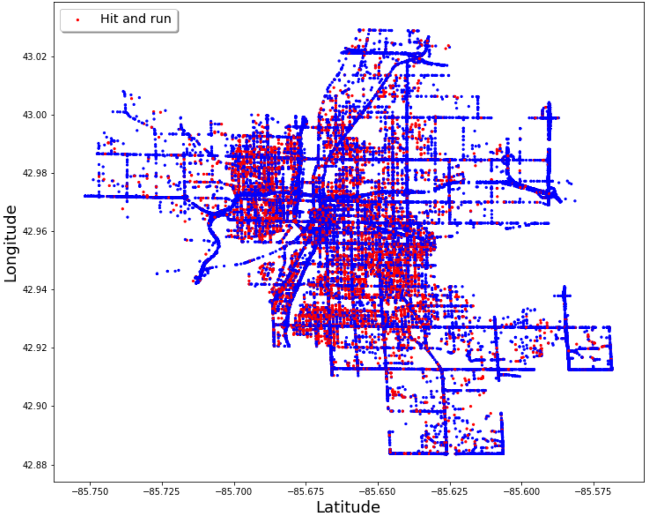
With our dataset, on car crashes, a logical place to begin would be the first two columns, containing latitudes and longitudes of each crash. This is just a snapshot of the data on the left side, and on the right, each dot represents a car crash.

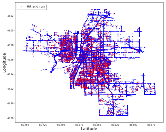

43.02
43.00
42.98
42.96
42.94
42.92
42.90
42.88
−85.750
−85.725
−85.700
−85.675
−85.650
−85.625
−85.600
−85.575


Comstock Park
Walker
Grand Rapids Charter Township
The Grand River Hotel, an Ascend Hotel...
Casual riverfront hotel...
Frederik Meijer Gardens
Holiday Inn Grand Rapids Downtown
Contemporary hotel...
Homewood Suites by Hilton Grand...
Classic hotel with a...
EASTOWN
East Grand Rapids
Millennium Park
EAST BELTLINE
EAST PARIS
Wyoming
Grandville



Now, this is pretty telling about our data, remember, there is nearly 73,000 crashes recorded, and if we juxtapose this plot with the city of GR, we actually see that the plot of crashes outline the city boundaries!


Hit and run
43.02
43.00
42.98
42.96
42.94
42.92
42.90
42.88
Longitude
−85.750
−85.725
−85.700
−85.675
−85.650
−85.625
−85.600
−85.575
Latitude



We may be interested in hit and runs...

I should add a caption or title to this plot

Check the variable's distribution

In [34]:
```
crash = pd.read_csv('Data/CGR_Crash_Data.csv')
crash.head(3)
```

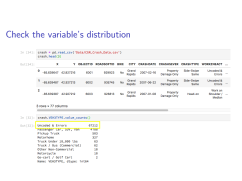
Out[34]:

	X	Y	OBJECTID	ROADSOFTID	BIKE	CITY	CRASHDATE	CRASHSEVER	CRASHTYPE	WORKZNEACT	...
0	-85.639647	42.927216	6001	929923	No	Grand Rapids	2007-02-16	Property Damage Only	Side-Swipe Same	Uncoded & Errors	...
1	-85.639487	42.927213	6002	935745	No	Grand Rapids	2007-06-22	Property Damage Only	Side-Swipe Same	Uncoded & Errors	...
2	-85.639387	42.927212	6003	926813	No	Grand Rapids	2007-01-08	Property Damage Only	Head-on	Work on Shoulder / Median	...

3 rows × 77 columns

In [32]:
```
crash.VEH3TYPE.value_counts()
```

Out[32]:
```
Uncoded & Errors             67212
Passenger Car, SUV, Van       4788
Pickup Truck                   503
Motorhome                      327
Truck Under 10,000 lbs          63
Truck / Bus (Commercial)        62
Other Non-Commercial            10
Motorcycle                      10
Go-cart / Golf Cart              2
Name: VEH3TYPE, dtype: int64
```



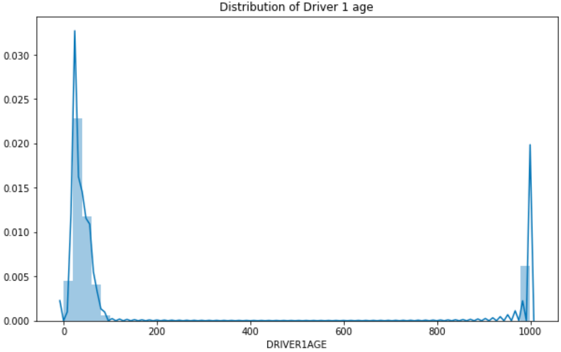
It's also often very helpful to check the distribution of the different variables. For example, our dataset contains many characteristics about what it defines as "DRIVER1", "DRIVER2", "DRIVER3". When I first came across that, I was impressed, like that's some seriously accurate data! But upon further examination, we find that many of the "DRIVER2" and "DRIVER3" columns are empty or contain errors. (This makes sense... not all crashes involve 3 driver!)

