Feature Engineering 101

Dan Ofer
https://ddofer.github.io

Overview

1. What is feature engineering (FE)? Why is it important for machine learning / statistics?
2. Feature engineering techniques for:
    - Categorical features
    - Text (Natural language)
    - Time Series
    - Geospatial

Why listen to me?

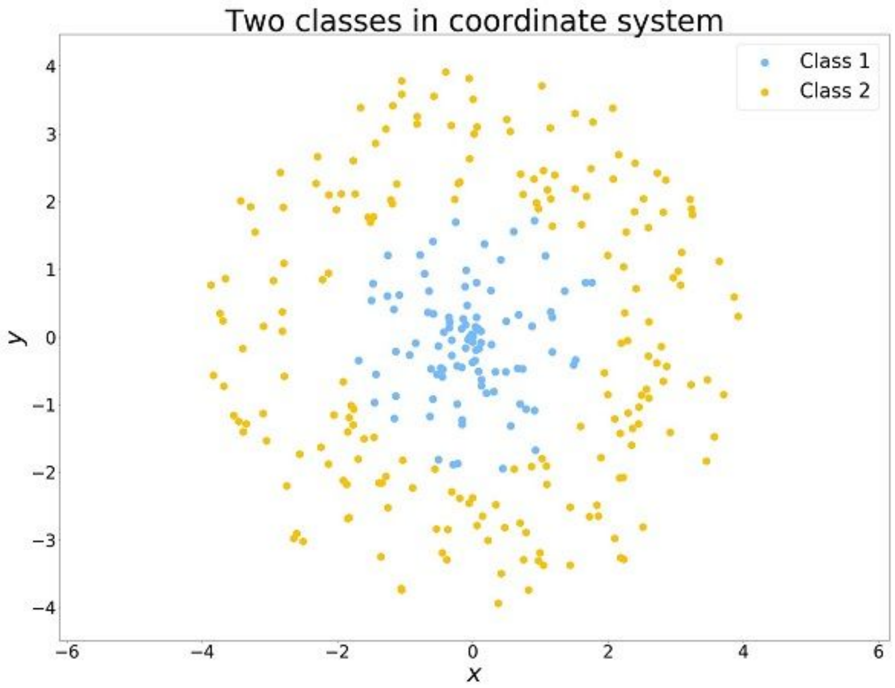
- Dan Ofer - Senior Data Scientist 4.5Y, now at Nutrino/Medtronic
- Sparkbeyond: AI with Fortune 500 & charities, including healthcare (Clalit), insurance, churn, chemicals etc’
- Top 0.8% on Kaggle (kaggle.com/danofer)
- 1st Place in WiDS 2020 challenge
- MsC: Neuroscience & Bioinformatics, thesis on protein feature engineering (HUJI)
- Probably took your picture at a convention/Midburn!

Feature Engineering

"Applied machine learning" is basically feature engineering

- Andrew Ng

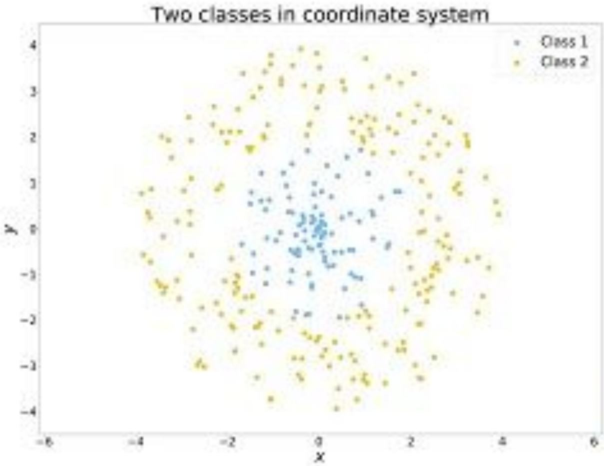
Feature Engineering: What is it good for?



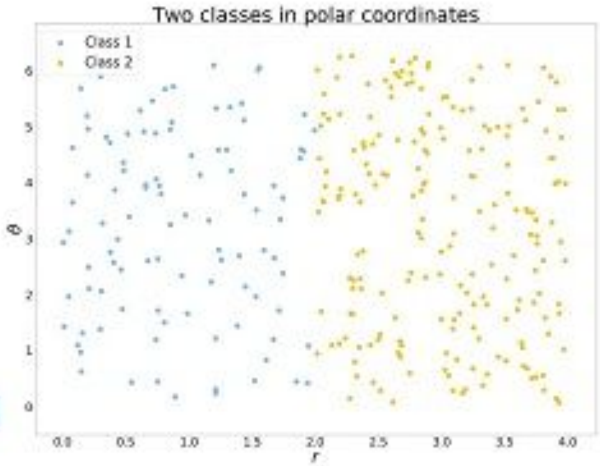
Feature Engineering: What is it good for?

Coordinate transformation

$$r = \sqrt{x^2 + y^2} \qquad \theta = \arctan \frac{y}{x}$$



Feature engineering



Tangled

Transparent

Feature engineering = transforming raw data into features that better represent the underlying problem, resulting in improved predictive model accuracy on unseen data

Feature Engineering

- Make *good* features
  - Highly predictive (of the target variable), succinct, interpretable, robust
- "Secret sauce of machine learning"
  - ~3d most important part of an machine learning project (after problem definition & data cleaning)
- Related, but different:
  - Feature extraction (raw features)
  - Feature selection
- Huge topic. More art than science
  - Extremely manual, handcrafted "dark magic" (For now)
- SparkBeyond (SOTA Automated Feature Engineering)
- Further Reading: Appendix + A few useful things to Know about machine Learning

