Features

David S. Rosenberg

Bloomberg ML EDU

October 18, 2017

Learning Objectives

- Understand where a *feature map* sits in a machine learning pipeline.
- Understand how feature extraction can be used to extend the power of linear methods.
- Build pipelines with expanded feature spaces using the sklearn ecosystem.

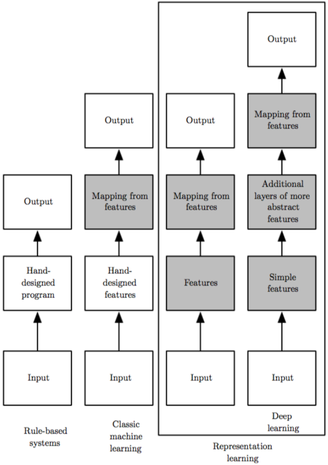


Figure 1.5: Flowcharts showing how the different parts of an AI system relate to each other within different AI disciplines. Shaded boxes indicate components that are able to learn from data.

From Goodfellow, Bengio, and Courville's *Deep Learning*.

Feature Extraction

The Input Space $\mathcal{X}$

- Our general learning theory setup: no assumptions about $\mathcal{X}$

The Input Space $\mathcal{X}$

- Our general learning theory setup: no assumptions about $\mathcal{X}$
- But $\mathcal{X} = \mathbf{R}^d$ for the specific methods we've developed:

The Input Space $\mathcal{X}$

- Our general learning theory setup: no assumptions about $\mathcal{X}$
- But $\mathcal{X} = \mathbf{R}^d$ for the specific methods we've developed:
    - Ridge regression
    - Lasso regression
    - Linear SVM

Motivation

- Two motivations for thinking about feature extraction:
  - Motive 1 – Improving performance.

```
Boston House Prices dataset
===========================

Notes
------
Data Set Characteristics:

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive

    :Median Value (attribute 14) is usually the target
```

Motivation

- Two motivations for thinking about feature extraction:
    - Motive 1 – Improving performance.

```
from sklearn.linear_model import ElasticNetCV
```

```
en = ElasticNetCV(cv = 5)
```

```
en.fit(np.log(train_X[['LSTAT']]), train_y)
en.score(np.log(test_X[['LSTAT']]), test_y)
```

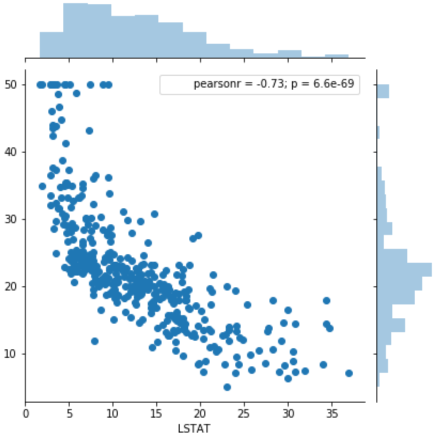
```
0.74651286928253746
```

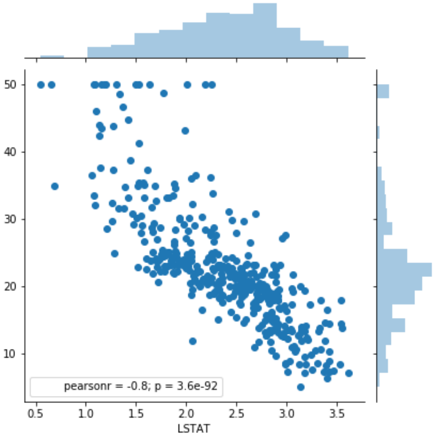
```
en.fit(train_X[['LSTAT']], train_y)
en.score(test_X[['LSTAT']], test_y)
```

```
0.57894475666257272
```

Motivation

- Key idea: instead of using more flexible models, use better features.