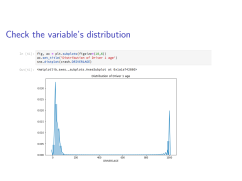
When we check the counts of the different values found in the "VEH3" column, we see that over 67,000 of them are errors! Now, this doesn't mean that the column is useless, but in terms of building a predictive model, this column probably won't be much help, so as we'll see in the data processing step, we'll end up dropping it.

Check the variable's distribution

```
In [41]: fig, ax = plt.subplots(figsize=(10,6))
         ax.set_title('Distribution of Driver 1 age')
         sns.distplot(crash.DRIVER1AGE)

Out[41]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1a742080>
```



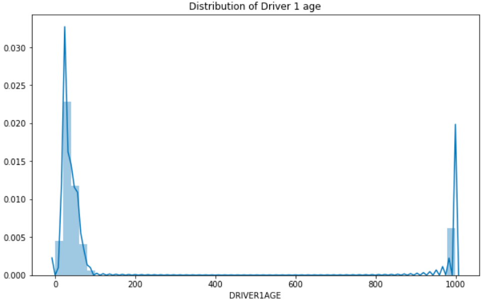


Continuing on, here we check the distribution of the age's of all the "DRIVER1"s recorded in the dataset. In my head, I would think that age would be an important feature in car crashes. And so we check it out, and i appears that there are a ton of instances bunched up between 1 and 100... that makes sense... then there is also a decent cluster of observations around 1000 years old... that does not make sense.

Check the variable's distribution

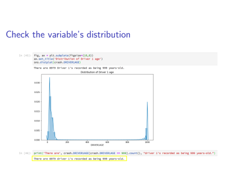
In [45]:
```
fig, ax = plt.subplots(figsize=(10,6))
ax.set_title('Distribution of Driver 1 age')
sns.distplot(crash.DRIVER1AGE)
```

There are 8979 Driver 1's recorded as being 999 years-old.



In [46]:
```
print('There are', crash.DRIVER1AGE[crash.DRIVER1AGE == 999].count(), "driver 1's recorded as being 999 years-old.")
```

There are 8979 driver 1's recorded as being 999 years-old.



Nearly 9,000 driver 1's are recorded as being 999 years-old.. That is a good thing to know before trying to build a predictive model, as it serously skews the data. We'll address that in our data processing stage.

Next, we move onto data preparation. This is the stage where we make the final manipulations to our data before feeding it into our ML algorithm. Now, you could make an argument that preparing the data is the most important part of the machine learning workflow. After all it's the data that fuels the algorithm, garbage in, garbage out! It's been shown time and time again that more/bigger data beats a better algorithm everytime. Though it usually appears to be straightforward, this step can often reqiure a lot of creativity.

Data Selection

Use as minimal features as possible

1. Computationally efficient
2. Easier to interpret
3. Simpler is better

Data Selection

Use as minimal features as possible

1. Computationally efficient
2. Easier to interpret
3. Simpler is better

```
In [8]: crash = pd.read_csv('Data/CGR_Crash_Data.csv')
        crash = crash[['X', 'Y', 'CRASHSEVER', 'DRIVER1AGE','DRIVER1SEX',
                       'EMRGVEH', 'HITANDRUN', 'SPEEDLIMIT', 'HOUR','MOTORCYCLE',
                       'NUMOFINJ', 'D1COND', 'D1DRINKIN']]
        crash.columns

Out[8]: Index(['X', 'Y', 'CRASHSEVER', 'DRIVER1AGE', 'DRIVER1SEX', 'EMRGVEH',
               'HITANDRUN', 'SPEEDLIMIT', 'HOUR', 'MOTORCYCLE', 'NUMOFINJ', 'D1COND',
               'D1DRINKIN'],
              dtype='object')
```

