MY474: Applied Machine Learning for Social Science

Lecture 5: Model Selection

Blake Miller

30 October 2019

Agenda

1. Feature selection
   - Subset selection
   - Forward subset selection
   - Backward subset selection
   - Regularization (more week 7)
2. Hyperparameter selection
   - Grid search
   - Random search
   - Bayesian search
3. Feature engineering
   - Text data
   - Image data
4. Feature learning
   - Text data
   - Image data

Feature selection

Feature selection

- Problem: Which features are informative?
  - Uninformative features add **noise**
  - Some features are highly **correlated**
- There are three main approaches to feature selection:
  1. **Subset Selection:** Identify a subset of p
  2. **Shrinkage/Regularization:** Use all p predictors, but **shrink** the size of coefficients towards zero (relative to the least squares estimates). We will discuss this in lecture next week.
  3. **Dimension Reduction:** Project the p predictors into a M-dimensional subspace, where $M < p$. We will discuss this in detail in lecture 9.

Why would we want fewer features?

- For **inference**, we may want to simplify the model
  - Some features may be **correlated** which can create problems when interpreting coefficients (**collinearity**)
- A large number of features may result in **longer training times**; similar performance may be achieved with dramatically faster training.
- Shrinking the number of predictors can **reduce variance** and prevent **overfilling**.
  - The chances of overfitting increase with more **irrelevant** or **redundant** features, as these add **noise.**
- One can avoid the **curse of dimensionality** with fewer predictors.

Linear Model Selection and Regularization

- Recall the linear model

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \epsilon.$$

- In the next lecture, we will discuss some extensions to the linear model framework that use **regularization**, one solution to the **model selection** problem.

Subset Selection

Best Subset Selection

1. Let $\mathcal{M}_0$ denote the **null model**, which contains no predictors. This model simply predicts the sample mean for each observation.
2. For $k = 1, 2, \cdots p$:
   - Fit all $\binom{p}{k}$ models that contain exactly k predictors.
   - Pick the best among these p models, and call it $\mathcal{M}_k$. Here best is defined as having the smallest RSS (for OLS), or **deviance** for **generalized linear models** such as **logistic regression** (negative two times the maximized log likelihood)
3. Select a single best model from among $\mathcal{M}_0, \cdots, \mathcal{M}_p$ using cross-validated prediction error.

Stepwise Selection

- For computational reasons, best subset selection cannot be applied with very large p. *Why not?*
- Best subset selection may also suffer from statistical problems when p is large: larger the search space, the higher the chance of finding models that look good on the training data, even though they might not have any predictive power on future data.
- Thus an enormous search space can lead to **overfitting** and **high variance** of the coefficient estimates.
- For both of these reasons, **stepwise methods**, which explore a far more restricted set of models, are attractive alternatives to best subset selection.

Forward Stepwise Selection

- Forward stepwise selection begins with a model containing no predictors, and then adds predictors to the model, one-at-a-time, until all of the predictors are in the model.
- In particular, at each step the variable that gives the greatest **additional** improvement to the fit is added to the model.

Forward Stepwise Selection: details

Forward Stepwise Selection

1. Let $\mathcal{M}_0$ denote the null model, which contains no predictors.
2. For $k = 0, \cdots, p-1$:
   - Consider all $p-k$ models that augment the predictors in $\mathcal{M}_k$ with one additional predictor.
   - Choose the *best* among these $p-k$ models, and call it $\mathcal{M}_{k+1}$. Here *best* is defined as having smallest RSS.
3. Select a single best model from among $\mathcal{M}_0, \cdots, \mathcal{M}_p$ using cross-validated prediction error.

More on Forward Stepwise Selection

- Computational advantage over best subset selection is clear.
- It is not guaranteed to find the best possible model out of all 2^p models containing subsets of the p predictors. *Why not? Give an example.*

Backward Stepwise Selection

- Like forward stepwise selection, **backward stepwise selection** provides an efficient alternative to best subset selection.
- However, unlike forward stepwise selection, it begins with the full least squares model containing all p predictors, and then iteratively removes the least useful predictor, one-at-a-time.

Backward Stepwise Selection: details

Backward Stepwise Selection

1. Let $\mathcal{M}_p$ denote the **full** model, which contains all p predictors.
2. For $k = p, p-1, \cdots, 1$:
   - Consider all k models that contain all but one of the predictors in $\mathcal{M}_k$, for a total of $k-1$ predictors.
   - Choose the best among these k models, and call it $\mathcal{M}_{k-1}$. Here best is defined as having smallest RSS.
3. Select a single best model from among $\mathcal{M}_0, \cdots, \mathcal{M}_p$ using cross-validated prediction error.

More on Backward Stepwise Selection

- Like forward stepwise selection, the backward selection approach searches through only $1 + p(p + 1)/2$ models, and so can be applied in settings where p is too large to apply best subset selection
- Like forward stepwise selection, backward stepwise selection is not guaranteed to yield the **best** model containing a subset of the p predictors.
- Backward selection **requires that the number of samples n is larger than the number of variables p** (so that the full model can be fit). In contrast, forward stepwise can be used even when $n < p$, and so is the only viable subset method when p is very large.

Choosing the Optimal Model

- The model containing all of the predictors will always have the smallest RSS since these quantities are related to the training error.
- We wish to choose a model with low test error, not a model with low training error. Recall that training error is usually a poor estimate of test error.
- Therefore, RSS is not suitable for selecting the best model among a collection of models with different numbers of predictors.

Validation and Cross-Validation

- Each of the procedures returns a sequence of models $\mathcal{M}_k$ indexed by model size $k = 0, 1, 2, \cdots$. Our job here is to select $\hat{k}$. Once selected, we will return model $\mathcal{M}_{\hat{k}}$
- We compute the validation set error or the cross-validation error for each model $\mathcal{M}_k$ under consideration, and then select the k for which the resulting estimated test error is smallest.

Shrinkage Methods

Ridge regression and Lasso

- The subset selection methods use least squares to fit a linear model that contains a subset of the predictors.
- As an alternative, we can fit a model containing all p predictors using a technique that constrains or regularizes the coefficient estimates, or equivalently, that shrinks the coefficient estimates towards zero.
- It may not be immediately obvious why such a constraint should improve the fit, but it turns out that shrinking the coefficient estimates can significantly reduce their variance.

Hyperparameter selection

Hyperparameters

- **Hyperparameters**, also called **tuning parameters**, refer to various attributes of a model that can be chosen in the **model selection** process.
- While **parameters** like β in regression are set during training, **hyperparameters** are set before training.
- **Model hyperparameters** are attributes of models, usually refer to flexibility of fit
- **Algorithm hyperparameters** refer to methods of training the model; these hyperparameters don't affect model performance, but rather the quality and speed of learning.

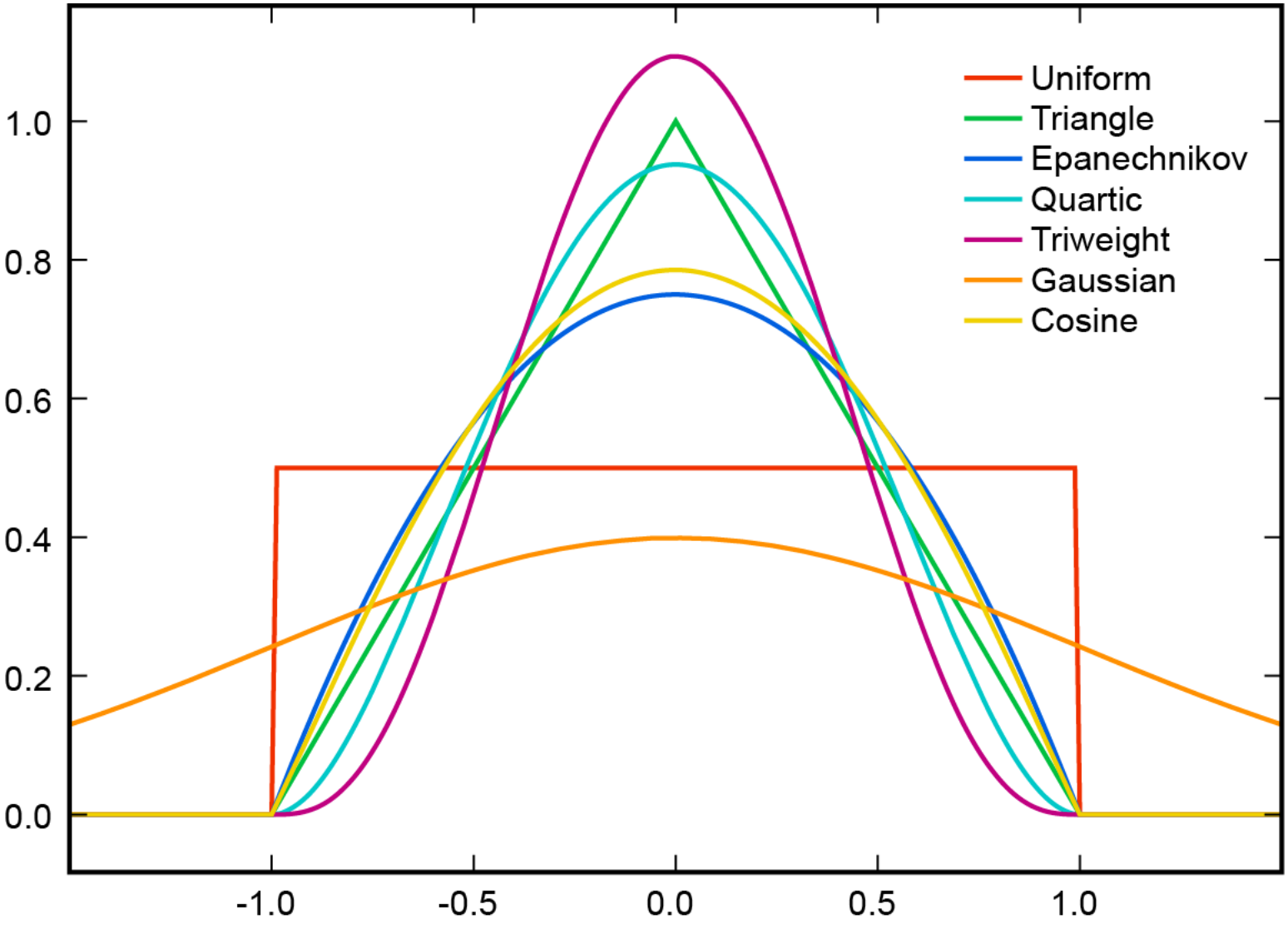
Model hyperparameter example: weighted KNN

- k in KNN is a hyperparameter! But we can use other hyperparameters with KNN.
- In previous examples, we have given equal weight to all neighbors, the equivalent of a "uniform" kernel.
- We could potentially improve performance by adding a weight to each of these neighbors based on their distance from the reference observation.
- Weighted KNN downweights neighbors farther from our target point:

$$\hat{C}(x) = \sum_{i=1}^{k} w_i * y_i; \ 0 < w_i < 1$$

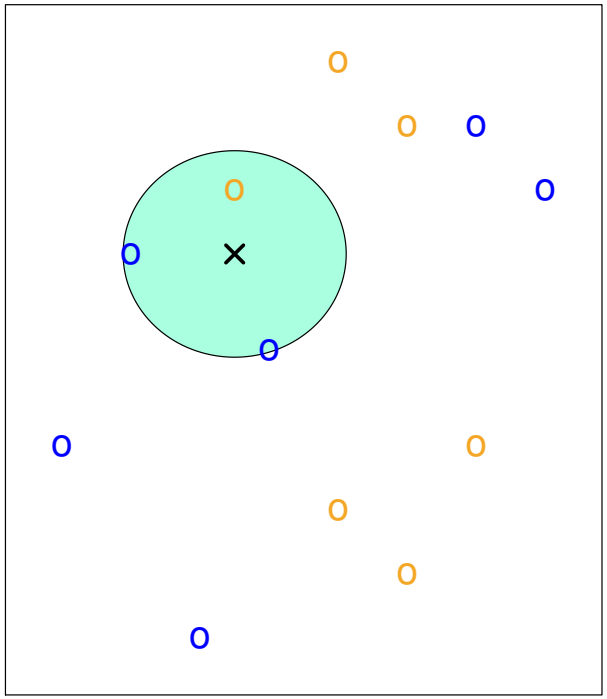
- We want our weight w_i to be
  - small if the distance to our reference point is large
  - large if the distance to our reference point is small

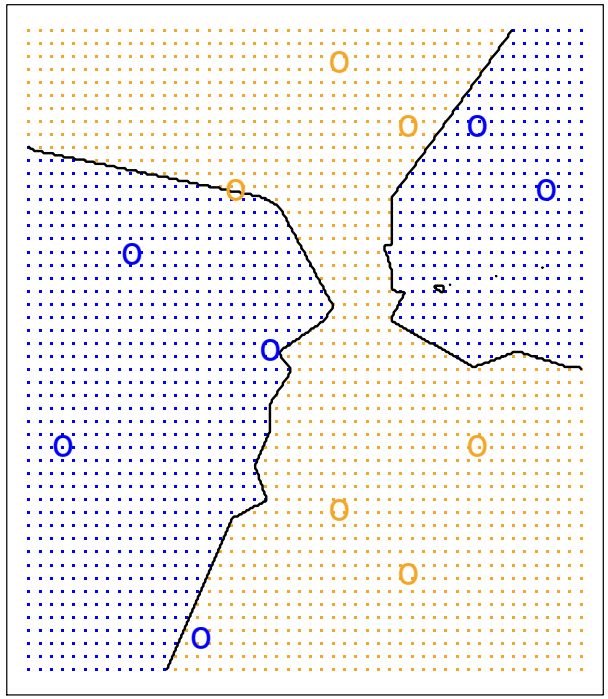
Model hyperparameter example: kernel KNN



We can use kernels that are functions of the distance between points. This is another hyperparameter in addition to k.

