Feature Engineering (Guest Lecture)

Pablo Duboue, PhD

CMPT 733: Big Data Programming II, SFU
March 23rd, Vancouver

Outline

Lecture in Detail

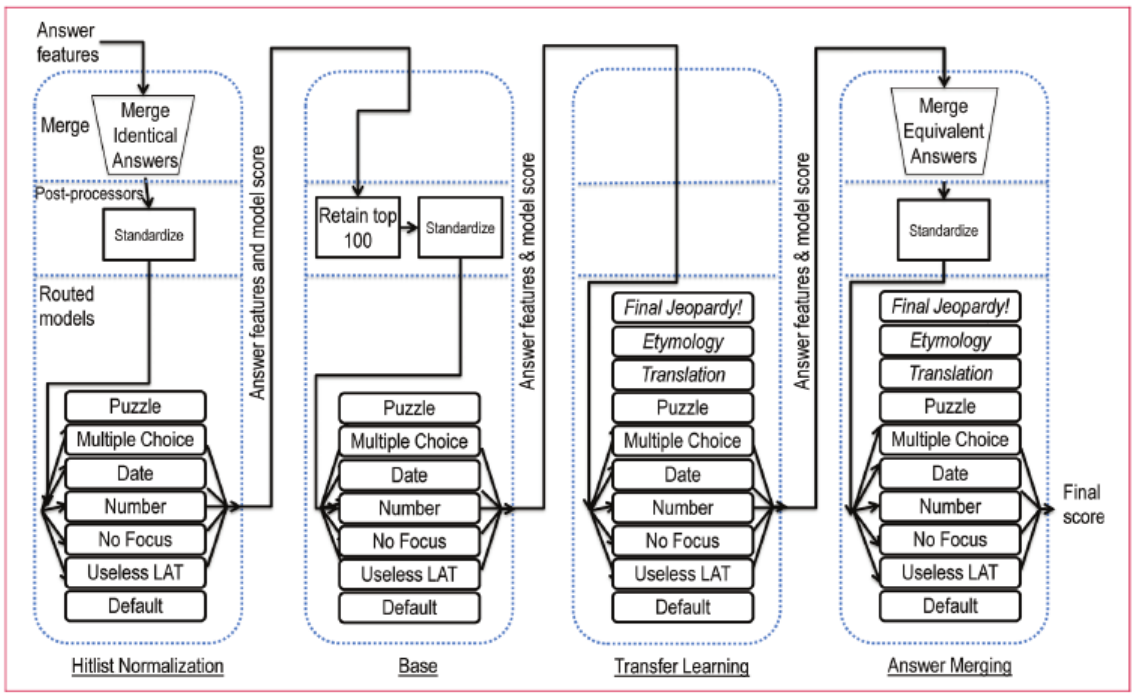
1. Concepts (expanding on lec. 5)
   - ML and Feature Engineering cycles
   - EDA and Error Analysis
   - Importance of domain knowledge (expanding on lec. 7)
2. Features combined and expanded (expanding on lec. 6)
   - Normalization, Discretization
   - Computable Features, Imputation and Decomposition
3. Features reduced and projected (expanding on lec. 6)
   - Feature Selection
   - Dimensionality Reduction (hashing features, clustering, random projections)
4. Closing (expanding on lec. 7)
   - Variable-length raw data, learning more

About the Instructor

- PhD in Computer Science, Columbia University, NY
  - Thesis on weakly supervised learning for text generation
- IBM Research Watson
  - Deep QA - Watson - Jeopardy! Show
- Good ML Practitioner
  - NLP uses tons of feature engineering
  - Consulting company in Vancouver (started it in Montreal from 2010-2016)
- Research contributions
  - Three workshop papers in 2019, including a poster at NeurIPS
  - This year reviewed for AAAI, ACL, INLG, and EMNLP
- Wrote a book on Feature Engineering coming out September 2020

My interest in Feature Engineering

- Jeopardy! show requirements
  - Machine Learning ought to be fast
  - Combines information from multiple sources



Gondek, Lally,Kalyanpur, Murdock, Duboue, Zhang, Pan, Qiu, Welty
(IBM Journal of R&D, 2012)

