Features

David S. Rosenberg and Ben Jakubowski

Bloomberg ML EDU

October 18, 2017

Learning Objectives

- Understand where a *feature map* sits in a machine learning pipeline.
- Understand that featurization/featuring mapping is inherently required to allow predictors to ingest many types of data.
- Understand how feature extraction can be used to extend the power of linear methods.
- Build pipelines with expanded feature spaces using the sklearn ecosystem.

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:

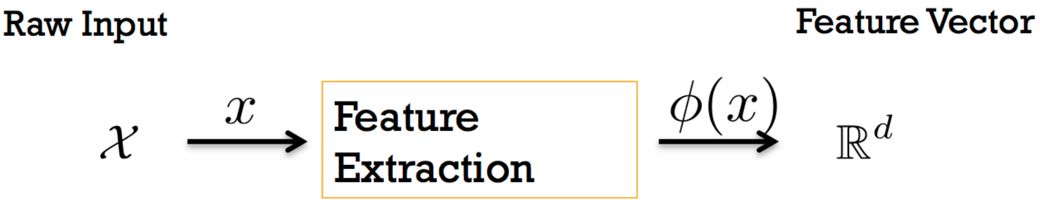
The Input Space $\mathcal{X}$

- 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

Feature Extraction

Definition

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



Motivation

- Two motivations for thinking about feature extraction:
  - Motivation 1 – consuming inputs that are not natively in $\mathbf{R}^d$ – examples?

Motivation

- Two motivations for thinking about feature extraction:
  - Motivation 1 – consuming inputs that are not natively in $\mathbf{R}^d$ – examples?
    - Text documents
    - Image files
    - Sound recordings
    - DNA sequences

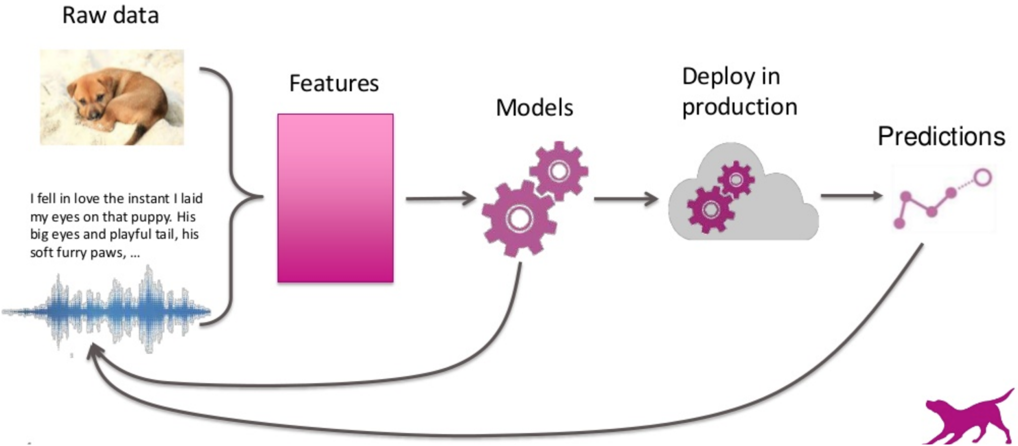
Motivation

- Two motivations for thinking about feature extraction:
  - Motivation 1 – 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?

Motivation

- Two motivations for thinking about feature extraction:
    - Motivation 1 – 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

The machine learning pipeline



https://www.slideshare.net/AliceZheng3/understanding-feature-space-in-machine-learning

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:

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

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

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.

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.

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.

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

abc@gmail.com

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 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.

Feature Map – ingesting inputs not natively in $\mathbf{R}^d$

Example: Classifying documents from 20 newsgroups

- Context: The newsgroups dataset comprises around 18000 newsgroups posts on 20 topics.

Example: Classifying documents from 20 newsgroups

- Context: The newsgroups dataset comprises around 18000 newsgroups posts on 20 topics.
- We'll restrict ourselves to classifying posts within 4 topics:
  - 'alt.atheism'
  - 'soc.religion.christian'
  - 'comp.graphics'
  - 'sci.med'.
- Thanks to the sklearn team for this worked example (at `http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html`).