Feature Extraction Vs Engineering

Given a Text, predict emotional sentiment

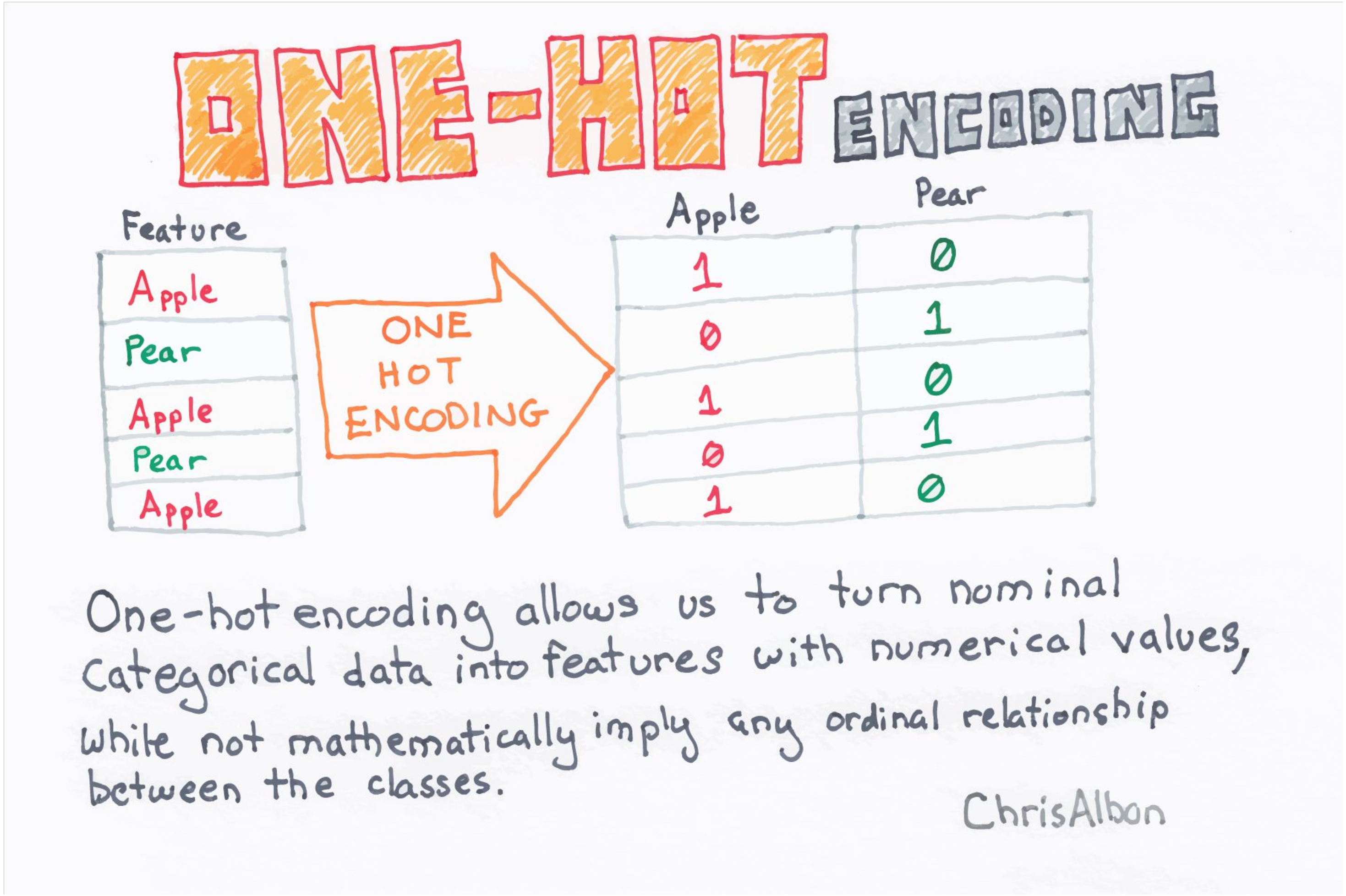
- “Star Wars = Greatest. movie. Ever!! Awesome! :D “ : Positive
- “Coronavirus quarantine/furlough makes me sad+bored” : Negative

- Feature extraction:
  - Characters or bytes in text
- (Simple) Feature engineering:
  - Lowercase text, tokenize (“split”) by whitespace & punctuation, count word frequencies etc’
    - What’s a word? What about Chinese? What about “New-York”?

Categorical Features

Categorical Features

- Categorical or ordinal features. “Moderate” cardinality (lower than free text, most elements expected to appear multiple times)
- **Cardinality** = How many unique values
- **Ordinal** features - have explicit “order”: 3>2>1. Old>Young…
  - Gender - [M/F/Rather not say].
  - Education level {Ordinal}. [Primary school/HS/college/graduate/PhD]
  - Ad category. [Sports/music/politics/cars/…]
  - Product ID. [Fuji XT3 / iPhone 10 / Galaxy S20/…]
- Some ML libraries handle “automatically”! E.g. Catboost, LGBM



Source: https://chrisalbon.com/machine_learning/preprocessing_structured_data/one-hot_encode_nominal_categorical_features/

One Hot Encoding (OHE)

- Naive - **One Hot Encoding - create a column for each possible value**
- Can result in a lot of columns, memory use. Can work best with simple models and huge amounts of data. Not ideal for tree models, when high cardinality.
- More approaches - feature hashing, etc'

One hot encoding methods:

```
df = pd.get_dummies(df,drop_first=True)
```

```
>>> from sklearn.preprocessing import OneHotEncoder
>>> X = [['Male', 1], ['Female', 3], ['Female', 2]]

>>> enc = OneHotEncoder(handle_unknown='ignore').fit(X)

>>> enc.categories_

[array(['Female', 'Male'], dtype=object), array([1, 2, 3], dtype=object)]
```

Ordinal encoding

- Replace string with (integer) number
- Use scikit-learn's [OrdinalEncoder]
- Can also use Pandas Dataframe's Categorical type - supports ordinals/ordering
- Works well for tree models that can "cut" arbitrarily. Unsuited for linear models

```
>>> from sklearn.preprocessing import OrdinalEncoder, LabelEncoder

>>> le = LabelEncoder()
>>> le.fit(["paris", "paris", "tokyo", "amsterdam"])
>>> le.transform(["tokyo", "tokyo", "paris"])
array([2, 2, 1]...)
```

Ordinal/Label vs One-Hot Encoding

Label Encoding

Food Name	Categorical #
Apple	1
Chicken	2
Broccoli	3

→

One Hot Encoding

Apple	Chicken	Broccoli
1	0	0
0	1	0
0	0	1

Big Idea - reduce dimensionality:

- **Reduce # unique variables to learn**
- **Handle rare variables (e.g. “singletons”)**

Transformations: Count/Frequency Encoding

- Replace variables with their count in the data ("Count encoding")
    - Combo! Replace variables that appear less than K times with their count.
- Efficient for reducing cardinality - e.g. if 70% of our values appear less than 2 times in the data.
- Relatively robust
- Adds some new information! (Popular/rare values)

```
data.apply(lambda x: x.map(x.value_counts()))

data.where(data.apply(lambda x: x.map(x.value_counts()))>=2, "other")
```

Transformations: Target Encoding

- Replace variables with target frequency - target/label encoding
- Some smoothing must be used, as otherwise will overfit badly!
- Many variants to technique: Weight Of Evidence, Bayesian target encoding, use of nested cross validation, etc'
  - Catboost does this internally
- Gotcha: just adding smoothing won't help for variables that appear just 1 time - they'll still be overfit, no matter the global prior!

```
df[“category_average_target”] = (0.5 +
df.groupby([“category”])[“target”].mean())/2
```

Interactions/Feature Crossings

- Statistical features over groups, or of a feature vs the group (e.g. an item's price vs mean price of all items in store-department).

```
df['Mean_Category_Price'] =
df.groupby(["Item_category"])['Price'].transform("mean")

df['relative_price'] = df['Price'].div(df['Mean_Category_Price'])
```

- Conjoined features:
    - Useful when we need a "unique ID" to distinguish an entity
    - E.g.: predict sales per item, per store:

```
df["joint_id"] = df["product_id"] + df["store_id"]
```

Interactions/Feature Crossings

Combinations ("crosses") of multiple features - "`A & B !C`"

- `e.g. "Age <20 & Education < college & profession == Doctor"` -> Fraud
  - sklearn's [PolynomialFeatures]
  - Can combine with bucketing and feature hashing - https://www.tensorflow.org/tutorials/structured_data/feature_columns#crossed_feature_columns
- Can easily overfit and create huge amount of noise features! Feature selection may not be enough
- I recomend restrict functions and columns used for generating features
- Consider your ml model's expressiveness, and if interaction expressed a non-linear relationship - e.g. BMI (Height/Weight^2)
  - This probably works well with huge amounts of data and regularized linear models

“Rounding down”

- Reduce cardinality (# unique variables) by aggregating to a higher “level”.
  - Reduces information (Bad 😨)
  - Reduces cardinality (good) -> reduces overfitting (good 😇). Domain specific...
- How to do it? - Take Prefix (truncate), round down granularity...
  - IP Addresses - take first few “blocks”.

