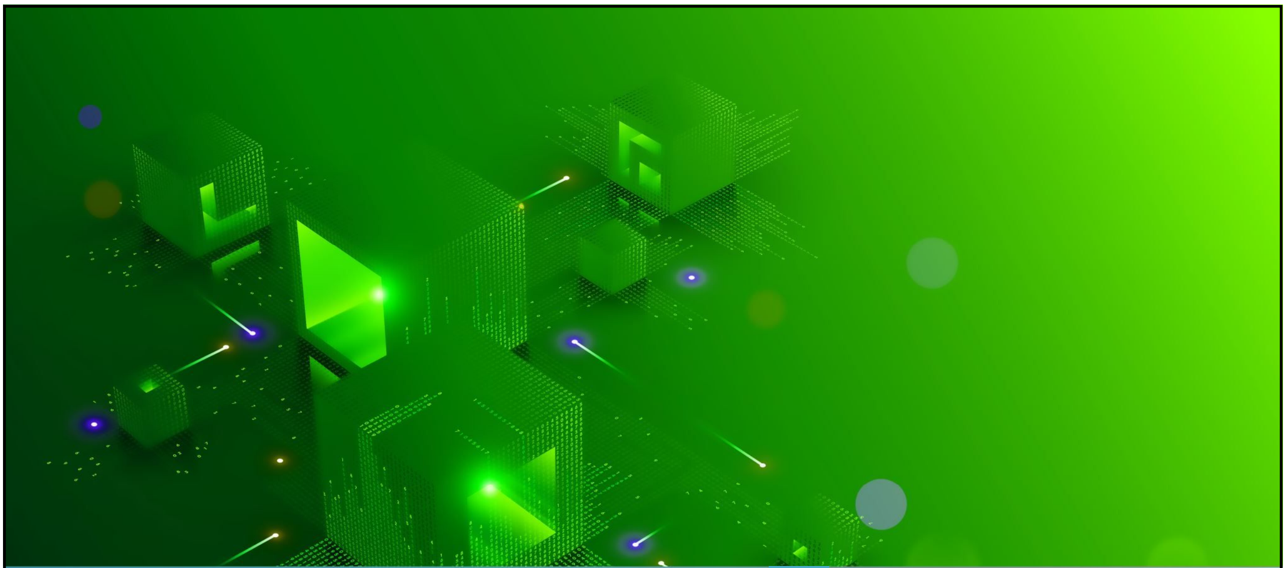


FEATURE SELECTION AND ENGINEERING
MODULE 6

Feature Selection: Filters

> Given *n* original features, how do you select size of subset
  - User can preselect a size *p* (< *n*) – not usually as effective
  - Usually try to find the smallest size where adding more features does not yield improvement

> Filters work independent of any learning algorithm

> Filters seek a subset of features which maximize some type of between class separability – or other merit score

> Can score each feature independently and keep best subset
  - e.g., 1st order correlation with output, fast, less optimal

Feature Selection: Filters

> Can score subsets of features together
  – Exponential number of subsets requires a more efficient, sub-optimal search approach
  – How to score features is independent of the ML model to be trained on and is an important research area
  – Decision Tree or another ML model pre-process

Feature Selection: Wrappers

> Optimizes for a specific learning algorithm
> The feature subset selection algorithm is a "wrapper" around the learning algorithm
  1. Pick a feature subset and pass it to learning algorithm
  2. Create training/test set based on the feature subset
  3. Train the learning algorithm with the training set
  4. Find accuracy (objective) with validation set
  5. Repeat for all feature subsets and pick the feature subset which gives the highest predictive accuracy

> Basic approach is simple
> Variations are based on how to select the feature subsets, since there are an exponential number of subsets

Feature Selection: Wrappers

> Exhaustive Search - Exhausting

> Forward Search – O(n^2 · learning/testing time) - Greedy

1. Score each feature by itself and add the best feature to the initially empty set *FS* (*FS* will be our final Feature Set)
2. Try each subset consisting of the current *FS* plus one remaining feature and add the best feature to *FS*
3. Continue until stop getting significant improvement (over a window)

