Module 4: Machine Learning
Part 1
Feature Engineering

Agenda

- Feature Selection & Engineering
- Math free overview of classification models
- Evaluating Model Performance
- Improving model performance

Machine Learning Terms

- **Features:** The mathematical representation of the original data. The features are the columns in your data set. Since the features will be a matrix, the are often written as X.
- **Observations:** The rows of your feature set.
- **Target:** The variable that you are trying to predict.
  - Often represented as y.

Features

http://www.google.com

domain_length	vowel_count	digit_count
6	3	0

Representation of URL Knowledge

- Come up with a **representation/set of knowledge** that has **enough complexity** to accurately describe the problem for the computer
- Knowledge here does not mean hard-coded knowledge or formal set of rules
- **The computer rather uses the knowledge we provide to extract patterns and acquire own knowledge**
- We should provide knowledge about reality that has **high variance about the problem** it describes (e.g. a feature that is high when it rains and low when it's sunshine)

https://www.google.com/search?q=URL&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjcl6ut-lDUAhVEPCYKHdJGDsYQ_AUIDCgD&biw=1215&bih=652

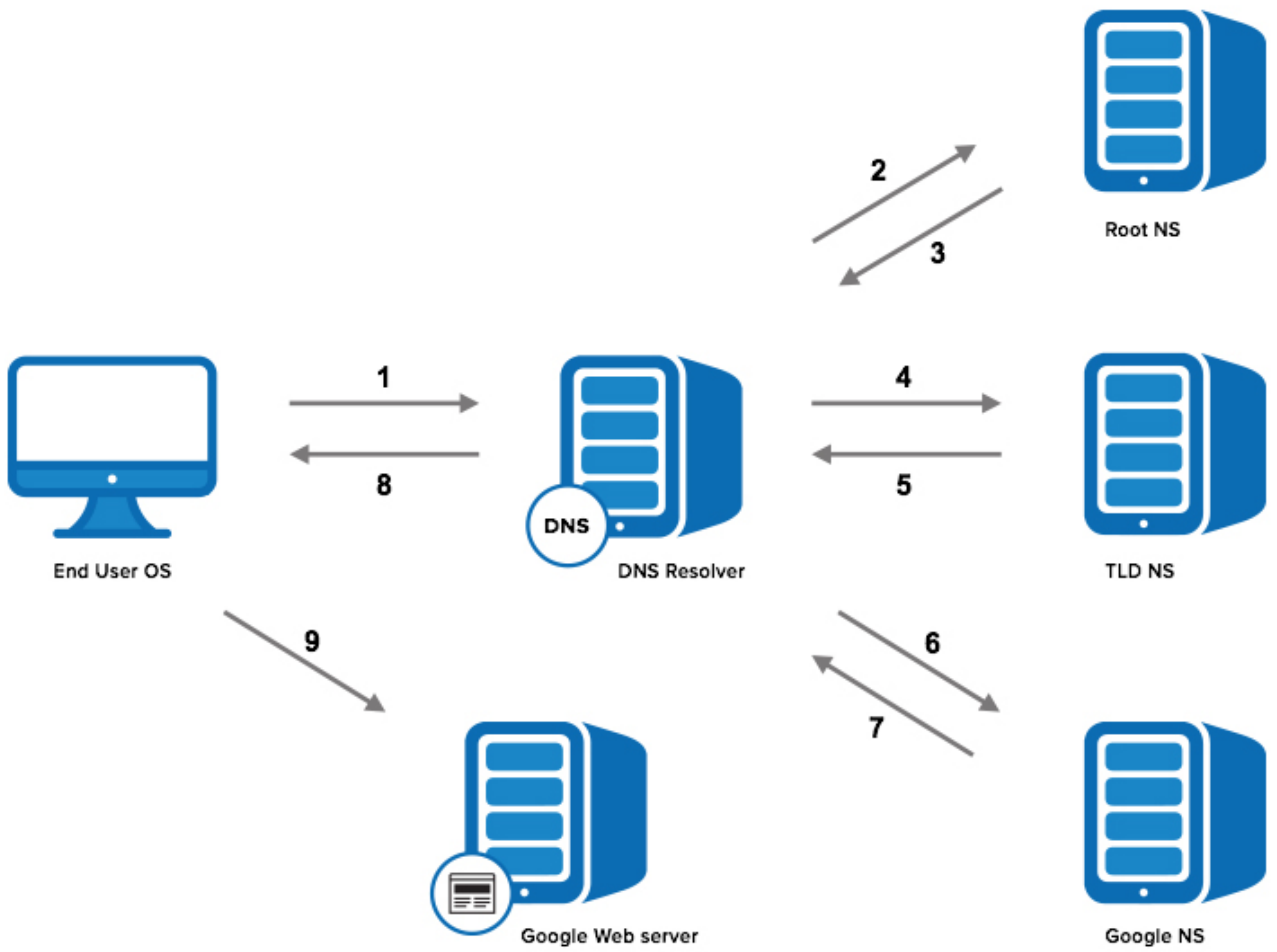
https://	protocol
www	subdomain
google.com	zone apex
google	domain
.com	top-level-domain (tld)
/search?q=URL…	path