    “192.168.42.1” -> “192.168”

    Map IPs to Country/state/city/zipcode..
  - Software version:

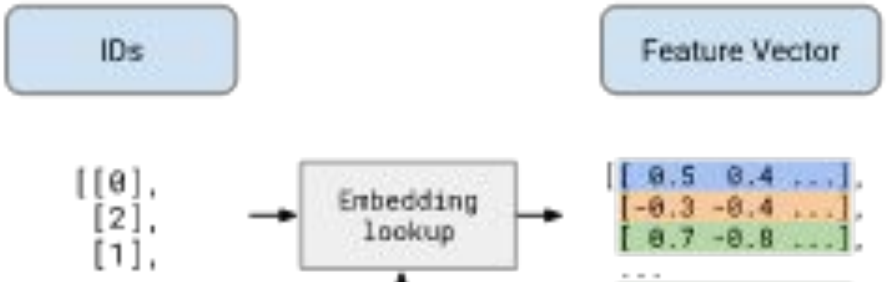
    “Windows 10.1.17.2358” -> “Windows 10.1.17”
  - Emails, websites - extract the domain.

    www.mashable.com/news/dan-ofer-wins-nobel-for-extreme-cleverness -> “[mashable](http://www.mashable.com)”
  - Hierarchical codes - zipcodes, medical codes (ICD9..) - truncate first X digits.

    Zipcode “90210” - Beverly hills neighbourhood. “902” -~ County level (larger area)
  - Age: 24.16 -> 24
  - Works best with semantic understanding

Entity Embedding

- Embedding using linear algebra/ML model. Word2Vec style
- Can also use SVD, matrix decomposition, graph methods, etc'.
- Used in Kaggle Rossman competition - "entity embedding"
  - Entity Embeddings of Categorical Variables (2016)
- Won't help with singletons/rare variables, unless you have a lot of unsupervised data to use. Can underperform simple one hot encoding - depends on ML model, variable cardinality, data size.



Predict emotional sentiment of a Text

- “Star Wars = Greatest. movie. Ever!! Awesome! :D“ : Positive
- “Coronavirus quarantine/furlough makes me sad+bored” : Negative

www.kaggle.com/danofer/reddit-comments-scores-nlp

https://github.com/ddofer/talk/blob/master/NLP%20101%20-%20ML%20Seminar%202017.pdf

Text Features

Text/NLP preprocessing

Huge subject in itself - transforming the input text before getting features.

- Lowercase text (or don't!). ("DAN Is grEAT" -> "dan is great")
- Normalize words, contractions, phrases, acronyms. E.g.:
  - "it's" -> "it is"
  - "{England, GB, Blighty}" -> "England".
  - Spelling correction
- Drop, keep or substitute placeholders for entities, phone numbers, emails..
  `"Call 052-9021042" -> "call PHONE_NUM"`
- Tokenizers (custom delimiters. E.g. Twitter tokenizer..).
- Stem or Lemmatizers: *"Cats" -> "Cat" ; "Octopii", "Octopuses" -> "Octopus"*
- Stop word removal - *("for and the or I")*

Excellent tools: Spacy, Textacy. Scikit-learn, Gensim, NLTK

https://chartbeat-labs.github.io/textacy/build/html/api_reference/text_processing.html

Bag of Words = count words in text

Document 1

The quick brown fox jumped over the lazy dog's back.

Document 2

Now is the time for all good men to come to the aid of their party.

Term	Document 1	Document 2
aid	0	1
all	0	1
back	1	0
brown	1	0
come	0	1
dog	1	0
fox	1	0
good	0	1
jump	1	0
lazy	1	0
men	0	1
now	0	1
over	1	0
party	0	1
quick	1	0
their	0	1
time	0	1

Stopword List
for
is
of
the
to

Image source: Quora: https://qr.ae/pNsdKl

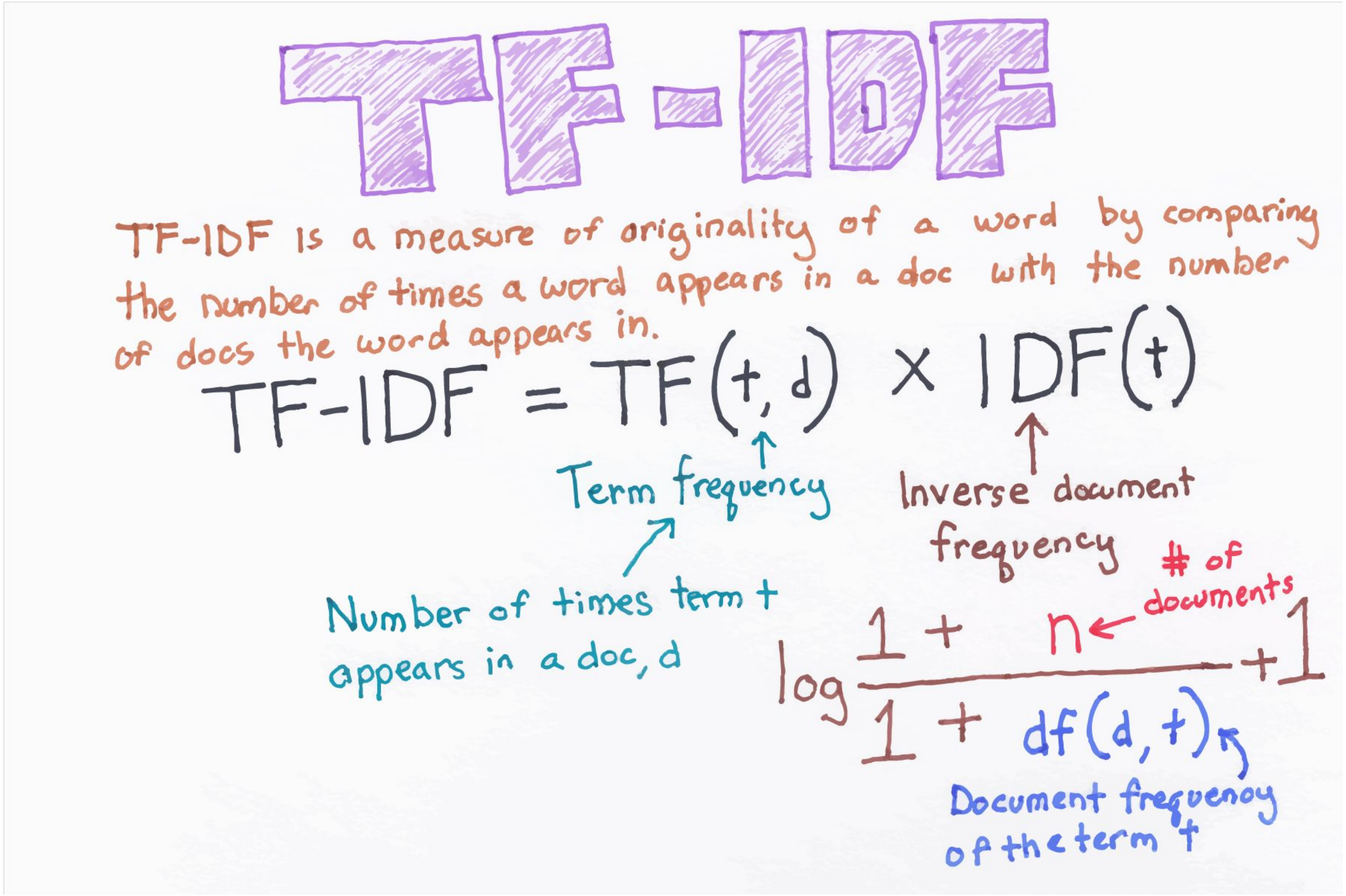
Improving on Bag of Words: TF/TF-IDF

- Bag of Words = count each words in text [0/1]
- TF - Term frequency (How many times a word appeared in the text)
- TF-IDF (IDF = Inverse document frequency: give less weight to very frequent words)