The Input Space $\mathcal{X}$

- Two motivations for thinking about feature extraction:
  - Motive 2 – consuming inputs that are not natively in $\mathbf{R}^d$ – examples?

The Input Space $\mathcal{X}$

- Two motivations for thinking about feature extraction:
  - Motive 2 – consuming inputs that are not natively in $\mathbf{R}^d$ – examples?
    - Text documents
    - Image files
    - Sound recordings
    - DNA sequences

The Input Space $\mathcal{X}$

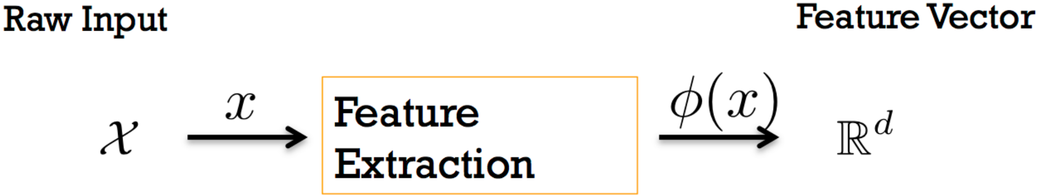
- Two motivations for thinking about feature extraction:
  - Motive 2 – consuming inputs that are not natively in $\mathbf{R}^d$ – examples?
    - Text documents
    - Image files
    - Sound recordings
    - DNA sequences
  - But everything in a computer is a sequence of numbers?

The Input Space $\mathcal{X}$

- Two motivations for thinking about feature extraction:
  - Motive 2 – consuming inputs that are not natively in $\mathbf{R}^d$ – examples?
    - Text documents
    - Image files
    - Sound recordings
    - DNA sequences
  - But everything in a computer is a sequence of numbers?
    - The ith entry of each sequence should have the same "meaning"
    - All the sequences should have the same length

Definition

Mapping an input from $\mathcal{X}$ to a vector in $\mathbf{R}^d$ is called **feature extraction** or **featurization**.



Feature Templates

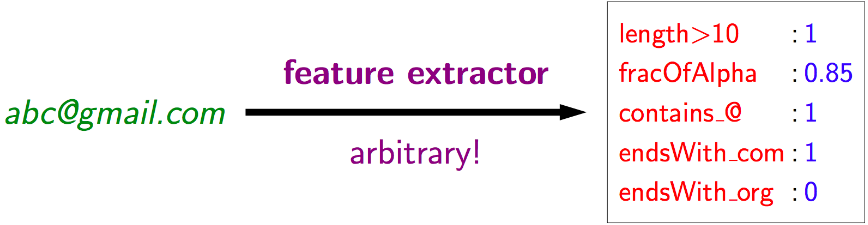
Example: Detecting Email Addresses

- Task: Predict whether a string is an email address

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Example: Detecting Email Addresses

- Task: Predict whether a string is an email address
- Could use domain knowledge and write down:



From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Example: Detecting Email Addresses

- Task: Predict whether a string is an email address
- Could use domain knowledge and write down:

abc@gmail.com $\xrightarrow[\text{arbitrary!}]{\textbf{feature extractor}}$

length>10	: 1
fracOfAlpha	: 0.85
contains_@	: 1
endsWith_com	: 1
endsWith_org	: 0

- But this was ad-hoc, and maybe we missed something.
- Could be more systematic?

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Feature Templates

Definition (informal)

A **feature template** is a group of features all computed in a similar way.

Based on Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Feature Templates

Definition (informal)

A **feature template** is a group of features all computed in a similar way.

- Input: *abc@gmail.com*

Based on Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Feature Templates

Definition (informal)

A **feature template** is a group of features all computed in a similar way.

- Input: *abc@gmail.com*

Feature Templates

- Length greater than ___

Based on Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Feature Templates

Definition (informal)

A **feature template** is a group of features all computed in a similar way.

- Input: *abc@gmail.com*

Feature Templates

- Length greater than ___
- Last three characters equal ___

Based on Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Feature Templates

Definition (informal)

A **feature template** is a group of features all computed in a similar way.