Data Selection

Use as minimal features as possible

1. Computationally efficient
2. Easier to interpret
3. Simpler is better

```
In [8]: crash = pd.read_csv('Data/CGR_Crash_Data.csv')
        crash = crash[['X', 'Y', 'CRASHSEVER', 'DRIVER1AGE','DRIVER1SEX',
                       'EMRGVEH', 'HITANDRUN', 'SPEEDLIMIT', 'HOUR','MOTORCYCLE',
                       'NUMOFINJ', 'D1COND', 'D1DRINKIN']]
        crash.columns

Out[8]: Index(['X', 'Y', 'CRASHSEVER', 'DRIVER1AGE', 'DRIVER1SEX', 'EMRGVEH',
               'HITANDRUN', 'SPEEDLIMIT', 'HOUR', 'MOTORCYCLE', 'NUMOFINJ', 'D1COND',
               'D1DRINKIN'],
              dtype='object')
```

The first step in preparing the data is simply choosing which features (you can think of as columns in the spreadsheet) you'll want to use in your model. As seen in our EDA some columns have a lot of missing data, and we'll drop them entirely.
It's generally regarded as best practice to use as minimal amount of features as possible, such that your predictive model still predicts as accurately as you need it to.
* computationally efficient
* more easily interpretable
* simpler is better
So how do we actually determine which features to use?, we can use various statistical tests, and even some algorithms to determine which features are going to be most relevant to predicting our target variable (which for us is whether or not the driver who caused the crash was drinking).We won't go into detail here.
When you are first beginning to iterate through different ml models, it is okay to kind of eyeball it, or use what you know about the context of the

Feature Engineering

'Coming up with features is difficult, time-consuming, requires expert knowledge. Applied machine learning is basically feature engineering.' Prof. Andrew Ng

'At the end of the day, some machine learning projects succeed and some fail. What makes the difference? Easily the most important factor is the features used.' Prof. Pedro Domingos

Feature Engineering

'Coming up with features is difficult, time-consuming, requires expert knowledge. Applied machine learning is basically feature engineering.' Prof. Andrew Ng

'At the end of the day, some machine learning projects succeed and some fail. What makes the difference? Easily the most important factor is the features used.' Prof. Pedro Domingos

So now we have our 13 features that we've chosen to use, and we could just send these raw features into our algorithm, and it may perform well, but it may not. It's importatnt to remember that our ultimate goal is to build a predictive model that can make accurate predictions on new/unseen observations. When all the data comes in from a car crash that just happened, we want to be able to accurately predict whether or not it was a drunk driver that caused it.

Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models, resulting in improved model accuracy on unseen data. - Jason Brownlee [2]

We acknowledge, however, that the data that's being recorded (from those UD10 reports) isn't necessarily guaranteed to accurately represent reality. Which is an important thing if we want to build accurate, stable predictive models.

The examples that we show here are absolutely not exhaustive when it

Feature Engineering: *transforming "hour" variable*

$$\begin{bmatrix} 12 \\ 2 \\ 4 \\ 18 \\ 19 \\ 6 \\ \vdots \end{bmatrix}$$

Feature Engineering: *transforming "hour" variable*

$$\begin{bmatrix} 12 \\ 2 \\ 4 \\ 18 \\ 19 \\ 6 \\ \vdots \end{bmatrix} \implies \quad f(x) = \frac{2 \cdot \pi \cdot (\text{hour})}{24}$$

Feature Engineering: *transforming "hour" variable*

$$\begin{bmatrix} 12 \\ 2 \\ 4 \\ 18 \\ 19 \\ 6 \\ \vdots \end{bmatrix} \Longrightarrow \quad f(x) = \frac{2 \cdot \pi \cdot (\text{hour})}{24} \quad \Longrightarrow \begin{bmatrix} 3.14 \\ 0.52 \\ 1.03 \\ 4.71 \\ 4.98 \\ 1.57 \\ \vdots \end{bmatrix}$$

Feature Engineering: *transforming "hour" variable*

$$\begin{bmatrix} 3.14 \\ 0.52 \\ 1.03 \\ 4.71 \\ 4.98 \\ 1.57 \\ \vdots \end{bmatrix} \implies \underbrace{\begin{bmatrix} 0.00 \\ 0.47 \\ 0.86 \\ -0.99 \\ -0.97 \\ 1.0 \\ \vdots \end{bmatrix}}_{sin(f(x))}, \underbrace{\begin{bmatrix} -1.0 \\ 0.87 \\ 0.51 \\ -0.002 \\ -0.26 \\ 0.001 \\ \vdots \end{bmatrix}}_{cos(f(x))}$$