DNS 101

- **Domain Name Service** (DNS) resolves domain names to IP addresses (like a phone book)
- **Domain Registrars**: authority that signs unique domain names (GoDaddy, BlueGtaor)
- **State of Authority** (SOA): Contains for example name of server for zone, administrator of zone, default time-to-live (ttl = time a DNS record is cached), seconds of secondary name server should wait before checking for updates
- **Root Zone** controlled by Internet Assigned Numbers Authority (IANA)
- **Name Servers** (NS Records): used by tld servers to direct traffic to DNS server (which contains authoritative DNS records)
- **A records** (part of DNS record): "A" stands for IP Address
- **CNAME** (part of DNS record): resolves one domain name to another
- **Autonomous System** (AS) and Border gateway Protocol (BGP) info

Python libraries: `python-whois`, `dnspython`, `tldextract`, `ipaddress`

DNS Flow



Root NS
2
3
1
4
8
5
DNS
End User OS
DNS Resolver
TLD NS
9
6
7
Google Web server
Google NS

What makes them different?

URL BlockList

amazon-sicherheit.kunden-ueberpruefung.xyz

eclipsehotels.com/language/en-GB/eng.exe

bohicacapital.com/page

summerweb.net

ad.getfond.info

vdula.czystykod.pl/rxdjna2.html

svision-online.de/mgfi/administrator/
components/com_babackup/classes/fx29id1.txt

URL AllowList

gurufocus.com/stock/PNC

dvdtalk.ru/review

333cn.com/zx/zhxw.html

made-in-china.com/special/led-lighting

google.com/u/0/112261544981697332354/posts

youtube.com/watch?v=Qp8MQ4shN6U

unesco.org/themes/education-sustainable-developm

thisisthefirst.com/page/5

Malicious URL Detection Features (Literature)

1. **BlackList Features**: BlackLists suffer from a high false negative rate, but can still be useful as machine learning feature.
2. **Lexical Features**: Capture the property that malicious URLs tend to "look different" from benign URLs. **Contextual information** such as the length of the URL, number of digits, lengths of different parts, entropy of domain name.
3. **Host-based Features**: Properties of web site host. **"Where"** the site is hosted, **"who" owns it** and **"how" it is managed**. API queries are needed (WHOIS, DNS records). Examples: Date of registration, the geolocations, autonomous system (AS) number, connection speed or time-to-live (TTL).
4. **Content-based Features**: Less commonly used feature as it requires **execution of web-page**. Can be not only be not safe, but also increases the computational cost. Examples: HTML or JavaScript based.

Sahoo et al. 2017: Malicious URL Detection using Machine Learning: A Survey

Domain Generating Algorithm (DGA) Detection

What is DGA?

