Using unsupervised learning to improve supervised learning

Kevyn Collins-Thompson

Associate Professor of Information and Computer Science
School of Information, University of Michigan

How can unsupervised learning help supervised learning?

- We've seen unsupervised methods like:
  - Density estimation
  - Clustering
  - Dimensionality reduction
  - Manifold learning
  - And more..
- At the same time, we've seen challenges faced by supervised learning
  - e.g. curse of dimensionality
- Let's explore how problems in supervised learning could benefits from the methods in this course.

Example: How can clustering help classification?

- Suppose your data looks like this (in 1D):

AA A AA A A BBB B B B BB BB BB AA AA A A AAA

- Run a clustering algorithm on each class.
- You find out there are two different variants of A.
- This suggests learning two separate classifiers for A1 and A2 in those two different regions of feature space.
- You can drop the cluster distinction for the final output.

Common problems in supervised learning
Challenge #1: Not enough labeled data

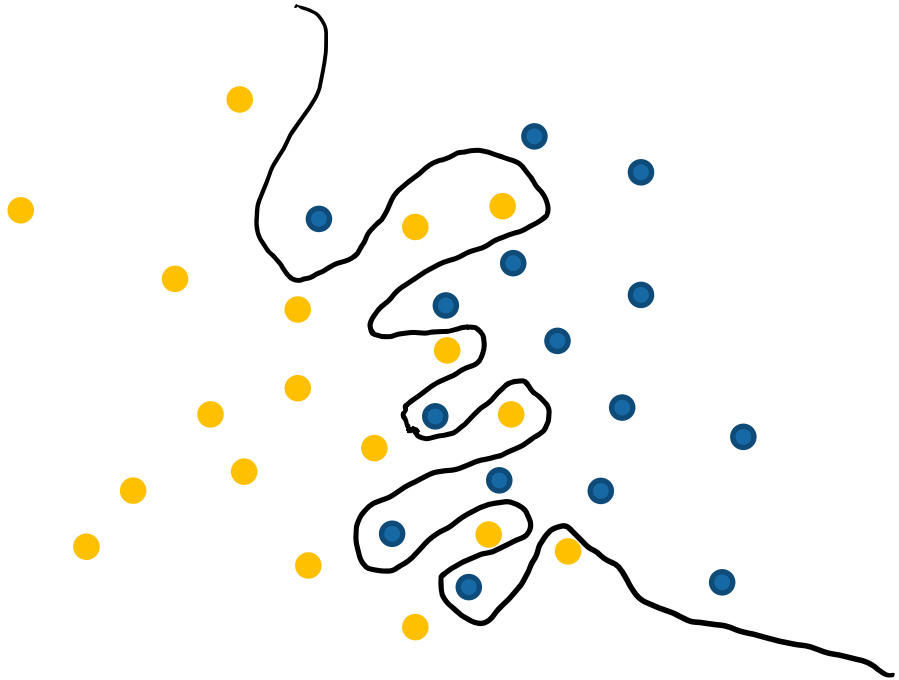
- Suppose we have one labeled example but many unlabeled examples.
- How could we label related examples automatically?
- Do clustering to find the related examples (no labels required).
- Related → same cluster.
- Apply the labels from labeled examples to the unlabeled ones in the same cluster.
- Can be viewed as a form of semi-supervised learning.
- We cover label propagation in a separate module.

Adapted from source: https://www.oreilly.com/library/view/hands-on-unsupervised-learning/9781492035633/ch01.html

Supervised learning challenge #2: Overfitting

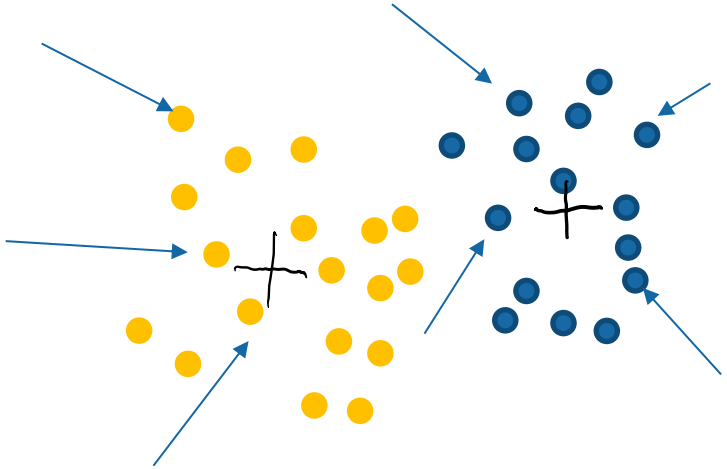
- Regularization: selecting or tuning the preferred level of model complexity so that the model is better at generalizing (predicting accurately for unseen examples)
- Use unsupervised learning as a regularizer.
- Unsupervised pre-training:
  - Compute a transformed representation of the input, train on that.
  - e.g. Do dimensionality reduction, then classification.