Feature Engineering: *transforming "hour" variable*

$$\begin{bmatrix} 3.14 \\ 0.52 \\ 1.03 \\ 4.71 \\ 4.98 \\ 1.57 \\ \vdots \end{bmatrix} \implies \underbrace{\begin{bmatrix} 0.00 \\ 0.47 \\ 0.86 \\ -0.99 \\ -0.97 \\ 1.0 \\ \vdots \end{bmatrix}}_{sin(f(x))}, \underbrace{\begin{bmatrix} -1.0 \\ 0.87 \\ 0.51 \\ -0.002 \\ -0.26 \\ 0.001 \\ \vdots \end{bmatrix}}_{cos(f(x))}$$

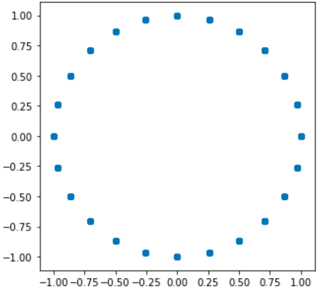
Okay, so we have this new vector.. it doesn't look too helpful at the moment. But now we map that vector to two other vectors

Feature Engineering: *transforming "hour" variable*

```
In [193]: crash['HOUR_X']=np.sin(2. * np.pi * crash.HOUR / 24.)
          crash['HOUR_Y']=np.cos(2. * np.pi * crash.HOUR / 24.)

In [194]: # Hence, the time of day is now cyclic (just as in reality)
          plt.figure(figsize = (5,5))
          plt.scatter(crash.HOUR_X, crash.HOUR_Y)

Out[194]: <matplotlib.collections.PathCollection at 0x1a0daeca20>
```

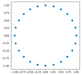


Feature Engineering: *transforming "hour" variable*

```
In [193]: crash['HOUR_X']=np.sin(2. * np.pi * crash.HOUR / 24.)
          crash['HOUR_Y']=np.cos(2. * np.pi * crash.HOUR / 24.)

In [194]: # Hence, the time of day is now cyclic (just as in reality)
          plt.figure(figsize = (5,5))
          plt.scatter(crash.HOUR_X, crash.HOUR_Y)

Out[194]: <matplotlib.collections.PathCollection at 0x1a0daeca20>
```

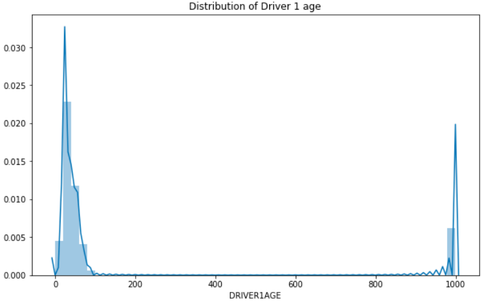


transferring that all into code...

Feature Engineering: *imputing missing ages*

In [45]:
```
fig, ax = plt.subplots(figsize=(10,6))
ax.set_title('Distribution of Driver 1 age')
sns.distplot(crash.DRIVER1AGE)
```

```
There are 8979 Driver 1's recorded as being 999 years-old.
```



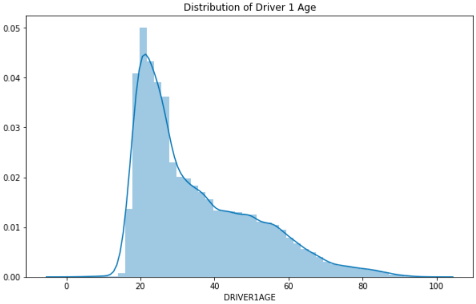
In [46]:
```
print('There are', crash.DRIVER1AGE[crash.DRIVER1AGE == 999].count(), "driver 1's recorded as being 999 years-old.")
```

```
There are 8979 driver 1's recorded as being 999 years-old.
```

Feature Engineering: *imputing missing ages*

In [90]:
```
# This age distribution looks much better!
fig, ax = plt.subplots(figsize=(10,6))
ax.set_title('Distribution of Driver 1 Age')
sns.distplot(crash['DRIVER1AGE'])
```