- Domain Generating Algorithms (DGA) are algorithms which generate pseudo-random domains that are used for infected machines to communicate with the controller.
- The seeds on both the victim and command server are synchronized
- First seen with the Conficker worm.
- "Hello world" of cyber machine learning.

ML Feature Engineering

Lexical Features

1. Length of URL
2. Length of domain
3. Count of digits
4. Entropy of domain
5. Position (or index) of the first digit
6. Bag-of-words for tld, domain and path parts of the URL

Host-based Features

1. Time delta between today's date and creation date
2. Check if it is an IP address

Data Set (Features and Target)

				09:53:07
60112	teothemes.com/html/mp3pl/blue-preview.jpg	1	teothemes.com	2011-09-08 21:43:00
66946	kfj.cc:162/17852q	1	kfj.cc	2013-08-18 05:52:47
81906	verapdpf.info/db/6d1b281b5c4bbcfe3b99228680c232fa	1	verapdpf.info	2016-08-18 07:09:03

Features or X

Target (y)

	isMalicious	isIP	Length	LengthDomain	DigitsCount	EntropyDomain	FirstDigitIndex	com	or
73320	1	0	27	21	0	3.558519	0	0	1
30785	0	0	77	11	14	3.095795	22	1	0
60789	1	0	141	11	5	3.459432	103	0	1
19495	0	0	59	13	20	3.546594	31	1	0
[illegible]	1	0	[illegible]	11	7	[illegible]	[illegible]	0	0

Encode Text: Counting

- CountVectorizer: Convert a collection of text documents to a matrix of token counts

```
CountVectorizer_tlds = CountVectorizer(analyzer='word', vocabulary=top_tlds)
CountVectorizer_tlds = CountVectorizer_tlds.fit(tlds)
matrix_tlds = CountVectorizer_tlds.transform(tlds)
```

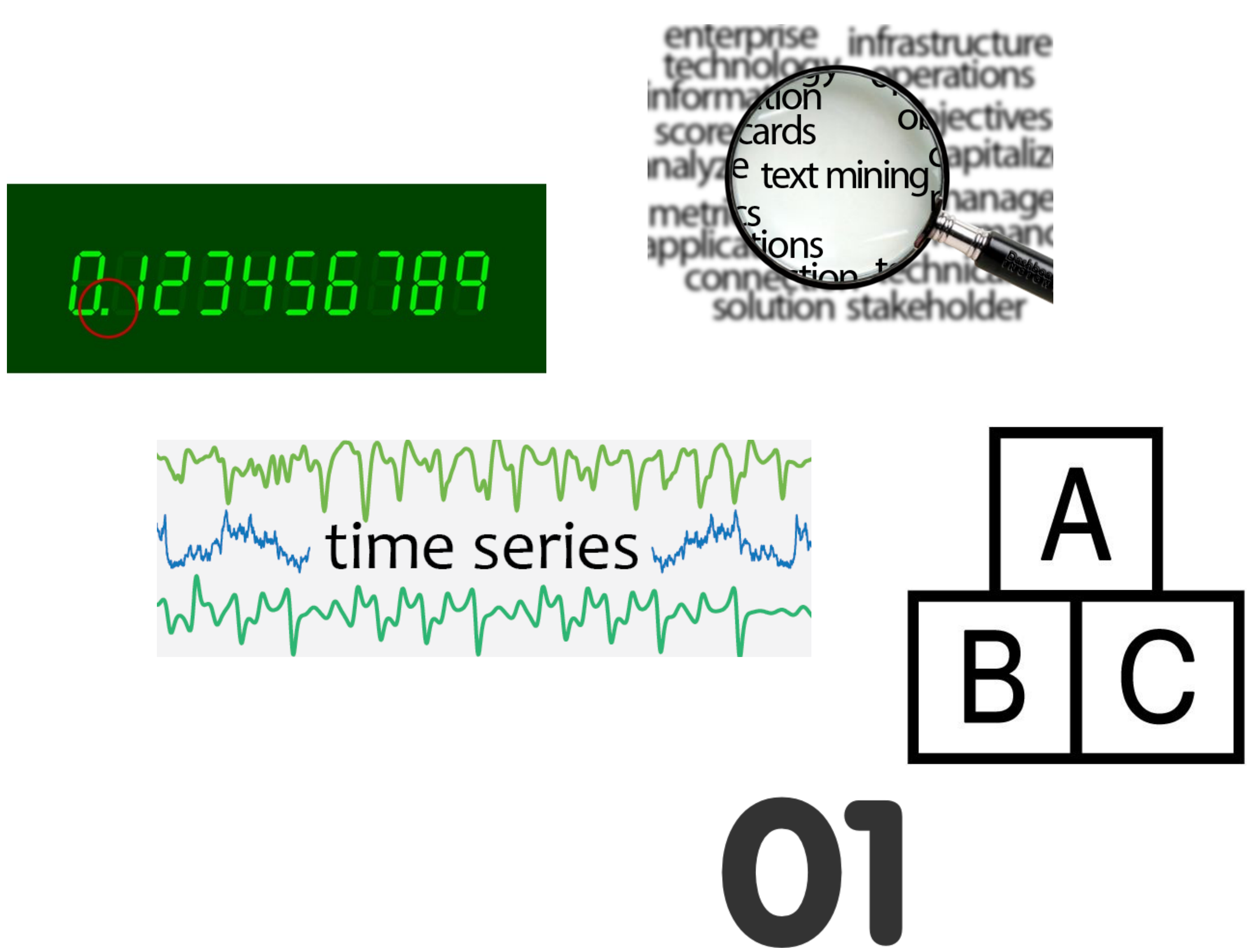
URL string
…google.ru…
…facebook.com…
…google.de…

