FEATURE ENGINEERING

HJ van Veen - Data Science - Nubank Brasil

"More data beats clever algorithms, but better data beats more data."

—Peter Norvig

Feature Engineering

- **Most creative aspect of Data Science.**
- Treat like any other creative endeavor, like writing a comedy show:
- Hold brainstorming sessions
- Create templates / formula's
- Check/revisit what worked before

Categorical Features

- Nearly always need some treatment
- High cardinality can create very sparse data
- Difficult to impute missing

Onehot encoding

- **One-of-K encoding on an array of length K.**
- Basic method: Used with most linear algorithms
- Dropping first column avoids collinearity
- Sparse format is memory-friendly
- Most current implementations don't gracefully treat missing, unseen variables

Onehot encoding

```
Sample: ["BR"]

country     country=NL country=BR country=US
--------    ---------- ---------- ----------
NL       => [         0,         1,         0]
BR
US

Encoded dense: [0, 1, 0]
Encoded sparse: 2:1
```

Hash encoding

- **Does "OneHot-encoding" with arrays of a fixed length.**
- Avoids extremely sparse data
- May introduce collisions
- Can repeat with different hash functions and bag result for small bump in accuracy
- Collisions usually degrade results, but may improve it.
- Gracefully deals with new variables (eg: new user-agents)

Hash encoding

```
Sample: ["BR"]

hash("BR") => `2`

country      hash1 hash2 hash3 hash4 hash5
-------      ----- ----- ----- ----- -----
NL       => [    0     1,    0     0,    0]
BR
US

Encoded dense: [0, 1, 0, 0, 0]
Encoded sparse: 2:1
```

Label encoding

- Give every categorical variable a unique numerical ID
- Useful for non-linear tree-based algorithms
- Does not increase dimensionality
- Randomize the cat_var -> num_id mapping and retrain, average, for small bump in accuracy.

Label encoding

```
Sample: ["Queenstown"]

city                 city
-----------          ----
Cherbourg               1
Queenstown    =>        2
Southampton             3

Encoded: [2]
```

Count encoding

- Replace categorical variables with their count in the train set
- Useful for both linear and non-linear algorithms
- Can be sensitive to outliers
- May add log-transform, works well with counts
- Replace unseen variables with `1`
- May give collisions: same encoding, different variables

Count encoding

```
Sample: ["A6GHBD78"]

teacher_ID      teacher_ID
----------      ----------
DEADB33F                 4
A6GHBD78                 3
DEADB33F                 4
FCKGWRHQ    =>           1
DEADB33F                 4
A6GHBD78                 3
A6GHBD78                 3
DEADB33F                 4

encoded: [3]
```

LabelCount encoding

- **Rank categorical variables by count in train set**
- Useful for both linear and non-linear algorithms
- Not sensitive to outliers
- Won't give same encoding to different variables
- Best of both worlds

LabelCount encoding

```
tld      tld
---      ---
nl         3
nl         3
nl         3
nl   =>    3
de         2
de         2
fr         1
fr         1
```

Target encoding

- **Encode categorical variables by their ratio of target (binary classification or regression)**
- Be careful to avoid overfit!
- Form of stacking: single-variable model which outputs average target
- Do in cross-validation manner
- Add smoothing to avoid setting variable encodings to 0.
- Add random noise to combat overfit
- When applied properly: Best encoding for both linear and non-linear

Target encoding

role	y		role
manager	1		0.5
engineer	1		0.66
scientist	1	=>	1.
manager	0		0.5
engineer	0		0.66
engineer	1		0.66

Category Embedding

- **Use a Neural Network to create dense embeddings from categorical variables.**
- Map categorical variables in a function approximation problem into Euclidean spaces
- Faster model training.
- Less memory overhead.
- Can give better accuracy than 1-hot encoded.
- https://arxiv.org/abs/1604.06737