https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html

https://chrisalbon.com/machine_learning/preprocessing_text/bag_of_words/

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
```



Source: https://chrisalbon.com/machine_learning/preprocessing_text/tf-idf/

Improving on Bag of Words: TF/TF-IDF

- Bag of Words = count each words in text [0/1]
- TF - Term frequency (How many times a word appeared in the text)
- TF-IDF (IDF = Inverse document frequency: downweighs common words)

My advice: Try this first (with defaults and lowercasing).

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Term x within document y

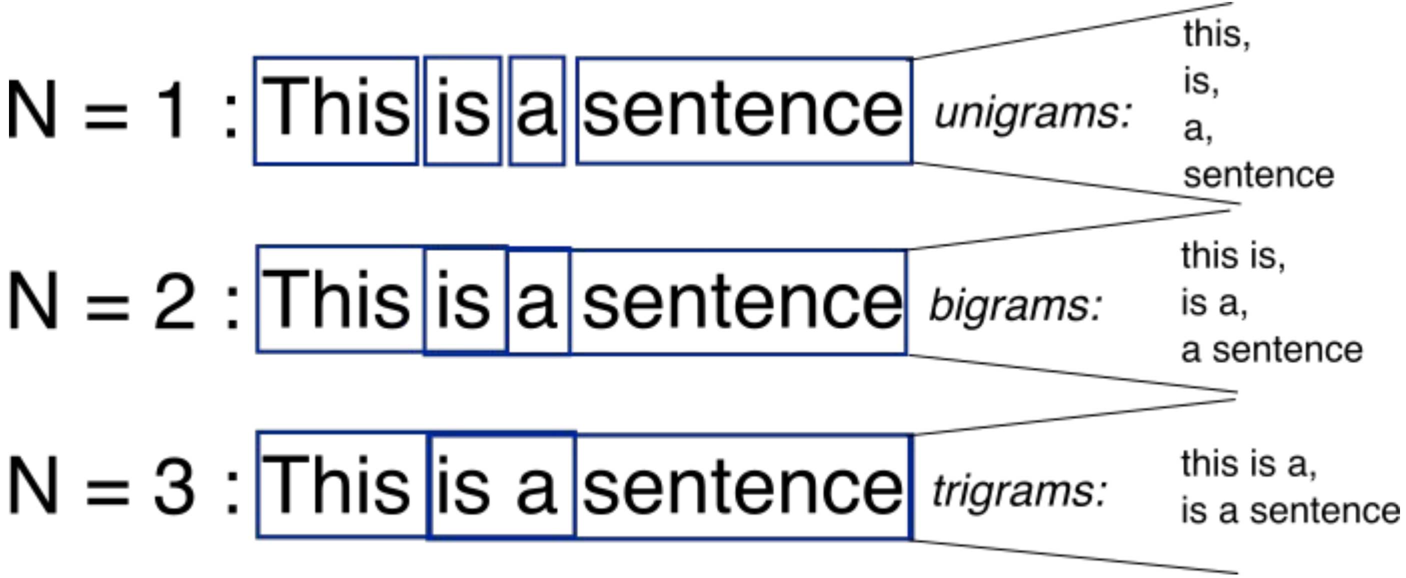
$tf_{x,y}$ = frequency of x in y

df_x = number of documents containing x

N = total number of documents

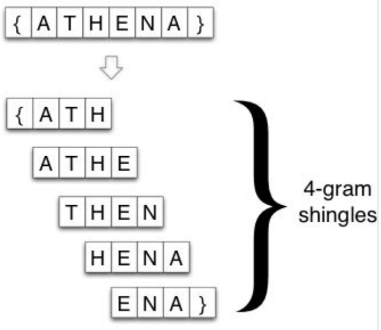
N-grams

- N-grams - N words together
  - Unigrams (1-gram/"default") - "New York" -> {"New":1, "York":1}
  - 2-Grams: "New York" -> {""New York":1}



Character level N-grams (shingles)

- Use letters/characters as “tokens”/basic unit, instead of words
- **Character level n-grams (shingles)** can be VERY useful for some domains!
  - Gives “free” count of special characters, punctuation etc’ (#,@,!,$...)
  - E.g. 3-gram characters shingles: “Danny” -> {“dan”, “ann”,”nny”}