'.com'	'.de'	'.uk'
0	0	0
1	0	0
0	1	0

Preprocessing - many options

- Imputing missing values
- Scaling/Normalization
- One-Hot Encoding (Encoding categorical features)
- Embedding (e.g. word2vec)
- Binarizing (e.g. needed for Deep Learning multi-class target vector encoding)
- Encoding strings as integers
- Dimensionality Reduction (e.g. PCA)
- Augmentation (e.g. tild/zoom images)
- Feature selection based on classifier
- Variance threshold

Data Types



Primary Python libraries: `pandas`, `sklearn`, `scipy`

http://scikit-learn.org/stable/modu

In Class Exercise

Please take 20 minutes and complete

Lab 4 Task A - Feature Engineering

(load data, create the features)

Missing Values

```
# using the most_frequent value
df['src_bytes'] = df['src_bytes'].fillna
# print the results
df['src_bytes'].value_counts().index[0])


# using the mean value
df['dst_bytes'] = df['dst_bytes'].fillna(df['dst_bytes'].mean())
```

Categorical Variables
One-Hot Encoding

One Hot Encoding

Color
Red
Red
Blue
Green
Yellow
Red

One Hot Encoding

4 Categories

Color
Red
Red
Blue
Green
Yellow
Red

4 Columns with 1 when Category is True and delete original column!

Color_Red	Color_Blue	Color_Yellow	Color_Green
1	0	0	0
1	0	0	0
0	1	0	0
0	0	0	1
0	0	1	0
1	0	0	0

One Hot Encoding

```
colors = ['Red', 'Red', 'Blue', 'Green', 'Yellow', ‘Red']
series_data = pd.Series( colors )

pd.get_dummies(series_data)

df = pd.get_dummies(df,
                     prefix=None,
                     prefix_sep=‘_’,
                    dummy_na=False,
                    columns=['protocol_type','flag'],
                    sparse=False
                    )
```

Better way: Feature Engine

- Feature Engine is a module which helps automate the complexities of feature engineering.
  - Missing Value Imputation
  - One Hot Encoding
  - Outlier Capping
  - More…
- Docs here: https://feature-engine.readthedocs.io/en/latest/
- Blog post: https://thedataist.com/when-categorical-data-goes-wrong/

In Class Exercise

Please take 20 minutes and complete

Lab 4 Task B - Feature Engineering

Calculate more features

Selecting Features

Should we use all of them?