Category Embedding

role	role 3-D embedding
manager	[0.05, 0.10, 0.96]
engineer	[0.72, 0.66, 0.17]
scientist	[0.75, 0.62, 0.15]
manager	[0.05, 0.10, 0.96]
engineer	[0.72, 0.66, 0.17]
engineer	[0.72, 0.66, 0.17]

NaN encoding

- **Give NaN values an explicit encoding instead of ignoring**
- NaN-values can hold information
- Be careful to avoid overfit!
- Use only when NaN-values in train and test set are caused by the same, or when local validation proves it holds signal

NaN encoding

```
Sample = [NaN]

UA          UA=mobile UA=tablet UA=NaN
--------    --------- --------- ------
mobile                0         0      1
tablet
mobile  =>
NaN
mobile

Encoded = [0, 0, 1]
```

Polynomial encoding

- **Encode interactions between categorical variables**
- Linear algorithms without interactions can not solve the XOR problem
- A polynomial kernel *can* solve XOR
- Explodes the feature space: use FS, hashing and/or VW

Polynomial encoding

A	B		y		A=1*B=1	A=0*B=1	A=1*B=0	A=0*B=0		y
1	1	|	1		1	0	0	0	|	1
0	1	|	0	=>	0	1	0	0	|	0
1	0	|	0		0	0	1	0	|	0
0	0	|	1		0	0	0	1	|	1

Expansion encoding

- **Create multiple categorical variables from a single variable**
- Some high cardinality features, like user-agents, hold far more information in them:
  - is_mobile?
  - is_latest_version?
  - Operation_system
  - Browser_build
  - Etc.

Expansion encoding

```
Mozilla/5.0 (Macintosh; Intel Mac OS X
10_10_4) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/53.0.2785.143 Safari/537.36

|
v

UA1     UA2                UA3      UA4 UA5
------- ---------------- -------- --- --------
Chrome 53.0.2785.143 Desktop Mac 10_10_4
```

Consolidation encoding

- **Map different categorical variables to the same variable**

- Spelling errors, slightly different job descriptions, full names vs. abbreviations

- Real data is messy, free text especially so

Expansion encoding

```
company_desc                desc1 company_desc2
--------------------        ----- -------------
Shell                       Shell   Gas station
shel                        Shell   Gas station
SHELL                       Shell   Gas station
Shell Gasoline              Shell   Gas station
BP                  =>        BP   Gas station
British Petr.               BP      Gas station
B&P                         BP      Gas station
BP Gas Station              BP      Gas station
bp                          BP      Gas station
Procter&Gamble              P&G  Manufacturer
```

“Coming up with features is difficult, time-consuming, requires expert knowledge. "Applied machine learning" is basically feature engineering.”

—Andrew Ng

Numerical Features

- Can be more readily fed into algorithms
- Can constitute floats, counts, numbers
- Easier to impute missing data

Rounding

- **Round numerical variables**
- Form of lossy compression: retain most significant features of the data.
- Sometimes too much precision is just noise
- Rounded variables can be treated as categorical variables
- Can apply log-transform before rounding

Rounding

```
age                age1 age2
-------            ---- ----
23.6671              23    2
23.8891              23    2
22.1261   =>         22    2
19.5506              19    1
18.2114              18    1
```