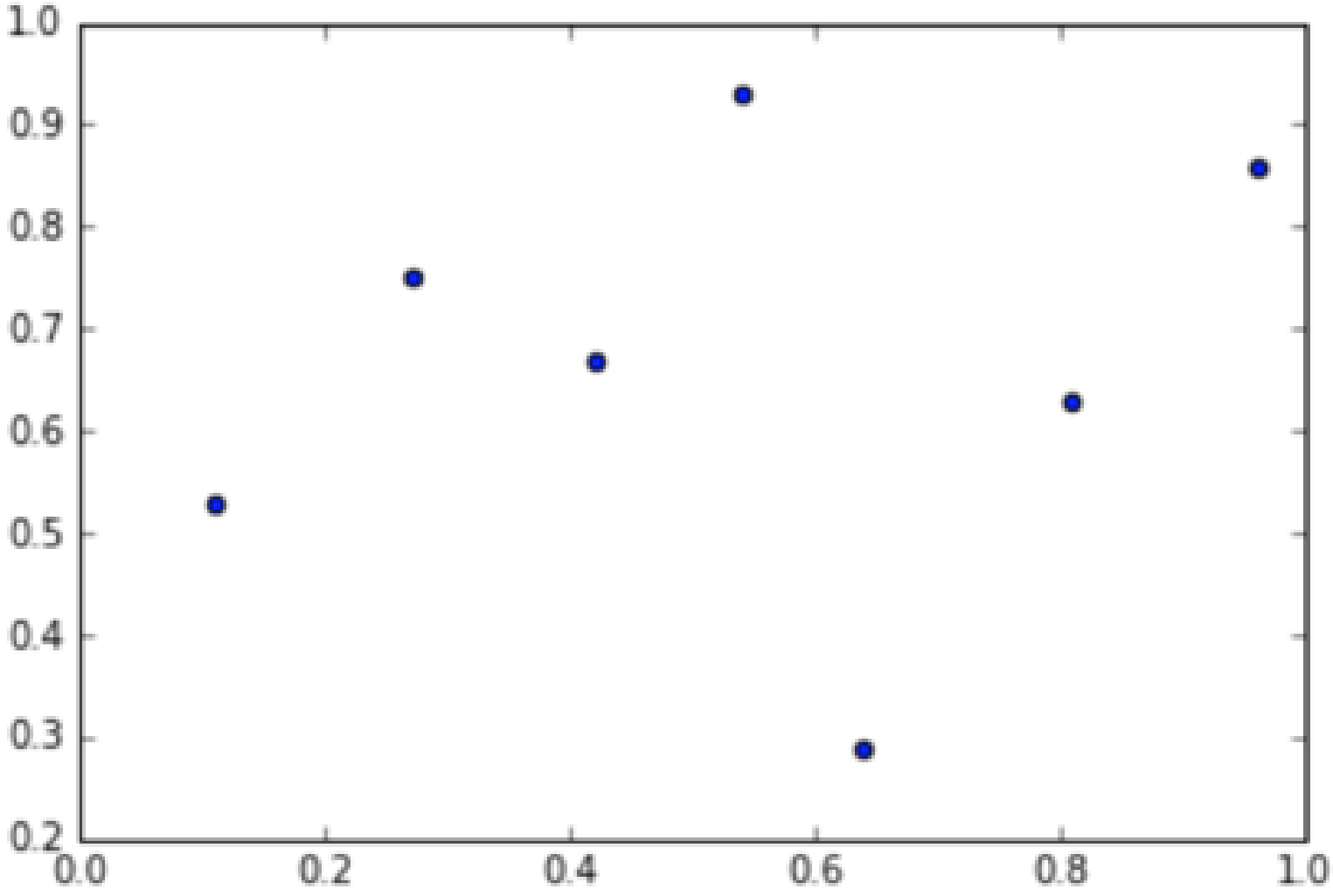
Algorithm hyperparameters: K-D tree vs. brute force





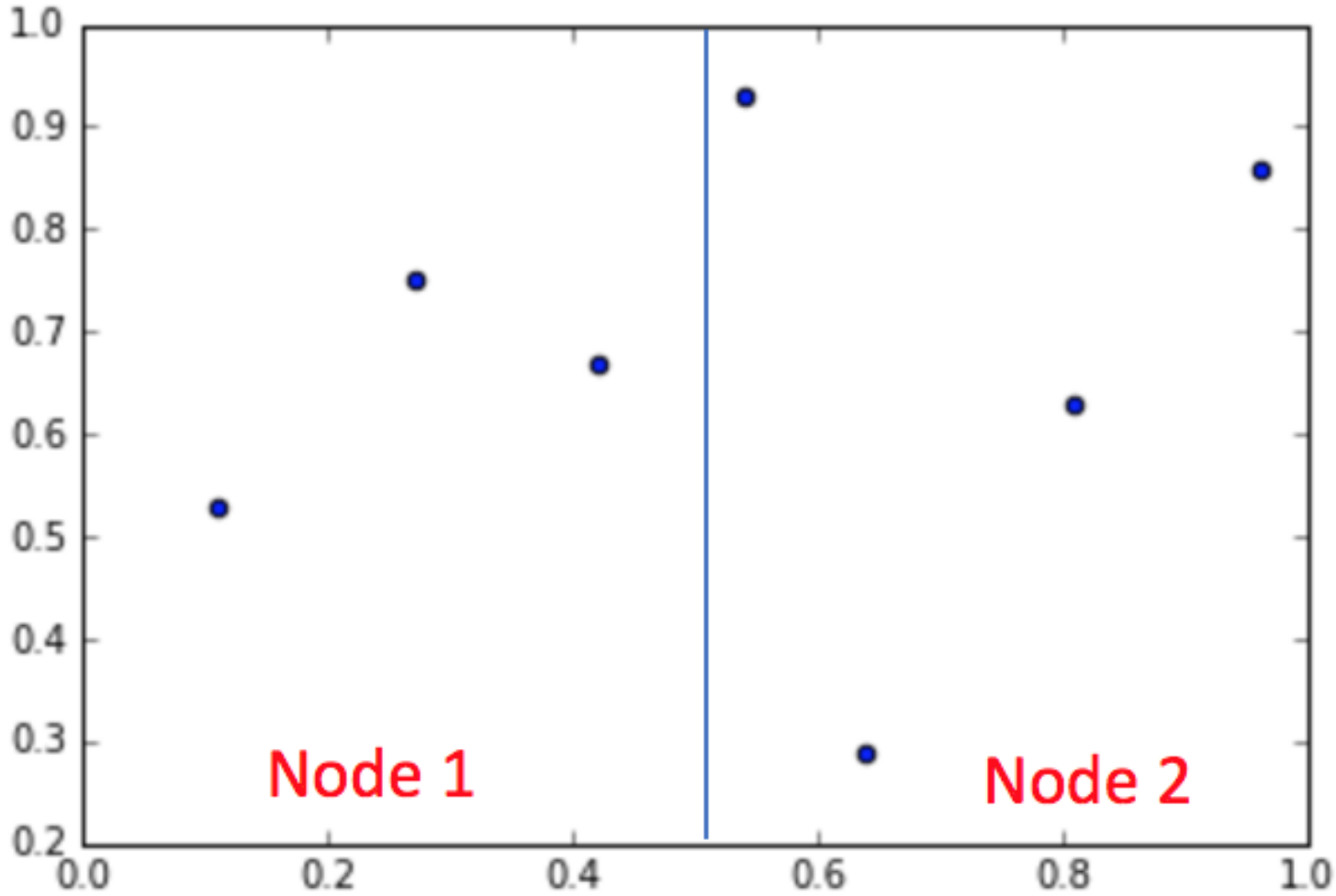
- Recall the KNN algorithm: in order to find the k nearest neighbors, we have to calculate the distance between a reference point and all of our training observations.
- This can be VERY expensive.
- As with many other learning algorithms, often there are computational shortcuts that get you close to the same answer as the original, less efficient algorithm.

Algorithm hyperparameters: K-D tree vs. brute force



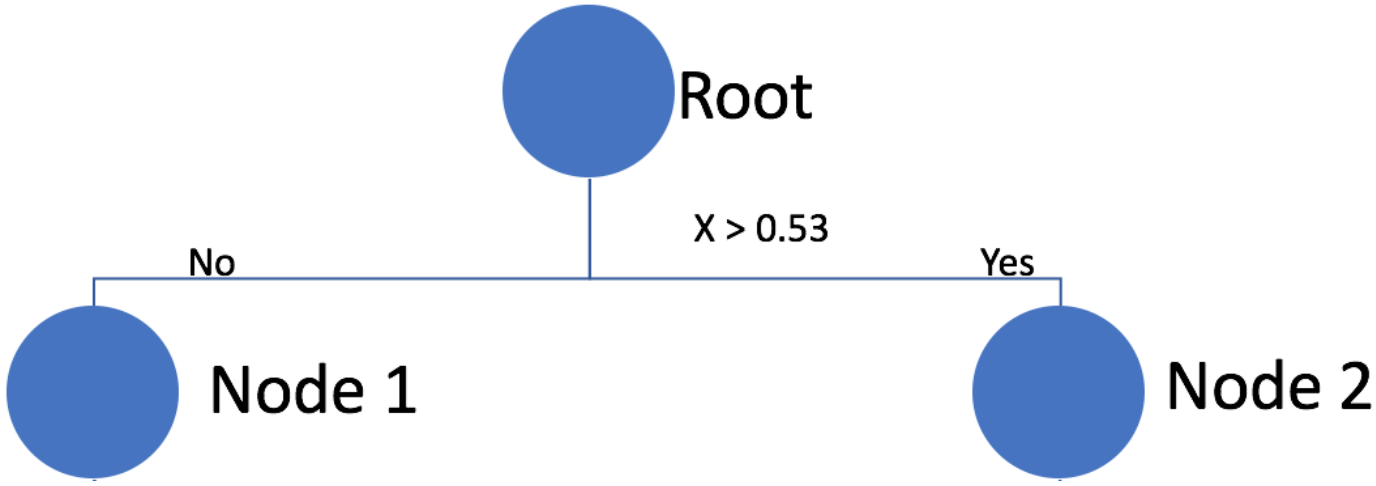
Say we have 2-dimensional training data like the following.

Algorithm hyperparameters: K-D tree vs. brute force



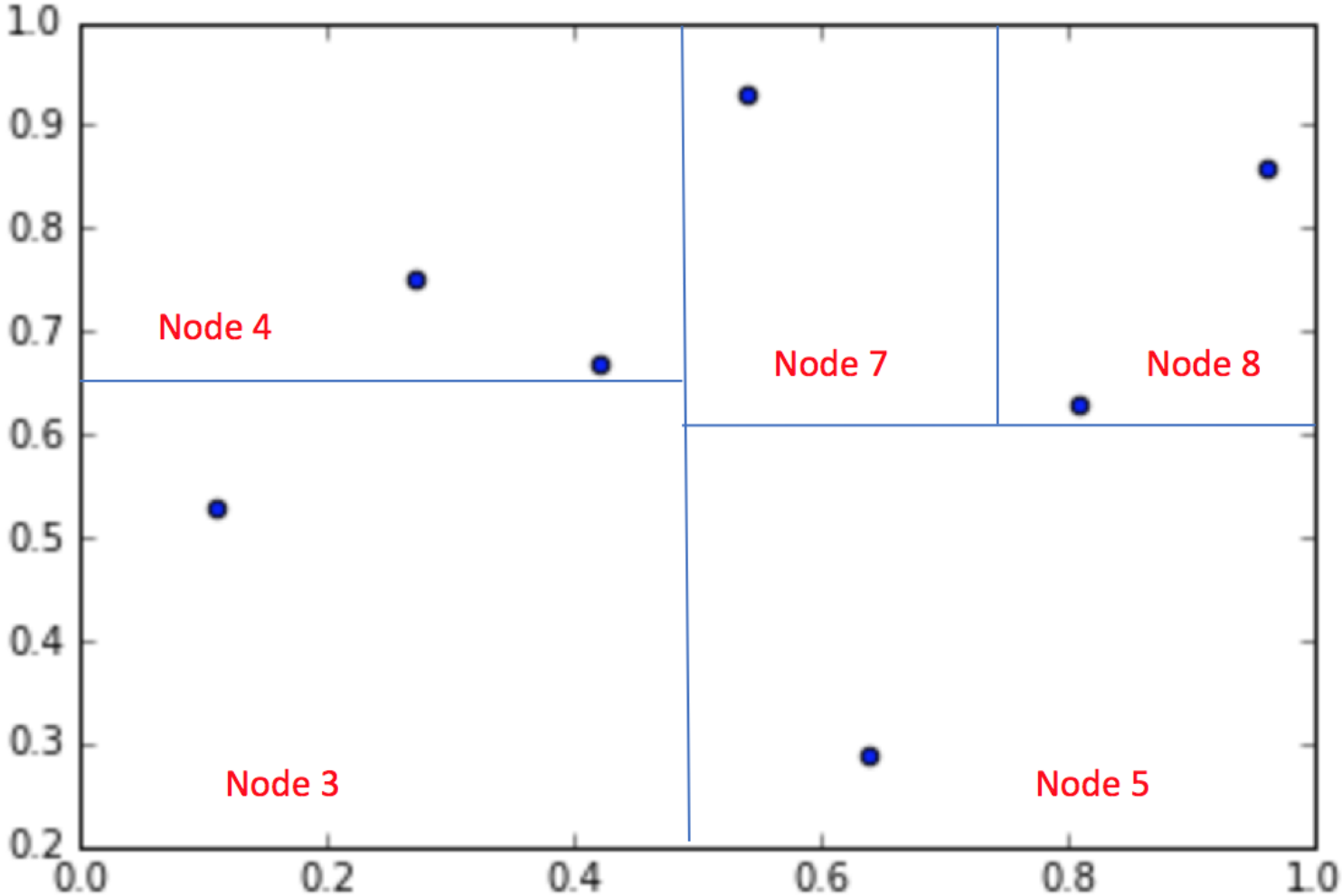
First split a randomly selected feature at the median.