How do we know which features to use and which to discard?

Before using automation, you should remove:

- Unnecessary features that are uninformative or repetitive
- Irrelevant features
- Duplicate observations

Selects k features according to the highest score

```
best_features = SelectKBest(score_func=chi2,k=3).fit_transform(features,target)
```

Selects all features above a given threshold in the scoring function

```
best_features =
SelectPercentile(score_func=chi2,percentile=3).fit_transform(features,target)
```

Available Scoring Functions:

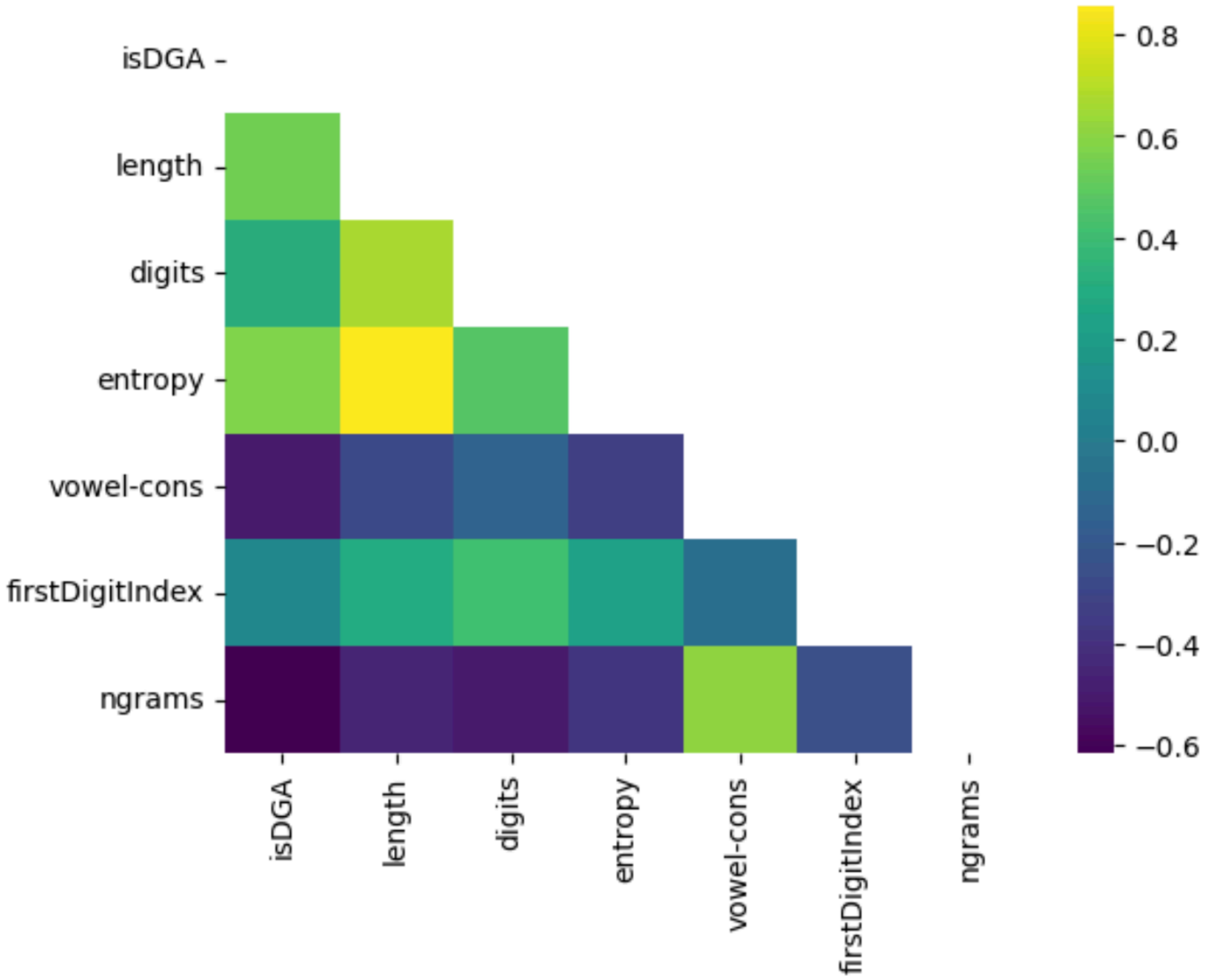
- **For regression:** f_regression, mutual_info_regression
- **For classification:** chi2, f_classif, mutual_info_classif