Out[90]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a21b50240>
```



Now, earlier when we were exploring our data and checking the distribution of the ages, we found that there are about 9,000 erraneous ages.

When you have erraneous/outright missing data you have a couple different options.

1. drop them entirely
2. impute mean/median (make this choice depending on distribution of data)

For most cases, one of those two options will work just fine. In our case, with nearly 9,000 errors, I'm not sure any of those would make sense, so we'll opt for a bit more involved technique. (The technical term is "Multiple Imputation by Chained Equations", affectionately known as MICE). What this means is that we will fit a linear regression model to our data to predict what the missing ages could be. We can think of these as making an educated guess.

Data Processing

Two general types of data to deal with:

- Numerical variables (Quantitative)
  - Driver 1 age, number of injuries, etc
- Categorical variables (Qualitative)
  - Hit and run, motorcycle involved, etc

Data Processing

Two general types of data to deal with:

- Numerical variables (Quantitative)
  - Driver 1 age, number of injuries, etc
- Categorical variables (Qualitative)
  - Hit and run, motorcycle involved, etc

Now, we move onto processing the data that we've selcted. the data processing stage naturally diverges into two substeps: dealing with numerical variables, and dealing with categorical variables.

It's important to note that some of the preprocessing steps we'll talk about here may actually be necessary for you to do to get the data in a format where you can explore it.

Numerical variables are quantitative – it's something you can measure. In our case, some numerical variables are Driver1 age, and number of injuries.

Categorical variables are qualitative, in our case, we have some binary categorical variables: like whether or not the crash was a hit and run, whether or not a mototcycle was involved. It's either one or the other. Variables can of course have multiple categories, for example, which car insurance provider you choose: (Nationwide, Statefarm, BlueCross BlueShield) There is obviously a lot of them, but all drivers fit into one of those categories... or at least they should.

There is some grey area between the two – maybe mention speedlimits or

Data Processing: *numerical variables*