```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(analyzer=”char”, ngram_range=(4,4))
```

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

Basic Text Featurization - tips

- EDA: frequency of majority/minority classes, text length, most frequent words, empty sentences, duplicates, html crud etc'
- Using N-grams - limit max N-size, max vocabulary, consider iterative text cleaning (e.g. conjoin entities ("New_York") in advance, remove stop-words)
- Don't "blindly" drop stop words in advance
- TF-IDF/BoW - try counts or frequency. Min count 3-5. Max_df ~0.98 %
- CountVectorizer also useful to featurize categorical "like" features (e.g. list of entities)
- Note - sklearn vectorizers use sparse matrices (Pandas supports them)
- Feature selection (e.g. Chi2, mutual information, max vocab size)
- SVD on term/BoW matrix

Text features

- Phrases/coallocations (Gensim; “New_York”) - expand n-gram space cheaply
- Semantic attributes - reading level (Textacy), named entities (Company, date, location etc’), Parts of speech (e.g. nouns, adjectives), % Camel Cased, % ALL CAPITALS, words in a predefined list (profanity)
- Language model - score sentence likelihood with KenLM (fast), Bert/GPT NN etc’
- Named entity extraction (NER - Spacy) - extract entities, and features about them:
    - E.g. “contains phone number -> Number is in New York area”.

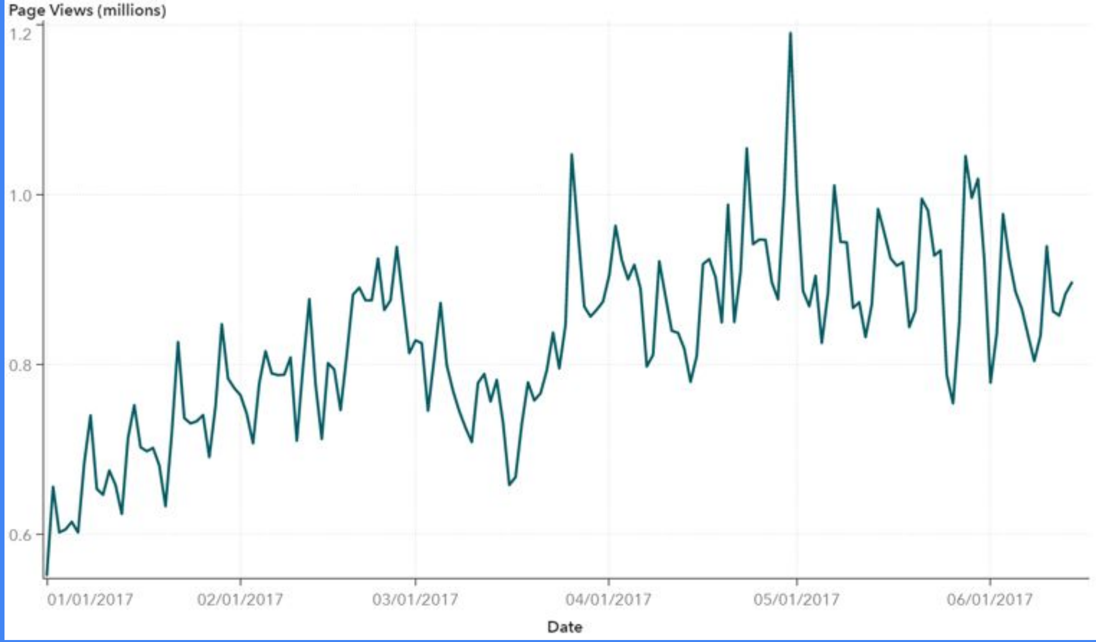
My example features notebook with code:
https://www.kaggle.com/danofer/reddit-comments-scores-nlp#Text-features-engineering

Word2Vec Text Features

- Word2Vec/FastText/Glove/Doc2Vec embeddings - get mean/max/sum over entire text, use vectors as new features
  - Train from scratch or use pre-trained embeddings.
    - Positional NN model embeddings like BERT - won't necessarily be better!
  - Multiple by word level TF-IDF score to improve - "A simple but tough to beat baseline"
  - Consider domain specific pretrained embeddings, +- fine-tuning.
  - Fast to train (unlike BERT)
  - Recomended library - many, but start with Gensim

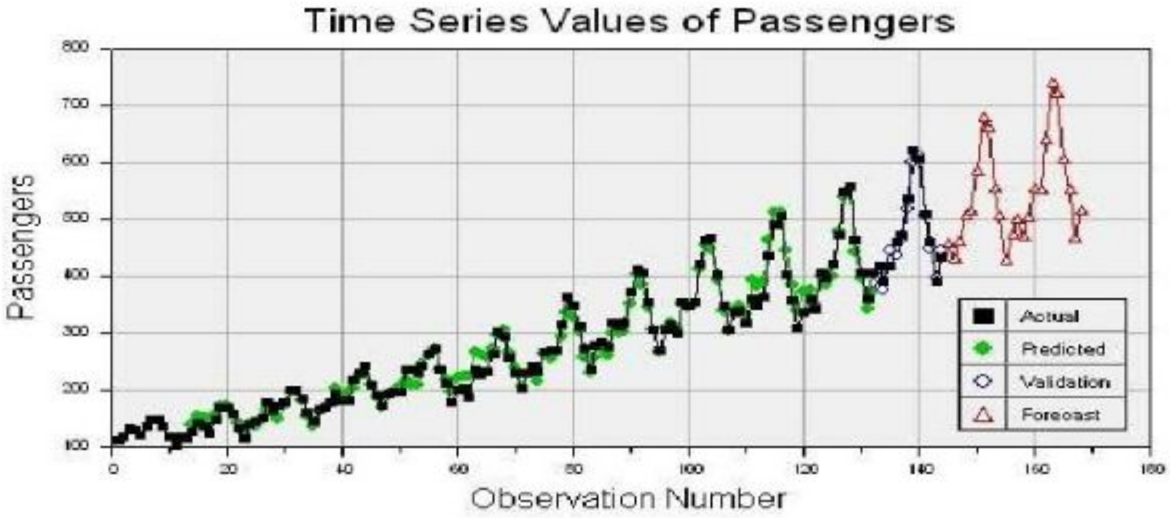
Advanced NLP augmentation - External data

- Simple: Emotion/Sentiment Lexicons.
  - {“Great”:+3, good:”+1, “fantastic!”:+5, “terrible”:-2,”shit”:-3, “horrible”:-5}
  - AFINN, Depechemood, McDonald finance sentiment lexicons..
- Medium: Semantic abstractions/”clusters” per word - lemmatization, Brown clusters..
- Advanced: word2vec embeddings per word, wordnet clusters, synsets..
- Data augmentation (more data points) - translate back/forth, word2vec synonym replacement..
- Look up features about words/entities from knowledge graphs/ontologies - e.g. Wikidata/Wikipedia, Wordnet, Google

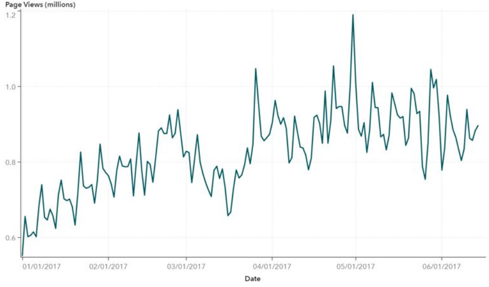

Page Views (millions)
1.2
1.0
0.8
0.6
01/01/2017
02/01/2017
03/01/2017
04/01/2017
05/01/2017
06/01/2017
Date

Time Series

TIME SERIES ANALYSIS



Time Series - Data over Time



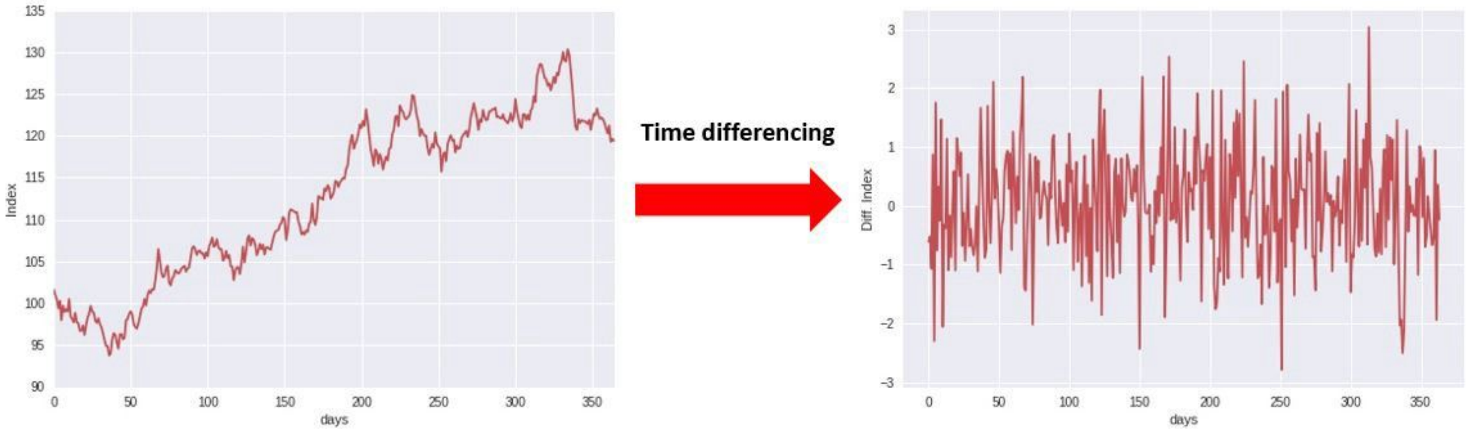
Can be seen as:

- (X) Data with time
  - E.g. “Given each customers internet history, predict if they will click an Ad”
- (Y) Target over time
  - E.g.: “Evaluate a financial trading strategy, predict daily stock market price, per stock, for each day, for the next month”
  - How many coronavirus patients..

Time-series: Some vital statistics

- Stationary (over time)?
  - Do statistical properties (e.g. mean, VAR) change? (e.g. inflation, growth)
  - Dickey–Fuller test (ADF)
  - If non stationary over time - detrend!

    https://people.duke.edu/~rnau/411diff.htm
    http://people.duke.edu/~rnau/whatuse.htm

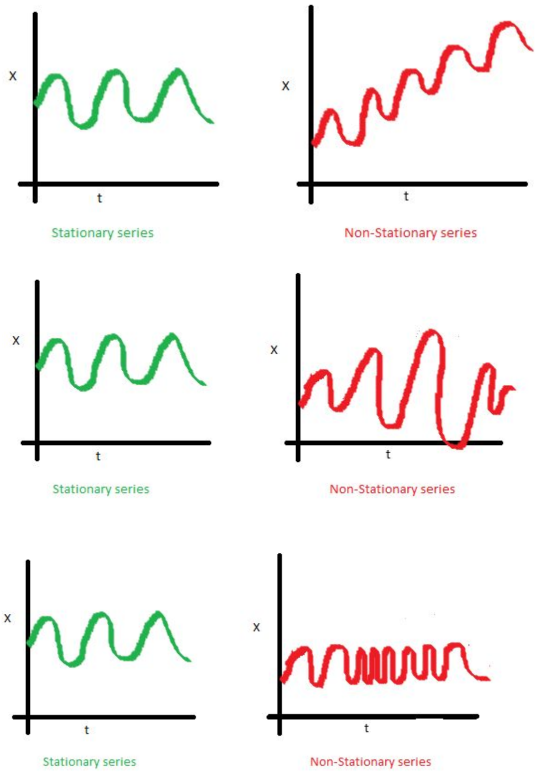


What does being stationary mean?

1. The mean of the series should not be a function of time. The red graph below is not stationary because the mean increases

2. The **variance** of the series should not be a function of time. This is known as **homoscedasticity**. Notice the varying **spread** of data over time.

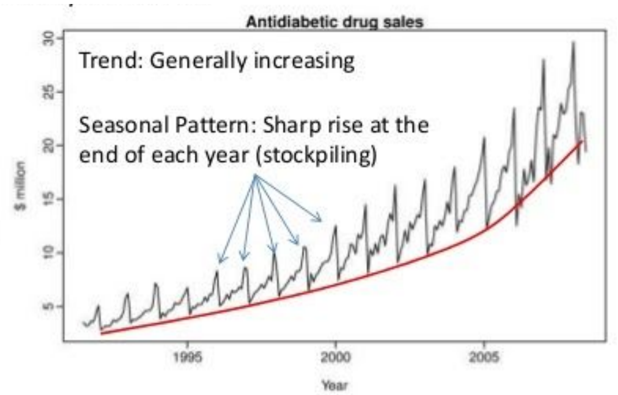
3. Finally, the **covariance** of the i th term and the (i + m) th term should not be a function of time. In the graph, notice the spread becomes closer as the time increases.



Source: https://www.seanabu.com/2016/03/22/time-series-seasonal-ARIMA-model-in-python/

Time-series - vital stats

- Seasonality?
  - Seasonal components: Daily/monthly/hourly? Holiday?
- Noise?
  - Regularly sampled intervals? Future non-causal noise?
- Multiple variables or univariate?
  - Predict a stock by its history: univariate (+AutoRegression)
  - Using other stocks: multivariate
- How much history?
  - Less than a year?
- Outliers?
- [Breakpoints](), Level shifts?



Evaluating Forecasts: Common Gotchas

- Target/Feature leakage:
  - Calculating features without accounting for **prediction horizon**.
  - E.g. Aggregate mean of value, including future data points.
- Not splitting test-set by time
  - E.g. Stock prices.
- Forgetting simple baselines
  - Last value, mean of value..
  - Overcomplicated models (often inferior to naive baselines!)
- Assuming everything is predictable
- Not accounting for non stationary values (Trends..)
  - Make it stationary by 1st/2d order differencing! https://people.duke.edu/~rnau/411diff.htm
  - Random forests & regression: can't predict target outside of range

Simple Time-Series Features

DateTime/Calendar features