References:
http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html
http://scikit-learn.org/stable/modules/feature_selection.html#univariate-feature-selection

How do we know which features to use and which to discard?

Visualize Them!!

Heatmap

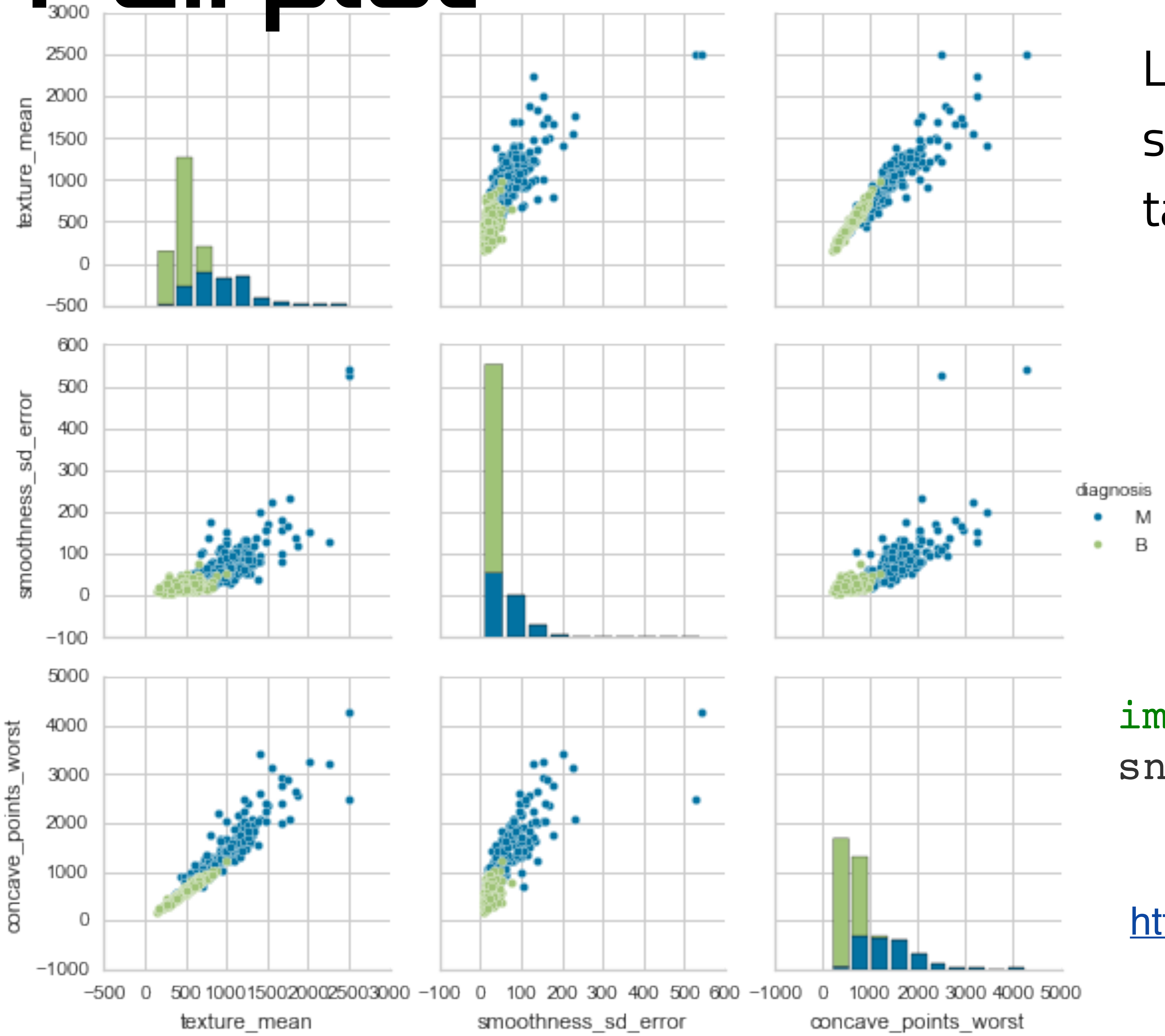


Look at the correlations, covariance, etc between features by calculating the stats and plotting a heatmap of the matrix

```
import seaborn as sns
import pandas as pd

#Calculate correlations between features
corr_matrix = df_final.corr()

#Generate a mask for the upper
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

#Plot the matrix as a heatmap
sns.heatmap(corr_mat, mask=mask, cmap='viridis')
```