Toy example of unsupervised pre-training



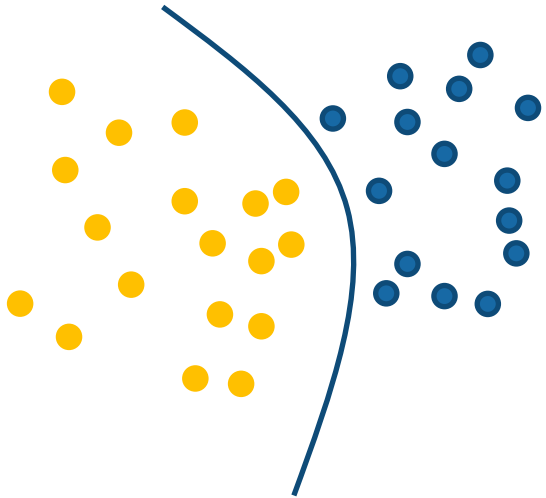
Toy example of unsupervised pre-training

Perform k-means clustering with two clusters.
Shrink the original points toward the cluster centroids

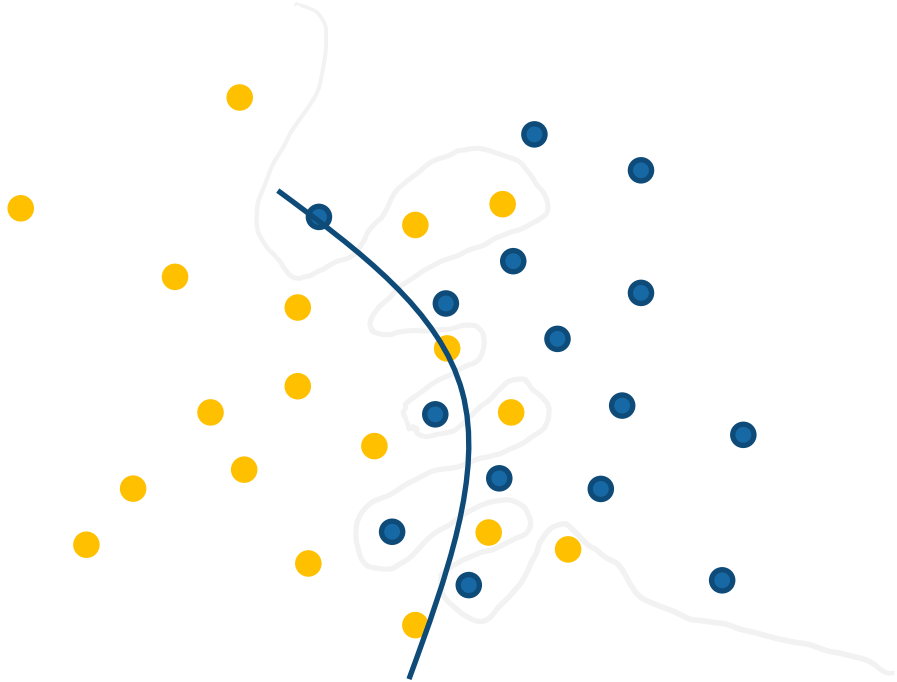


Toy example of unsupervised pre-training

Train on the new representation, which results in a lower-variance decision boundary between the classes.