Algorithm hyperparameters: K-D tree vs. brute force



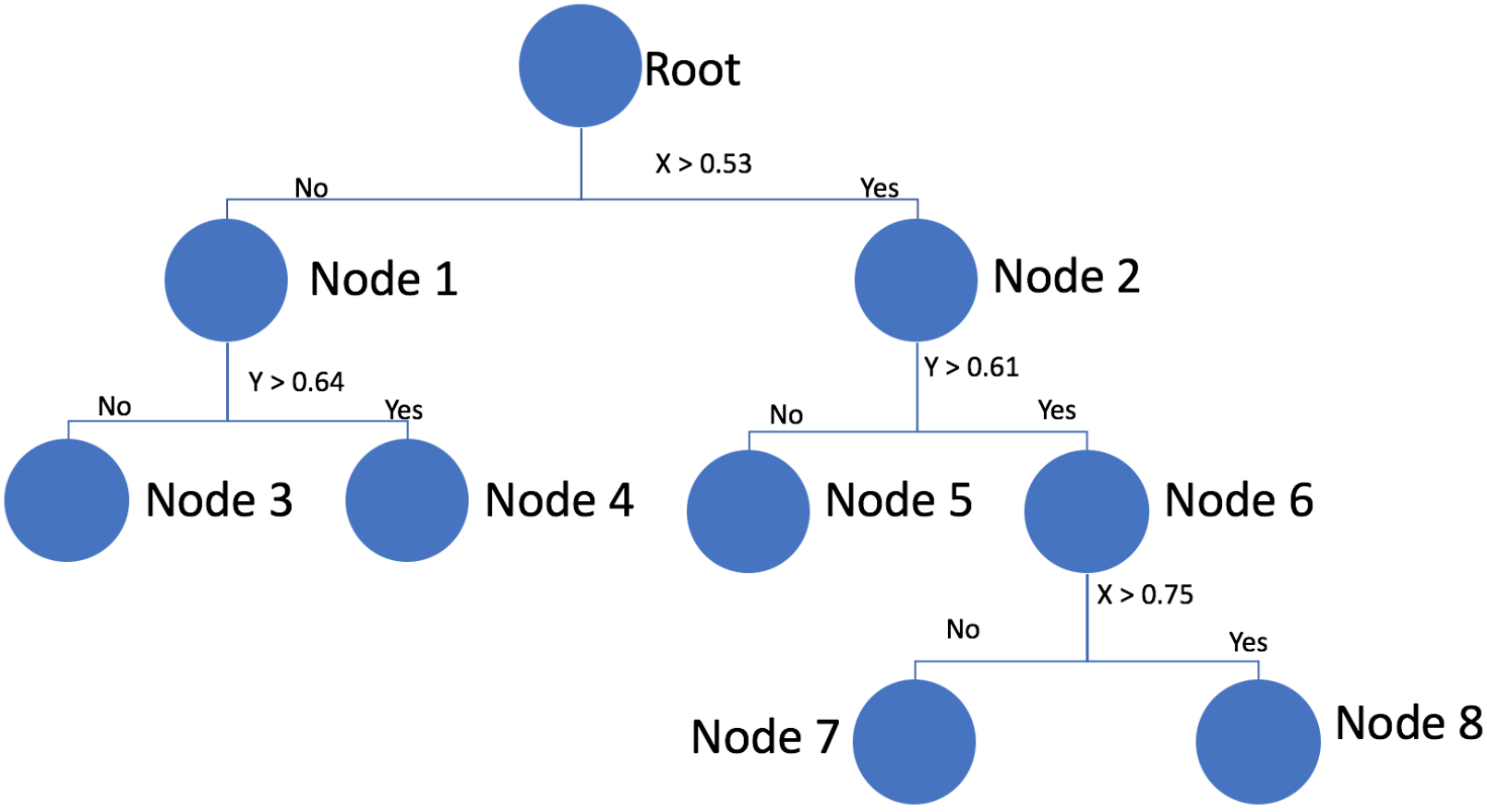
Now you have a tree with two nodes.

Algorithm hyperparameters: K-D tree vs. brute force



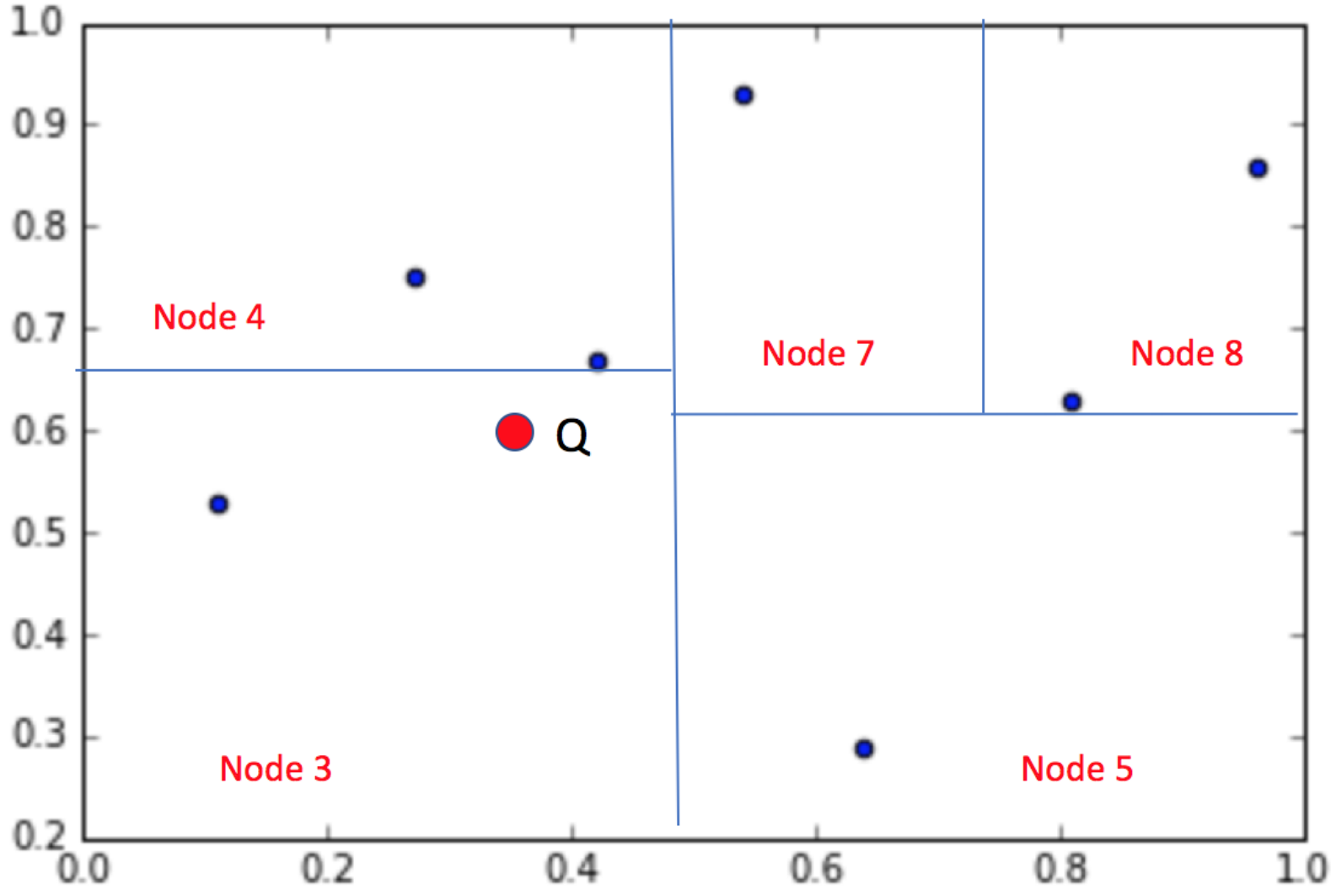
Continue to split the feature space along a random dimension until you have no more than m observations in each bounding box. In this case $m = 2$.

Algorithm hyperparameters: K-D tree vs. brute force



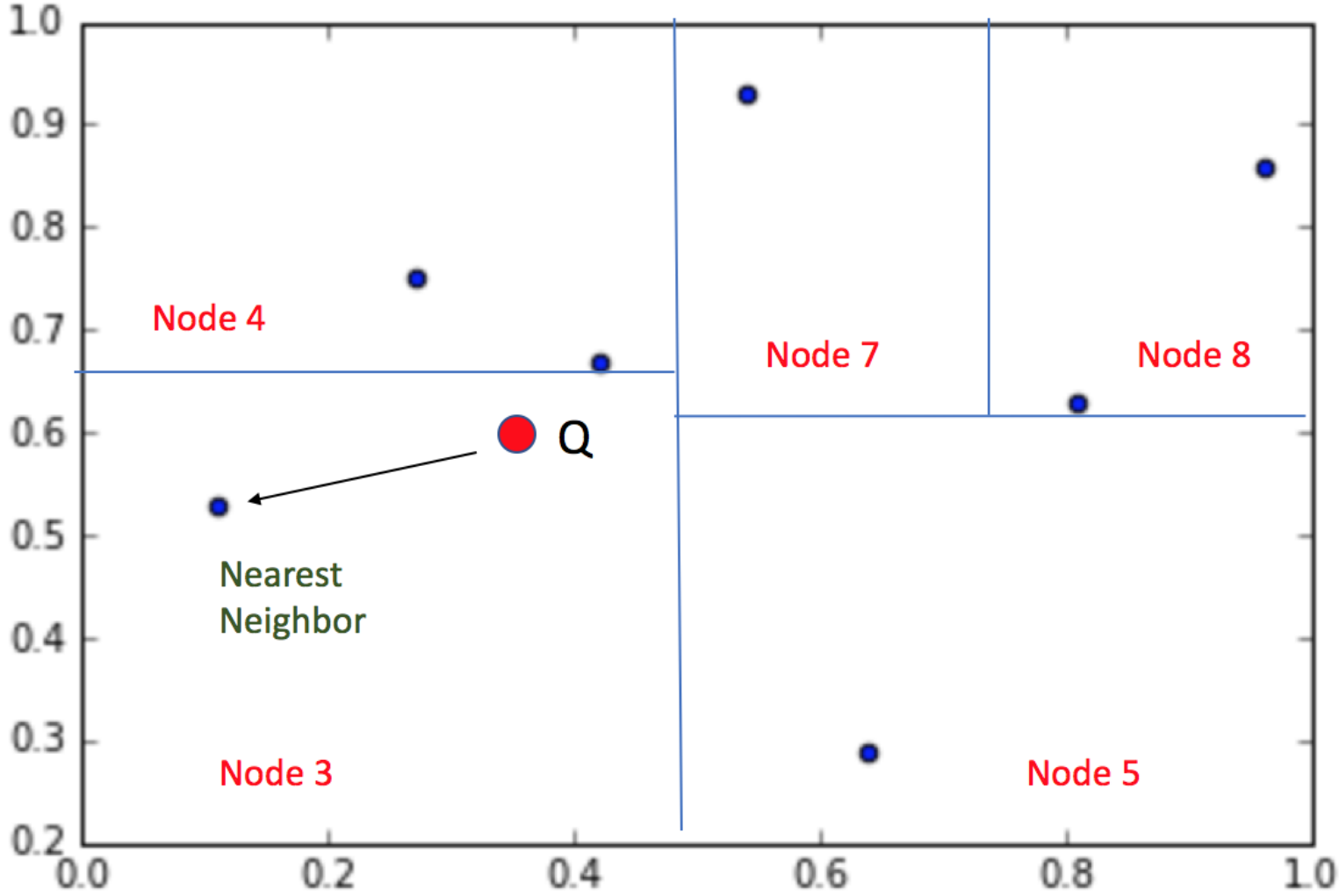
Each of these bounding boxes are represented by terminal nodes on the following tree.