ML CYCLE

1. Exploratory Data Analysis (*Raw data*)
   - *Data insights*
   - *Model to train*
   - *Metrics to evaluate*
2. Featurizer (*Raw data*)
   - *Feature vectors*
3. Train/test split (*Feature vectors*)
   - *Trainset*
   - *Testset*
4. Build model (*Trainset*)
   - *Trained model*
5. Evaluate(*Trained model*, *Testset*)
   - *Evaluation results*

What is Feature Engineering

- Feature Engineering is the **process** of
  - **representing** a problem domain as to make it
  - **amenable for learning** techniques.
- This process involves
  - the **initial discovery** of features and their
  - **step-wise improvement**
- based on
  - **domain knowledge** and the
  - **observed performance** of a given ML algorithm over specific training data.

Trading Amount of Data with Human Time

- In computer science many times, it is possible to trade **time** with **space** and viceversa
  - Speed things up by using a cache (that uses more RAM)
  - Use less RAM by doing multiple passes on the data (which takes more time)
- Similarly, if you have large amounts of data, it might be possible to dispense with Feature Engineering
  - The premise of Deep Learning
  - Synthesize better feature representations in neural network layers
- Adding human knowledge helps solving problems using computers
  - Does not help building autonomous intelligent systems

Good Features

1. **Informative**
   - the feature describes something that makes sense to a human
2. **Useful**
   - the feature is defined for as many instances as possible and
3. **Discriminant**
   - the feature divides instances into the different target classes.

Feature Engineering Cycle

1. Final evaluation split(Raw data)
   - *Raw data for feature enginering*
   - *Raw data for final test*
2. **Iterate** over ML cycle(*Raw data for feature engineering*)
   - *Featurizer*
   - *Trained model*
3. Featurize(*Featurizer*, *Raw data for final test*)
   - *Feature vectors for final test*
4. Evaluate(Trained model, Feature vectors for final test)
   - *Final evaluation*

- **Feature engineering can overfit terribly and harm your ML.** Use data in stages to avoid that.

EDA

- You have seen EDA in lec. 5
- Exploratory Data Analysis
- Analyze data sets to summarize their characteristics
  - Usually through visual means
- Help formulate hypothesis about the data
- In the ML case
  - Help pick a ML model
  - Help with initial featurization

Error Analysis

- Looking at aggregate metrics is a good start but for feature engineering you want to look in detail
- Identify individual erroneous instances or classes of instances that contribute substantively to the error
  - Using the contingency table
  - And a properly instrumented system
- Drill down on hypothesis for the reasons behind the error
- Assess whether such hypothesis will *actually* cover a lot of error cases
  - Do not overoptimize
  - Do not overengineer

Feature Drilldown

- Based on the Error Analysis results you might want to...
  - Drop poorly performing features
    - Simpler models have less parameters
    - They are more informed over the same amount of data
  - Expand good ones
    - Provide variations over them
    - Combine them
- Go back to your raw data and do more EDA on it to hypothesise new features

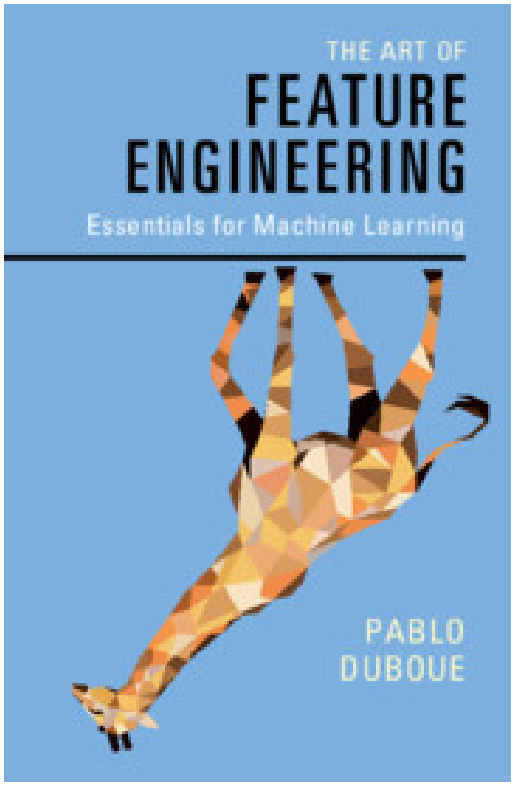
Feature Ideation

- Expanding on the discussion in lec. 5

1. Throw everything (and the kitchen sink) at it
2. Stop and think
   - 2.1 What information would **you** us to solve that problem?
   - 2.2 Look for published work
     - Papers: http://aclweb.org/anthology-new/
     - Blog postings
     - Open source projects
3. Add computable features
   - Learning to sum takes an incredible amount of training!