```
df["Datetime"] = pd.to_datetime(df["Datetime"],
infer_datetime_format=True)

df["interval"] = df['end_date']-df['start_date']

df['year'] = df['Datetime'].dt.year

df['month'] = df['Datetime'].dt.month

df['week'] = df['Datetime'].dt.week

df['day'] = df['Datetime'].dt.day

df['hour'] = df['Datetime'].dt.hour

df['dayofweek'] = df['Datetime'].dt.dayofweek
```

Datetime Properties

Series.dt.date	Returns numpy array of python datetime.date objects (namely, the date part of Timestamps without timezone information).
Series.dt.time	Returns numpy array of datetime.time.
Series.dt.year	The year of the datetime
Series.dt.month	The month as January=1, December=12
Series.dt.day	The days of the datetime
Series.dt.hour	The hours of the datetime
Series.dt.minute	The minutes of the datetime
Series.dt.second	The seconds of the datetime
Series.dt.microsecond	The microseconds of the datetime
Series.dt.nanosecond	The nanoseconds of the datetime
Series.dt.week	The week ordinal of the year
Series.dt.weekofyear	The week ordinal of the year
Series.dt.dayofweek	The day of the week with Monday=0, Sunday=6
Series.dt.weekday	The day of the week with Monday=0, Sunday=6
Series.dt.dayofyear	The ordinal day of the year
Series.dt.quarter	The quarter of the date
Series.dt.is_month_start	Logical indicating if first day of month (defined by frequency)
Series.dt.is_month_end	Indicator for whether the date is the last day of the month.
Series.dt.is_quarter_start	Indicator for whether the date is the first day of a quarter.
Series.dt.is_quarter_end	Indicator for whether the date is the last day of a quarter.
Series.dt.is_year_start	Indicate whether the date is the first day of a year.
Series.dt.is_year_end	Indicate whether the date is the last day of the year.
Series.dt.is_leap_year	Boolean indicator if the date belongs to a leap year.
Series.dt.daysinmonth	The number of days in the month
Series.dt.days_in_month	The number of days in the month
Series.dt.tz	
Series.dt.freq	

https://www.kaggle.com/danofer/datetime-embeddings-for-end-to-end-deep-learning

Source:
https://pandas.pydata.org/docs/user_guide/timeseries.html#time-date-components

Lag

- Variable's value, X "steps" ago.
- Strong baseline to beat.
  - Momentum strategy in stocks: "Stock will be the same as yesterday" (Lag 1 day)
  - Weather will be the same as it was last year (lag 365)

```
df["value_lag1"] = df["Value"].shift(1)
```

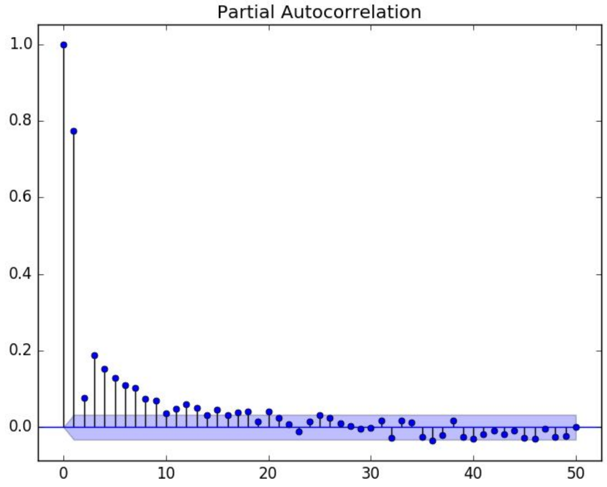
Date	Value	$Value_{t-1}$	$Value_{t-2}$
1/1/2017	200	NA	NA
1/2/2017	220	200	NA
1/3/2017	215	220	200
1/4/2017	230	215	220
1/5/2017	235	230	215
1/6/2017	225	235	230
1/7/2017	220	225	235
1/8/2017	225	220	225
1/9/2017	240	225	220
1/10/2017	245	240	225

Lag: How to pick lags?

1. Domain knowledge:
   - Store sales: same weekday last week (lag7) ; last month (lag30), last year (lag 365)
2. Partial autocorrelation plot: pick points with highest (absolute) correlation with target:

- `Pandas.plotting.autocorrelation_plot`