Feature Selection: Wrappers

> Backward Search – O(n^2 · learning/testing time) - Greedy

1. Score the initial complete *FS*
2. Try each subset consisting of the current *FS* minus one feature in *FS* and drop the feature from *FS* causing least decrease in accuracy
3. Continue until dropping any feature causes a significant decreases in accuracy

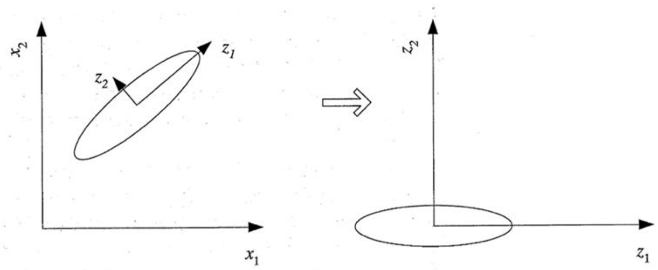
> Branch and Bound and other heuristic approaches available

PCA – Principal Components Analysis

> PCA is one of the most common feature reduction techniques
> A linear method for dimensionality reduction
> Allows us to combine much of the information contained in *n* features into *p* features where $p < n$
> PCA is *unsupervised* in that it does not consider the output class/value of an instance – There are other algorithms which do (e.g. Linear Discriminant Analysis)
> PCA works well in many cases where data features have mostly linear correlations
> Non-linear dimensionality reduction is also a successful area and can give better results for data with significant non-linear correlations between the data features

PCA Overview

> Seek new set of bases which correspond to the highest variance in the data
> Transform *n*-dimensional *normalized* data to a new *n*-dimensional basis
  – The new dimension with the most variance is the first principal component
  – The next is the second principal component, etc.
  – Note z_1 combines/fuses significant information from both x_1 and x_2
> Can drop dimensions for which there is little variance



Variance and Covariance

> Variance is a measure of data spread in one feature/dimension
  – *n* features, *m* instances in data set
  – Note *n* in variance/covariance equations is number of instances in the data set, apologies

> Covariance measures how two dimensions (features) vary with respect to each other

> Normalize data features so they have similar magnitudes else covariance may not be as informative

$$\mathrm{var}(X) = \frac{\sum_{i=1}^{n}\left(X_i - \overline{X}\right)\left(X_i - \overline{X}\right)}{(n-1)}$$

$$\mathrm{cov}(X,Y) = \frac{\sum_{i=1}^{n}\left(X_i - \overline{X}\right)\left(Y_i - \overline{Y}\right)}{(n-1)}$$

Covariance and the Covariance Matrix

> Considering the sign (rather than exact value) of covariance:
  – Positive value means that as one feature increases or decreases the other does also (positively correlated)
  – Negative value means that as one feature increases the other decreases and vice versa (negatively correlated)
  – A value close to zero means the features are independent
  – If highly covariant, are both features necessary?

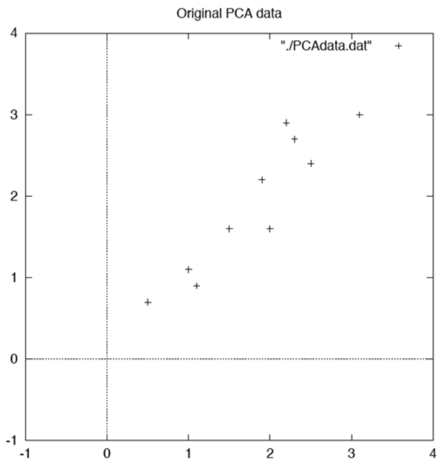
> Covariance matrix is an $n \times n$ matrix containing the covariance values for all pairs of features in a data set with *n* features (dimensions)