- Input: *abc@gmail.com*

Feature Templates

- Length greater than ___
- Last three characters equal ___
- Contains character ___

Based on Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Feature Template: Last Three Characters Equal ___

- Don't think about which 3-letter suffixes are meaningful...
- Just **include them all.**

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Feature Template: Last Three Characters Equal _ _ _

- Don't think about which 3-letter suffixes are meaningful...
- Just **include them all.**

abc@gmail.com $\longrightarrow$

endsWith_aaa : 0
endsWith_aab : 0
endsWith_aac : 0
...
endsWith_com : 1
...
endsWith_zzz : 0

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Feature Template: Last Three Characters Equal ___

- Don't think about which 3-letter suffixes are meaningful...
- Just **include them all.**

abc@gmail.com $\longrightarrow$

endsWith_aaa	: 0
endsWith_aab	: 0
endsWith_aac	: 0
...	
endsWith_com	: 1
...	
endsWith_zzz	: 0

- With regularization, our methods will not be overwhelmed.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Feature Template: One-Hot Encoding

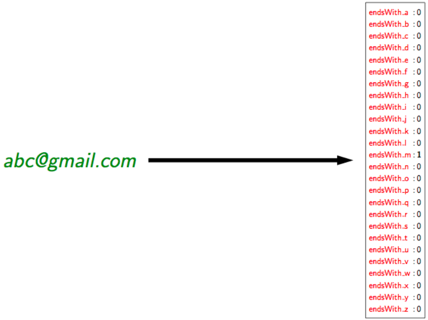
Definition

A **one-hot encoding** is a set of features (e.g. a feature template) that always has **exactly one** non-zero value.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Definition

A **one-hot encoding** is a set of features (e.g. a feature template) that always has **exactly one** non-zero value.



From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Feature Template: Count-based

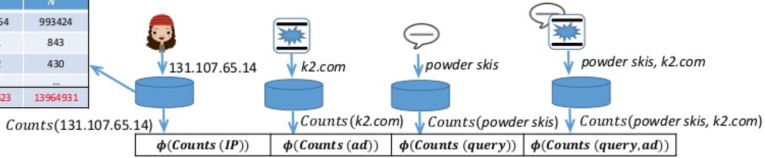
- Imagine you are trying to validate a credit card transaction.
- One feature in raw data consumed by pipeline: zip code of transaction.
- There are approximately 43,000 zip codes in the USA.
- What might be a problem with one-hot encoding this feature?

Feature Template: Count-based

- Imagine you are trying to validate a credit card transaction.
- One feature in raw data consumed by pipeline: zip code of transaction.
- There are approximately 43,000 zip codes in the USA.
- What might be a problem with one-hot encoding this feature?
- Even worse – think about internet scale problems.

Learning with counts

IP	N^+	N^-
173.194.33.9	46964	993424
87.250.251.11	31	843
131.107.65.14	12	430
...	...	...
REST	745623	13964931



- Features are per-label counts [+odds] [+backoff]

$$\phi = [N^+ \quad N^- \quad \log(N^+)-\log(N^-) \quad \text{IsRest}]$$

- $\log(N^+)-\log(N^-) = \log\frac{p(+)}{p(-)}$: log-odds/Naïve Bayes estimate
- N^+, N^-: indicators of confidence of the naïve estimate
- **IsFromRest:** indicator of back-off vs. "real count"

From Misha Bilenko/Microsoft Azure ML's "Many Shades of Scale: Big Learning Beyond Big Data"

Feature Vector Representations

fracOfAlpha	: 0.85
contains_a	: 0
...	
contains_@	: 1
...	