```
In [91]: crash[['X', 'Y', 'DRIVER1AGE', 'NUMOFINJ']].head()
```

Out[91]:

	X	Y	DRIVER1AGE	NUMOFINJ
0	-85.639647	42.927216	62.0	0
1	-85.639487	42.927213	31.0	0
2	-85.639387	42.927212	22.0	0
3	-85.639288	42.927210	30.0	0
4	-85.639288	42.927210	44.0	0

Data Processing: *numerical variables*

In [96]:
```
crash[['X', 'Y', 'DRIVER1AGE', 'NUMOFINJ']].head()
```

Out[96]:

	X	Y	DRIVER1AGE	NUMOFINJ
0	0.406318	-0.996140	1.685157	-0.416816
1	0.411006	-0.996237	-0.269018	-0.416816
2	0.413936	-0.996298	-0.836360	-0.416816
3	0.416866	-0.996358	-0.332056	-0.416816
4	0.416866	-0.996358	0.550474	-0.416816

Data Processing: *numerical variables*

In [96]:
```
crash[['X', 'Y', 'DRIVER1AGE', 'NUMOFINJ']].head()
```

Out[96]:

	X	Y	DRIVER1AGE	NUMOFINJ
0	0.406318	-0.996140	1.685157	-0.416816
1	0.411006	-0.996237	-0.269018	-0.416816
2	0.413936	-0.996298	-0.836360	-0.416816
3	0.416866	-0.996358	-0.332056	-0.416816
4	0.416866	-0.996358	0.550474	-0.416816

Here, we have our numerical variables. One problem that we still face with these numerical variables is that when we feed our data through the algorithm, the computer will view an increase in 1 latitude of latitude as equivalent to an increase in 1 year of Age of the driver. In actuality and increase in 1 degree of latitude could land you in an entirely different neighborhood, while 37 y/o and 38 y/o are basically exactly the same. We will rescale the numerical variables through a process called standardization. This is not necessary for all ml algorithms, but generally will not hurt if we do it.

When we standardize the variables, we rescale them so that they all have a mean of 0 and a S.D. of 1.

Data Processing: *categorical variables*

In [111]:
```
crash[['CRASHSEVER', 'DRIVER1SEX','EMRGVEH', 'HITANDRUN',
         'MOTORCYCLE', 'D1COND','D1DRINKIN']].head()
```

Out[111]:

	CRASHSEVER	DRIVER1SEX	EMRGVEH	HITANDRUN	MOTORCYCLE	D1COND	D1DRINKIN
0	Property Damage Only	F	No	Yes	No	Appeared Normal	No
1	Property Damage Only	M	No	Yes	No	Unknown	No
2	Property Damage Only	F	No	No	No	Appeared Normal	No
3	Property Damage Only	M	No	Yes	No	Appeared Normal	No
4	Property Damage Only	M	No	No	No	Appeared Normal	No

Data Processing: *categorical variables*

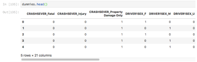
```
In [135]: dummies.head()
```

Out[135]:

	CRASHSEVER_Fatal	CRASHSEVER_Injury	CRASHSEVER_Property Damage Only	DRIVER1SEX_F	DRIVER1SEX_M	DRIVER1SEX_U
0	0	0	1	1	0	0
1	0	0	1	0	1	0
2	0	0	1	1	0	0
3	0	0	1	0	1	0
4	0	0	1	0	1	0

5 rows × 21 columns

Data Processing: *categorical variables*



In [135]: dummies.head()

Out[135]:

	CRASHSEVER_Fatal	CRASHSEVER_Injury	CRASHSEVER_Property Damage Only	DRIVER1SEX_F	DRIVER1SEX_M	DRIVER1SEX_U
0	0	0	1	1	0	0
1	0	0	1	0	1	0
2	0	0	1	1	0	0
3	0	0	1	0	1	0
4	0	0	1	0	1	0

5 rows × 21 columns

Here we have our categorical variables. In our excel spreadsheet, this looks great, nice and clean.., but if we want to squeeze this data into a model, we need to manipulate it into a format that makes sense for math. Basically, we are going to turn all of our categories into 1's and 0's, representing yes' and no's. You can think of this as transforming the data to turn everything into a yes or no question.

Was the driver sex male? No. Was the driver sex female? yes.

I've seen this called creating dummy variables, or one-hot-encoding

This is just a snapshot, you see that our columns of categorical variables grew from 7 to 21

Come up with more concrete explanation

Choosing a model.. or should i say algorithm??

	Classification	**Regression**
Supervised	• Logistic Regression • Naive-Bayes • KNN • SVM	• Linear Regression • Decision Trees • Random Forests
Unsupervised	• Apriori • Hidden Markov Model	• PCA • K-means • SVD

Choosing a model.. or should i say algorithm??

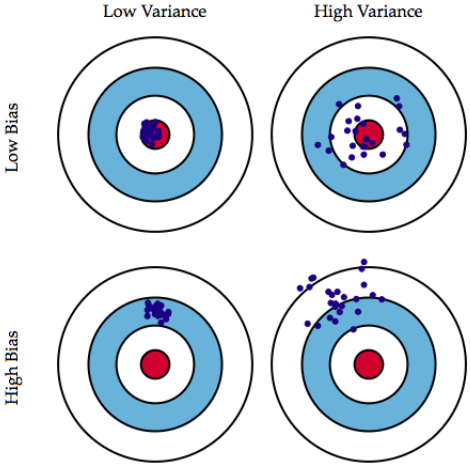
	Classification	Regression
Supervised	• Logistic Regression • Naive-Bayes • KNN • SVM	• Linear Regression • Decision Trees • Random Forests
Unsupervised	• Apriori • Hidden Markov Model	• PCA • K-means • SVD

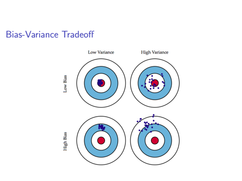
And with that, our data is ready to be fed into a predictive model! The final step is to choose which predictive model to use. There are hundreds to choose from, and trade-offs associated with each. The good news is, as we alluded to earlier there are different types of machine learning problems and each come with their own set of algorithms – so this narrows down our search considerably.

In our case, we're working on a supervised classification problem, so the upper left hand corner displays some common algorithms for problems like ours. It's very common to test a handful of them and choose the one that performs best on your dataset (or even combine some of them into an ensemble model.)

Remeber, the ultimate objective to choose a model that learns a predictive rule (which comes in the form of a equation) that can be generalized to new observations (both for classification and regression)

Bias-Variance Tradeoff





One critical aspect to take into consideration when choosing a model is called the bias-variance tradeoff

So first, I'm going to define what I mean when I talk about the terms "bias" and "variance" in the context of machine learning, then we are going to look at some simplified examples (using this dataset here) in the regression context that I think really well convey the essence of applied machine learning:

Maybe still show my graphs?

mention statistical techniques necessary to combat variance, like bagging

- https://www.sisense.com/glossary/data-exploration/
- https://towardsdatascience.com/understanding-feature-engineering-part-2-categorical-data-f54324193e63
- http://scott.fortmann-roe.com/docs/BiasVariance.html