Algorithm hyperparameters: K-D tree vs. brute force



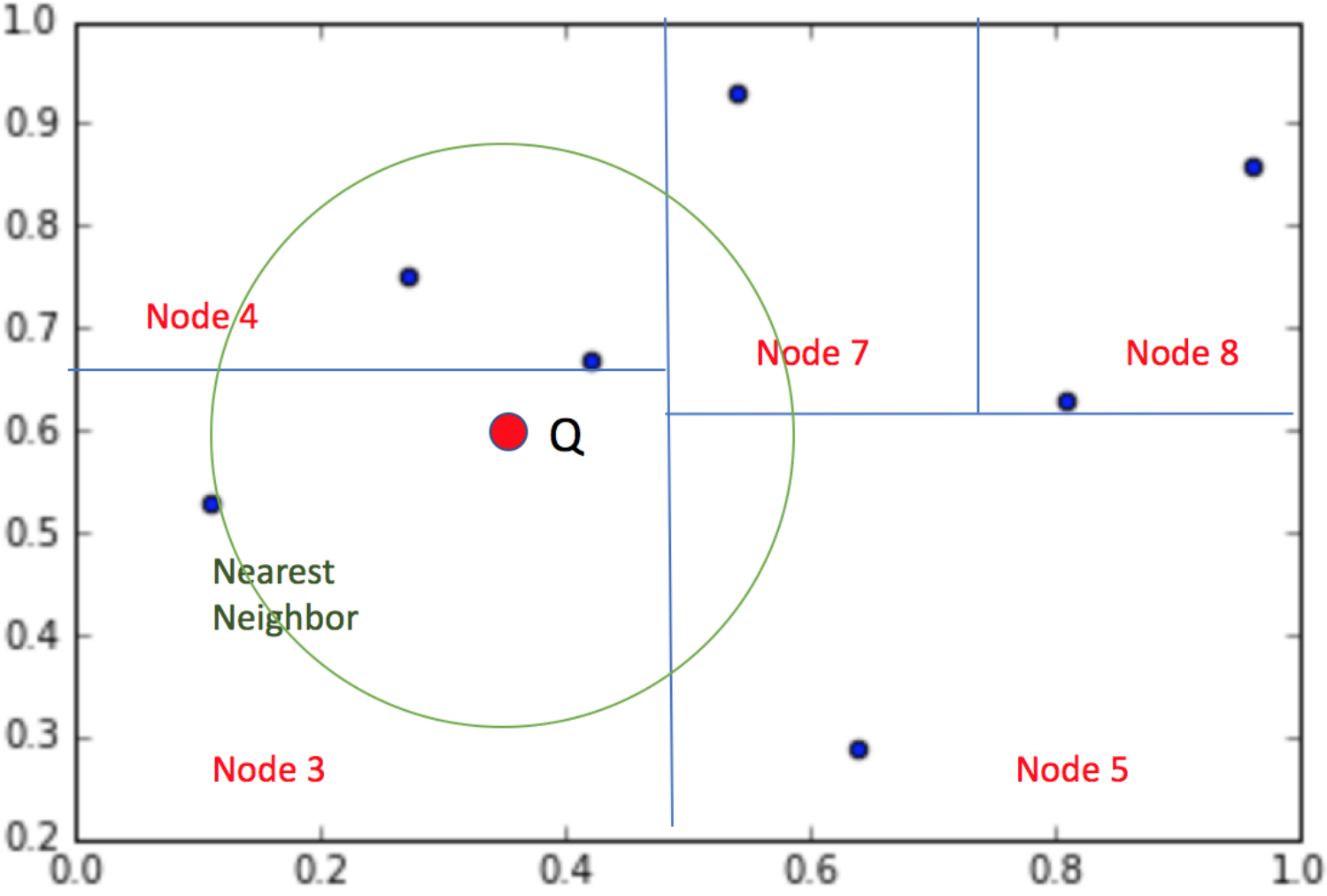
When predicting the class of a new observation, we first determine which bounding box it is in.

Algorithm hyperparameters: K-D tree vs. brute force



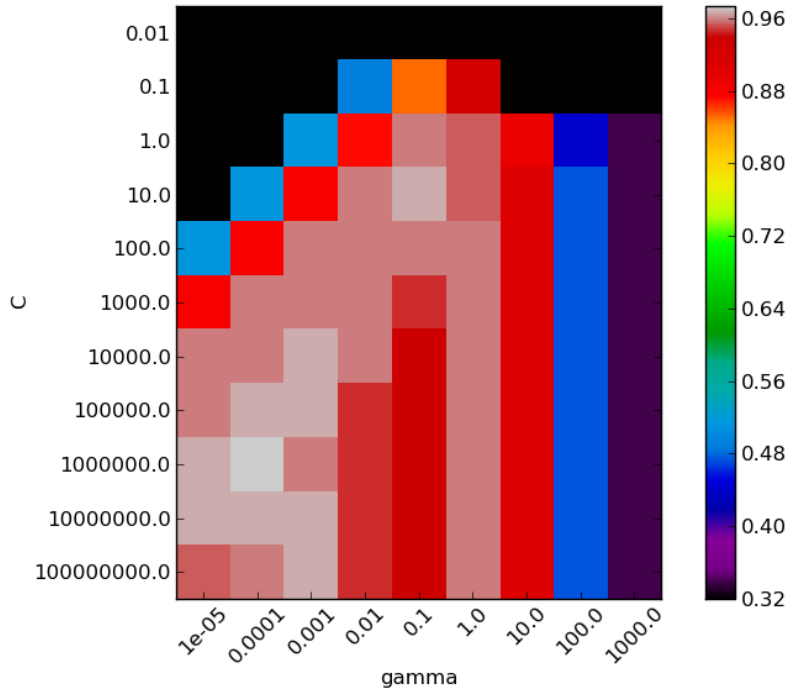
Instead of calculating the distance between all observations in the training data, we restrict our search to the bounding box where the query/reference point resides.

Algorithm hyperparameters: K-D tree vs. brute force



Unfortunately, we oftentimes do not find the true nearest neighbor, but we get close!

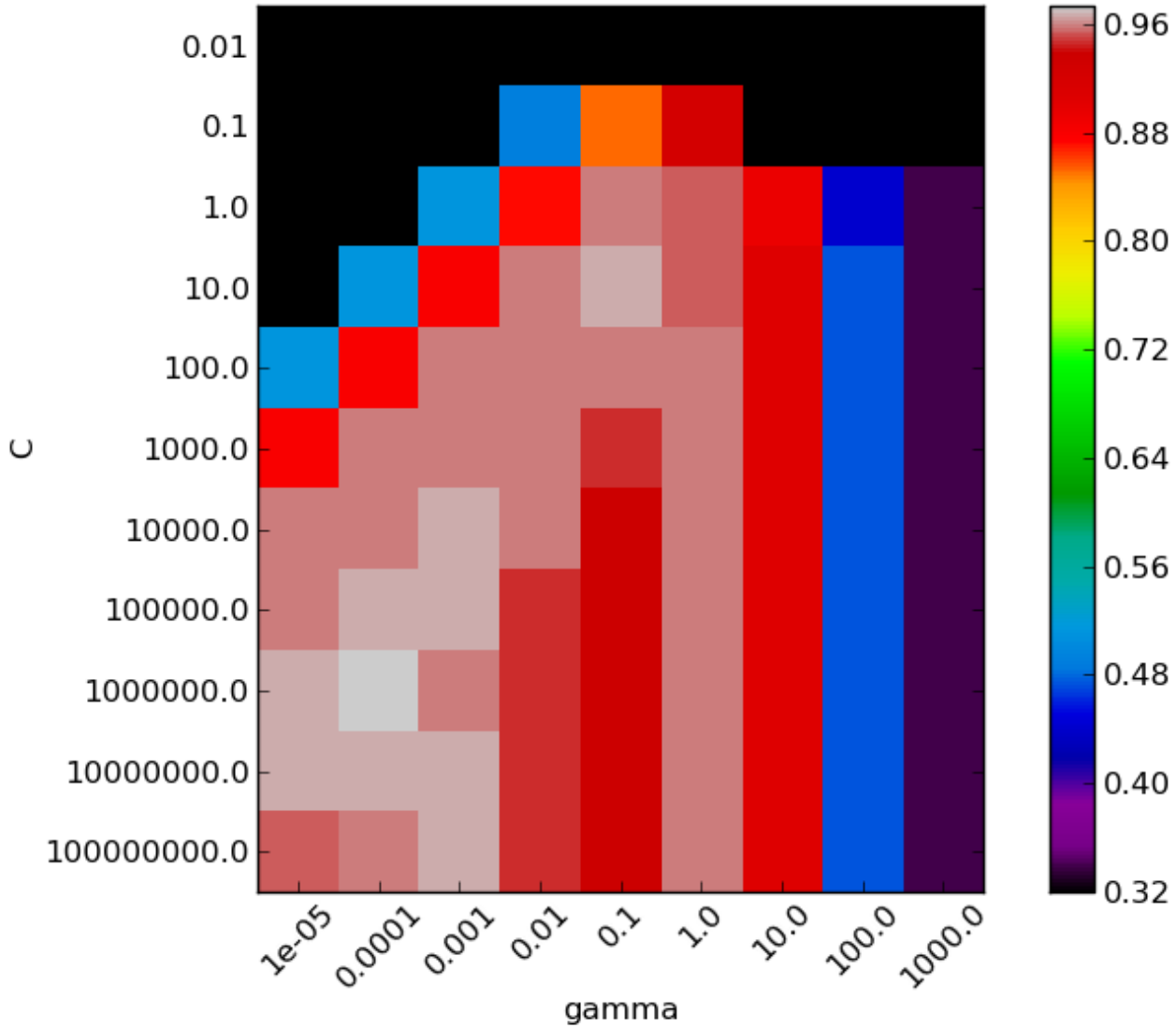
Grid search



Here we see an example of a grid search using two hyperparameters for support vector machines.

- Exhaustive search of all potential parameter combinations.
- Parameters, even when continuous, must be **discretized**
- Suffers from the **curse of dimensionality**. *Why?*

Grid search



Here we see an example of a grid search using two hyperparameters for support vector machines.

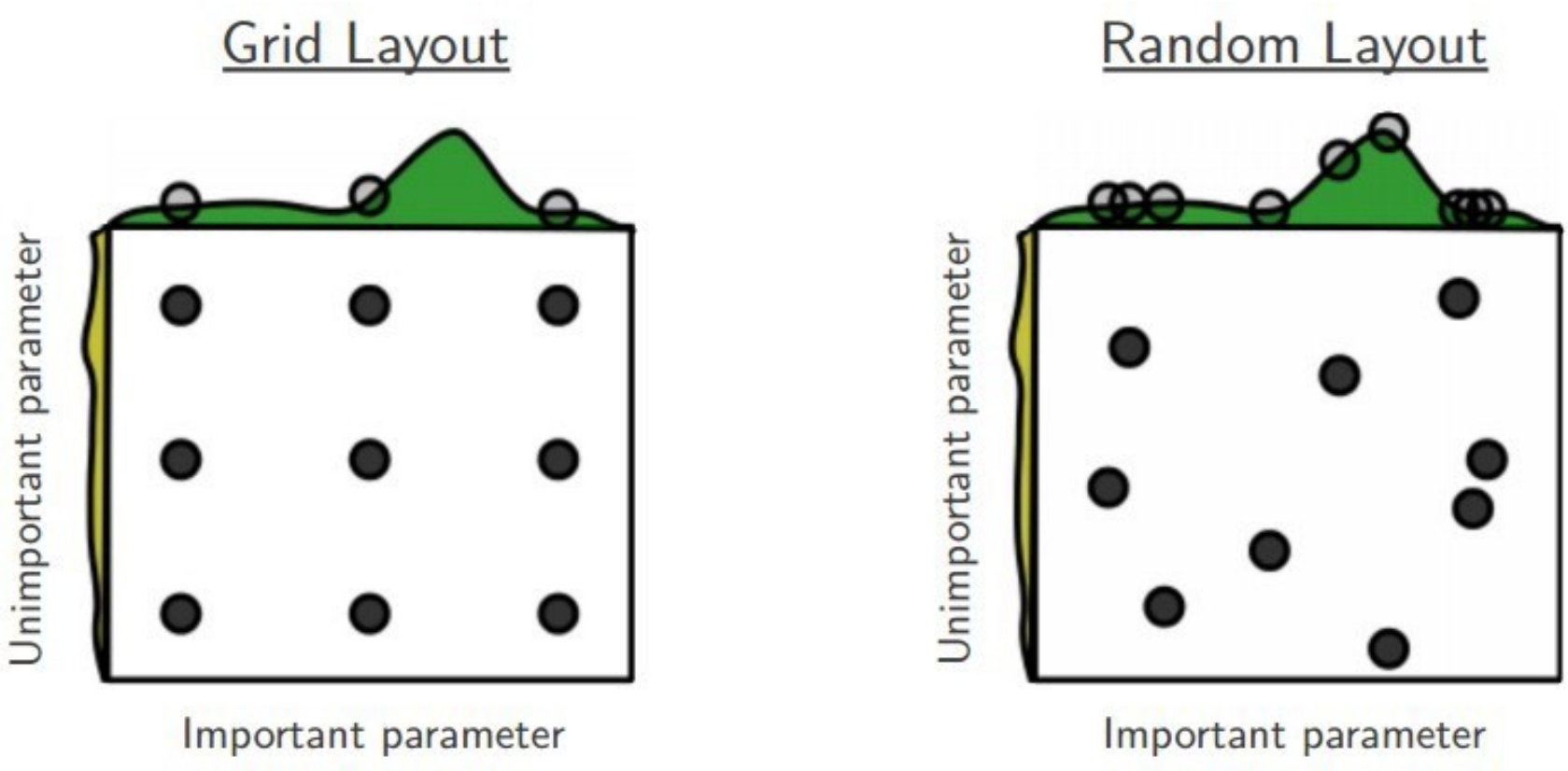
Problems with Grid Search

- The search space can become extremely large as the number of hyperparameters or feature selection/preprocessing steps increases
- One solution is to randomly sample a subset of possible models using a **randomized hyperparameter search**
- Another solution is to use a **surrogate model** in **sequential model-based optimization (SMBO)**

Randomized hyperparameter search

- In randomized search, we treat each hyperparameter as a **distribution** and sample values from it at random.
- Randomized search does not exhaustively search the space of possible hyperparameter combinations, instead taking repeated random samples from distributions of hyperparameters.
- Because of this, a **computational budget**—the number of random samples to be drawn—can be set ahead of time.
- Randomized search can **discretize** parameters, but does not necessarily have to.

Randomized hyperparameter search



On the left we see the difference in sampling of parameters in grid search vs. random search. The green distribution represents a hyperparameter that is important if tuned. The high peak represents the area of the optimal hyperparameter value.

Problems with randomized hyperparameter search

- It can become quite expensive to check many different model configurations.
- Each new sample of hyperparameters must be used to train a new model on training data, make predictions with the validation data, and calculate our performance metric.
- With randomized search, the algorithm has no memory of past successes and failures, wanders randomly through the hyperparameter space
- An alternative, **bayesian hyperparameter search**, uses past records of hyperparameters and performance metrics to decide where to look next.