Array representation (good for dense features):

```
[0.85, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
```

Map representation (good for sparse features):

```
{"fracOfAlpha": 0.85, "contains_@": 1}
```

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Feature Vector Representations

- Arrays
  - assumed fixed ordering of the features

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Feature Vector Representations

- Arrays
  - assumed fixed ordering of the features
  - appropriate when significant number of nonzero elements ("**dense feature vectors**")

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Feature Vector Representations

- Arrays
  - assumed fixed ordering of the features
  - appropriate when significant number of nonzero elements (“**dense feature vectors**”)
  - very efficient in space and speed (and you can take advantage of GPUs)

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Feature Vector Representations

- Arrays
  - assumed fixed ordering of the features
  - appropriate when significant number of nonzero elements ("**dense feature vectors**")
  - very efficient in space and speed (and you can take advantage of GPUs)
- Map (a "dict" in Python)
  - best for **sparse feature vectors** (i.e. few nonzero features)

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Feature Vector Representations

- Arrays
  - assumed fixed ordering of the features
  - appropriate when significant number of nonzero elements (“**dense feature vectors**”)
  - very efficient in space and speed (and you can take advantage of GPUs)
- Map (a “dict” in Python)
  - best for **sparse feature vectors** (i.e. few nonzero features)
  - features not in the map have default value of zero

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Feature Vector Representations

- Arrays
    - assumed fixed ordering of the features
    - appropriate when significant number of nonzero elements (“**dense feature vectors**”)
    - very efficient in space and speed (and you can take advantage of GPUs)
- Map (a “dict” in Python)
    - best for **sparse feature vectors** (i.e. few nonzero features)
    - features not in the map have default value of zero
    - Python code for “ends with last 3 characters”:

```
{"endsWith_"+x[-3:]: 1}.
```

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Feature Vector Representations

- Arrays
  - assumed fixed ordering of the features
  - appropriate when significant number of nonzero elements ("**dense feature vectors**")
  - very efficient in space and speed (and you can take advantage of GPUs)
- Map (a "dict" in Python)
  - best for **sparse feature vectors** (i.e. few nonzero features)
  - features not in the map have default value of zero
  - Python code for "ends with last 3 characters":

    ```
    {"endsWith_"+x[-3:]: 1}.
    ```

  - On `"example string"` we'd get `{"endsWith_ing": 1}`.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Feature Vector Representations

- Arrays
  - assumed fixed ordering of the features
  - appropriate when significant number of nonzero elements (“**dense feature vectors**”)
  - very efficient in space and speed (and you can take advantage of GPUs)
- Map (a “dict” in Python)
  - best for **sparse feature vectors** (i.e. few nonzero features)
  - features not in the map have default value of zero
  - Python code for “ends with last 3 characters”:

```
{"endsWith_"+x[-3:]: 1}.
```

  - On `"example string"` we’d get `{"endsWith_ing": 1}`.
  - Has overhead compared to arrays, so much slower for dense features.
  - Question: if we have a sparse feature vector, what are the implications for preprocessing?

From Percy Liang’s "Lecture 3" slides from Stanford’s CS221, Autumn 2014.

Handling Nonlinearity with Linear Methods

Example Task: Predicting Health

- General Philosophy: Extract every feature that might be relevant

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Example Task: Predicting Health

- General Philosophy: Extract every feature that might be relevant
- Features for medical diagnosis
    - height
    - weight
    - body temperature
    - blood pressure
    - etc...

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Issues for Linear Predictors

- For linear predictors, it's important **how** features are added

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Issues for Linear Predictors

- For linear predictors, it's important **how** features are added
- Three types of nonlinearities can cause problems:

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Issues for Linear Predictors

- For linear predictors, it's important **how** features are added
- Three types of nonlinearities can cause problems:
  1. Non-monotonicity
  2. Saturation
  3. Interactions between features

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Non-monotonicity: The Issue

- Feature Map: $\phi(x) = [1, \text{temperature}(x)]$

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Non-monotonicity: The Issue

- Feature Map: $\phi(x) = [1, \text{temperature}(x)]$
- Action: Predict health score $y \in \mathbf{R}$ (positive is good)