Example: Classifying documents from 20 newsgroups

Example Document:

```
From: sd345@city.ac.uk (Michael Collier)
Subject: Converting images to HP LaserJet III?
Nntp-Posting-Host: hampton
Organization: The City University
Lines: 14

Does anyone know of a good way (standard PC application/PD utility) to
convert tif/img/tga files into LaserJet III format.  We would also like to
do the same, converting to HPGL (HP plotter) files.

Please email any response.

Is this the correct group?

Thanks in advance.  Michael.
--
Michael Collier (Programmer)                 The Computer Unit,
Email: M.P.Collier@uk.ac.city                The City University,
Tel: 071 477-8000 x3769                      London,
Fax: 071 477-8565                            EC1V 0HB.
```

Example: Classifying documents from 20 newsgroups

```
From: sd345@city.ac.uk (Michael Collier)
Subject: Converting images to HP LaserJet III?
Nntp-Posting-Host: hampton
Organization: The City University
Lines: 14

Does anyone know of a good way (standard PC application/PD utility) to
convert tif/img/tga files into LaserJet III format.  We would also like to
do the same, converting to HPGL (HP plotter) files.

Please email any response.
```

- What feature maps could we apply over these sorts of documents?

Example: Classifying documents from 20 newsgroups

```
From: sd345@city.ac.uk (Michael Collier)
Subject: Converting images to HP LaserJet III?
Nntp-Posting-Host: hampton
Organization: The City University
Lines: 14

Does anyone know of a good way (standard PC application/PD utility) to
convert tif/img/tga files into LaserJet III format.  We would also like to
do the same, converting to HPGL (HP plotter) files.

Please email any response.
```

- What feature maps could we apply over these sorts of documents?
- A simple approach – bag-of-words (BOW).
  - Assign a fixed integer id to each word occurring in any document of the training set.
  - For each document i, count the number of occurrences of each word w and store it (sparsely) as $doc_i[w] = j_w ==$ count of word w in document i.
  - The BOW representation implies that $n_{features}$ is the number of distinct words in the corpus.
  - What is the feature map $\phi(x)$?

Example: Classifying documents from 20 newsgroups

```
From: sd345@city.ac.uk (Michael Collier)
Subject: Converting images to HP LaserJet III?
Nntp-Posting-Host: hampton
Organization: The City University
Lines: 14

Does anyone know of a good way (standard PC application/PD utility) to
convert tif/img/tga files into LaserJet III format.  We would also like to
do the same, converting to HPGL (HP plotter) files.

Please email any response.
```

- What feature maps could we apply over these sorts of documents?
- A simple approach – bag-of-words (BOW).
  - Assign a fixed integer id to each word occurring in any document of the training set.
  - For each document i, count the number of occurrences of each word w and store it (sparsely) as $doc_i[w] = j_w ==$ count of word w in document i.
  - The BOW representation implies that $n_{features}$ is the number of distinct words in the corpus.
  - What is the feature map $\phi(x)$?
  - $\phi(x) = [j_{word_1}, \cdots, j_{word_{n_{words}}}]$

Example: Classifying documents from 20 newsgroups

- Here's the classifier we'll fit (note we're adding the `TfidfTransformer` to scale by inverse document frequency, since it improves performance on this task – if you're not familiar with TF-IDF see the docs).

```
from sklearn.linear_model import SGDClassifier
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.pipeline import Pipeline
text_clf = Pipeline([('vect', CountVectorizer()),
                     ('tfidf', TfidfTransformer()),
                     ('clf', SGDClassifier(loss='hinge', penalty='l2',
                                           alpha=1e-3, random_state=42,
                                           max_iter=5, tol=None)),
])
text_clf.fit(twenty_train.data, twenty_train.target)
```

- Which named steps in this Pipeline comprise our feature map ϕ?