The Art of Feature Engineering

- Focus
  - Helping practitioners
  - People already familiar with ML algos
  - Helping also approach new domains
- Material seldom presented in book format
- Coming early 2020 through Cambridge University Press
- 286 pages, 330 references, 10 chapters
- 10,000 lines of Python in 5 case studies
  - Available open source at `http://artfeateng.ca`



Food for Thought

[C]onstruct your own toolbox [...] Then, instead of looking at a hard job and getting discouraged, you will perhaps seize the correct tool and get immediately to work.

Stephen King, "On Writing"

WikiCities

- WikiCities dataset
  - a resource put together for the book
  - to be presented at the ML for All workshop at NeurIPS
  - Goal: exercising different Feature Engineering techniques
- Predicting the population of towns and cities based on:
  - Their properties
  - Their textual description
  - Their historical population
  - Their satellite imagery

Outline

Combining Features

- How to consider all the features together
  - Normalizing them, so their variabilities are meaningful
  - Finding discrete representation to continuous values
  - Computing representative statistics over larger amounts of feature data

Normalizing Features

- Feature Normalization is a great way of reducing the variations on feature values
  - less *nuisance variations*
  - concentrate on variations that contain a signal to the target class
- It also achieves feature values on a specific range
  - For example, floating point between 0 and 1 (or -1 and 1)
  - Crucial for SVMs and ANNs that need input data scaled on certain values
- You have seen normalization (feature scaling) in lec. 5:
  - Scaling, centering, scaling to unit length and standardization

Decorrelation, Smoothing and Feature Weighting

- For signals acquired from sensor data, it is common to have artifacts, like repetitions of the past data (echo in the case of acoustic data)
- Decorrelation is a technique to reduce such artifacts
  - Usually by discounting past versions of the data over the current data
  - A type of linear filter
- Smoothing: assume feature perturbed by indepedent
  - Bring the feature value closer to its true value, by comparing it to other feature values in the *vicinity* of the instance
  - For space, nearby points should have similar feature values: take the average
- Feature weighting: boost certain features based on domain knowledge
  - TF-IDF, camera callibration

Discretization

- Discretization is the process of transforming a continuous signal into discrete values
  - Every discretization incurs a discretization error
  - We expect that discretization will help reduce the number of parameters from the machine learning model
    - Thus boosting the signal in the training data
    - Improving generalization
    - less *nuisance variations*
- Discretization can be supervised (using the target class) or unsupervised
  - Unsupervised: truncation, rounding, binning (equal frequency or equal width)

Supervised Discretization

- There are a number of supervised discretization algorithms proposed
- They will keep meaningful differences among feature values separated
  - It is important to estimate its parameters in held-out data
    - Otherwise they will be preferred by the ML as a feature leak
- Of note: MDLP and CAIM, both examples of top-down **adaptive quantizers**
  - Observed feature values are sorted, then split recursively, starting from an interval equal to the whole range
  - Criteria to choose where to split an interval and whether to continue splitting
  - Candidate places to split (*cut-points*) are the observed values or a "smart" subset of them
  - Do not require number of internals known beforehand
  - You can even train a full classifier on each partition and choose the cut-point where the classifier performs better.