> The diagonal contains the covariance of a feature with itself which is the variance (i.e. the square of the standard deviation)

> The matrix is symmetric

PCA Example

- First step is to center the original data around 0 by subtracting the mean in each dimension – normalize first if needed



Data	*x*	*y*	*x'*	*y'*
	2.5	2.4	0.68	0.49
	0.5	0.7	-1.32	-1.21
	2.2	2.9	0.38	0.99
	1.9	2.2	0.08	0.29
	3.1	3.0	1.28	1.09
	2.3	2.7	0.48	0.79
	2.0	1.6	0.18	-0.31
	1.0	1.1	-0.82	-0.81
	1.5	1.6	-0.32	-0.31
	1.2	0.9	-0.62	-1.01
Mean	1.82	1.91	0	0

PCA Example

- Second: Calculate the covariance matrix of the centered data
- Only 2 × 2 for this case

Data	*x*	*y*	*x'*	*y'*
	2.5	2.4	0.68	0.49
	0.5	0.7	-1.32	-1.21
	2.2	2.9	0.38	0.99
	1.9	2.2	0.08	0.29
	3.1	3.0	1.28	1.09
	2.3	2.7	0.48	0.79
	2.0	1.6	0.18	-0.31
	1.0	1.1	-0.82	-0.81
	1.5	1.6	-0.32	-0.31
	1.2	0.9	-0.62	-1.01
Mean	1.82	1.91	0	0

$$\mathrm{cov}(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{(n-1)}$$

Covariance	Matrix
0.60177778	0.60422222
	0.71655556

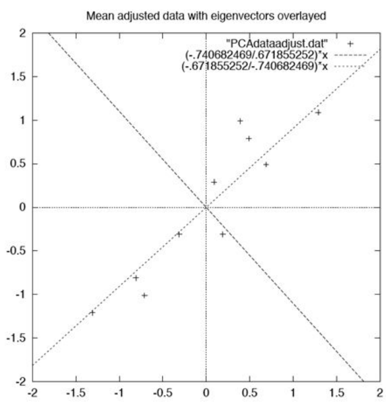
PCA Example

> Third: Calculate the unit eigenvectors and eigenvalues of the covariance matrix (remember your linear algebra)

- Covariance matrix is always square $n \times n$ and positive semi-definite, thus n non-negative eigenvalues will exist
- All eigenvectors (principal components) are orthogonal to each other and form the new set of bases/dimensions for the data (columns)
- The magnitude of each eigenvalue corresponds to the variance along each new dimension – Just what we wanted!
- We can sort the principal components according to their eigenvalues
- Just keep those dimensions with the largest eigenvalues

PCA Example

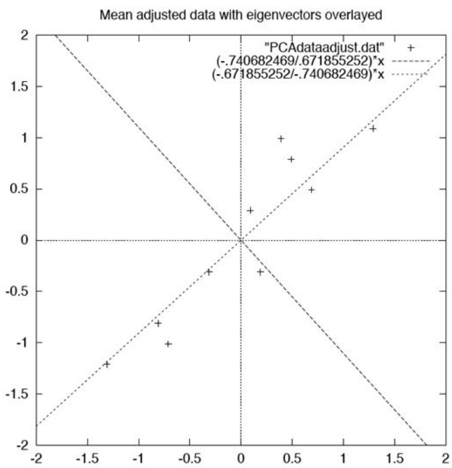
Principal Component	Eigenvalues	Eigenvectors
1	1.26610816	-0.67284685, -0.7397818
2	0.05222517	-0.7397818, 0.67284685



PCA Example

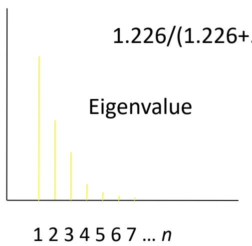
- Below are the two eigenvectors overlaying the centered data
- Which eigenvector has the largest eigenvalue?
- Fourth Step: Just keep the *p* eigenvectors with the largest eigenvalues
  - Do lose some information, but if we just drop dimensions with small eigenvalues then we lose only a little information, hopefully noise
  - We can then have *p* input features rather than *n*
  - The *p* features contain the most pertinent *combined* information from all *n* original features
  - How many dimensions *p* should we keep?