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Non-monotonicity: The Issue

- Feature Map: $\phi(x) = [1, \text{temperature}(x)]$
- Action: Predict health score $y \in \mathbf{R}$ (positive is good)
- Hypothesis Space $\mathcal{F} = \{\text{affine functions of temperature}\}$

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Non-monotonicity: The Issue

- Feature Map: $\phi(x) = [1, \text{temperature}(x)]$
- Action: Predict health score $y \in \mathbf{R}$ (positive is good)
- Hypothesis Space $\mathcal{F} = \{\text{affine functions of temperature}\}$
- Issue:

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Non-monotonicity: The Issue

- Feature Map: $\phi(x) = [1, \text{temperature}(x)]$
- Action: Predict health score $y \in \mathbf{R}$ (positive is good)
- Hypothesis Space $\mathcal{F} = \{\text{affine functions of temperature}\}$
- Issue:
  - Health is not an affine function of temperature.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Non-monotonicity: The Issue

- Feature Map: $\phi(x) = [1, \text{temperature}(x)]$
- Action: Predict health score $y \in \mathbf{R}$ (positive is good)
- Hypothesis Space $\mathcal{F} = \{\text{affine functions of temperature}\}$
- Issue:
  - Health is not an affine function of temperature.
- Affine function can either say
  - Very high is bad and very low is good, or
  - Very low is bad and very high is good,
  - But here, both extremes are bad.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Non-monotonicity: Solution 1

- Transform the input:
$$\phi(x) = \left[1, \{\text{temperature(x)-37}\}^2\right],$$
where 37 is "normal" temperature in Celsius.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Non-monotonicity: Solution 1

- Transform the input:
$$\phi(x) = \left[1, \{\text{temperature(x)-37}\}^2\right],$$
where 37 is "normal" temperature in Celsius.
- Ok, but this requires domain knowledge
  - Do we really need that?

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Non-monotonicity: Solution 2

- Think less, put in more:

$$\phi(x) = \left[1, \text{temperature}(x), \{\text{temperature}(x)\}^2\right].$$

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Non-monotonicity: Solution 2

- Think less, put in more:

$$\phi(x) = \left[1, \text{temperature}(x), \{\text{temperature}(x)\}^2\right].$$

- **More expressive** than Solution 1.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Non-monotonicity: Solution 2

- Think less, put in more:

$$\phi(x) = \left[1, \text{temperature}(x), \{\text{temperature}(x)\}^2\right].$$

- **More expressive** than Solution 1.

General Rule

Features should be simple building blocks that can be pieced together.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Saturation: The Issue

- Setting: Find products relevant to user's query

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Saturation: The Issue

- Setting: Find products relevant to user's query
- Input: Product x
- Action: Score the relevance of x to user's query

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Saturation: The Issue

- Setting: Find products relevant to user's query
- Input: Product x
- Action: Score the relevance of x to user's query
- Feature Map:
$$\phi(x) = [1, N(x)],$$
where $N(x) =$ number of people who bought x.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Saturation: The Issue

- Setting: Find products relevant to user's query
- Input: Product x
- Action: Score the relevance of x to user's query
- Feature Map:
$$\phi(x) = [1, N(x)],$$
where $N(x) =$ number of people who bought x.
- We expect a monotonic relationship between $N(x)$ and relevance, but...

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Saturation: The Issue

Is relevance linear in N(x)?

- Relevance score reflects confidence in relevance prediction.
- Are we 10 times more confident if $N(x) = 1000$ vs $N(x) = 100$?

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Saturation: The Issue

Is relevance linear in N(x)?

- Relevance score reflects confidence in relevance prediction.
- Are we 10 times more confident if $N(x) = 1000$ vs $N(x) = 100$?