Histograms

- A histogram is a simple representation of the distribution of values for a set of features
- It is amply used in image processing
  - It transforms feature values to frequency values
  - It can handle continous data by using bins to define each of the frequency columns
- Histograms are usually associated with images
  - However, a popular representation for text processing (BoW) is a histogram of words
- Computing a histogram is a form of moving from the observational domain to a frequency domain
  - Other forms of applying this idea exist in the form of Fourier Transformation

Descriptive Features

- Maximum, Minimum, Average, Length
- Descriptive features are not named that way for all domain and problems
  - A very common (and succesful) feature in natural language processing is text length
  - It can predicts correctly many classes of interest
    - For example, is the customer happy or unhappy?
    - Unhappy customers tend to leave much longer reviews, with plenty of details to justify their unhappiness
- Other Statistics
  - skewness: lack of symmetric in the distribution
  - kurtosis: whether the data is heavy tailed compared to a normal distribution

Dense vs. Sparse Feature Values

- Using descriptive features has the advantage of being reliable, dense features
  - For example, if the user comment is in a language different from the trainset, the length feature will still be informative
- Dense feature vectors, however, might slow down considerably machine learning algorithms designed to operate over sparse feature vectors

Expanding Features

- Poor domain modelling results in too many features
  - Feature reduction (e.g., feature selection) is key
- However, many times adding new features can have plenty of value
- We will discuss
  - Computable features
  - Imputation
  - Smoothing

Computable Features

- The simpler way of expanding features is by operating over them
  - That is, by computing new features using the existing ones as input
  - These small programs encode domain knowledge about the problem and dataset
- For example,
  - if we believe the multiplication between two features might be a better predictor to the target class than any of the two features alone
  - we might want to add an extra feature that contains the result of that operation
  - more so if multiplying two features is not part of the representation used by the machine learning algorithm (e.g., Naive Bayes)

Single Feature Transformations

- In health domain, too heavy or too light is a problem, take the sqrt
- Proportional features? Take the log
- Operator palette, $e^x, \log x, x^2, x^3, \tanh x, \sqrt{x}$, sigmoid
- Computable features are more suited for linear models (e.g., Logistic Regression)
- You have seen plenty of computable features operations in lec. 7
- One Hot Encoding (discussed in lec. 6) falls in this category

Computable Features and DL

Name	Expression	Error			
		std.dev.	min	mean	max
Difference	$x_1 - x_2$	0.788	0.013	0.410	0.814
Logarithm	$\log(x)$	0.721	0.052	0.517	0.779
Polynomial	$2 + 4x_1x_2 + 5x_1^2x_2^2$	0.810	0.299	0.770	1.142
Power	x^2	0.001	0.298	0.299	0.300
Ratio	x_1/x_2	0.044	27.432	27.471	27.485
Ratio Difference	$(x_1-x_2)/(x_3-x_4)$	0.120	22.980	23.009	23.117
Ratio Polynomial	$1/(5x_1^2x_2^2+4x_1x_2+2)$	0.092	0.076	0.150	0.203
Square Root	$\sqrt{x}$	0.231	0.012	0.107	0.233

from Heaton (2017)

Drilling down on a feature

- If you have a feature that seems effective, consider providing variations over it
  - If it is binary, transform it on a probability based on how well it correlates with target classes
  - If it is categorical, consider reducing the number of categories (by adding an "other" category)
  - If it is discrete, threshold it or split it into bins
  - If it is continuous, discretize it or apply a squashing function to it
- Realize that the meaning of a feature might change due to refinement

Target Rate Encoding

- If 80% of the times a binary feature is true then the binary target class is also true, instead of having the original feature as true or false, set it to 0.8
- If there are multiple target categories, have a target rate for each of the categories
- Might need to estimate these firing rates on a separate set from the training set
  - Otherwise the machine learning will trust these features too much
- An alternative is to let go of the target altogether and just represent each categorical by its frequency counts
  - The fact that it is a rare or a frequent category might be more informative than the category itself

Target Rate Encoding: Example

Dataset

F	T
A	+
B	-
C	+
A	+
B	+
C	+
A	+

→

F	T
A	+
B	-
C	-
A	+
B	-
C	-
B	+

→

F	TRE
A	1.0
B	**0.4**
C	*0.5*

TRE dataset

→

F_{TRE}	T
1.0	+
0.4	-
0.5	+
1.0	+
0.4	+
0.5	+
1.0	+

F_{TRE}	T
1.0	+
0.4	-
0.5	-
1.0	+
0.4	-
0.5	-
0.4	+

Full example: `https://github.com/DrDub/artfeateng/blob/master/Chapter6.ipynb`, Cell #30