PCA Example



PC	Eigenvalues	Eigenvectors
1	1.26610816	-0.67284685, -0.7397818
2	0.05222517	-0.7397818, 0.67284685

1.226/(1.226+.052) = .96



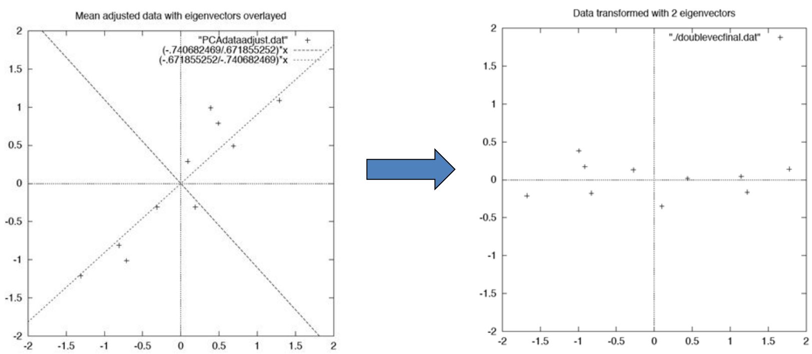
Proportion of Variance

$$\frac{\sum_{i=1}^{p} \lambda_i}{\sum_{i=1}^{n} \lambda_i} = \frac{\lambda_1 + \lambda_2 + \ldots + \lambda_p}{\lambda_1 + \lambda_2 + \ldots + \lambda_p + \ldots + \lambda_n}$$

PCA Example

- Last Step: Transform the *n* features to the *p* (< *n*) chosen bases (Eigenvectors)
- Transform data (*m* instances) with a matrix multiply $T = A \times B$
  - *A* is a $p \times n$ matrix with the *p* principal components in the rows, component one on top
  - *B* is a $n \times m$ matrix containing the transposed centered original data set
  - T^T is a $m \times p$ matrix containing the transformed data set
- Now we have the new transformed data set with *p* features
- Keep matrix *A* to transform future centered data instances
- Below is the transform of both dimensions. Would if we just kept the 1st component for this case?

PCA Example



PCA Summary

- PCA is a linear transformation, so if the features have highly non-linear correlations, the transformed data will be less useful
  - Nonlinear dimensionality reduction techniques can sometimes handle these situations better (e.g. LLE, Isomap, Manifold-Sculpting)
  - PCA is good at removing redundant linearly correlated features
- With high dimensional data the eigenvector is a hyper-plane
- Interesting note: The 1st principal component is the multiple regression plane that delta rule will always discover
- Caution: Not a "cure all" and can lose important info in some cases
  - How would you know if it is effective?
  - Just compare accuracies of original vs transformed data set

Overview

- Feature engineering overview
- Common approaches to featurizing with text
- Feature selection
- Iterating and improving (and dealing with mistakes)

Goals of Feature Engineering

> Convert ‘context’ -> input to learning algorithm.

> Expose the structure of the concept to the learning algorithm.

> Work well with the structure of the model the algorithm will create.

> Balance number of features, complexity of concept, complexity of model, amount of data.

Sample from SMS Spam

> SMS Message (arbitrary text) -> 5-dimensional array of binary features

1 if message is longer than 40 chars, 0 otherwise

1 if message contains a digit, 0 otherwise

1 if message contains word ‘call’, 0 otherwise

1 if message contains word ‘to’, 0 otherwise

1 if message contains word ‘your’, 0 otherwise

“SIX chances to win CASH! From 100 to 20,000 pounds txt> CSH11 and send to 87575. Cost 150p/day, 6days, 16+ TsandCs apply Reply HL 4 info”

Long?	HasDigit?	ContainsWord(Call)	ContainsWord(to)	ContainsWord(your)

Basic Feature Types

Binary Features

- > ContainsWord(call)?
- > IsLongSMSMessage?
- > Contains(*#)?
- > ContainsPunctuation?