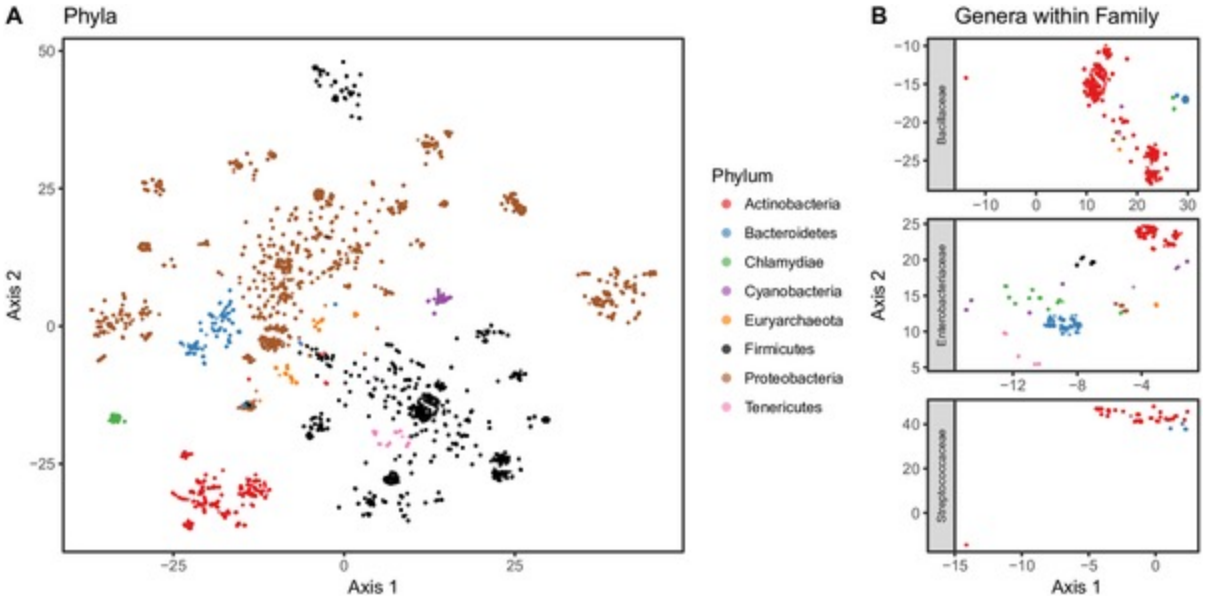
Toy example of unsupervised pre-training



Supervised learning challenge #3: curse of dimensionality

- Processing dataset with billions of ROWS is not too bad: can be parallelized
- Billions of columns (features) is much harder
- Searching for nearest neighbors or optimizing a loss function in very high dimensions is time and compute expensive
- Solution: dimensionality reduction

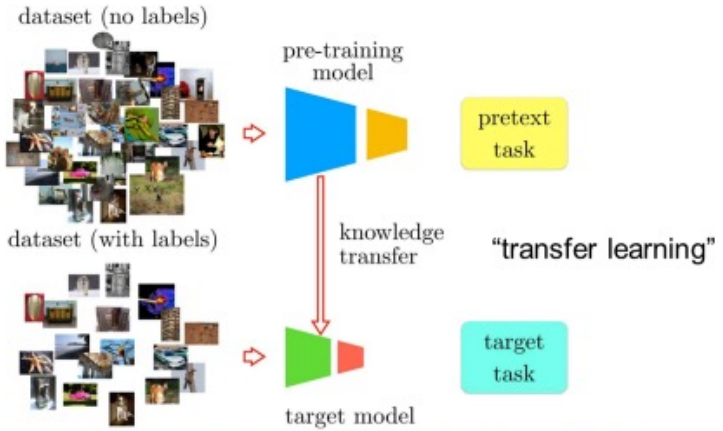
Long RNA sequences → 2d embedding



Source: S. Woloszynek, Z. Zhao, J. Chen, G. L. Rosen. 16S rRNA sequence embeddings: Meaningful numeric feature representations of nucleotide sequences that are convenient for downstream analyses. https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1006721