Bayesian hyperparameter search

- Uses a probabilistic model of hyperparameter values. Maps these hyperparameters probabilistically to a score on a chosen performance metric.
- The approach works as follows:
  1. Build a **surrogate probability model** that predicts how hyperparameter values will impact the model performance based on **prior knowledge**.
  2. Find the hyperparameters that perform best on the surrogate
  3. Train the model with these hyperparameters on the **training data**; evaluate on the **validation set.**
  4. Update the surrogate model incorporating the new information about how hyperparameter values impact the model's performance.
  5. Repeat steps 2–4 until max iterations or time is reached

AutoML

- Why not just take humans out of the loop altogether?
- ML algorithms (KNN, logistic regression, neural networks) can be treated like hyperparameters (choose the algorithm that minimizes validation error)
- Features and preprocessing can also be treated like hyperparameters (choose feature representations that minimize validation error)

Feature engineering and feature learning

Representing complexity

- We rarely have complete information about the **data generating process**
- Recall the Monroe et. al. big data symposium piece, data must necessarily be simplified and represented.
- The **no free lunch theorem** applies to this process as well: "all models are wrong, some are useful"
- In the next part of this lecture, I'll introduce two approaches to representing the complexity of language and vision.
- There are two ways to do this: **engineer features**, or **automatically learn features**

Feature engineering vs. feature learning

- We must find a way to turn language or images into features that can be inputs to a machine learning algorithm.
- **Feature engineering:** Manually transforming, reducing dimensionality, or augmenting input data.
  - Analyst expertise used to construct features based on theory or models of the input (e.g. document term matrices)
- **Feature learning/representation learning:** automatically learn useful representations of the input data (example word2vec, CNNs)
  - Use learning algorithms to organize raw inputs into a more useful feature representation (e.g. Word2Vec)

Feature engineering

Example 1: Text data

- Data generation process for text is unknown
- Complexity of language:
  - Time flies like an arrow, fruit flies like a banana
  - The old man the ship
- Models necessarily fail to capture language useful for specific tasks

Feature engineering vs. feature learning

- We must find a way to turn texts into features that can serve as inputs to a machine learning algorithm
- **Feature engineering:** Manually transforming, reducing dimensionality, or augmenting input data according to expertise or a model of the input (example document term matrices)
- **Feature learning/representation learning:** automatically learning useful representations of the input data (example word2vec)

Document Vectors: The Bag of Words Assumption

Bag of Words: Treat documents as bags of words, ignoring nuance, structure and context of the text.

Assumptions:

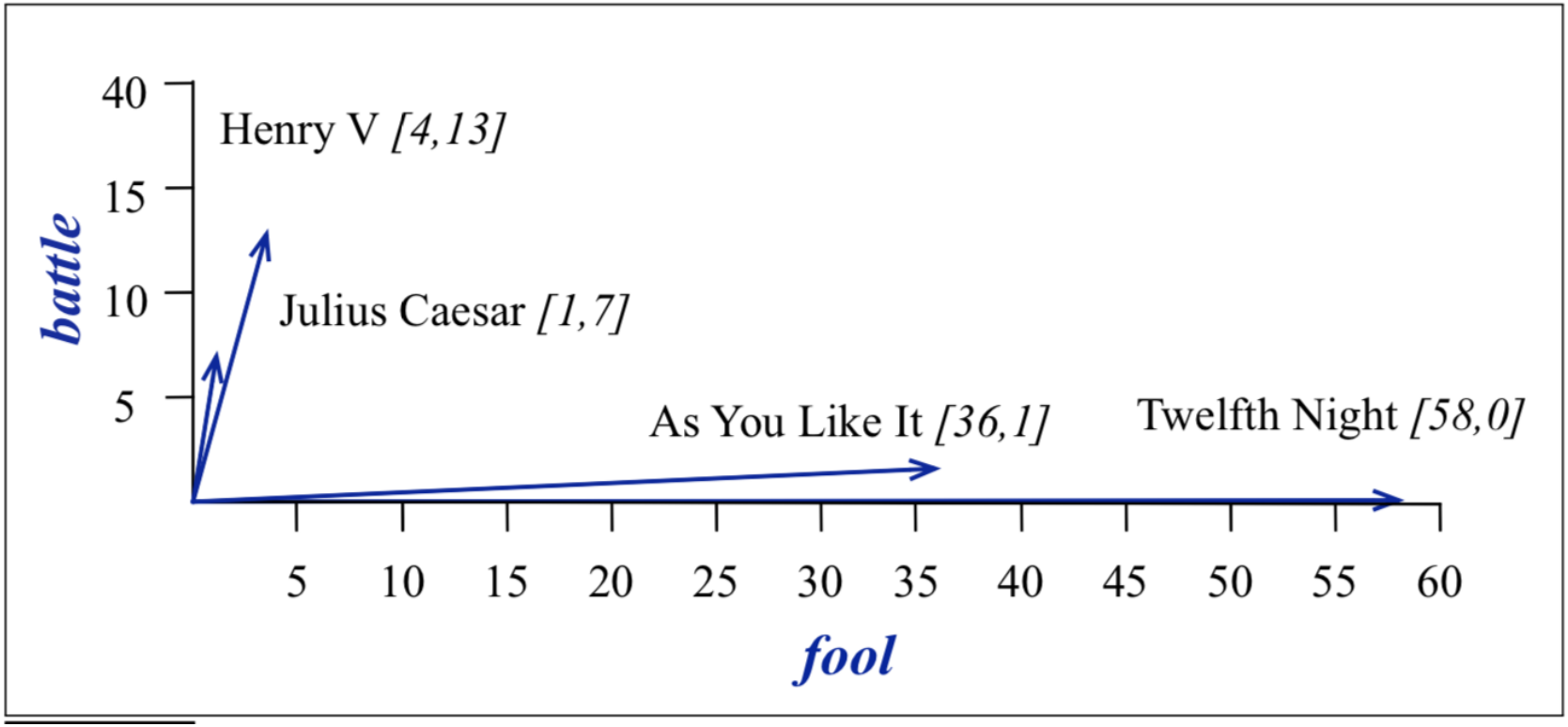
- Word order is not important
- Counts of words in documents contain enough information for the task at hand

Document Term Matrices

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

- The document term matrix for four words in four Shakespeare plays.
- The red boxes show that each document is represented as a column vector of length four.
- Each document is represented by a vector of words
- Vectors are similar for the two comedies
- Both are different than the two historical plays
- Comedies have more fools and wit and fewer battles.

Document Vectors Visualized



Document Term Matrices

- A common way of **preprocessing** text to make it useful for analysis
- Raw texts are difficult to quantitatively analyze without first transforming it
- Document term matrices lower the dimensionality of text data
- Remember that our goal is to characterize the hay stack
- If you want to analyze a straw of hay, these methods are unlikely to work
- But even if you want to closely read texts, characterizing hay stack can be useful

Reducing dimensionality of Document Term Matrices

- Oftentimes document term matrices are enormous, as the vocabulary (all words in a corpus) is large in most corpora.
- This large number of features can get in the way of inference and prediction because not all words in documents carry the same amount of meaning.
- Many words are also used for the same concept.
- Many words are so rare or frequent that they are not useful for the text analysis problem at hand.

What Counts as a "Term" in Document Term Matrices

- Can be anything you choose
- Phrases can be identified manually by an analyst or identified using phrase-detection algorithms
- A term can be a single word (in English, you could split text by white space)
- A term can be a fixed length sequence of characters (character n-grams)
- A term can be a fixed length sequence of words (word n-grams)

bigrams trigrams, 4-grams, n-grams

- Grouping words (or characters) into sequences of *n* units
- In general this is an insufficient model of language because language has long-distance dependencies:

"The computer(s) which I had just put into the machine room on the fifth floor is (are) crashing."

- But we can often get away with N-gram models

n-gram example

- **Input:** "Rubio suspended his campaign after losing Florida"
- **Word Unigrams:** (Rubio, suspended, his, campaign, after, losing, Florida)
- **Word Bigrams:** (* Rubio, Rubio suspended, suspended his, his campaign, campaign after, after losing, losing Florida, Florida STOP)
- **Character 4-grams:**: ('Rubi', 'ubio', 'bio', 'io s', 'o su', ' sus', 'susp', 'uspe', 'spen', 'pend', 'ende', 'nded', 'ded', 'ed h', 'd hi', ' his', 'his', 'is c', 's ca', ' cam', 'camp', 'ampa', 'mpai', 'paig', 'aign', 'ign', 'gn a', 'n af', ' aft', 'afte', 'fter', 'ter', 'er l', 'r lo', ' los', 'losi', 'osin', 'sing', 'ing', 'ng F', 'g Fl', ' Flo', 'Flor', 'lori', 'orid', 'rida')

Stop Words

- Stop Words: low information words
- Example: the, it, if, a, able, at, be, because...
- These words add "noise" to documents (without conveying much information)
- You may need to customize a stop word list based on domain expertise or term frequency

Reduce dimensionality further

- Create equivalence class between words
- Words used to refer to same basic concept (family, families, familial $\rightarrow$ famili)
- Stemming/Lemmatizing algorithms: Many-to-one mapping from words to stem/lemma

Comparing Stemming and Lemmatizing

Stemming algorithm:

- Simplistic algorithms: Porter stemmer, Lancaster stemmer, Snowball stemmer
- Fast approximation
- How it works
  - Chop off end of word

Lemmatizing algorithm:

- More complicated algorithm
- Slow to compute
- How it works
  - Condition on part of speech (noun, verb, etc)
  - Verify result is a word

A potential recipe for preprocessing

1. Remove capitalization, punctuation
2. Segment into words, characters, morphemes
3. Discard Order ("Bag of Words" Assumption)
4. Discard stop words
5. Create Equivalence Class: stem, lemmatize, or synonym
6. Discard less useful features
7. Other reduction, specialization

A Complete Example

"Political power grows out of the barrel of a gun" - Mao

A Complete Example

"Political power grows out of the barrel of a gun" - Mao

Compound Words: With substantive justification, words can be combined or split to improve inference.

A Complete Example

"Political power grows out of the barrel of a gun" - Mao

Compound Words: An analyst may want to combine words into a single term that can be analyzed.

A Complete Example

"Political power grows out of the barrel of a gun" - Mao

Compound Words: An analyst may want to combine words into a single term that can be analyzed.

A Complete Example

[Political], [power], [grows], [out], [of], [the], [barrel of a gun]

Compound Words: An analyst may want to combine words into a single term that can be analyzed.

A Complete Example

[Political], [power], [grows], [out], [of], [the], [barrel of a gun]

Stopword Removal: Removing terms that are not related to what the author is studying from the text.

A Complete Example

[Political], [power], [grows], [out], [of], [the], [barrel of a gun]

Stopword Removal: Removing terms that are not related to what the author is studying from the text.

A Complete Example

[Political], [power], [grows], [out], [barrel of a gun]

Stopword Removal: Removing terms that are not related to what the author is studying from the text.

A Complete Example

[Political], [power], [grows], [out], [barrel of a gun]

Stemming: Takes the ends off conjugated verbs or plural nouns, leaving just the "stem."

A Complete Example

[Political], [power], [grows], [out], [barrel of a gun]

Stemming: Takes the ends off conjugated verbs or plural nouns, leaving just the "stem."

A Complete Example

[Polit], [power], [grow], [out], [barrel of a gun]

Stemming: Takes the ends off conjugated verbs or plural nouns, leaving just the "stem."

A Complete Example

Finally, we can turn tokens and documents into a "document-term matrix."

Imagine we have a second document in addition to the Mao quote, "the political science students study politics", which tokenizes as follows.

Document #1: [polit], [power], [grow], [out], [barrel of a gun]
Document #2: [polit], [scien], [student], [studi], [polit]

Output: Document Term Matrix

	Doc 1	Doc 2
power	1	0
grow	1	0
out	1	0
barrel of a gun	1	0
student	0	1
studi	0	1
polit	1	2
scien	0	1

But raw frequency is a bad representation

- Frequency is clearly useful; if sugar appears a lot near apricot, that's useful information.
- But overly frequent words like the, it, or they are not very informative about the context
- Some terms carry more information about contents
- Need a function that resolves this frequency paradox!

Term Frequency (tf)

- A term is more important if it occurs more frequently in a document
- tf: term frequency. frequency count (usually log-transformed):

$$\text{tf}_{t,d} = \begin{cases} 1 + log(\,\text{count}(t,d)) & \text{if count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Inverse Document Frequency (idf)

- A term is more discriminative if it occurs only in fewer documents. Why this is true?

$$idf_i = log\left(\frac{N}{df_i}\right)$$

- N is the total number of documents in the collection
- df_i is the number of documents in the collection that contain the word i
- Note that IDF is document independent while TF is document dependent!
- Words like “the” or “and” have a very low idf

Tf-idf Weighting

- tf-idf value for word t in document d:

$$w_{t,d} = tf_{t,d} \times idf_t$$

- Term is common in doc $\rightarrow$ high tf $\rightarrow$ high weight
- Term is rare in collection $\rightarrow$ high idf $\rightarrow$ high weight
- Imagine what kind of terms would have high weights?