Binning

- **Put numerical variables into a bin and encode with bin-ID**

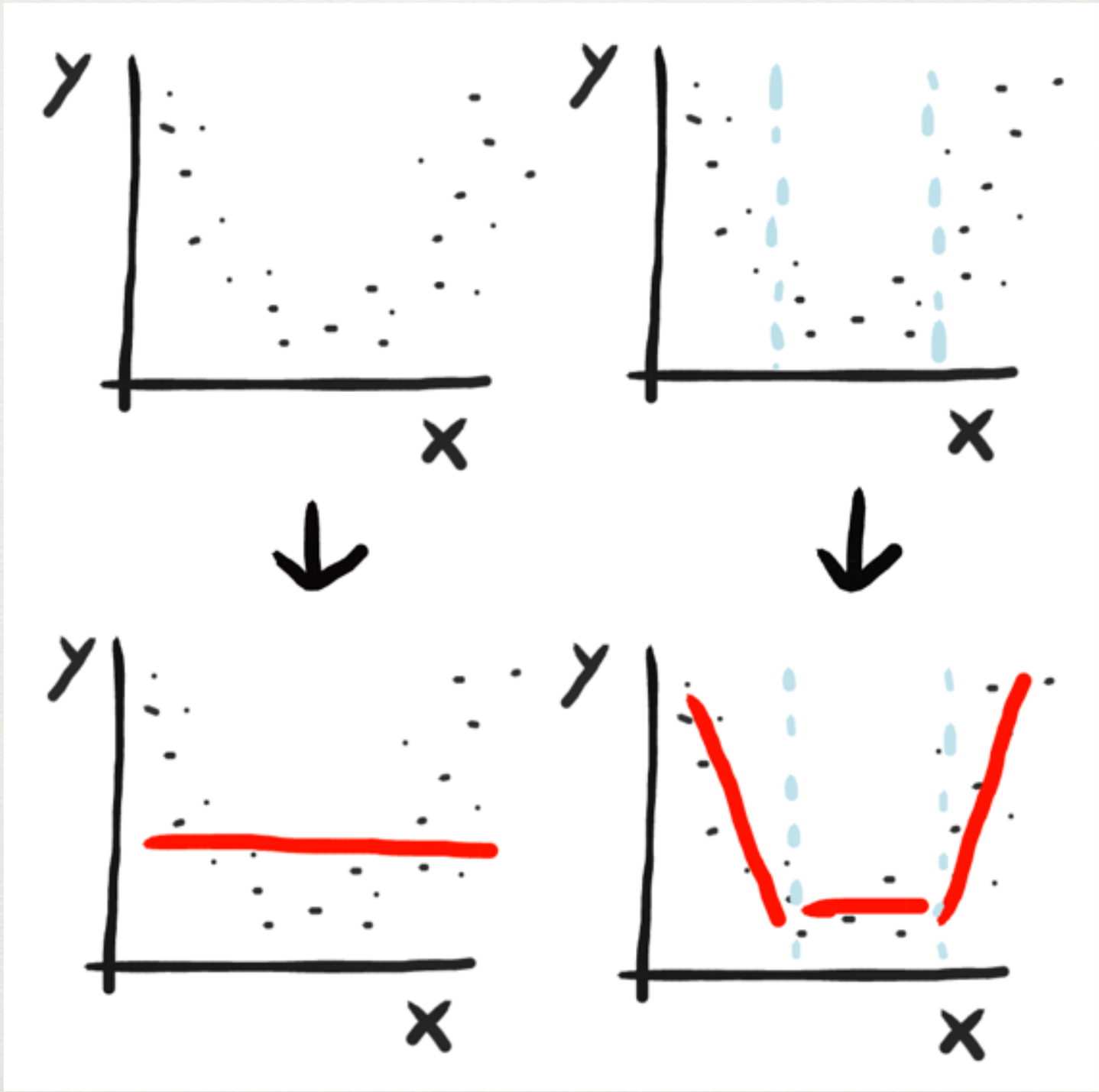
- Binning can be set pragmatically, by quantiles, evenly, or use models to find optimal bins

- Can work gracefully with variables outside of ranges seen in the train set

Binning

risk_score		rs[-inf,33]	rs[33,66]	rs[66,inf]
15		1	0	0
77		0	0	1
78	=>	0	0	1
55		0	1	0
42		0	1	0

Binning



Scaling

- **Scale to numerical variables into a certain range**

  - Standard (Z) Scaling
  - MinMax Scaling
  - Root scaling
  - Log scaling

Imputation

- **Impute missing variables**
- Hardcoding can be combined with imputation
- Mean: Very basic
- Median: More robust to outliers
- Ignoring: just postpones the problem
- Using a model: Can expose algorithmic bias

Imputation

wage	hours	gender
1600	40	0
2200	50	1
1800	36	0
2100	45	1
2050	60	NaN
1650	36	0

y
1
1
0
0
0
1

=>

wage	hours
1600	40
2200	50
1800	36
2100	45
2050	60
1650	36

gender_y
0
1
0
?
?
?