Supervised learning challenge #4: feature engineering

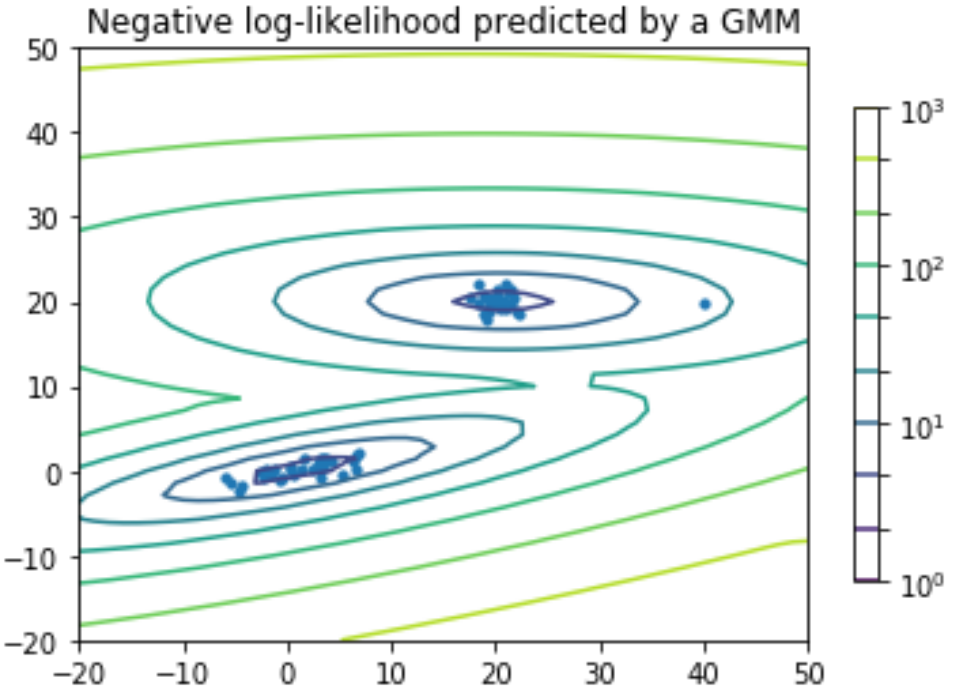
- Representation learning from unsupervised data
- Pretext tasks (self-supervised learning) are used to learn a latent representation.
- This pre-trained model is frozen, partly ‘unthawed’ to be tuned for a different but related target task.
- Embeddings are a good example of this.



Supervised learning challenge #5: outliers

- Rare examples can distort results (especially for generative models that are focused on modeling the classes, not just the decision boundary between them)
- Do outlier detection (e.g. UMAP example)
  - Apply one solution for outliers.
  - Apply a second solution for ‘normal’ instances.

Detecting outliers



(See Kernel Density outlier detection and Gaussian Mixture Model examples from Week 1 notebook)

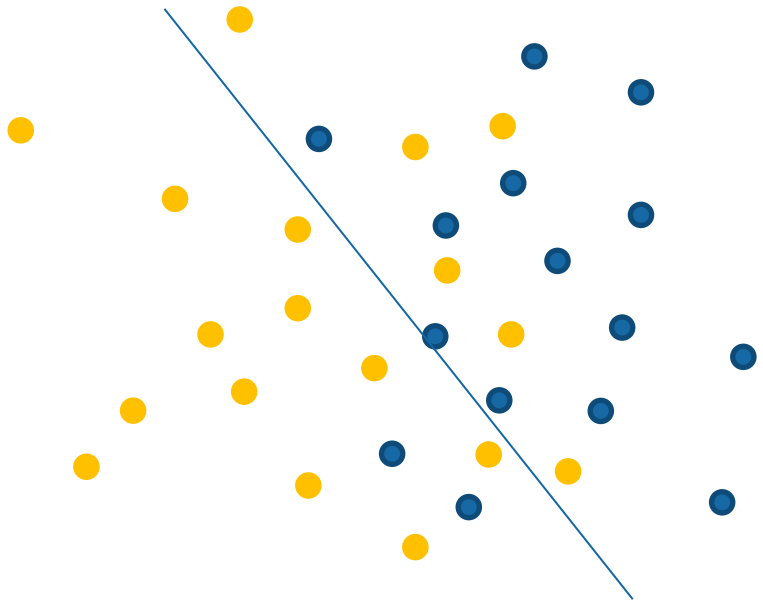
Supervised learning challenge #6: data drift

- Incoming new test data is drawn from a different underlying distribution than the model was trained and tuned on.
- **Idea**: Do density estimation to get probability distributions for assessing how different the current data is from the training set data.
- If too high a fraction of incoming new instances are "low probability events" relative to the training data, trigger a retraining.
- Also called covariate shift or feature drift in the case where previously unseen or rare features become common or vice versa.
- A very significant machine learning pipeline issue!