Tf-idf Weighted Document Term Matrix

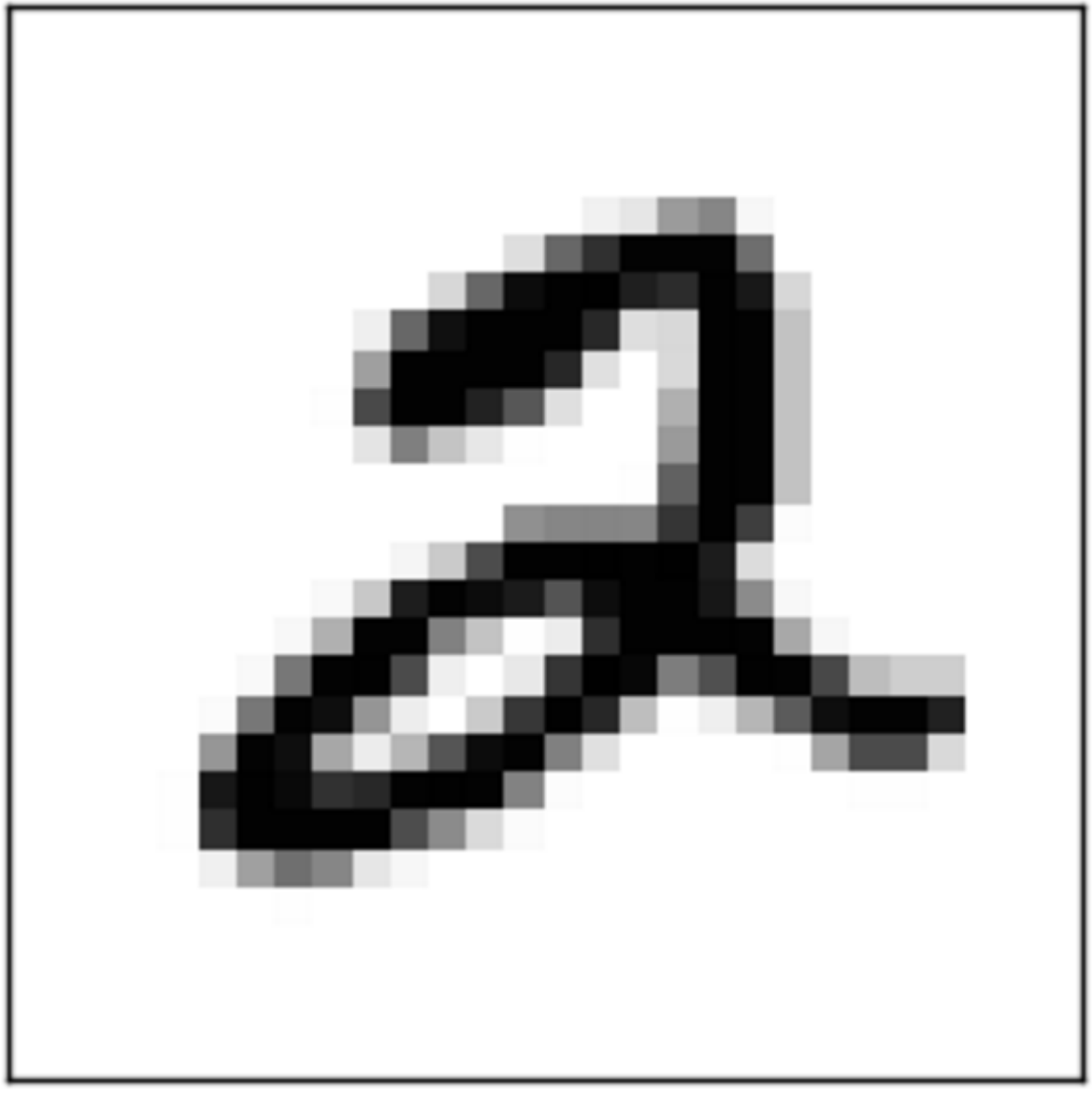
	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

- A tf-idf weighted document term matrix for four words in four Shakespeare plays, using the counts from the earlier document term matrix
- The idf weighting has eliminated the importance of the ubiquitous word good and vastly reduced the impact of the almost-ubiquitous word fool.

Computer vision example: digit recognition

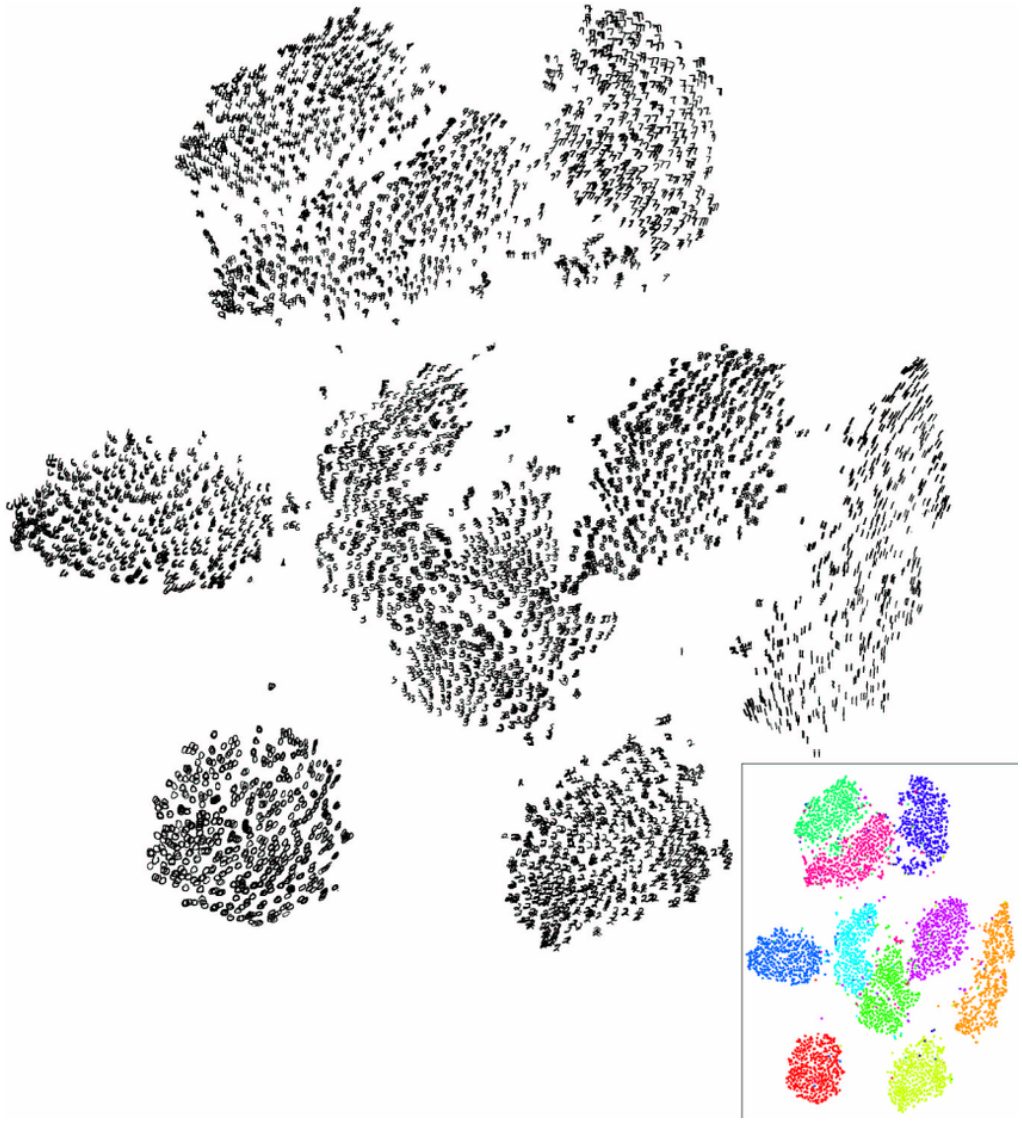
Handwritten digits.

Computer vision example: digit recognition



Can be transformed into a vector of grayscale pixel intensities.

Computer vision example: digit recognition

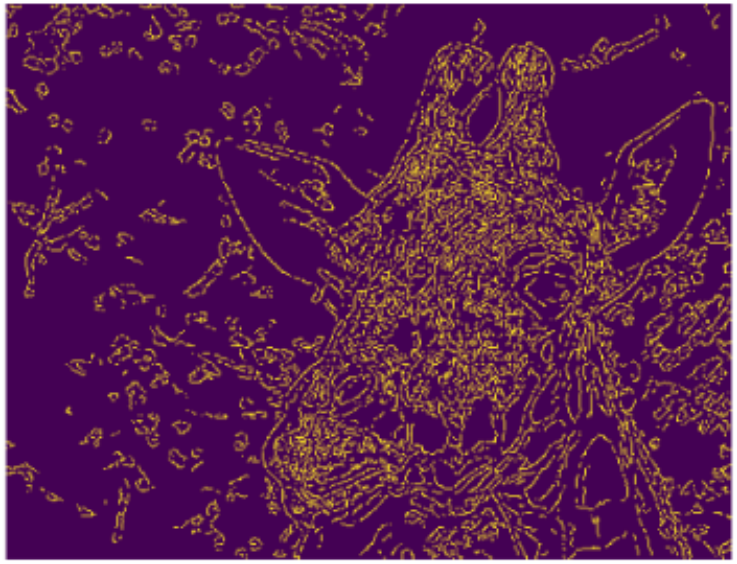


Grayscale values are a good feature representation, each digit is neatly clustered ttogether (projection onto 2 principal components)

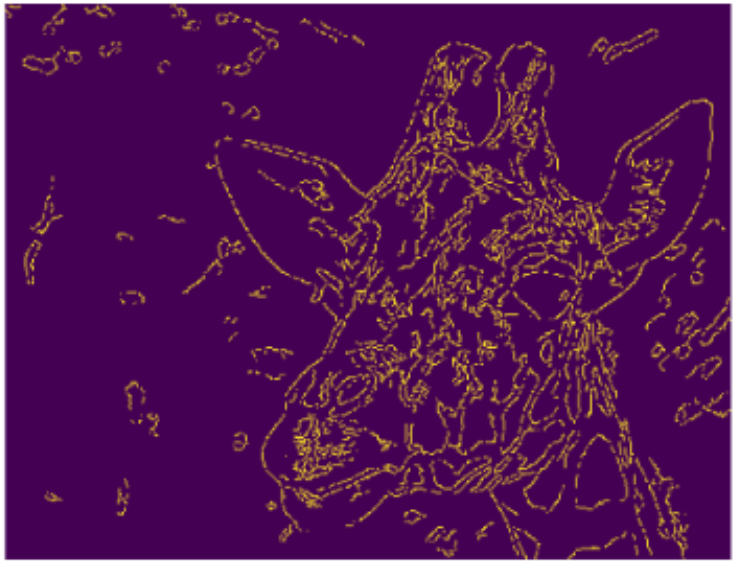
Computer vision example: other possible features

- Edges
- Corners
- Faces
- RGB color

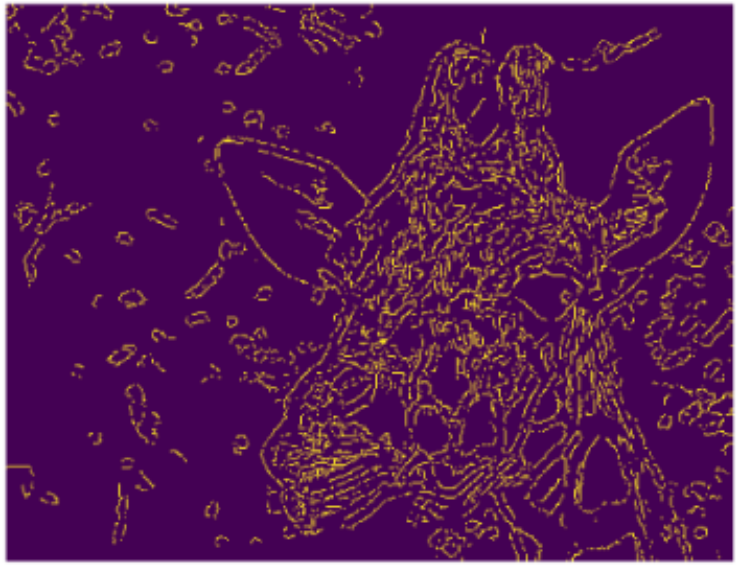
Computer vision example: edges



ksize = (5, 5)
lower, upper



ksize = (5, 5)
lower, upper + 100



ksize = (9, 9)

ksize = (9, 9)

Computer vision example: corners

Original Image

Shi-Tomasi Corner Detection

Feature learning

Text example: Word vectors/Word2vec

- Synonyms like oculist and eye-doctor tend to occur in the same environment (e.g., near words like eye or examined) with the amount of meaning difference between two words "corresponding roughly to the amount of difference in their environments" (Harris, 1954, 157).
- "You shall know a word by the company it keeps" (Rupert Firth)

Text example: Word vectors/Word2vec

```
... lemon,  a [tablespoon of apricot jam,      a] pinch ...
               c1         c2    t     c3       c4
```

Text example: Word2vec

- Instead of **counting** how often each word *w* occurs near "apricot"
- Train a classifier on a binary prediction task:
  - Is w likely to show up near "apricot"?
- We don't actually care about this task, but we'll take the learned classifier weights as the word embeddings
- Dense vector representation of words

Text example: Word2vec

- A word s near apricot acts as "correct answer" to the question
  - "Is word w likely to show up near apricot?"
- No need for hand-labeled supervision
- The idea comes from neural language modeling
  - Bengio et al. (2003)
  - Collobert et al. (2011)

Skip-gram algorithm

1. Treat the target word and a neighboring context word as positive examples.
2. Randomly sample other words in the lexicon to get negative samples
3. Use logistic regression to train a classifier to distinguish those two cases
4. Use the weights as the embeddings

Setup

```
... lemon,  a [tablespoon of apricot jam,      a] pinch ...
               c1         c2    t     c3       c4
```

- Let's represent words as vectors of some length (say 300), randomly initialized.
- So we start with $300 \times V$ random parameters
- Over the entire training set, we'd like to adjust those word vectors such that we
  - Maximize the similarity of the target word, context word pairs (t,c) drawn from the positive data
  - Minimize the similarity of the (t,c) pairs drawn from the negative data.