Example: Classifying documents from 20 newsgroups

```
predicted = text_clf.predict(docs_test)
np.mean(predicted == twenty_test.target)
```

```
0.912782956058585
```

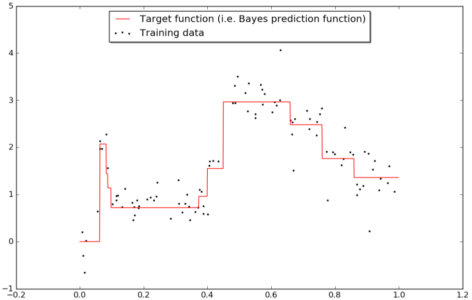
```
from sklearn import metrics
print(metrics.classification_report(twenty_test.target, predicted,
      target_names=twenty_test.target_names))
```

```
                        precision    recall  f1-score   support

           alt.atheism       0.95      0.81      0.87       319
         comp.graphics       0.88      0.97      0.92       389
               sci.med       0.94      0.90      0.92       396
soc.religion.christian       0.90      0.95      0.93       398

           avg / total       0.92      0.91      0.91      1502
```

- Key takeaway: need feature map ϕ when dealing with inputs not natively in $\mathbf{R}^d$.

Motivation

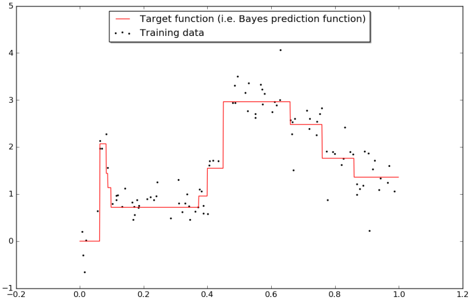
- Two motivations for thinking about feature extraction:
    - Motive 2 – Improving performance. Think about HW2.



- What was our feature map $\phi(x)$? $\phi(x) \in \mathbf{R}^k$ for what k?

Motivation

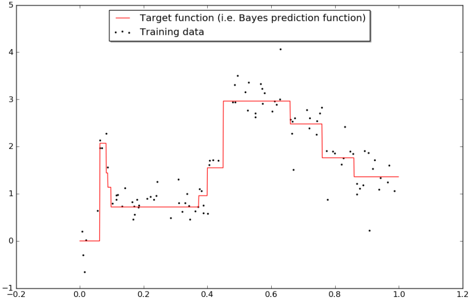
- Two motivations for thinking about feature extraction:
  - Motive 2 – Improving performance. Think about HW2.



- What was our feature map $\phi(x)$? $\phi(x) \in \mathbf{R}^k$ for what k?
- $\phi(x) = \left[\mathbb{1}(x \geqslant \frac{1}{400}), \cdots, \mathbb{1}(x \geqslant \frac{399}{400})\right]$

Motivation

- Two motivations for thinking about feature extraction:
    - Motive 2 – Improving performance. Think about HW2.



- Why did we use this feature map instead of just learning a prediction function $y \sim x$?

Motivation

- Two motivations for thinking about feature extraction:
  - Motive 2 – Improving performance. Toy Example:

```
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 2 – Improving performance. Toy Example:

```
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

- We'll be looking at regression examples throughout this lab.
- Using Elastic Net in sklearn, the default score method returns the coefficient of determination R^2 of the prediction.
- Recall:

  - The total sum of squares (proportional to the variance of the data):

$$SS_{\text{tot}} = \sum_i (y_i - \bar{y})^2,$$

  - The regression sum of squares, also called the explained sum of squares:

$$SS_{\text{reg}} = \sum_i (f_i - \bar{y})^2,$$

  - The sum of squares of residuals, also called the residual sum of squares:

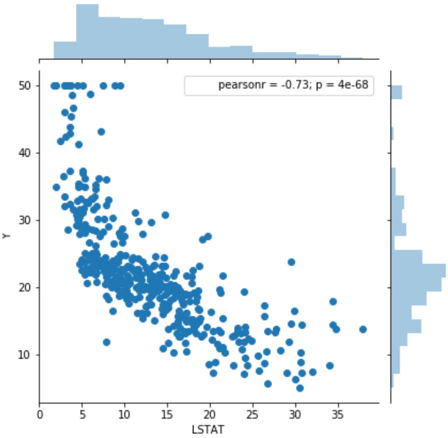
$$SS_{\text{res}} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2$$