Interactions

- **Specifically encodes the interactions between numerical variables**
- Try: Substraction, Addition, Multiplication, Divison
- Use: Feature selection by statistical tests, or trained model feature importances
- Ignore: Human intuition; weird interactions can give significant improvement!

"…some machine learning projects succeed and some fail. What makes the difference? Easily the most important factor is the features used."

—*Pedro Domingos*

Non-linear encoding for linear algo's

- **Hardcode non-linearities to improve linear algorithms**
- Polynomial kernel
- Leafcoding (random forest embeddings)
- Genetic algorithms
- Locally Linear Embedding, Spectral Embedding, t-SNE

Row statistics

- **Create statistics on a row of data**
- Number of NaN's,
- Number of 0's
- Number of negative values
- Mean, Max, Min, Skewness, etc.

"The algorithms we used are very standard for Kagglers. […] We spent most of our efforts in feature engineering. [...] We were also very careful to discard features likely to expose us to the risk of over-fitting our model."

Xavier Conort

Temporal Variables

- Temporal variables, like dates, need better local validation schemes (like backtesting)
- Easy to make mistakes here
- Lots of opportunity for major improvements

Projecting to a circle

- **Turn single features, like day_of_week, into two coordinates on a circle**

- Ensures that distance between max and min is the same as min and min +1.

- Use for day_of_week, day_of_month, hour_of_day, etc.

Trendlines

- **Instead of encoding: total spend, encode things like: Spend in last week, spend in last month, spend in last year.**

- Gives a trend to the algorithm: two customers with equal spend, can have wildly different behavior — one customer may be starting to spend more, while the other is starting to decline spending.

Closeness to major events

- **Hardcode categorical features like: date_3_days_before_holidays:1**
- Try: National holidays, major sport events, weekends, first Saturday of month, etc.
- These factors can have major influence on spending behavior.

"feature engineering is another topic which doesn't seem to merit any review papers or books, or even chapters in books, but it is absolutely vital to ML success. […] Much of the success of machine learning is actually success in engineering features that a learner can understand."

Scott Locklin

Spatial Variables

- Spatial variables are variables that encode a location in space
- Examples include: GPS-coordinates, cities, countries, addresses

Categorizing location

- Kriging
- K-means clustering
- Raw latitude longitude
- Convert cities to latitude longitude
- Add zip codes to streetnames

Closeness to hubs

- **Find closeness between a location to a major hub**

- Small towns inherit some of the culture/context of nearby big cities

- Phone location can be mapped to nearby businesses and supermarkets

Spatial fraudulent behavior

- **Location event data can be indicative of suspicious behavior**
- Impossible travel speed: Multiple simultaneous transactions in different countries
- Spending in different town than home or shipping address
- Never spending at the same location

“you have to turn your inputs into things the algorithm can understand”

Shayne Miel

Exploration

- **Data exploration can find data health issues, outliers, noise, feature engineering ideas, feature cleaning ideas.**
- Can use: Console, Notebook, Pandas
- Try simple stats: Min, max
- Incorporate the target so find correlation between signal.

Iteration / Debugging

- **Feature engineering is an iterative process: Make your pipelines suitable for fast iteration.**
- Use sub-linear debugging: Output intermediate information on the process, do spurious logging.
- Use tools that allow for fast experimentation
- More ideas will fail, than ideas will work

Label Engineering

- **Can treat a label/target/dependent variable as a feature of the data and vice versa.**
- Log-transform: y -> log(y+1) | exp(y_pred) - 1
- Square-transform
- Box-Cox transform
- Create a score, to turn binary target in regression.
- Train regressor to predict a feature not available in test set.