strong correlation -> could mean similar information

Pairplot

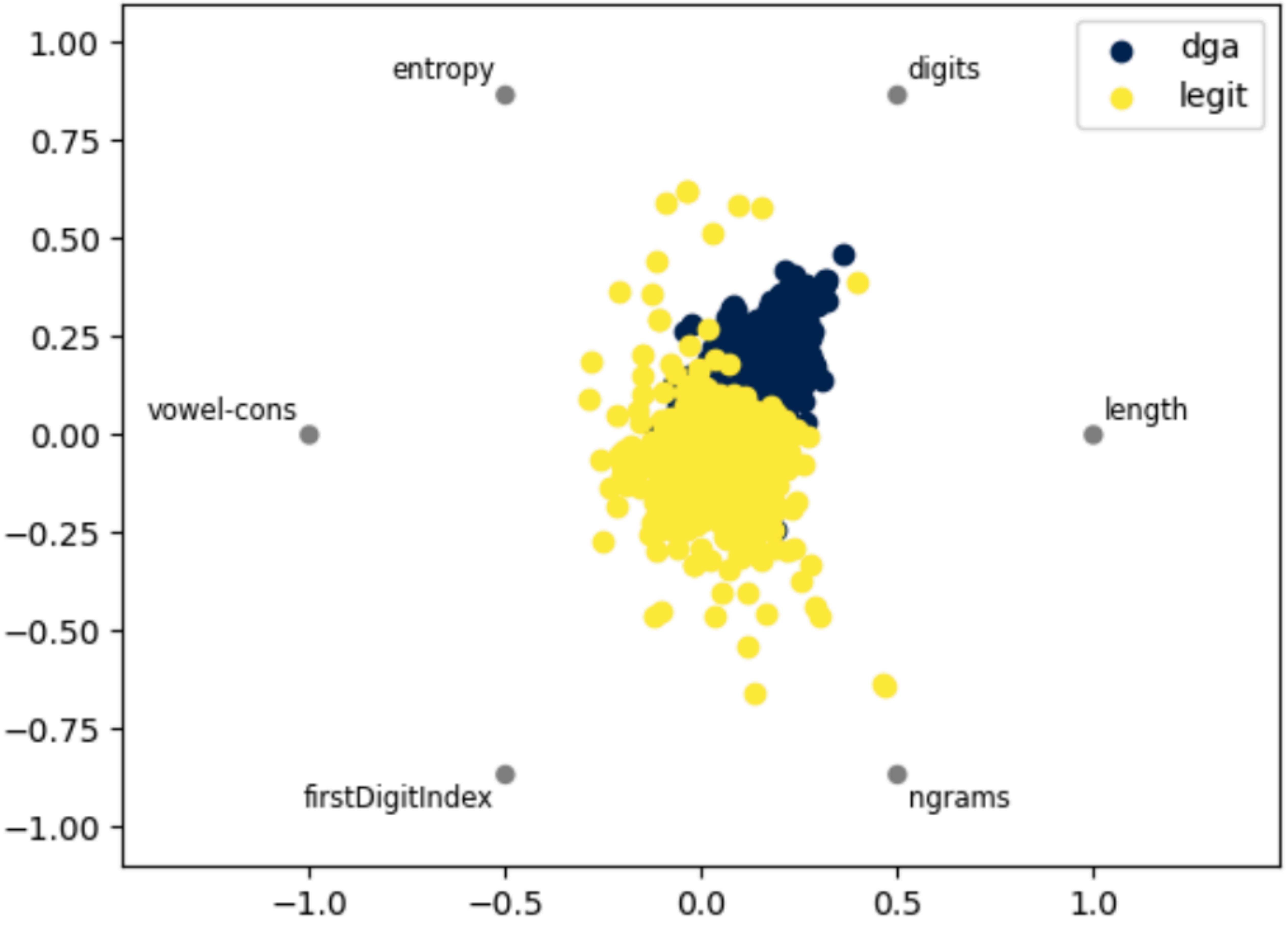


Look at the correlations between features with scatterplots of the targets, colored by the target class.

```
import seaborn as sns
sns.pairplot(<features>, hue='<target>')
```

http://seaborn.pydata.org/generated/seaborn.pairplot.html

Radviz



Veiw the contributions that each feature is making to the target via Pandas Radviz

```
import pandas as pd

pd.plotting.radviz(df, '<target_column')
```

Feature Scaling

Standard Scaling

Standardization - rescale so that mean = 0, std = 1

$$x_{scaled} = \frac{x - \mu}{\sigma}$$

```
scaled_feature = (feature - column_mean) / standard_deviation
```

Standard Scaling

$$x_{scaled} = \frac{x - \mu}{\sigma}$$

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(features)
features_scaled = scaler.transform(features)
```

Standard Scaling

```
original values:
 [[  0.9    0.1   40. ]
 [   0.3    0.2   50. ]
 [   0.6    0.8   60. ]]

Mean of each column:
 [   0.6   0.3667   50.]

SD of each column:
 [ 0.2449  0.3091  8.165 ]
```

```
scaled values:
 [[ 1.2247 -0.8627 -1.2247]
 [-1.2247 -0.5392  0.     ]
 [ 0.      1.4018  1.2247]]

Means of scaled data, per column:
 [ 0. -0.  0.]

SD's of scaled data, per column:
 [ 1.  1.  1.]
```