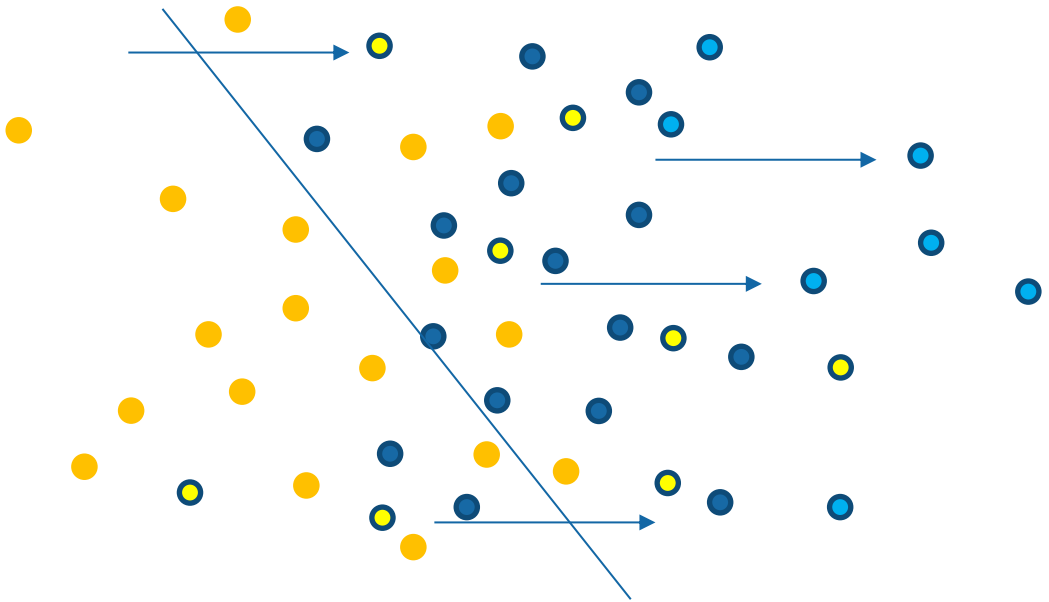
Data drift and generalization

- If we assume the training and test sets were drawn from the same underlying distribution, this means each pair of observations (x, y) is drawn independently from the same joint distribution p(X,Y).
- Intuitively, this means that the training set S is a reasonable protocol through which we can get partial information about the underlying true function f (e.g. a classifier) that we're trying to learn.
- If the above assumption does not hold, the empirical risk on the training set is no longer a robust estimation of the expected empirical risk on new unseen data. In other words, the test error is no longer a good estimation of future errors.
- See the topic 'Empirical Risk Minimization' for deeper theoretical justification of the above.