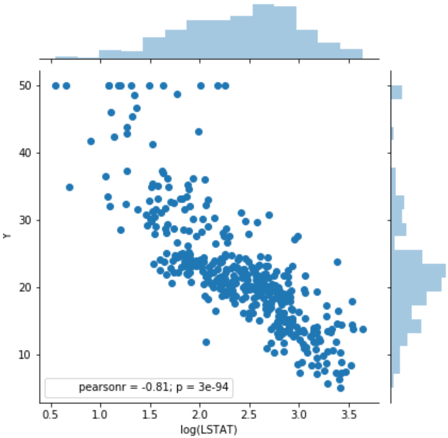
The most general definition of the coefficient of determination is

$$R^2 \equiv 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}.$$

Motivation

- Key idea: instead of using more flexible (i.e. non-linear) models, build better features.





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):

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

- 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 on the parameters for rare buckets?

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 on 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 $P : \mathcal{X} \to \{\text{True}, \text{False}\}$.

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:

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

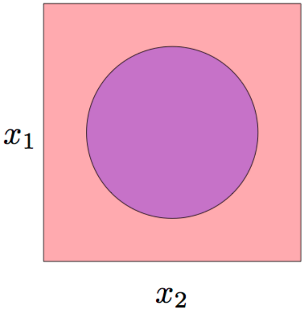
- Linear in w? Yes.
- Linear in $\phi(x)$? Yes.
- Linear in x? No.
  - Linearity not even defined unless $\mathcal{X}$ is a vector space

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.

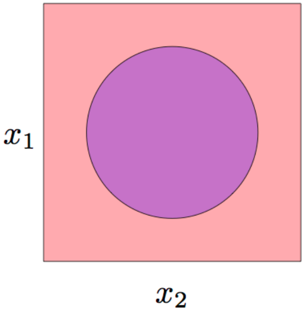
Geometric Example: Two class problem, nonlinear boundary



- 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.

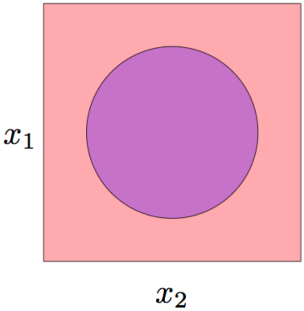
Geometric Example: Two class problem, nonlinear boundary



- 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

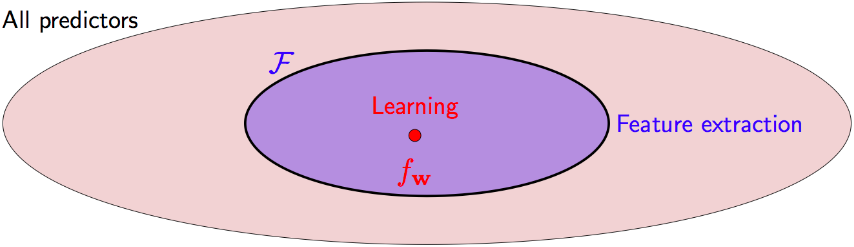
- Consider a linear hypothesis space with a feature map $\phi : \mathcal{X} \to \mathbf{R}^d$:

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

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

- 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.

Example 1: Boston housing and Abalone

Boston Housing

- Let's revisit the Boston housing dataset from the start of lab.
- We're going to be predicting the median house values in Boston suburbs.
- We'll build our feature map using `sklearn` and `sklearn_pandas`

```
import pandas as pd
import numpy as np

from sklearn.base import TransformerMixin
from sklearn.preprocessing import OneHotEncoder,LabelEncoder
from sklearn_pandas import DataFrameMapper
from sklearn.pipeline import Pipeline
from sklearn.linear_model import ElasticNetCV
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

- Set up data:

```
from sklearn.datasets import load_boston
```

```
data = load_boston()
df = data.data
cols = ['CRIM','ZN','INDUS','CHAS','NOX',
        'RM','AGE','DIS','RAD','TAX',
        'PTRATIO','B','LSTAT']
df = pd.DataFrame(df, columns=cols)
train_X, test_X, train_y, test_y = train_test_split(df, data.target,
                                                    test_size=0.2,
                                                    random_state = 2142018)
```

```
categorical = ['CHAS','RAD']
numeric = ['CRIM','ZN','INDUS','NOX','RM',
        'AGE','DIS','TAX','PTRATIO','B',
        'LSTAT']
```

Boston Housing

- Feature map 1– looking at the code, what is the feature map ϕ_1?

```
mapper = DataFrameMapper(
    [(col, None) for col in numeric] + \
    [(col, OneHotStrings()) for col in categorical])

pipe = Pipeline([
    ('mapper',mapper),
    ('clf', ElasticNetCV(cv=10,
                         l1_ratio=[.5, .7, .9, .95, .99, 1],
                         normalize=True))
])

pipe.fit(train_X, train_y)

print('Train score: ',pipe.score(train_X, train_y))
print('Test score: ',pipe.score(test_X, test_y))
```

```
Train score:  0.762945577283
Test score:  0.645054949322
```

- Feature map 1– looking at the code, what is the feature map ϕ_1?

```
mapper = DataFrameMapper(
    [(col, None) for col in numeric] + \
    [(col, OneHotStrings()) for col in categorical])

pipe = Pipeline([
    ('mapper',mapper),
    ('clf', ElasticNetCV(cv=10,
                         l1_ratio=[.5, .7, .9, .95, .99, 1],
                         normalize=True))
])

pipe.fit(train_X, train_y)

print('Train score: ',pipe.score(train_X, train_y))
print('Test score: ',pipe.score(test_X, test_y))
```

```
Train score:  0.762945577283
Test score:  0.645054949322
```

- $\phi_1(X)$ dummy encodes categoricals and passes numeric features untouched.

- Feature map 2– looking at the code, what is the feature map ϕ_2?

```
mapper2 = DataFrameMapper(
    [(numeric, PolynomialFeatures(degree=2))] + \
    [(col, OneHotStrings()) for col in categorical])

pipe2 = Pipeline([
    ('mapper',mapper2),
    ('clf', ElasticNetCV(cv=10,
                         l1_ratio=[.1, .5, .7, .9, .95, .99, 1],
                         normalize=True))
])

pipe2.fit(train_X, train_y)

print('Train score: ',pipe2.score(train_X, train_y))
print('Test score: ',pipe2.score(test_X, test_y))
```

```
Train score:  0.879364391327
Test score:  0.816386882677
```

Boston Housing

- Feature map 2– looking at the code, what is the feature map ϕ_2?

```
mapper2 = DataFrameMapper(
    [(numeric, PolynomialFeatures(degree=2))] + \
    [(col, OneHotStrings()) for col in categorical])

pipe2 = Pipeline([
    ('mapper',mapper2),
    ('clf', ElasticNetCV(cv=10,
                         l1_ratio=[.1, .5, .7, .9, .95, .99, 1],
                         normalize=True))
])

pipe2.fit(train_X, train_y)

print('Train score: ',pipe2.score(train_X, train_y))
print('Test score: ',pipe2.score(test_X, test_y))
```

```
Train score:  0.879364391327
Test score:  0.816386882677
```

- $\phi_2(X)$ dummy encodes categoricals and maps numeric features to polynomial features of degree $d \leqslant 2$.

Abalone

- Here we are using the abalone dataset – predicting the number of rings on an abalone (a kind of shellfish).
- Set up data:

```
df = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/' +
                 'abalone/abalone.data',
                 header=None)

df = df.rename(columns={
    0:'sex', 1:'length', 2:'diameter', 3:'height',
    4:'whole_weight', 5:'shucked_weight', 6:'viscera_weight',
    7:'shell_weight', 8:'rings'
})

categorical = ['sex']

numeric = ['length', 'diameter', 'height',
        'whole_weight', 'shucked_weight',
        'shell_weight']

train_X, test_X, train_y, test_y = train_test_split(df.drop('rings', axis=1),
                                                    df['rings'],
                                                    random_state=42)
```

Abalone

- $\phi_1(X)$ dummy encodes categoricals and passes numeric features untouched.