Categorical Features

- FirstWordPOS ->
  { Verb, Noun, Other }
- MessageLength ->
  { Short, Medium, Long, VeryLong }
- TokenType ->
  { Number, URL, Word, Phone#, Unknown }
- GrammarAnalysis ->
  - { Fragment, SimpleSentence, ComplexSentence }

Numeric Features

- CountOfWord(call)
- MessageLength
- FirstNumberInMessage
- WritingGradeLevel

Converting Between Feature Types

Numeric Feature => Binary Feature

Length of text + [40] => { 0, 1 } — Single threshold

Numeric Feature => Categorical Feature

Length of text + [20, 40] => { short or medium or long } — Set of thresholds

Categorical Feature => Binary Features — One-hot encoding

{ short or medium or long } => [1, 0, 0] or [0, 1, 0] or [0, 0, 1]

Binary Feature => Numeric Feature — …

{ 0, 1 } => { 0, 1 }

Sources of Data for Features

- System State
  - App in foreground?
  - Roaming?
  - Sensor readings
- Content Analysis
  - Stuff we've been talking about
  - Stuff we're going to talk about next
- User Information
  - Industry
  - Demographics
- Interaction History
  - User's 'report as junk' rate
  - # previous interactions with sender
  - # messages sent/received
- Metadata
  - Properties of phone #s referenced
  - Properties of the sender
  - Run other models on the content
    - Grammar
    - Language
    - …

Feature Engineering for Text

- Tokenizing
- Bag of Words
- N-grams
- TF-IDF
- Embeddings
- NLP

Tokenizing

- Breaking text into words
  “Nah, I don't think he goes to usf” ->
  [‘Nah,’ ‘I’, ‘don't’, ‘think’, ‘he’, ‘goes’, ‘to’, ‘usf’]

- Dealing with punctuation
  “Nah,” ->
  [‘Nah,’] or [‘Nah’, ‘,’] or [‘Nah’]
  “don't” ->
  [‘don't’] or [‘don’, ‘'’, ‘t’] or [‘don’, ‘t’] or [‘do’, ‘n't’]

- Normalizing
  “Nah,” ->
  [‘Nah,’] or [‘nah,’]
  “1452” ->
  [‘1452’] or [<number>]

Some tips for deciding

- If you have lots of data / optimization…
  – Keep as much information as possible
  – Let the learning algorithm figure out what is important and what isn’t
- If you don’t have much data / optimization...
  – Reduce the number of features you maintain
  – Normalize away irrelevant things
- Focus on things relevant to the concept…
  – Explore data / use your intuition
  – Overfitting / underfitting ← much more later

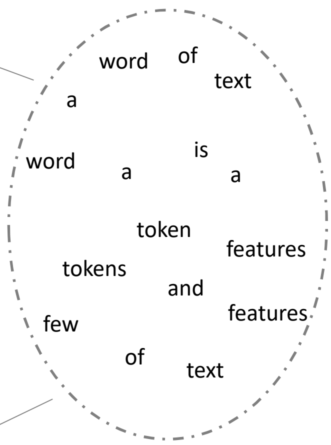
Bag of Words

One feature per unique token

Bag of words

m1: A word of text.
m2: A word is a token.
m3: Tokens and features.
m4: Few features of text.

Training data



Tokens

x_1	a
x_2	word
x_3	of
x_4	text
x_5	is
x_6	token
x_7	tokens
x_8	and
x_9	features
x_{10}	few

Features

Bag of Words: Example

test1: Some features for a text example.