Notice that the mean of standard scaled data is zero and the StdDev is 1.

Min/Max Scaling

$$x_{scaled} = \frac{x - min}{max - min}$$

```
normed_feature = (feature - col_min) / (col_max - col_min)
```

Min/Max Scaling

```
from sklearn.preprocessing import MinMaxScaler
minmax = MinMaxScaler()

minmax.fit(features)
features_scaled_minmax = minmax.transform(features)
```

Min/Max Scaling

```
original values:
 [[  0.9   0.1  40. ]
 [   0.3   0.2  50. ]
 [   0.6   0.8  60. ]]

Mean of each column:
 [   0.6  0.3667  50.]

SD of each column:
 [ 0.2449  0.3091  8.165 ]
```

```
scaled values:
 [[ 1.       0.       0.     ]
 [ 0.       0.1429   0.5    ]
 [ 0.5      1.       1.     ]]

Means of scaled data, per column:
[ 0.5     0.381   0.5   ]

SD's of scaled data, per column:
[ 0.4082  0.4416  0.4082]
```

Dealing with Imbalanced Classes

Dealing with Imbalanced Classes

- A lot of real-world security data will have very imbalanced targets.
- Fortunately, there is a library called imbalanced-learn which can assist.
- Imbalanced-learn provides a series of options to resample the data so that you have more balanced classes
- Docs available here: http://imbalanced-learn.org/

Dealing with Imbalanced Classes

```
from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler(random_state=0)

X_resampled, y_resampled = ros.fit_resample(X, y)
```