```
mapper = DataFrameMapper(
    [(col, None) for col in numeric] + \
    [(col, OneHotStrings()) for col in categorical])

pipe = Pipeline([
    ('mapper',mapper),
    ('clf', ElasticNetCV(cv=10,
                         l1_ratio=[.1, .5, .7, .9, .95, .99, 1],
                         normalize=True))
])

pipe.fit(train_X, train_y)

print('Train score: ',pipe.score(train_X, train_y))
print('Test score: ',pipe.score(test_X, test_y))
```

```
Train score:  0.528393673016
Test score:  0.534127249172
```

Abalone

- $\phi_2(X)$ dummy encodes categoricals and maps numeric features to polynomial features of degree $d \leqslant 2$.

```
poly_mapper = DataFrameMapper(
    [(numeric, PolynomialFeatures(degree=2))] + \
    [(col, OneHotStrings()) for col in categorical])

pipe_poly = Pipeline([
    ('mapper',poly_mapper),
    ('clf', ElasticNetCV(cv=10,
                         l1_ratio=[.1, .5, .7, .9, .95, .99, 1],
                         normalize=True))
])

pipe_poly.fit(train_X, train_y)

print('Train score: ',pipe_poly.score(train_X, train_y))
print('Test score: ',pipe_poly.score(test_X, test_y))
```

```
Train score:  0.559442492704
Test score:  0.554318625419
```

Comparing performance

- Why did the performance improve much more for the Boston Housing dataset versus the Abalone dataset when we used map 2?

Comparing performance

- Why did the performance improve much more for the Boston Housing dataset versus the Abalone dataset when we used map 2?

- What is the Bayes prediction function for square loss?

Comparing performance

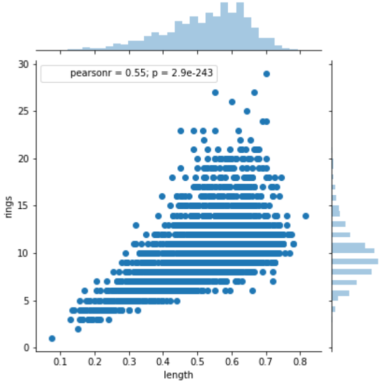
- Why did the performance improve much more for the Boston Housing dataset versus the Abalone dataset when we used map 2?

- What is the Bayes prediction function for square loss?
- If $E[Y|X]$ is linear in $\phi_1(X)$, will we improve performance using $\phi_2(X)$?

Comparing performance

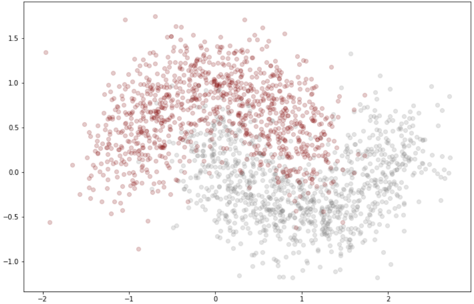
- Why did the performance improve much more for the Boston Housing dataset versus the Abalone dataset when we used map 2?

- What is the Bayes prediction function for square loss?
- If $E[Y|X]$ is linear in $\phi_1(X)$, will we improve performance using $\phi_2(X)$?
- Do we typically know in advance the structure of $E[Y|X]$?



Example 2: Two moons data

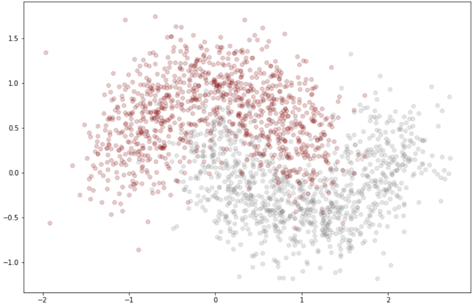
Two moons setup



- What feature maps might be helpful for this problem?

From Alice Zheng, Amanda Casari, *Feature Engineering for Machine Learning*

Two moons setup



- What feature maps might be helpful for this problem?
- We'll try binning data instances using k-means (and for fun random tree embeddings) – let's look at the transformers (*in notebook*).

From Alice Zheng, Amanda Casari, *Feature Engineering for Machine Learning*

Two moons setup

- Here's the pipeline. First, notice the sklearn class `FeatureUnion`, which let's us easily apply multiple feature maps over an input array.
- What is the feature map $\phi(X)$? What will `transformed.shape[1]` equal?