Out of vocabulary

m1: A word of text.
m2: A word is a token.
m3: Tokens and features.
m4: Few features of text.

x_1	a
x_2	word
x_3	of
x_4	text
x_5	is
x_6	token
x_7	tokens
x_8	and
x_9	features
x_{10}	few

Selected Features

	m1	m2	m3	m4
x_1	1	1	0	0
x_2	1	1	0	0
x_3	1	0	0	1
x_4	1	0	0	1
x_5	0	1	0	0
x_6	0	1	0	0
x_7	0	0	1	0
x_8	0	0	1	0
x_9	0	0	1	1
x_{10}	0	0	0	1

Training X

	test1
x_1	1
x_2	0
x_3	0
x_4	1
x_5	0
x_6	0
x_7	0
x_8	0
x_9	1
x_{10}	0

Test X

Use bag of words when you have a lot of data, can use many features

N-Grams: Tokens

> Instead of using single tokens as features, use series of N tokens

> “down the bank” vs “from the bank”

Message 1: “Nah I don't think he goes to usf”
Message 2: “Text FA to 87121 to receive entry”

Message 2:

Nah I	I don’t	don’t think	think he	he goes	goes to	to usf	…	Text FA	FA to	87121 to	To receive	receive entry
0	0	0	0	0	0	0	…	1	1	1	1	1

Use when you have a LOT of data, can use MANY features

N-Grams: Characters

> Instead of using series of tokens, use series of characters

Message 1: “Nah I don't think he goes to usf”
Message 2: “Text FA to 87121 to receive entry”

Message 2:

Na	ah	h <space>	<space> I	I <space>	<space> d	do	…	<space> e	en	nt	tr	ry
0	0	0	0	0	0	0	…	1	1	1	1	1

Helps with out of dictionary words & spelling errors
Fixed number of features for given N (but can be very large)

TF-IDF
Term Frequency – Inverse Document Frequency

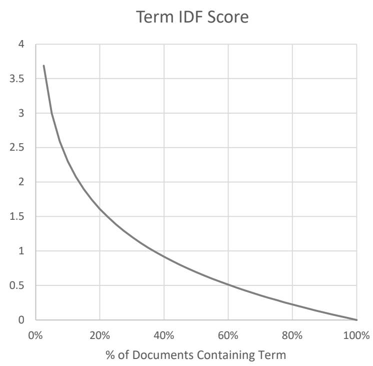
> Instead of using binary: ContainsWord(<term>)
> Use numeric importance score TF-IDF:

Importance to Document

TermFrequency(<term>, <document>) =
% of the words in <document> that are <term>

Novelty across corpus

InverseDocumentFrequency(<term>, <documents>) =
log (# documents / # documents that contain <term>)



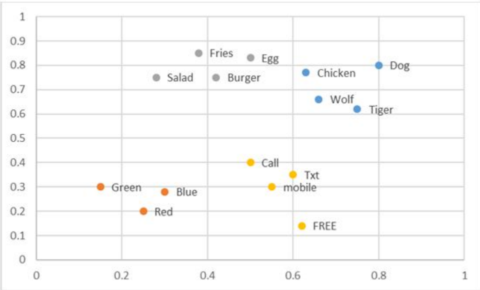
Words that occur in many documents have low score (x_i)

Message 1: “Nah I don't think he goes to usf”
Message 2: “Text FA to 87121 to receive entry”

Message 2:

	Nah	I	don't	think	he	goes	to	usf	Text	FA	87121	receive	entry
BOW	0	0	0	0	0	0	1	0	1	1	1	1	1
TF-IDF	0	0	0	0	0	0	0	0	.099	.099	.099	.099	.099

Embeddings -- Word2Vec and FastText



> Word -> Coordinate in N dimension
> Regions of space contain similar concepts
> Creating Features Options:
  – Average vector across words
  – Count in specific regions
> Commonly used with neural networks

Replaces words with their 'meanings' – sparse -> dense representation

Normalization (Numeric => Better Numeric)