Other Techniques

- Coalescing Categories
  - For categorical features with many categories, not all categories might be needed
    - Dropping them might help amplify the signal
    - Seek to coalesce infriquent categories or categories that will present a rather poor Correct Firing Rate
  - The coalesced categories then go to a "other" category
- Winsorising (thresholding)
  - A simple way to binnarize or coalesce ordered feature values is to apply a threshold over them
    - Values below the threshold become an indicator feature with a value of false
- Changes of the coordinate system
  - Cartesian to polar (if the angle is meaningful, Cartesian coordinates will not be very helpful)
  - RGB to HSV (If the luminisance is meaningful, pure red, green and blue values will not be very helpful)

Expading Features using External Data

- Many times there will be value to add external data from separate sources
- This is equivalent to add common sense knowledge
- For example, if the problem involves Geographical data, adding distance to major cities might help
  - For example, if the target is to predict fuel prediction during a road trip
- This is very domain dependent

Full example: `https://github.com/DrDub/artfeateng/blob/master/Chapter10.ipynb`, Cell #25

Handling Missing Data

- Many times the data has instances where a particular feature value is unknown
- This might be due to artifacts on the acquisition, merging from different data sources or limitations with the feature extractors
- Some machine learning libraries (e.g., Weka) have excellent functionality to deal with missing data
  - Others (e.g., scikit-learn) have very poor tools
- Beware of its impact

Missing Data Indicator Feature

- A first approach to handle missing data is to explicitly signal to the machine learning algorithm about the existence of the missing feature
- The feature vector is thus expanded with an indicator feature for each feature that might contain missing values
  - Those features will be true when the value was missing in the original data
  - This feature is indepdent of whether the value is replaced by a default (imputed) value as discussed next

Imputation

- Imputation is the process of choosing what value to use to complete a missing value
- In the simpler case, the median value for the feature can be used
- In the most complex case, a classifier can be trained on the remaining features to predict the value of the missing feature
- In practice, a combination of both approaches can be used, employing a simple threshold classifier
  - But the median approach is still better than leave the feature as "0" as with scikit-learn

Full example: `https://github.com/DrDub/artfeateng/blob/master/Chapter7.ipynb`, Cell #7

Decomposing Complex Features

- Machine learning algorithms expect their features to be low-level values
  - Not unlike database columns
- Many times, raw data contains fields were multiple information has been aggregated
- This step involves decomposing such features
- By far the most common case is date/time values
  - While it is possible to have enough data to learn what ranges of seconds from January first 1970 constitute weekends, it is doubtful you will have such data available
  - Instead, if your feature is a unix date time, expand it as multiple features for day, day of the week, etc
- Strings can also profit from decomposition, for example, realizing that 36f is an age/gender encoded string

Pivoting

- Another technique to decompose complex features is to apply a pivot
  - Transform data rows into an aggregated, larger tuple

Original Data

ID	Activity	Target
1	read	n
1	save	n
1	search	y
2	save	n
2	read	n
3	save	n
4	search	n
5	read	y
5	search	y

$\rightarrow$

Data after Pivot

ID	Read?	Save?	Search?	Target
1	y	y	y	y
2	y	y	n	n
3	n	y	n	n
4	n	n	y	n
5	y	n	y	y

Tidy Data

- The decomposition of complex features aligns itselfs with the goals of the Tidy Data concept in Big Data (Hadley, 2013):
  1. Each feature should be in one column
  2. Each instance should be in one row
  3. There should be one table for each kind of feature
  4. If there are multiple tables, they should have a column that allows them to be linked

Outline

Reducing Features

- Central to feature engineering are efforts to reduce the number of features
- An excess of uninformative features bloats the machine learning model with unnecessary parameters
- Too many parameters then either produces suboptimal results or require large amounts of training data
- These efforts are either by
  - Explicitly dropping certain features (**feature selection**)
  - Mapping the (usually sparse) feature vector into a lower (much dense) dimension (**dimensionality reduction**)