```
pipe = Pipeline([
    ('feats',FeatureUnion([
        ('kmeans', KMeansFeaturizer(k=100,random_state=2052018)),
        ('ID', IdentityFeaturizer())
    ])),
    ('clf', LogisticRegressionCV())
])

pipe.fit(training_data, training_labels)

# Just to make sure it's clear what this does:
transformed = pipe.named_steps['feats'].transform(training_data)
```

```
transformed.shape
```

Two moons setup

- Here's the pipeline. First, notice the sklearn class `FeatureUnion`, which let's us easily apply multiple feature maps over an input array.
- What is the feature map $\phi(X)$? What will `transformed.shape[1]` equal?

```
pipe = Pipeline([
    ('feats',FeatureUnion([
        ('kmeans', KMeansFeaturizer(k=100,random_state=2052018)),
        ('ID', IdentityFeaturizer())
    ])),
    ('clf', LogisticRegressionCV())
])

pipe.fit(training_data, training_labels)

# Just to make sure it's clear what this does:
transformed = pipe.named_steps['feats'].transform(training_data)
```

```
transformed.shape
```

- $\phi(X) = [X_1, X_2, \mathbb{1}[(X_1, X_2) \text{ binned to centroid } 1], \cdots, \mathbb{1}[(X_1, X_2) \text{ binned to centroid } 100]]$

Two moons setup

- Here's the pipeline. First, notice the sklearn class `FeatureUnion`, which let's us easily apply multiple feature maps over an input array.
- What is the feature map $\phi(X)$? What will `transformed.shape[1]` equal?

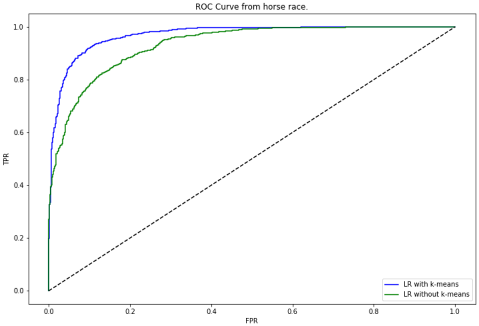
```
pipe = Pipeline([
    ('feats',FeatureUnion([
        ('kmeans', KMeansFeaturizer(k=100,random_state=2052018)),
        ('ID', IdentityFeaturizer())
    ])),
    ('clf', LogisticRegressionCV())
])

pipe.fit(training_data, training_labels)