“Developing good models requires iterating many times on your initial ideas, up until the deadline; you can always improve your models further. Your final models will typically share little in common with the solutions you envisioned when first approaching the problem, because a-priori plans basically never survive confrontation with experimental reality.”

Francois Chollet

Natural Language Processing

- Can use the same ideas from categorical features.
- Deep learning (automatic feature engineering) increasingly eating this field, but shallow learning with well-engineered features is still competitive.
- High sparsity in data introduces you to “curse of dimensionality”
- Many opportunities for feature engineering:

Natural Language Processing

- Lowercasing,
- Removing non-alphanumeric,
- Repairing,
- Encoding punctuation marks,
- Tokenizing,
- Token-grams,
- skipgrams,
- char-grams,
- Removing stopwords,
- Removing rare words
- and very common words,
- Spelling Correction,
- Chopping,
- Stemming,
- Lemmatization,
- Document features,
- Entitity Insertion & Extraction
- Simplification,
- Word2Vec and GloVe / Doc2Vec,
- String Similarity,
- Reading level,
- Nearest Neighbors,
- TF*IDF,
- BayesSVM, Vectorization, LDA, LSA.

Cleaning

- **Lowercasing**: Make tokens independant of capitalisation: “I work at NASA” -> “i work at nasa”.
- **Unidecode**: Convert accented characters to their ascii-counterparts: “Memórias Póstumas de Brás Cubas” -> “Memorias Postumas de Bras Cubas”
- **Removing non-alphanumeric**: Clean text by removing anything not in [a-z] [A-Z] [0-9]. “Breaking! Amsterdam (2009)” -> “Breaking Amsterdam 2009”
- **Repairing**: Fix encoding issues or trim intertoken spaces. “C a s a C a f é” -> “Casa Café”

Tokenizing

- **Encode punctuation marks**: Hardcode “!” and “?” as tokens.
- **Tokenize**: Chop sentences up in word tokens.
- **N-Grams**: Encode consecutive tokens as tokens: “I like the Beatles” -> [“I like”, “like the”, “the Beatles”]
- **Skip-grams**: Encode consecutive tokens, but skip a few: “I like the Beatles” -> [“I the”, “like Beatles”]
- **Char-grams**: Same as N-grams, but character level: “Beatles” -> [“Bea”, “eat”, “atl”, “tle”, “les”]
- **Affixes**: Same as char-grams, but only the postfixes and prefixes

Removing

- **Stopwords**: Remove words/tokens that appear in stopword lists.
- **Rare words**: Remove words that only appear few times in training set.
- **Common words**: Remove extremely common words that may not be in a stopword list.

Roots

- **Spelling correction**: Change tokens to their correct spelling.

- **Chop**: Take only the first n (8) characters of a word.

- **Stem**: Reduce a word/token to its root. “cars” -> “car”

- **Lemmatize**: Find semantic root “never be late” -> “never are late”

Enrich

- **Document features**: Count number of spaces, tabs, newlines, characters, tokens, etc.
- **Entity insertion**: Add more general specifications to text “Microsoft releases Windows” -> “Microsoft (company) releases Windows (application)”
- **Parse Trees**: Parse a sentence into logic form: “Alice hits Bill” -> Alice/Noun_subject hits/Verb Bill/Noun_object.
- **Reading level**: Compute the reading level of a document.

Similarities

- **Token similarity**: Count number of tokens that appear in two texts.
- **Compression distance**: Look if one text can be compressed better using another text.
- **Levenshtein/Hamming/Jaccard Distance**: Check similarity between two strings, by looking at number of operations needed to transform one in the other.
- **Word2Vec / Glove**: Check cosine similarity between two averaged vectors.

TF-IDF

- **Term Frequency**: Reduces bias to long documents.

- **Inverse Document Frequency**: Reduces bias to common tokens.

- **TF-IDF:** Use to identify most important tokens in a document, to remove unimportant tokens, or as a preprocessing step to dimensionality reduction.