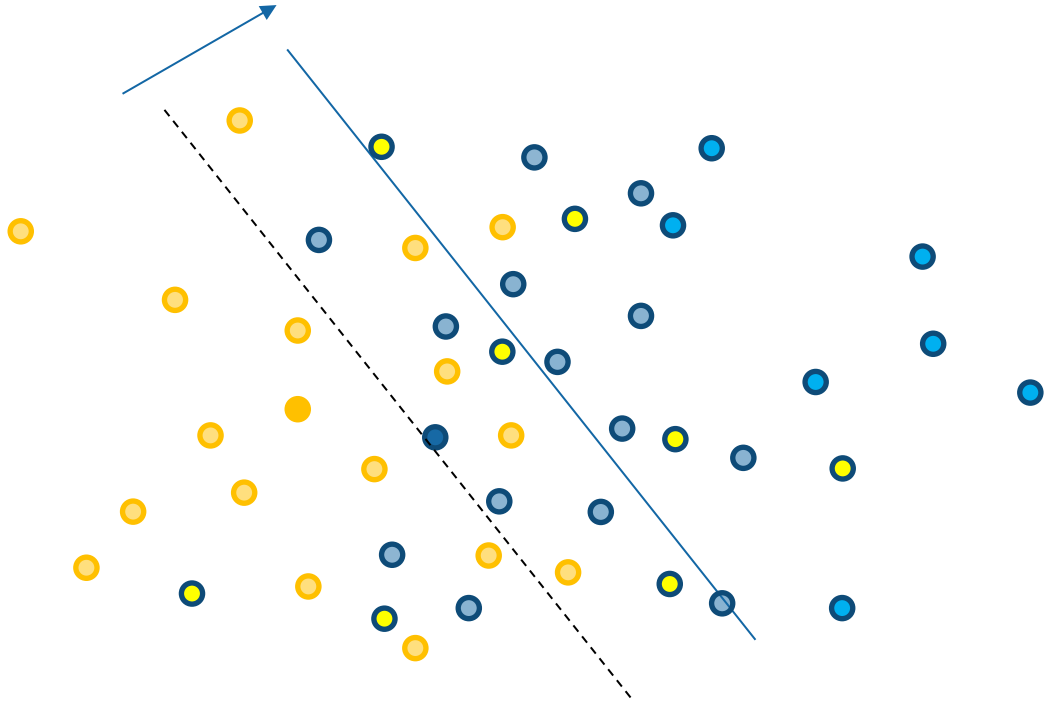
Covariate shift



Covariate shift: change in ‘x’ feature distribution for new points



Covariate shift: classifier must be retrained on up-to-date examples



Why does data drift / covariate shift happen?

- Changing environment
  - Time-dependent tasks!
  - Business forecasting during evolving market trends.
  - Monthly / seasonality effects.
- Selection bias
  - Training data was sampled from an unrepresentative group.
- Changes in data quality
  - Previously reliable feature is lost or becomes noisy/unreliable.
- Concept drift: feature distribution is unchanged
  BUT the relationship with the label changes…
  - e.g. new labels come from a different set of judges or based on a different criterion.

7. Supervised learning challenge: missing data

- Data imputation methods.
- Basic vs iterative.
- Single-column vs multi-column.
- Can itself use supervised learning to predict missing features!

Summary of how unsupervised learning can help supervised learning

1. Not enough labels: semi-supervised methods like label propagation.
2. Overfitting: smooth the data by transforming it (e.g. dim reduction)
3. Curse of dimensionality: dimensionality reduction.
4. Feature engineering: self-supervised methods.
5. Outliers: detect them with UMAP, density estimation.
6. Data/covariate shift: density estimation to detect differences in train/test distribution.
7. Missing features: data imputation

Kevyn Collins-Thompson

kevynct@umich.edu

School of Information

Semi-Supervised Learning

Kevyn Collins-Thompson

Associate Professor of Information and Computer Science
School of Information, University of Michigan

Semi-supervised classification

- Traditional classifiers use only labeled data to learn from.
- This labeled data is often **expensive** and/or difficult to obtain since it requires experienced human annotators.
- Unlabeled data is **much easier** to collect.
- How could we use unlabeled data to build better classifiers, and get higher accuracy with less effort/cost?

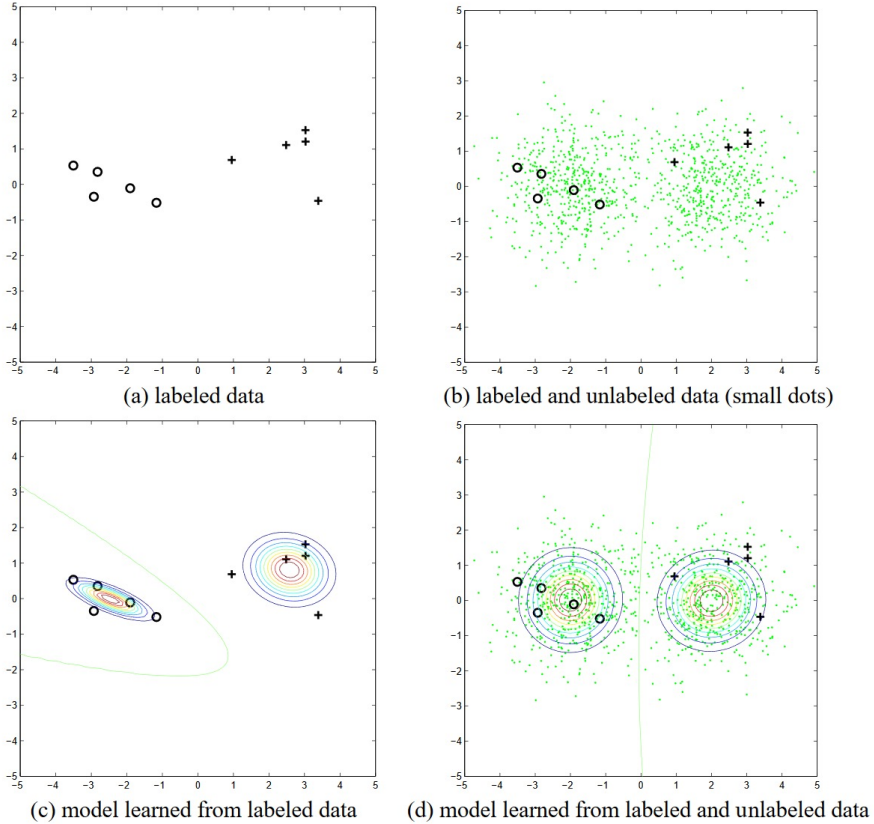
Does using unlabeled data really help?