Curse of Dimensionality

- The main issue with higher dimensional spaces is that everything starts to become very close, even if just by chance
- That is the case as meaningful differences on a few key coordinates are drown upon similar values for the remaining coordinates
- Moreover, the more dimensions are present in the data, the more training data is needed
  - A rule of thumb says about 5 training instances per dimension

Feature Selection

- It is common to approach a machine learning problem from instances derived from databases or other structured data sources
- These instances contain spurious columns (in the sense that they are unrelated to the target)
  - Social security numbers, unrelated IDs, etc
- The machine learning algorithm will have to sort out through large amounts of data to learn that these columns are irrelevant
  - Worse yet, some algorithms (like Naive Bayes) will have to use all features, irrespective of whether they are relevant or not
  - Plenty of value of pre-filtering the features
- Also, a model with fewer features is easy to understand by humans
  - You have seen feature selection in lec. 6

Defining Feature Utility

- Key to filtering features is to measure how well a particular subset behaves
- On the simpler case, we can define a metric over each individual feature
  - Fast, but no interactions
- However, in the general case we know many times features operate jointly to predict a class
  - For example, if the target class is a square in a two dimensional feature space, both features will be needed
  - In the worse case, it might be XOR of all the features
    - Theoretical worse case

Feature Filtering

- Maximum Likelihood Estimator
  - How well the feature (alone) predicts the target class
- Chi-Square correlation between the feature and the target class
- TPR, TFR (true positive rate, false positive rate)
- Firing rate (how many times it is defined)

Feature Utility: Mutual Information

- Mutual information is the amount of bits of information about a second random variable that are known by knowning the value of a first random variable
  - Related to information theory's **entropy** of a random variable, the amount of information held by the variable

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \left(\frac{p(x,y)}{p(x)p(y)} \right)$$

- To define use MI as a feature utility metric, we seek to compute the mutual information between the feature being evaluated and the target class
- The probabilities in the formula can be estimated using counts on the train data

Random Feature Cut-off

- When operating directly with feature utility metrics, it is sometimes necessary to define a "minimum utility value" to help drop features and stop greedy methods (discussed next)
- One such technique is to incorporate a completely random feature to the set and evaluate it together with the other features
  - The utility values obtained for this artificial feature can be used as a cut-off value

Greedy Methods

- Finding the best feature subset is NP-hard
- Instead, we can define an iterative process using heuristics
  - We can start from an empty set and add one feature at a time (**forward feature selection**)
  - We can start from the full set and drop one feature at a time (**backward feature selection**)
- At each step, we can use the metrics directly or retrain the full system (**wrapper methods**)

Forward Feature Selection

- Evaluate each individual feature
- Start from the feature that better predicts the target class by itself
- Keep adding features inasmuch the selected set of features keeps improving
  - either on the utility metrics or when run end-to-end
- Also called "incremental construction" of the feature subset

Backward Feature Selection

- Start from the full set
- Either drop the feature with less computed utility or evaluate the system end-to-end without each feature
- Keep dropping features as long as the performance improves or stays the same
  - That is, stop when it gets worse
- Also called "ablation study" of the feature set

Full example: `https://github.com/DrDub/artfeateng/blob/master/Chapter6.ipynb`, Cell #28

Wrapper Methods

- As mentioned, greedy methods can be used either with direct feature utility metrics or by training end-to-end with each subset
- Training end-to-end is more informative but it is very onerous
  - Some algorithms allow re-using information from one run into the other

Blacklisting Features

- A simpler approach to feature selection involves having a list of features known to be bad in particular domain and always exclude them
- The most popular such approach in NLP and IR is to use lists of "stopwords", function words that contain little to no semantic information content for theme assignment of full documents
  - For example "for, a, to, in, them, the, of, that, be, not, or, and, but"
  - Of course function words are very important for human understanding of documents and there are expressions only composed of stop words ("to be or not to be")

Embedded Methods

- Some machine learning algorithms can perform feature selection as part of their system
- The most common manner this is achieved is by regularization, described next

Regularization

- Another way to reduce the number of features employed by the model is not to remove them from the training data but to discourage the model from using too many features
- Extend the evaluation function for the different representations being searched by the machine learning model to include a penalty score based on the complexity of the model
  - Different penalties produce different regularization approaches (compare: how many features being used with sum of the weights for all the features being used)
- These techniques are called regularization techniques and are not easily applicable to all machine learning models