# Just to make sure it's clear what this does:
transformed = pipe.named_steps['feats'].transform(training_data)
```

```
transformed.shape
```

```
(2000, 102)
```

Two moons feature map

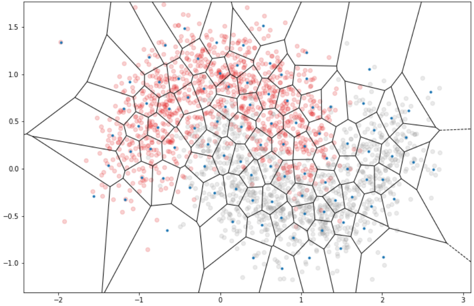
- Let's fit this pipe, and compare to a baseline logistic regression over just $\phi_I(X) = X$.



- We see performance improve.

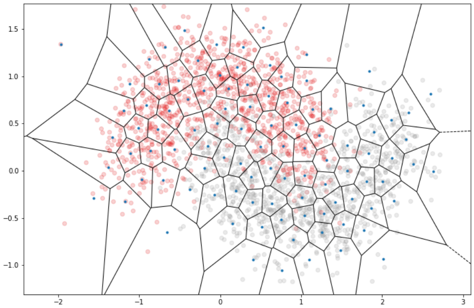
Two moons feature map

- Here's the voronoi diagram after fitting the `KMeansFeaturizer` (fit in `pipe.fit` call).



- Intuitively, why did this improve performance?

Two moons feature map



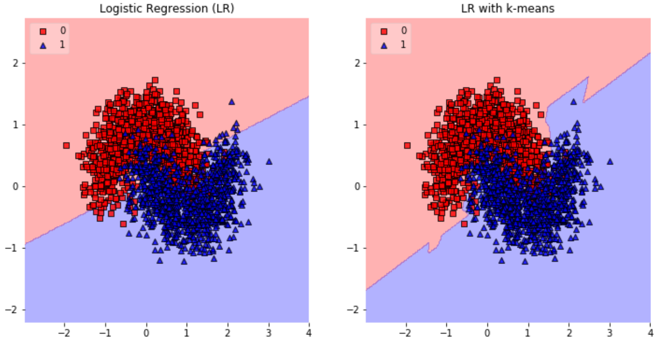
- Think back to the 1D discretization discussed earlier – which map is this analogous to?

$$\phi_2(x) = (1(5 \leqslant N(x) < 10), 1(10 \leqslant N(x) < 100), 1(100 \leqslant N(x)))$$

$$\phi_1(x) = (1(5 \leqslant N(x)), 1(10 \leqslant N(x)), 1(100 \leqslant N(x)))$$

Two moons feature map

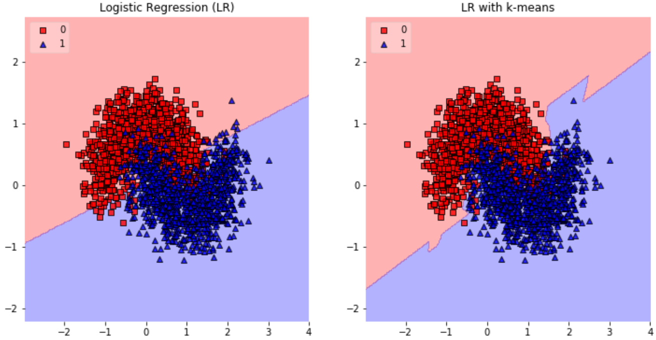
- Here's a comparison of decision boundaries (note made using `mlexend`).



- What's with the plotted decision boundaries? I thought logistic regression was linear?

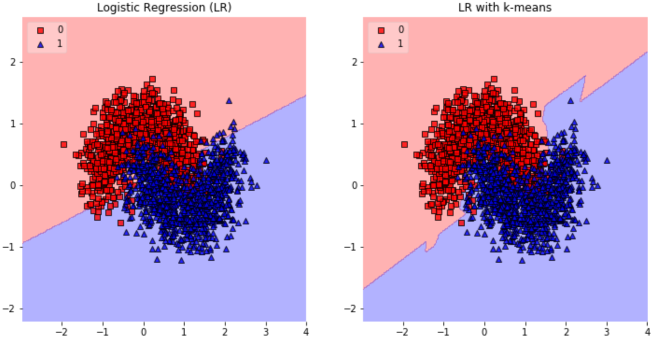
Two moons feature map

- What's with the plotted decision boundaries? I thought logistic regression was linear?



Two moons feature map

- What's with the plotted decision boundaries? I thought logistic regression was linear?



- Both decision boundaries are affine, but with k-means embedding it's affine in $\mathbf{R}^{102}$.

Learning Objectives

- Understand where a *feature map* sits in a machine learning pipeline.
- Understand that featurization/featuring mapping is inherently required to allow predictors to ingest many types of data.
- Understand how feature extraction can be used to extend the power of linear methods.
- Build pipelines with expanded feature spaces using the sklearn ecosystem.