```
from matplotlib import pyplot
from statsmodels.graphics.tsaplots import plot_pacf
series = pd.read_csv('daily-temperature.csv')
plot_pacf(series, lags=50)
pyplot.show()
```



Sliding Window Statistics

"{Statistic} Over the last X points"

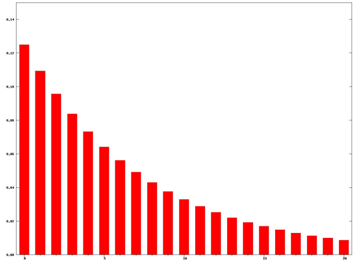
- Examples:
  - **Mean** sales over last **month**
  - **Max** sales over last **year**
  - Count unique visitors in past day
  - **Sum**, Var, STD, skew, curtosis, etc' ...
- Can be combined with different window/weighting methods :
  https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.rolling.html#pandas.DataFrame.rolling

Example: Feature of Mean sales over past 30 days, with a prediction horizon of 7 days into the future, using daily data:

```
df["mean_30day_sales"] = df["sales"].shift(7).rolling(window=”30D”,on=”Datetime”).mean()
```

Sliding Window: EWMA (exponential weighted moving average)

- Like sliding window, but give different weights to more recent points.
- E.g. average over last 3 years, but points 3 years ago have ¼ weight, 2 years ago ½ weight..
- https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.ewm.html



Many more time series features...

- Variable transformations - detrend, `diff`, scale for inflation, percent change...
- Time between X1 and X2. e.g. “Time between `date_started` and `current_date`”)

```
df[“interval_days_elapsed”] = (df[‘end_date’]-df[‘start_date’]).dt.days
```

- Total time elapsed (Can help “learn” inflation over time)
- Time since last occurence of X, counts of X in last interval
- Recent X vs historical X

```
df[“monthly_sales_vs_history”] = df[“sales”].rolling(window=”30D”).mean() /
df[“sales”].expanding().mean()
```

- X vs seasonal X - “Sales on this sunday vs previous sundays”
- Count peaks (“max”), troughs (“min”), time between min/max (local or global peak or trough)

Many more time series features...

- Signal analysis/Decomposition - FFT (Fourier transform), DWT (Discrete wavelets), Haar Wavelets...
- Autocorrelation, cross correlation with self or other time series
  - Entropy, change in autocorrelation or entropy over time...
- Many domain specific features for continuous series over time - finance - “Quant” features. E.g. “Candlesticks”.
  - There are many libraries with hundreds of such features. Beware of overfit..
- Be careful of temporal leakage!!!

TSFresh - nice time series features library

Advanced Time-Series Features

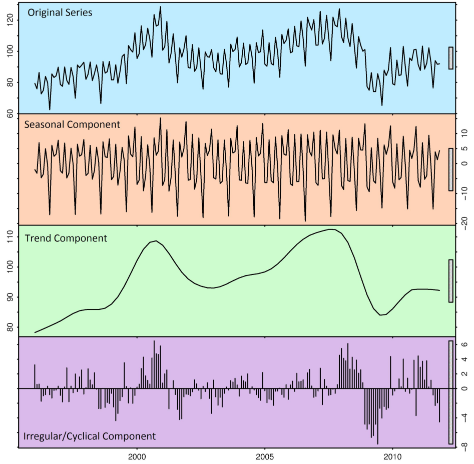
Grouped time-series

- Features for entities/subsets within groups
- Features for entities across groups/time-series
- Store sales example:
    - Sales **per item within** the store
    - Sales of an item **across** stores

```
df[“mean_item_sales”] =
df.set_index(“datetime”).groupby([“store”,”item”])[“sales”].rolling(window=30).mean()
```

Classical statistical time-series models

- Can outperform ML approaches, especially on small data, noisy problems, studied domains, highly seasonal problems, etc'
- We can use these models, or combine them with ML models!
- E.g. extract Trend, seasonal components from ARIMA/ statsmodels.seasonal decomposition/ FB prophet, and use as features!



Interpolation

- Fill in missing values - required for most classical models
- This can cause a lot of bias, especially when the data is very uneven.
- I suggest only doing this when your data is from regular intervals overall, and with high temporal resolution (e.g. weekly, daily).
- Typical approaches: backfill, forward fill , fill in by average...
  - All supported in pandas

Transform temporal variables/inputs

- Differentiate/gradient, normalize by Z-score, percent change, etc’
  - E.g. Δ/rate of change in coronavirus patients (1st order diff) is more important than absolute count, in order to predict if rate (Δ) of infection is slowing or increasing

- Transform continuous variables/target to a normal distribution
  - Hit it with a log!
  - Important for many linear models!
  - [Box-cox, other power transformations]

Target Transformations: Change Y

- Make target stationary for modelling - e.g. subtract/divide/differentiate
    - Very important for most models!
        - Reading material in appendix
    - 1st/2d order differencing is common approach to detrending

- "Remove baseline"
    - "Subtract" mean
    - "Subtract"/divide by top feature (e.g. moving average)
    - Normalize (Z-score) by group
    - Decomposition components/forecast
      https://otexts.com/fpp2/decomposition.html

Categorical variables & Text over time

- “Contains”.
- “Contains within last X”
- Frequency statistics (within time-window)
- Trends (e.g. breakout topics/keywords - a la Twitter)
- Word2Vec embeddings + window over time (No NN needed!)
- Order may be less important for these features, vs recency... problem dependent!
    - Workaround - looks at it as a set (ignore most ordering)
- Lots more!

Deep learning - time series

- LSTM, RNNs, Convolutional neural networks (CNN), BiLSTM+Dilated CNN + Attention, Etc'...
- In many forecasting competitions DL **lost** to classical TS approaches! But **combination** of both often **won**!

Geospatial (Geography) Features

Geospatial features - “samples near me”

Think about predicting taxi trip travel time, or housing prices..

- Features about other samples “near me”:
  E.g. “average price of houses nearby”, “Average price of properties within same Zipcode”...
- Consider different “sizes” to draw a shape around - K nearest, weighting by distance, etc’.
- These features can get very computationally expensive VERY easily with naive approaches!
- (My) Example notebooks with code/features:
  https://www.kaggle.com/danofer/fare-prediction-baseline-feat-eng

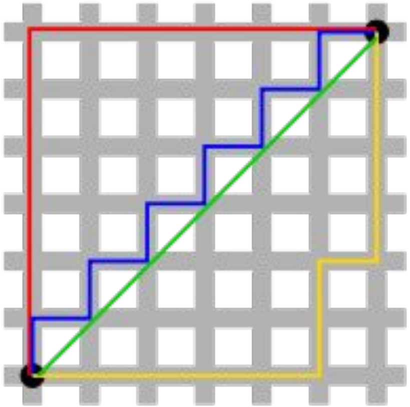
Geospatial features

- Combine **Lat**itude and **Lon**gitude columns together => new categorical feature
- Reduce Lat/Long granularity (e.g. 3d decimal place is ~± 110 meters).
  https://gis.stackexchange.com/questions/8650/measuring-accuracy-of-latitude-and-longitude