- Bigger is better... but not that much better.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Saturation: Solve with nonlinear transform

- Smooth nonlinear transformation:

$$\phi(x) = [1, \log\{1 + N(x)\}]$$

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Saturation: Solve with nonlinear transform

- Smooth nonlinear transformation:

$$\phi(x) = [1, \log\{1 + N(x)\}]$$

- $\log(\cdot)$ good for values with large dynamic ranges

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Saturation: Solve with nonlinear transform

- Smooth nonlinear transformation:

$$\phi(x) = [1, \log\{1 + N(x)\}]$$

- $\log(\cdot)$ good for values with large dynamic ranges
- *Does it matter what base we use in the log?*

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Saturation: Solve by discretization

- Discretization (a discontinuous transformation):

$$\phi(x) = (1(5 \leqslant N(x) < 10), 1(10 \leqslant N(x) < 100), 1(100 \leqslant N(x)))$$

Saturation: Solve by discretization

- Discretization (a discontinuous transformation):

$$\phi(x) = (1(5 \leqslant N(x) < 10), 1(10 \leqslant N(x) < 100), 1(100 \leqslant N(x)))$$

- Sometimes we might prefer one-sided buckets

$$\phi(x) = (1(5 \leqslant N(x)), 1(10 \leqslant N(x)), 1(100 \leqslant N(x)))$$

  - Why?

Saturation: Solve by discretization

- Discretization (a discontinuous transformation):

$$\phi(x) = (1(5 \leqslant N(x) < 10), 1(10 \leqslant N(x) < 100), 1(100 \leqslant N(x)))$$

- Sometimes we might prefer one-sided buckets

$$\phi(x) = (1(5 \leqslant N(x)), 1(10 \leqslant N(x)), 1(100 \leqslant N(x)))$$

  - Why? Hint: What's the effect of regularization the parameters for rare buckets?

Saturation: Solve by discretization

- Discretization (a discontinuous transformation):

$$\phi(x) = \left(1(5 \leqslant N(x) < 10), 1(10 \leqslant N(x) < 100), 1(100 \leqslant N(x))\right)$$

- Sometimes we might prefer one-sided buckets

$$\phi(x) = \left(1(5 \leqslant N(x)), 1(10 \leqslant N(x)), 1(100 \leqslant N(x))\right)$$

  - Why? Hint: What's the effect of regularization the parameters for rare buckets?
- Small buckets allow quite flexible relationship

Interactions: The Issue

- Input: Patient information x
- Action: Health score $y \in \mathbf{R}$ (higher is better)
- Feature Map

$$\phi(x) = [\text{height}(x), \text{weight}(x)]$$

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Interactions: The Issue

- Input: Patient information x
- Action: Health score $y \in \mathbf{R}$ (higher is better)
- Feature Map
$$\phi(x) = [\text{height}(x), \text{weight}(x)]$$
- Issue: It's the weight **relative** to the height that's important.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Interactions: The Issue

- Input: Patient information x
- Action: Health score $y \in \mathbf{R}$ (higher is better)
- Feature Map

$$\phi(x) = [\text{height}(x), \text{weight}(x)]$$

- Issue: It's the weight **relative** to the height that's important.
- Impossible to get with these features and a linear classifier.
- Need some **interaction** between height and weight.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Interactions: Approach 1

- Google "ideal weight from height"

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Interactions: Approach 1

- Google "ideal weight from height"
- J. D. Robinson's "ideal weight" formula (for a male):

$$\text{weight(kg)} = 52 + 1.9\,[\text{height(in)} - 60]$$

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Interactions: Approach 1

- Google “ideal weight from height”
- J. D. Robinson’s “ideal weight” formula (for a male):

$$\text{weight(kg)} = 52 + 1.9\,[\text{height(in)} - 60]$$

- Make score square deviation between height(h) and ideal weight(w)

$$f(x) = (52 + 1.9\,[h(x) - 60] - w(x))^2$$

From Percy Liang’s "Lecture 3" slides from Stanford’s CS221, Autumn 2014.