Raw X		Normalize Mean		Normalize Variance
36		-38.875		-1.31696
74		-0.875		-0.02964
22		-52.875		-1.79123
81	Subtract Mean ⇒	6.125	Divide by Stdev ⇒	0.207495
105		30.125		1.020536
113		38.125		1.29155
77		2.125		0.071988
91		16.125		0.546262
Mean: 74.875		Mean: 0 Std: 29.5188		Mean: 0 Std: 1

Helps make model's job easier

- No need to learn what is 'big' or 'small' for the feature
- Some model types benefit more than others

To use in practice:

- Estimate mean/stdev on training data
- Apply normalization using those parameters to validation /train

Feature Selection

> Which features to use?

> How many features to use?

Approaches:

- Frequency
- Mutual Information
- Accuracy

Feature Selection: Frequency

Take top N most common features ***in the training set***

Feature	Count
to	1745
you	1526
I	1369
a	1337
the	1007
and	758
in	400
…	…

Feature Selection: Mutual Information

*Take N that contain most information about target **on the training set***

$$MI(X,Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) log(\frac{p(x,y)}{p(x)p(y)})$$

x	y
1	0
0	0
0	0
0	0
1	1
1	1
0	1
0	1
1	1
1	1

Training Data

	$x = 0$	$x = 1$
$y = 0$	3	1
$y = 1$	2	4

Contingency Table

$$p(x=0,y=0) log\left(\frac{p(x=0,y=0)}{p(x=0)p(y=0)}\right) = .3 * log\left(\frac{.3}{.5 * .4}\right) = 0.122$$

Sum over all combinations: MI = 0.086

	$x = 0$	$x = 1$
$y = 0$	10	0
$y = 1$	0	10

Perfect predictor → high MI

	x=0	x=1
$y = 0$	5	5
$y = 1$	5	5

No Information → 0 MI

Additive Smoothing to avoid 0s: $P(*) = \frac{Obs(*)+1}{N+2}$

Feature Selection: Accuracy (wrapper)

Take N that improve accuracy most ***on hold out data***

Greedy search, adding or removing features

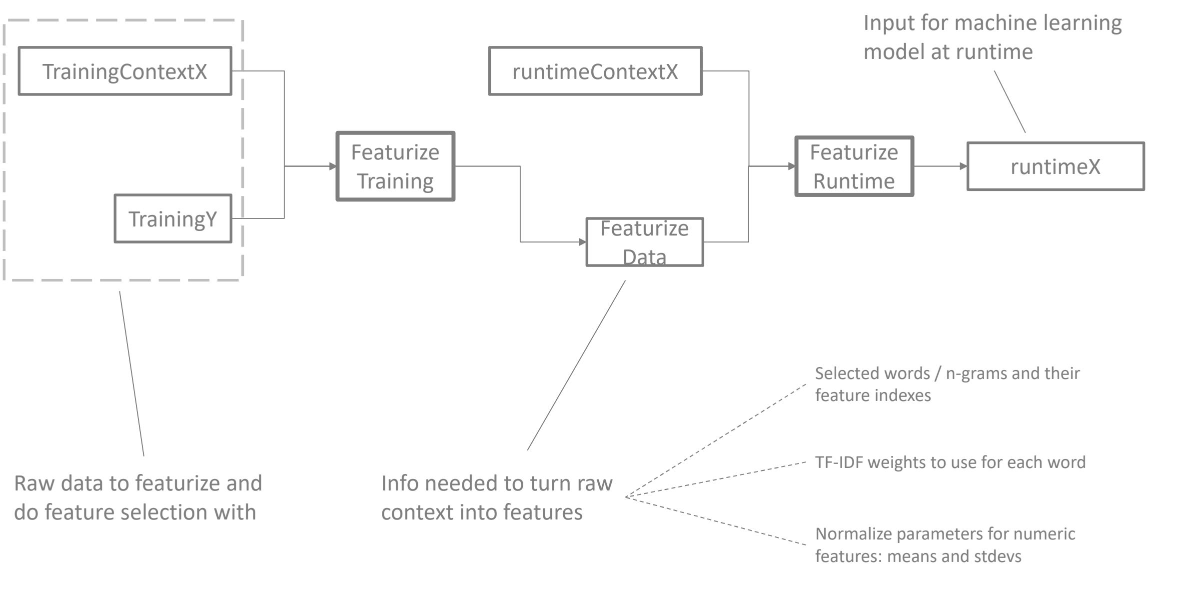
From baseline, try adding (removing) each candidate
Build a model
Evaluate ***on hold out data***
Add (remove) the best
Repeat till you get to N