ML Algorithms that regularize with ease

- Certain algorithms are regularized more easily than others
  - Neural networks
- Moreover, some algorithms can only operate under strong regularization as they are mathematically ill-defined
  - Logistic Regression
- In order to regularize we need to be able to extract the parameters of the representation in an explicit manner so as to compute its size

Implicit Feature Selection

- In general, regularization techniques are a particular case of machine learning algorithms that perform an implicit feature selection
- For example, neural networks in deep learning now are trained with dropout, a technique where a percentage of the connections are hardwired to zero in each training step
  - Different connections at each step
- Winnow (Littlestone, 1988) learns a linear classifier from labelled examples
  - Similar to perceptron
  - Uses a multiplicative scheme that handles many irrelevant dimensions
  - Simple algorithm that scales well to high-dimensional data
  - Feature selection is embedded into the algorithm, during its update step
    - Embedded feature selection that does *not* use regularization
  - Algorithm of choice to use on very large datasets

Dimensionality Reduction

- Another technique to reduce the number of features is to find a related representation for the input features in a lower dimensional space
- This process has to be able to accommodate unseen data at testing/execution time and to generalize properly from there

Hashing Features

- The simpler dimensionality reduction technique is just to apply a hashing function
  - Either from data structures or from cryptography
  - For example SHA1
  - A hashing function is any function that maps data of arbitrary size to a fixed size
    - Non-identical input data that maps to the same hash are called "collisions"
    - Good hashing functions minimize collisions
- The intuition is that collisions will be rare and that when a collition happens other hashed features will inform the machine learning which of the original features is being seen
- It is incredible that it works as effectively as it does
  - A key component of the fastText machine learning system

Hashing Features: Example

- Take the last byte of md5sum for /usr/share/dict/spanish (reduction from 86,016 to 256 features)

```
cat /usr/share/dict/spanish | perl -e 'use Digest::MD5
qw(md5_hex);while(<STDIN>){chomp;$d=md5_hex($_);$d=substr($d,0,2);print
"$d $_\n"}'|sort
```

- The following words have the same hash (around with 300 other words):

00 abstersiva
00 acaudalar
00 aceleradamente
00 aciago
00 adenopatía
00 admixtión
00 adocenar
00 agostadero
00 ahoguío

Other Issues with Feature Hashing

- Hashing Features as Polysemy
  - When using Bag-of-Words approaches to Natural Language Processing, hashing features is equivalent to artificially increase the polysemy of words
  - If door and banana map to the same hash, for the machine learning, it is like English now has a bananadoor word and it has to distinguish between the two senses on a given instance
    - Not unlike if the word was bank (river bank vs. money bank)
- Interpretability (discussed in lec. 7) suffers highly when using feature hashing
  - And dimensionality reduction in general
- Beware python hash function is now randomized for security reasons
  - You will get different hashes in different executions

Full example: `https://github.com/DrDub/artfeateng/blob/master/Chapter8.ipynb`, Cell #14

Random Projection

- Also in the "very simple method that can curiously work sometimes" is to use a explicit projection matrix from the original feature space to a reduced space... at random
- The methods we will see next compute such matrices based on a number of desiderata
  - However, many times the value is in executing a projection
  - If the dimensions and training data is very big, computing a "good" projection might be prohibitively computationally expensive
- Note that a random projection is an actual projection
  - Random hashing is much more random than a random projection

Singular Value Decomposition

- A more meaningful projection can be found using the "direction" of the dataset
- And keeping only the directions that are stronger
  - Where the directions are the eigenvectors of the decomposition and their strength is given by their eigenvalues
- Also discussed in terms of PCA (Principal Components Analysis)
  - The stronger directions are the principal components
- Discussed in lec. 6

SVD: Details

- Computing the eigenvectors of the covariance matrix and their eigenvalues can be done with a numerical package
- The projection matrix is V and the dimensionality reduction is achieved by truncating V to the larger eigenvalues
- If the truncation only keeps the top r eigenvalues, then $\tilde{V}$ is a matrix of size $r \times r$ and we are reducing the number of features from m to r
- The projected representation of an instance i (row i of M) is the i row of $M\tilde{V}$
  - New data can be encoded by multiplying with $\tilde{V}$