Training Data

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	preserves
apricot	or

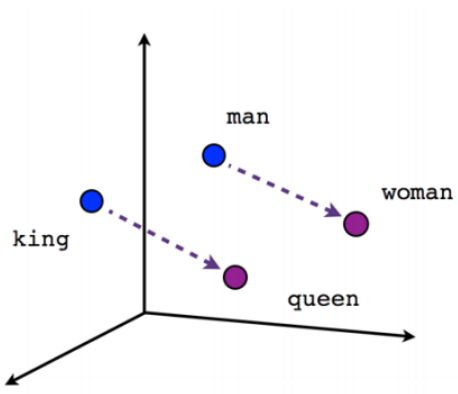
negative examples -

t	c	t	c
apricot	aardvark	apricot	twelve
apricot	puddle	apricot	hello
apricot	where	apricot	dear
apricot	coaxial	apricot	forever

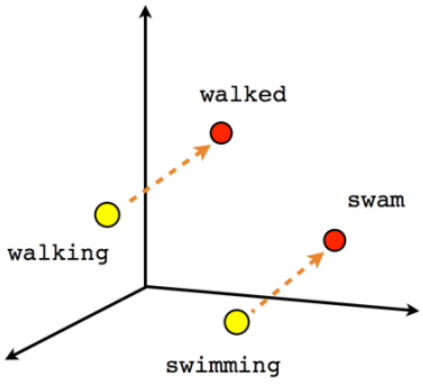
How to learn word2vec (skip-gram) embeddings

- Start with V random 300-dimensional vectors as initial embeddings
- Use logistic regression, the second most basic classifier used in machine learning after naive bayes
- Take a corpus and take pairs of words that co-occur as positive examples
- Take pairs of words that don't co-occur as negative examples
- Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
- Throw away the classifier and keep the embeddings.

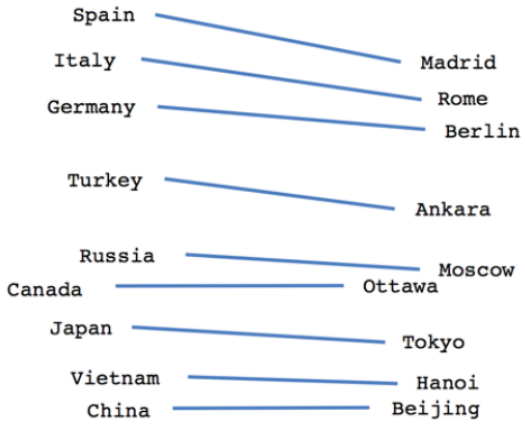
Vectors and Semantic Relationships



Male-Female



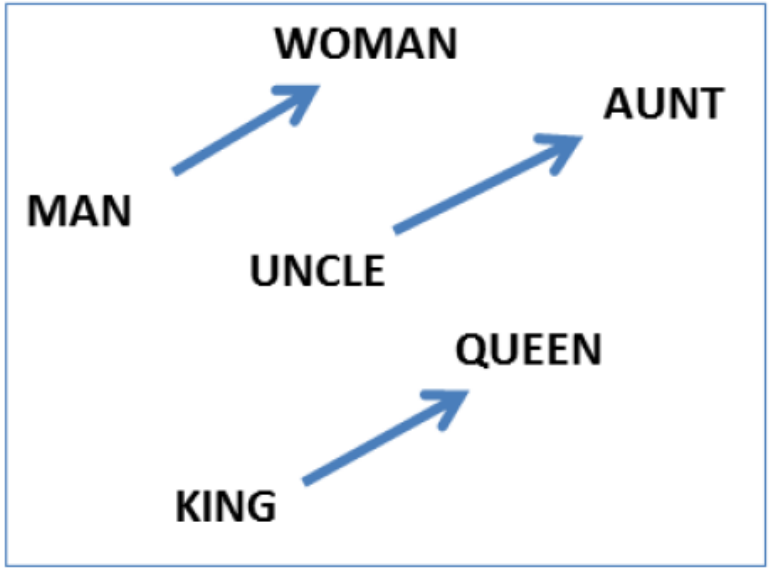
Verb tense

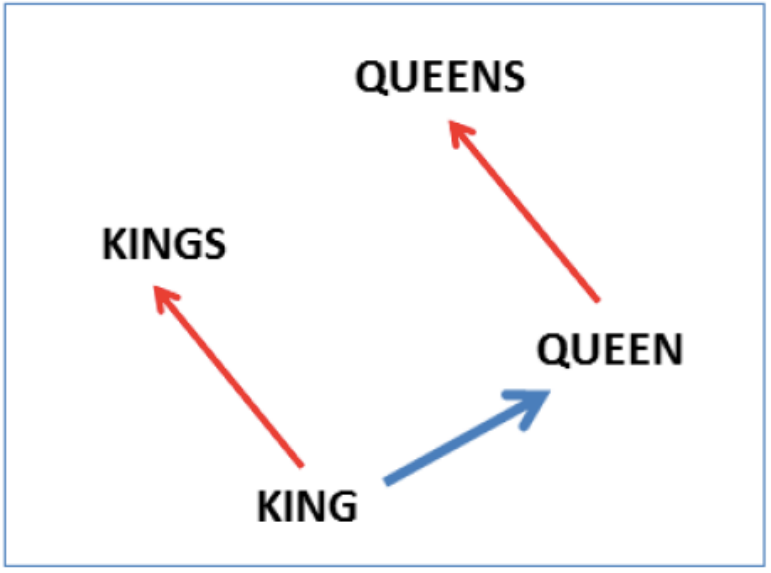


Country-Capital

- Vectors capture some general semantic information about words and their relationships to one another.

Analogy: Embeddings capture relational meaning!



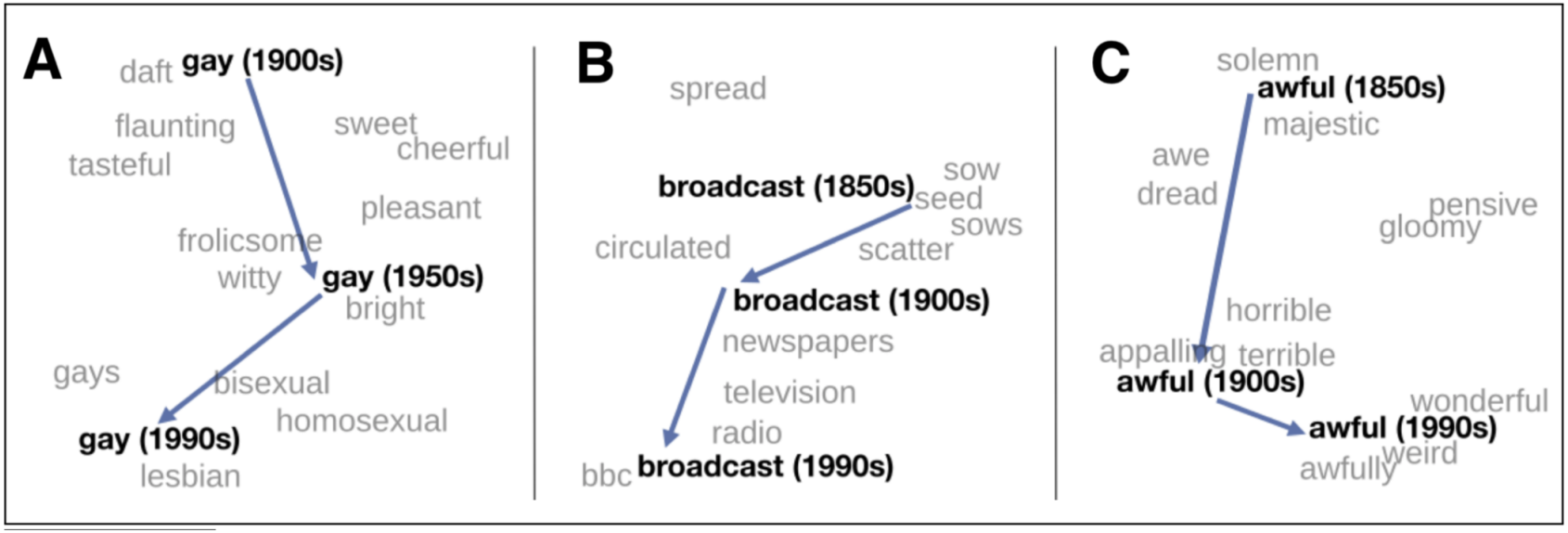


- vector('king') - vector('man') + vector('woman') = vector('queen')
- vector('Paris') - vector('France') + vector('Italy') = vector('Rome')

Embeddings reflect cultural bias (Bolukbasi et. al 2016)

- Ask "Paris : France :: Tokyo : x"
  - x = Japan
- Ask "father : doctor :: mother : x"
  - x = nurse
- Ask "man : computer programmer :: woman : x"
  - x = homemaker

Visualizing historical changes in word meaning



Computer vision example: convolutional neural networks (CNNs)

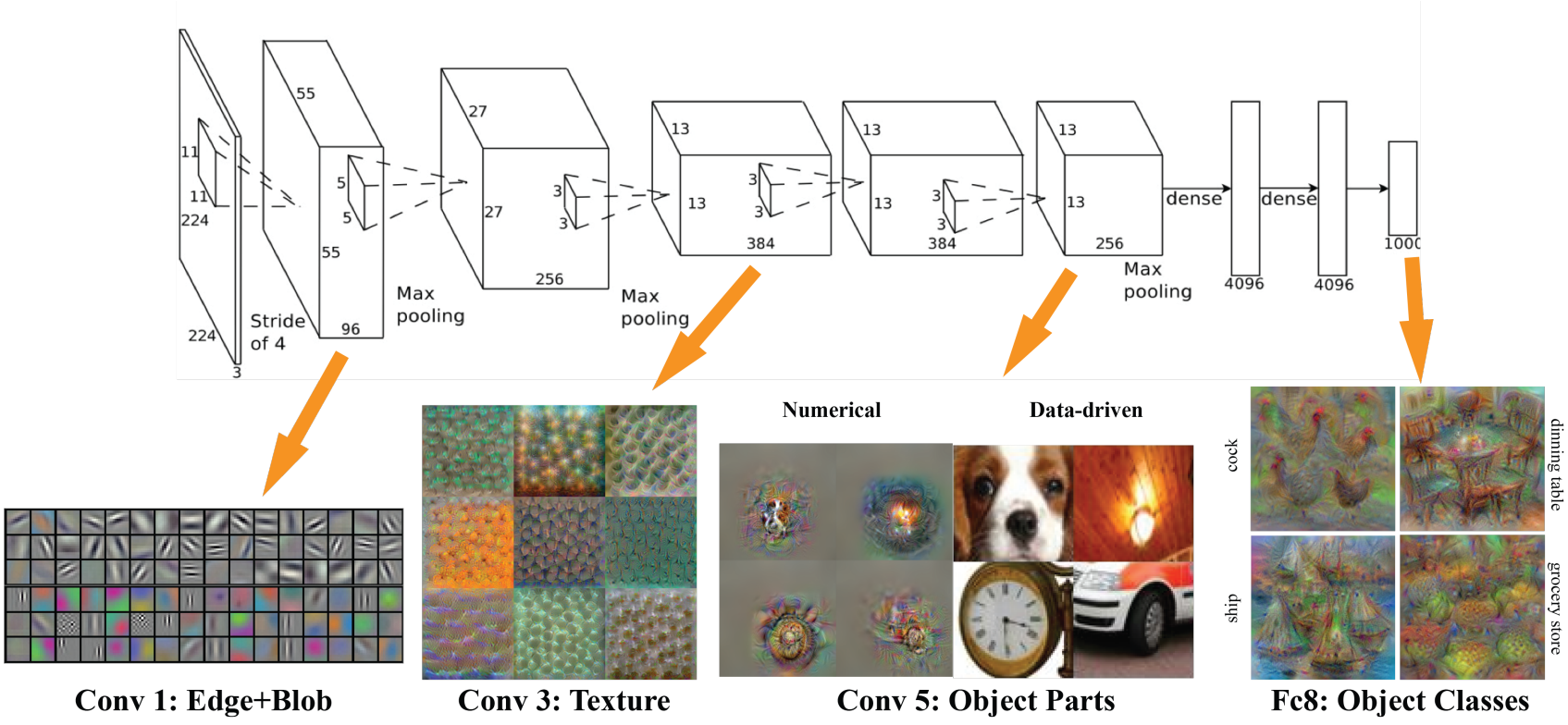
$F[x, y]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$G[x, y]$

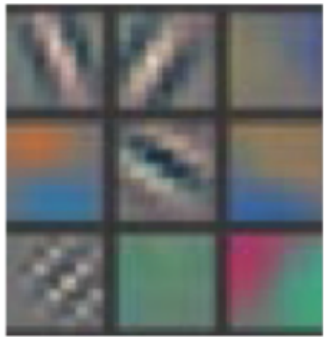
	0								

Computer vision example: convolutional neural networks (CNNs)