Interactions: Approach 1

- Google "ideal weight from height"
- J. D. Robinson's "ideal weight" formula (for a male):

$$\text{weight(kg)} = 52 + 1.9\,[\text{height(in)} - 60]$$

- Make score square deviation between height(h) and ideal weight(w)

$$f(x) = (52 + 1.9\,[h(x) - 60] - w(x))^2$$

- WolframAlpha for complicated Mathematics:

$$f(x) = 3.61h(x)^2 - 3.8h(x)w(x) - 235.6h(x) + w(x)^2 + 124w(x) + 3844$$

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Interactions: Approach 2

- Just include all second order features:

$$\phi(x) = \left[1, h(x), w(x), h(x)^2, w(x)^2, \underbrace{h(x)w(x)}_{\text{cross term}}\right]$$

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Interactions: Approach 2

- Just include all second order features:

$$\phi(x) = \left[1, h(x), w(x), h(x)^2, w(x)^2, \underbrace{h(x)w(x)}_{\text{cross term}}\right]$$

- More flexible, no Google, no WolframAlpha.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Interactions: Approach 2

- Just include all second order features:

$$\phi(x) = \left[1, h(x), w(x), h(x)^2, w(x)^2, \underbrace{h(x)w(x)}_{\text{cross term}}\right]$$

- More flexible, no Google, no WolframAlpha.

General Principle

Simpler building blocks replace a single "smart" feature.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Predicate Features and Interaction Terms

Definition

A **predicate** on the input space $\mathcal{X}$ is a function $P : \mathcal{X} \to \{\text{True}, \text{False}\}$.

Predicate Features and Interaction Terms

Definition

A **predicate** on the input space $\mathcal{X}$ is a function $P : \mathcal{X} \to \{\text{True}, \text{False}\}$.

- Many features take this form:
  - $x \mapsto s(x) = 1(\text{subject is sleeping})$
  - $x \mapsto d(x) = 1(\text{subject is driving})$

Predicate Features and Interaction Terms

Definition

A **predicate** on the input space $\mathcal{X}$ is a function $P : \mathcal{X} \to \{\text{True}, \text{False}\}$.

- Many features take this form:
  - $x \mapsto s(x) = 1(\text{subject is sleeping})$
  - $x \mapsto d(x) = 1(\text{subject is driving})$
- For predicates, interaction terms correspond to **AND** conjunctions:
  - $x \mapsto s(x)d(x) = 1(\text{subject is sleeping AND subject is driving})$

So What's Linear?

- Non-linear feature map $\phi : \mathcal{X} \to \mathbf{R}^d$
- Hypothesis space:

$$\mathcal{F} = \left\{ f(x) = w^T \phi(x) \mid w \in \mathbf{R}^d \right\}.$$

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

So What's Linear?

- Non-linear feature map $\phi : \mathcal{X} \to \mathbf{R}^d$
- Hypothesis space:
$$\mathcal{F} = \left\{ f(x) = w^T \phi(x) \mid w \in \mathbf{R}^d \right\}.$$
- Linear in w? Yes.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

So What's Linear?

- Non-linear feature map $\phi : \mathcal{X} \to \mathbf{R}^d$
- Hypothesis space:

$$\mathcal{F} = \left\{ f(x) = w^T \phi(x) \mid w \in \mathbf{R}^d \right\}.$$

- Linear in w? Yes.
- Linear in $\phi(x)$? Yes.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

So What's Linear?

- Non-linear feature map $\phi : \mathcal{X} \to \mathbf{R}^d$
- Hypothesis space:

$$\mathcal{F} = \left\{ f(x) = w^T \phi(x) \mid w \in \mathbf{R}^d \right\}.$$

- Linear in w? Yes.
- Linear in $\phi(x)$? Yes.
- Linear in x? No.
  - Linearity not even defined unless $\mathcal{X}$ is a vector space

Key Idea: Non-Linearity

- Nonlinear $f(x)$ is important for **expressivity**.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

So What's Linear?