```
df[“rounded_ll”] = df[“lat”].round(3).astype(str) +
df[“lon”].round(3).astype(str)
```

- Bin/discretize Lat, Long (as categorical feature) - mainly useful for linear or deep models

Geospatial distance



Different ways of calculating distance:

- Euclidean distance ("as the crow flies")
- Haversine (includes earth's curvature), Manhattan, etc'
- Travel distance/time by car (not) - e.g. Manhattan distance, real travel time (google maps, Bing APIs), street graph distance (using OSM)

```
from math import cos, sqrt
def quick_euclid_distance(Lat1, Long1, Lat2, Long2):
     x = Lat2 - Lat1
     y = Long2 - Long1
return sqrt(x* + y*y)
```

https://www.kaggle.com/danofer/fare-prediction-baseline-feat-eng

Isochrones

- **Isochrones** = “What’s within 5/15/K minutes walk/drive from me?”
    - Can be very different from “air distance” - What if there’s a highway crossing the street? Think Tel Aviv old Central bus station...
    - Use: [Open Street Map - OSMnx](geoffboeing.com/2017/08/isochrone-maps-osmnx-python/) - geoffboeing.com/2017/08/isochrone-maps-osmnx-python/ or google maps

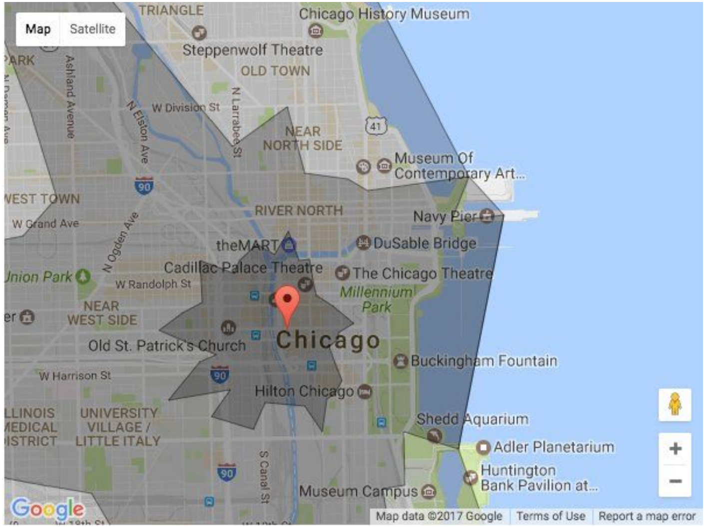


Image source: Yhat - http://blog.yhat.com/posts/isochrones-isocronut.html

Geospatial clusters

- Cluster lat-long points (k-means) - add as new feature or high level aggregator. Can help discover neighbourhoods.

- Use real world “clusters”! E.g. Zip codes, cities, Airbnb/Zillow business neighbourhoods, Nielsen…

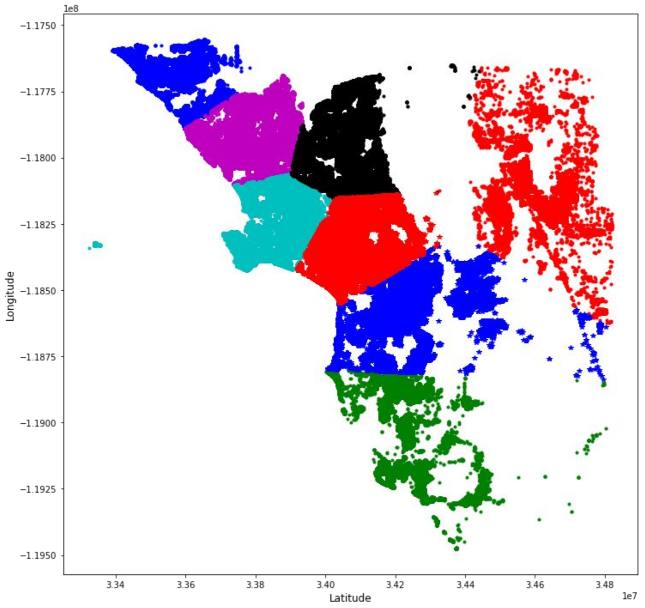
Geospatial K-means Clustering by latitude and longitude

```
from sklearn.cluster import KMeans
```

df = data of latitude and longitude for each row

```
X = df[['latitude','longitude']]
kmeans = KMeans(n_clusters=id_n).fit(X)
id_label=kmeans.labels_

## plot output ...
```



https://www.kaggle.com/xxing9703/kmean-clustering-of-latitude-and-longitude

More Geographic features

- Distance from: City center, ocean, beach, park, road, nearest other data point…
- # Points of interest nearby (OSM - Open Street Map POI, OSMnx)
- # tripadvisor/Yelp checkins in surrounding area, average review scores
- Geotemporal features (Great for fraud, crime!) - travel speed between points, events nearby in past hour…
- Geocode addresses into LatLong or reverse; LatLong into locations
- External data - census demographic data, ACS, Zillow, Open Street map (OSM) map, APIs

Thanks!

P.S - We’re Hiring!

ddofer.github.io

Further reading

Further reading

- https://github.com/solegalli/packt_featureengineering_cookbook
- Python Feature Engineering Cookbook
- Feature Engineering for Machine Learning, O'Reilly. https://github.com/alicezheng/feature-engineering-book
- https://github.com/aikho/awesome-feature-engineering
- https://github.com/Yimeng-Zhang/feature-engineering-and-feature-selection
- https://machinelearningmastery.com/basic-feature-engineering-time-series-data-python/
- Kaggle: Learn Feature extraction/"engineering" https://www.kaggle.com/learn/feature-engineering
- https://machinelearningmastery.com/basic-feature-engineering-time-series-data-python/
- https://www.slideshare.net/HJvanVeen/feature-engineering-72376750/11 - categorical embedding/transformations
- https://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/
- https://www.slideshare.net/HJvanVeen/feature-engineering-72376750/11
- https://blog.dataiku.com/2015/06/25/predicting_crime_sf (Geospatial features)
- https://www.kaggle.com/c/ieee-fraud-detection/discussion/108575#624914
- Feature Engineering for Machine Learning, Udemy Course

Further reading: Time Series

- https://people.duke.edu/~rnau/411diff.htm
- Forecasting: Principles and Practice - Rob J Hyndman and G Athanasopoulos - https://otexts.com/fpp2/
- http://www.svds.com/avoiding-common-mistakes-with-time-series/
- TimeSeriesSplit - scikit-learn Time Series cross-validator

Further reading: Libraries

- Pandas - designed for tabular data and time-series! https://pandas.pydata.org/
- Automatic extraction of relevant features from time series: http://tsfresh.readthedocs.io
- Facebook Prophet - very easy to use bayesian TS
- Irregularly spaced TS features : https://traces.readthedocs.io/en/latest/
- https://github.com/microsoft/forecasting
- Statsmodels (Inc. classical models for time series - Arima)
- **OSMNx - Open street map**
- **Hyndman (R) Blog - auto.arima, Feasts etc' -** https://robjhyndman.com/hyndsight
- Feature Tools - Open-source Feature engineering library. Time-aware. https://www.featuretools.com/
- H20 - http://docs.h2o.ai/driverless-ai/latest-stable/docs/userguide/time-series.html
- SparkBeyond (Automated feature engineering & insights including complex multivariate, non numeric irregular time-series)