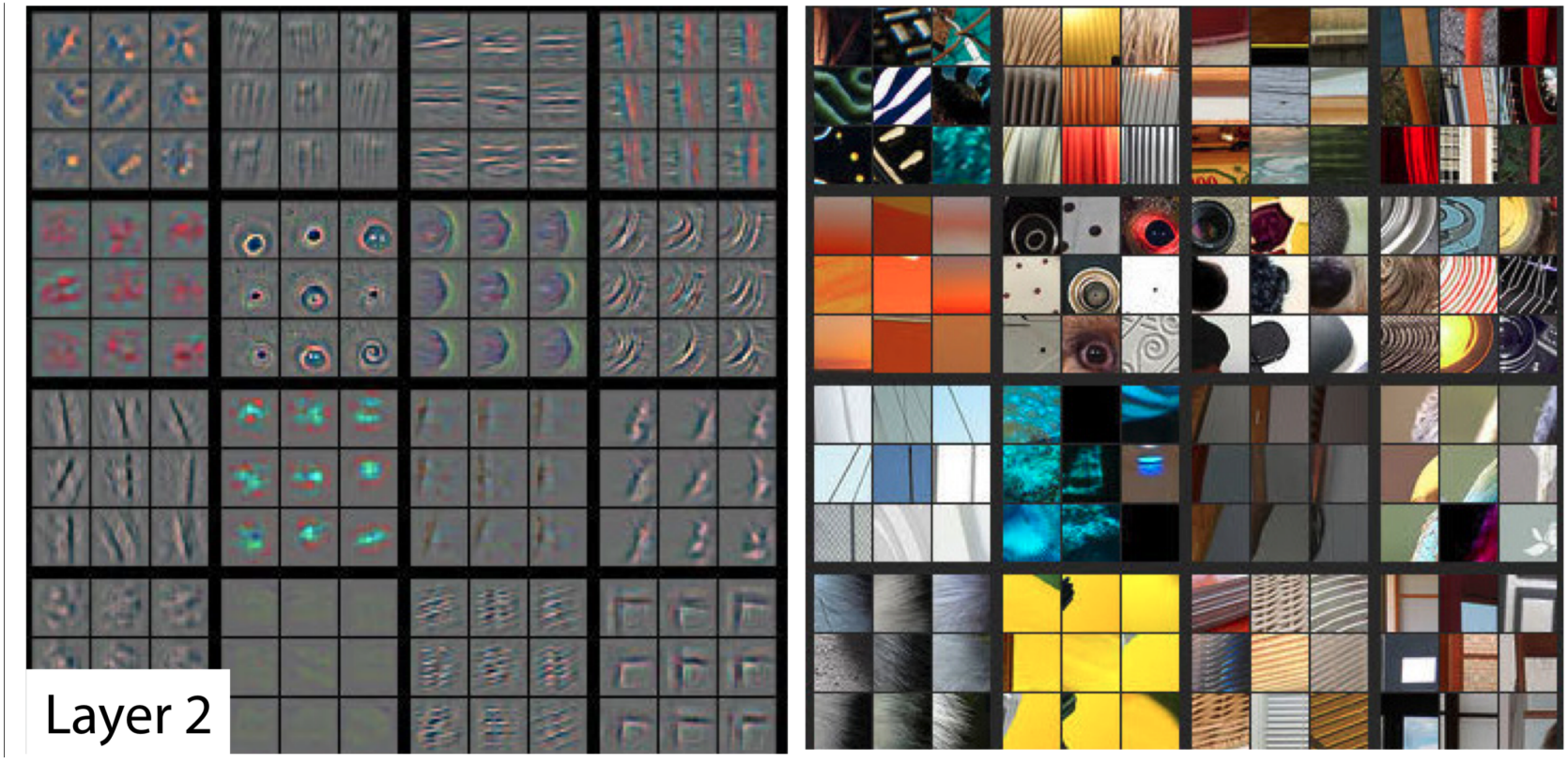
Source: AlexNet / VGG-F network visualized by mNeuron.

Visualizing learned features: Layer 1



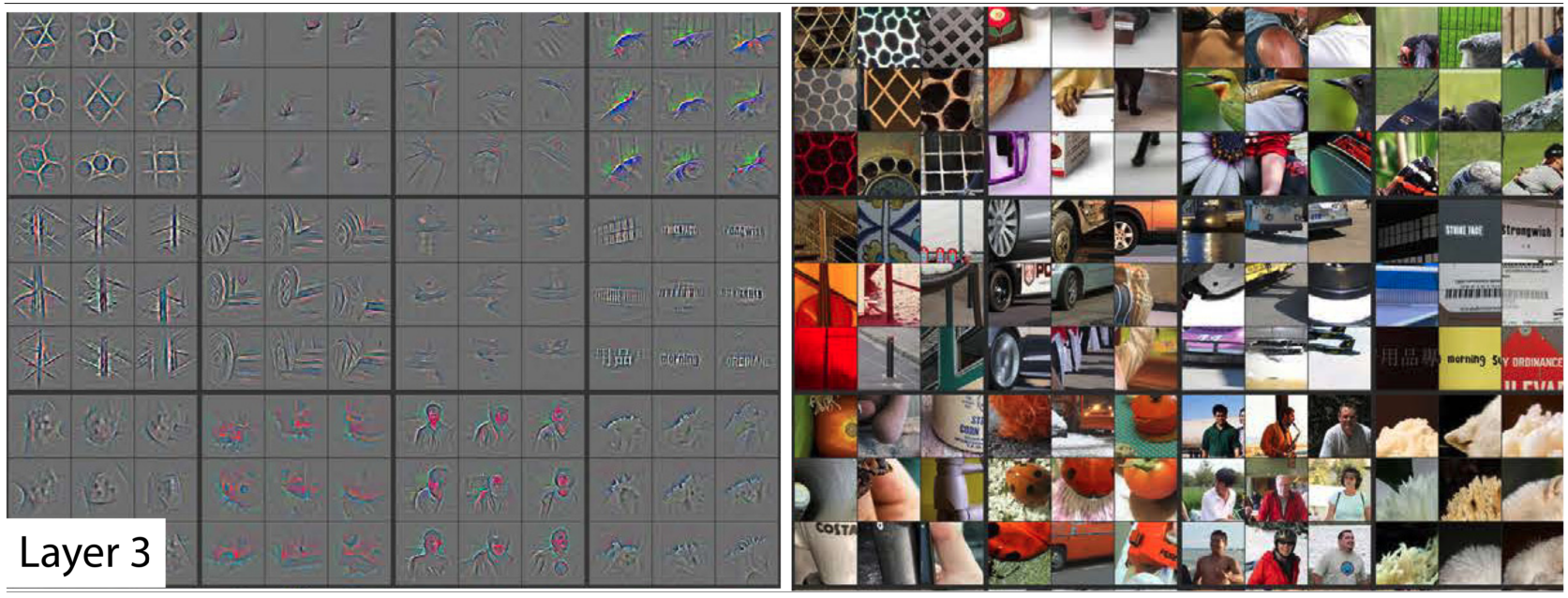
Layer 1

Visualizing learned features: Layer 2



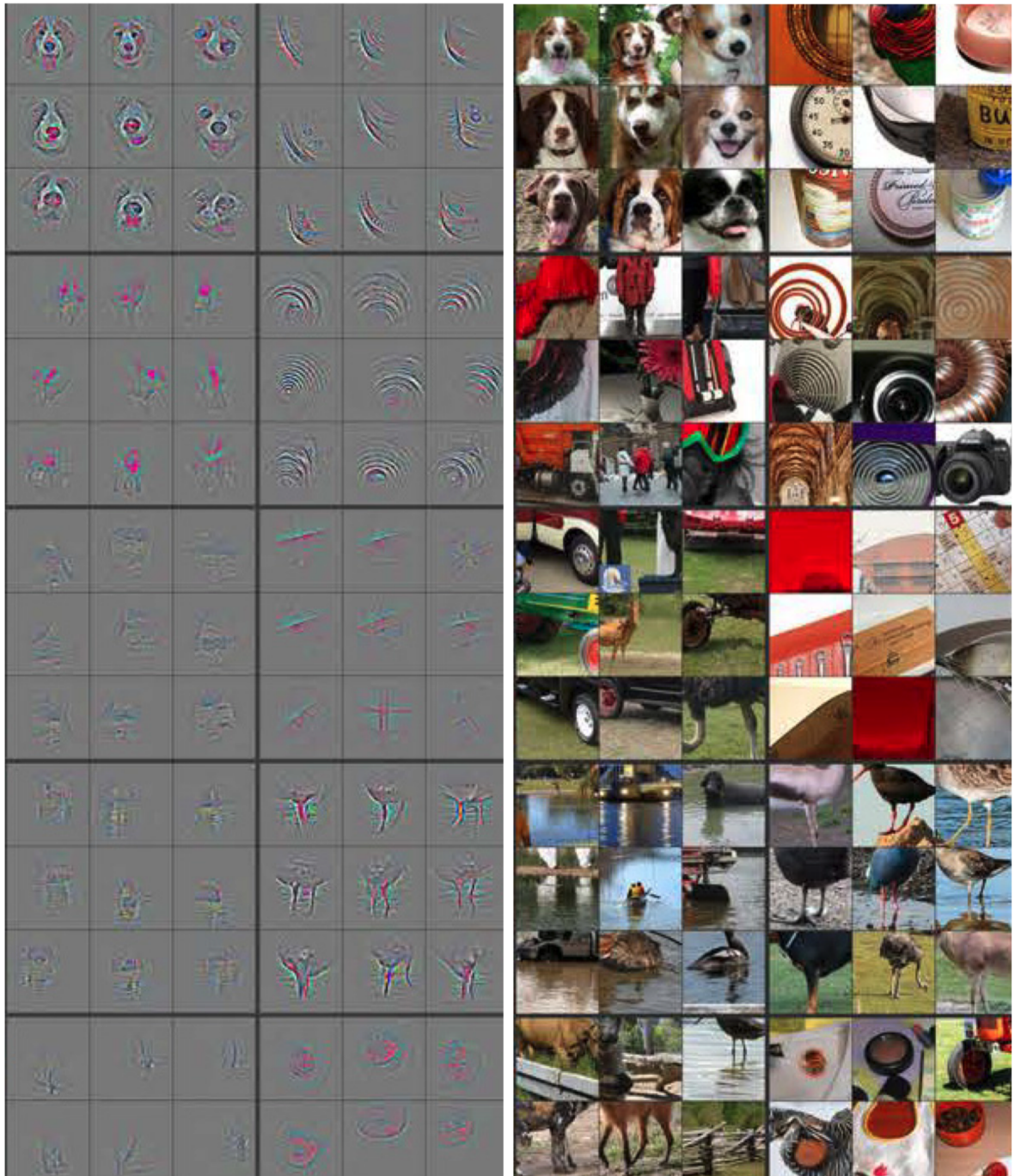
Source: Visualizing and Understanding Convolutional Networks, Matthew D Zeiler et al.

Visualizing learned features: Layer 3

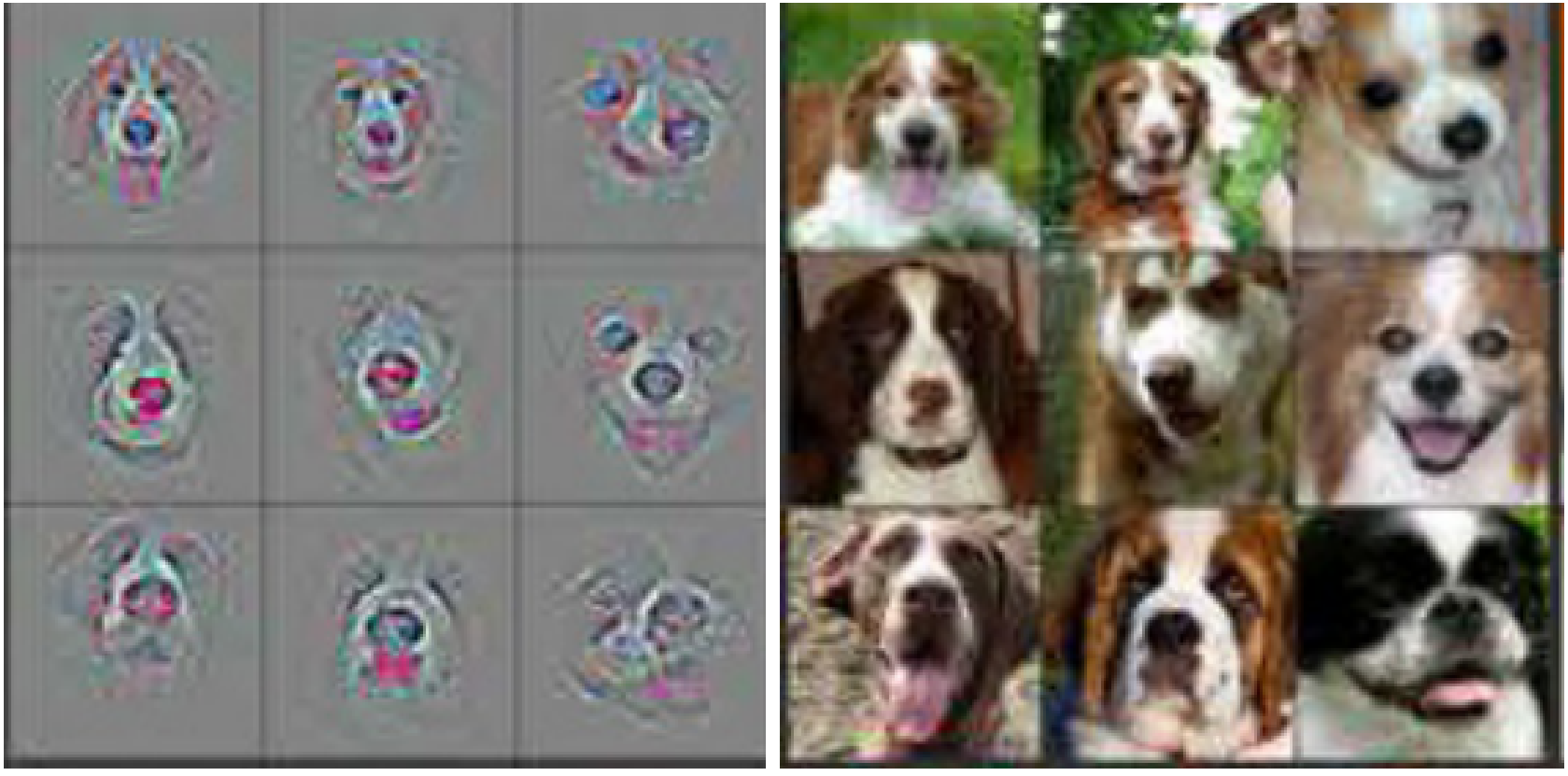


Source: Visualizing and Understanding Convolutional Networks, Matthew D Zeiler et al.

Visualizing learned features: Layer 4



Visualizing learned features: Layer 4 (Dogs)



Source: Visualizing and Understanding Convolutional Networks, Matthew D Zeiler et al.