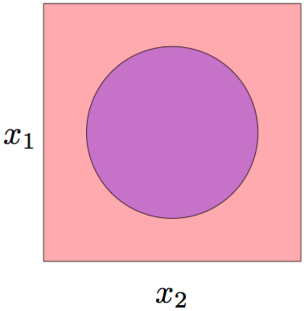
- Non-linear feature map $\phi : \mathcal{X} \to \mathbf{R}^d$
- Hypothesis space:
$$\mathcal{F} = \left\{ f(x) = w^T \phi(x) \mid w \in \mathbf{R}^d \right\}.$$
- Linear in w? Yes.
- Linear in $\phi(x)$? Yes.
- Linear in x? No.
    - Linearity not even defined unless $\mathcal{X}$ is a vector space

Key Idea: Non-Linearity

- Nonlinear $f(x)$ is important for **expressivity**.
- $f(x)$ linear in w and $\phi(x)$: makes finding $f^*(x)$ much easier

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

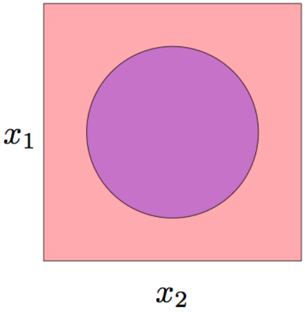
Geometric Example: Two class problem, nonlinear boundary



- With linear feature map $\phi(x) = (x_1, x_2)$ and linear models, no hope

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

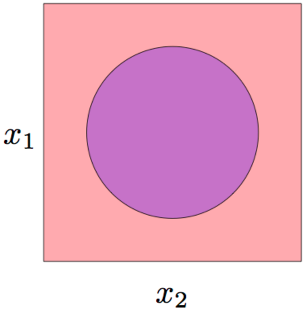
Geometric Example: Two class problem, nonlinear boundary



- With linear feature map $\phi(x) = (x_1, x_2)$ and linear models, no hope
- With appropriate nonlinearity $\phi(x) = \left(x_1, x_2, x_1^2 + x_2^2\right)$, piece of cake.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Geometric Example: Two class problem, nonlinear boundary



- With linear feature map $\phi(x) = (x_1, x_2)$ and linear models, no hope
- With appropriate nonlinearity $\phi(x) = \left(x_1, x_2, x_1^2 + x_2^2\right)$, piece of cake.
- Video: http://youtu.be/3liCbRZPrZA

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Expressivity of Hypothesis Space

- Consider a linear hypothesis space with a feature map $\phi : \mathcal{X} \to \mathbf{R}^d$:

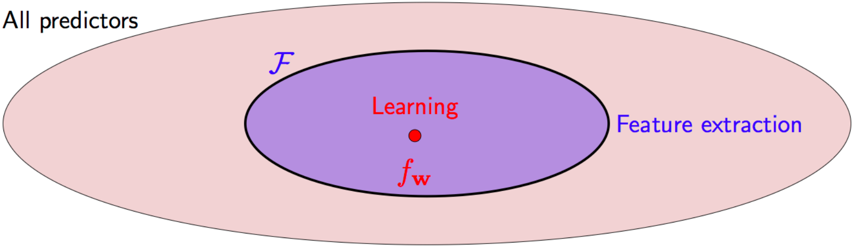
$$\mathcal{F} = \left\{ f(x) = w^T \phi(x) \right\}$$

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

Expressivity of Hypothesis Space

- Consider a linear hypothesis space with a feature map $\phi : \mathcal{X} \to \mathbf{R}^d$:

$$\mathcal{F} = \{f(x) = w^T \phi(x)\}$$



Question: does $\mathcal{F}$ contain a good predictor?

We can grow the linear hypothesis space $\mathcal{F}$ by adding more features.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.