Clustering

- Another dimensionality reduction technique (unrelated to the rest) is to cluster the data and use the number of the cluster as the feature itself
- For new data, we need a way to assign clusters to unseen data
  - For example, in k-Means we can pick the cluster with a sorter distance to its centroid
- The amount of dimensionality reduction achieved is equivalent to the number of clusters used
  - Assuming one hot encoding of the cluster
- Clustering techniques that assign overlaping clusters (e.g., Canopy clustering) can be used to provide dense representations
- Using synonyms of words (e.g., WordNet) is a way of using clusters

Outline

Variable Length Feature Vectors

- Many problems are easily expressed with complex features of variable length
- We will see some techniques to deal with such data when it comes in the form of:
  - Lists
  - Sets
  - Trees

Sets

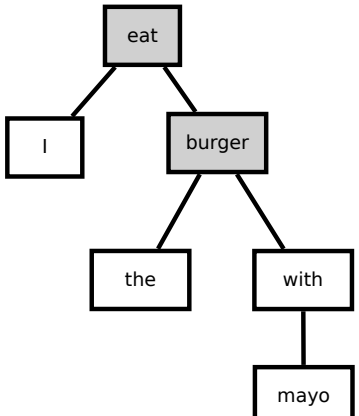
- The simpler case of variable feaure length complex features are sets
  - Multiple elements without ordering but the number of possible elements to be found is unknown
- List of Indicator Features
  - Transform the set into a list of indicator features for each of the potential elements in the set
  - Only available for sets of categorical features
- Addition and Multiplication
  - Use a machine learning model that can accommodate associative operations such as additions and multiplications
  - A common way to represent a set of word embeddings: add the vectors
- Sort the set, then encode it as a list

Lists

- Representing lists has two major issues:
  - Nuisance variations in terms of exact positions
  - Variable length itself
- Truncation to a fixed length, pad shorter lists with "null"
  - This will work if the position has an exact meaning
    - Fails for language, compare "cancel my account" with "please I would like you to cancel my account"
  - We can truncate at the beginning, at the end or from a designated marker
- Fixed Regions Histogram
  - Certain elements appeared at the beginning of the list vs. at the middle or at the end
- n-Grams
  - Unordered features encoding local ordering
  - The feature is a tuple that represents a fixed-size (of size n) window over the list

Trees

- Representing trees is further complicated with respect to lists as it includes a parent-node relation
- In a simpler case, it is possible to linearize the tree (for example, using a depth-first traversal) and encode the resulting list using the techniques discussed before
- Parent-Node pairs
  - A similar approach to using n-grams is to encode all parent-node relations as pairs



eat-I, **eat-burger**, burger-the, burger-with, with-mayo

Domain Dependence

- Subject matter experts know which data and features are important
- Domain independent approaches can produce good models very quickly, but...
  - "If you want to go fast, go alone; but if you want to go far, go together."
    - And if you want to go really fast, ask somebody who knows the way.
- Tapping into domain expertise will allow you to get much farther than if you need to reinvent the wheel for every little quirk and problem in your target domain

On Feature Engineering Success

- Not all the techniques presented in the case studies in the book worked well on the data at hand
  - Feature engineering involves much trial and error and that is reflected on these case studies
- At about 50% success rate, these case studies still paint an optimistic view of feature engineering
  - A rate between 10-20% is more likely

Food for Thought

Tools often redefine a problem.

Dr. Ursula Franklin on her 1989 CBC Massey Lectures "The Real World of Technology" (1989)

Learning more and an Invitation

- Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists (2018)
  - by Alice Zheng, Amanda Casari
- Feature Engineering for Machine Learning and Data Analytics (2018)
  - edited by Guozhu Dong, Huan Liu
- Meetup Learn Data Science Reading group
  - https://www.meetup.com/LearnDataScience
  - "Deep Double Descent: Where Bigger Models and More Data Hurt" `https://arxiv.org/abs/1912.02292`
  - 6:00pm, ONLINE, March 25
- Keep in touch!
  - @pablodubоue on TW