Dimensionality Reduction

- **PCA**: Reduce text to 50 or 100-dimensional vector.
- **SVD**: Reduce text to 50 or 100-dimensional vector.
- **LDA**: TF-IDF followed by SVD.
- **LSA**: Create topic vectors.

External models

- **Sentiment Analyzers**: Get a vector for negative or positive sentiment for any text.

- **Topic models:** Use another dataset to create topic vectors for a new dataset.

"So many papers: feature engineering is hard and time consuming. instead here's 8 pages in which we have to design a new weird neural net to do the same thing"

Hal Daume III

Neural Networks & Deep Learning

- **Neural networks claim end-to-end automatic feature engineering.**
- Feature engineering dying field?
- No! Moves the focus to architecture engineering
- And despite promise: computer vision uses features like: HOG, SIFT, whitening, perturbation, image pyramids, rotation, z-scaling, log-scaling, frame-grams, external semantic data, etc.

Leakage / Golden Features

- **Feature engineering can help exploit leakage.**
- Reverse engineer:
  - Reverse MD5 hash with rainbow tables.
  - Reverse TF-IDF back to Term Frequency
  - Encode order of samples data set.
  - Encode file creation dates.
- Rule mining:
  - Find simple rules (and encode these) to help your model.

Case Study: Quora Duplicate Questions Dataset

- **Classify ~440.000 question pairs as duplicate or non-duplicate.**

- **Benchmark 1:** 0.79 accuracy (Stacked Siamese Nets)
- **Benchmark 2:** 0.82 accuracy (Neural Bag of Words)
- **Benchmark 3:** 0.88 accuracy (Bilateral Multi-Perspective Matching)

Case Study: Quora Duplicate Questions Dataset

- **First attempt:** Simple bag of words with logistic regression.

- **0.75 accuracy**

- **Second attempt:** Polynomial feature interactions between the tokens in both questions.

- **0.80 accuracy**

Case Study: Quora Duplicate Questions Dataset

- **Third attempt:** Use stemming with SnowballStemmer from NLTK.
- **0.805 accuracy**
- **Fourth attempt:** Add 2-grams
- **0.81 accuracy**

Case Study: Quora Duplicate Questions Dataset

- **Fifth attempt: Add manual features**
- Normalized difference in length between question pairs
- Normalized compression distance between question pairs.
- Cosine distance between averaged word2vec vectors for the question pairs.
- Chargram co-occurence between question pairs.
- Token count of words: “which, what, where”
- **0.827 Accuracy**

Case Study: Quora Duplicate Questions Dataset

- **Can you think of any more features to engineer?**
- External & pre-trained models?
- Search engine models?
- Logic based models?

Questions?

This one went unusually smoothly. When I finished it, I remarked to a friend that I felt like an engineer who had designed a machine and then sat back and realized it did everything I'd set out to do.

Which made him say, quite emphatically, "No engineer has ever felt this."

Robert J. Bennett

Resources & Further Reading

- **Kaggle forums & kernels:** Far0n, KazAnova, Fchollet, Abhishek, Gilberto Titericz, Leustagos, Owen Zhang, Gert Jacobusse …
- **Introduction:** http://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/
- **Books:**
  - Mastering Feature Engineering (Alice Zheng),
  - Feature Extraction (Isabelle Guyon et al.)
- **Blogs**:
  - https://smerity.com/articles/2016/architectures_are_the_new_feature_engineering.html
  - http://hunch.net/~jl/projects/hash_reps/
  - https://blogs.technet.microsoft.com/machinelearning/2014/09/24/online-learning-and-sub-linear-debugging/
  - http://blog.kaggle.com/2015/12/03/dato-winners-interview-1st-place-mad-professors/
  - http://blog.kaggle.com/2016/08/24/avito-duplicate-ads-detection-winners-interview-1st-place-team-devil-team-stanislav-dmitrii/
  - http://www.slideshare.net/DataRobot/featurizing-log-data-before-xgboost
- **Data:** https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs
- **Software:** https://github.com/trevorstephens/gplearn