Remove	Accuracy
<None>	88.2%
claim	82.1%
FREE	86.5%
or	87.8%
to	89.8%
…	…

Important note about feature selection

> Do not use validation (or test) data when doing feature selection
> Use train data only to select features
> Then apply the selected features to the validation (or test) data

Simple Feature Engineering Pattern



Simple Feature Engineering Pattern: Pseudocode

```
for f in featureSelectionMethodsToTry:
    (trainX, trainY, featureData) = FeaturizeTraining(rawTrainX, rawTrainY, f)
    (validationX, validationY) = FeaturizeRuntime(rawValidationX, rawValidationY, f, featureData)

    for hp in hyperParametersToTry:
        model.fit(trainX, trainY, hp)
        accuracies[hp, f] = evaluate(validationY, model.predict(validationX))

(bestHyperParametersFound, bestFeaturizerFound) = bestSettingFound(accuracies)

(finalTrainX, finalTrainY, featureData) =
        FeaturizeTraining(rawTrainX + rawValidationX, rawTrainY + rawValidationY, bestFeaturizerFound)

(testX, testY) = FeaturizeRuntime(rawTextX, rawTestY, bestFeaturizerFound, featureData)

finalModel.fit(finalTrainX, finalTrainY, bestHyperParametersFound)

estimateOfGeneralizationPerformance = evaluate(testY, model.predict(testX))
```

Understanding Mistakes

> Noise in the data
  – Encodings
  – Bugs
  – Missing values
  – Corruption
> Noise in the labels
  Ham: As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been set as your callertune for all Callers. Press *9 to copy your friends Callertune
  Spam: I'll meet you at the resturant between 10 & 10:30 – can't wait!
> Model being wrong…
  – Reason?

Exploring Mistakes

> Examine N *random* false positive and N *random* false negatives

Reason	Count
Label Noise	2
Slang	5
Non-English	5
…	…

> Examine N *worst* false positives and N *worst* false negatives
  – Model predicts very near 1, but true answer is 0
  – Model predicts very near 0, but true answer is 1

Approach to Feature Engineering

> Start with 'standard' for your domain; 1 parameter per ~10 samples
> Try all the important variations ***on hold out data***
  – Tokenizing
  – Bag of words
  – N-grams
  – …
> Use some form of feature selection to find the best, evaluate
> Look at your mistakes…
> Use your intuition about your domain and adapt standard approaches or invent new features…
> Iterate
> When you want to know how well you did, evaluate on test data

Feature Engineering in Other Domains

Computer Vision:
> Gradients
> Histograms
> Convolutions

Time Series:
> Window aggregated statistics
> Frequency domain transformations

Internet:
> IP Parts
> Domains
> Relationships
> Reputation

Neural Networks:
> A whole bunch of other things we'll talk about later…

Summary of Feature Engineering

> Feature engineering converts raw context into inputs for machine learning

> Goals are:
– Match structure of concept to structure of model representation
– Balance number of feature, amount of data, complexity of concept, power of model

> Every domain has a library of proven feature engineering approaches

> Text's include: normalization, tokenizing, n-grams, TF-IDF, embeddings, & NLP

> Feature selection removes less useful features and can greatly increase accuracy