- Yes, but there is no ‘free lunch’.
- It’s not a good fit for some problems, in which case the classifier could get worse.
- An open question on how to tell automatically if it’s a good fit.
- You avoid labeling effort, but it’s replaced by effort to design effective models, features, and similarity functions for semi-supervised learning.
- Three scenarios where unlabeled data can help.

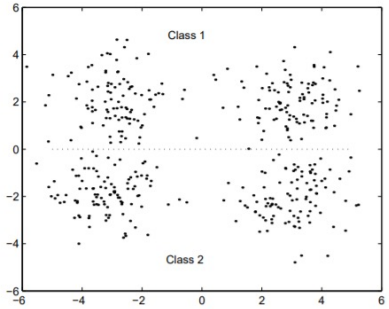
Scenario 1. Generative models

- $p(x,y) = p(y)\, p(x|y)$
- With lots of unlabeled data, the mixture components can be identified.
- Then we need only one labeled example per mixture component to fully determine the mixture distribution.
- We can apply EM to perform soft clustering.

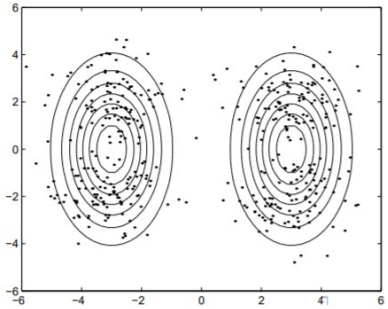
- Assumption: each class has a Gaussian distribution.
- If this reflects the truth, unlabeled data can help estimate the mixture model distribution.
- If it's a bad assumption, we will learn the wrong model which will have worse classification accuracy.



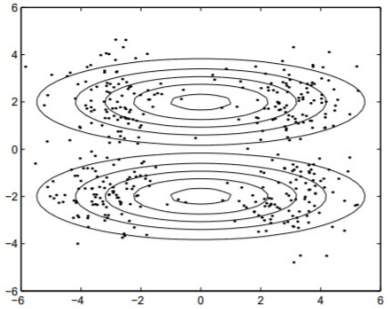
Source: X. Zhu, Semi-supervised literature survey (2008)



(a) Horizontal class separation



(b) High probability



(c) Low probability

(a) Clearly not generated from just two Gaussians (note Class 1, Class 2 labels and boundary)
(b) If we assume single Gaussians, this model has high probability (but 50% accuracy)
(c) Has lower probability but is much more accurate.

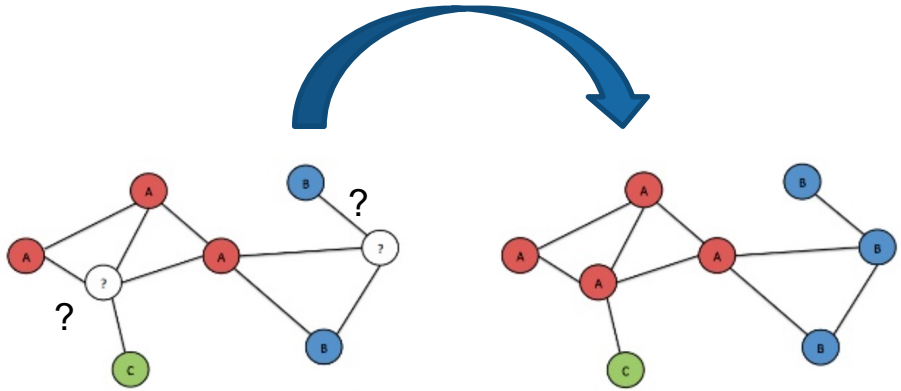
Source: X. Zhu, Semi-supervised literature survey (2008)

Scenario 2. Self-training

- First train a classifier with a small amount of labeled data.
- Then classify the unlabeled data.
- Add the most confident predictions on the unlabeled data to the training set.
- Re-train the classifier with the expanded training set and repeat.

Scenario 3. Graph-based methods

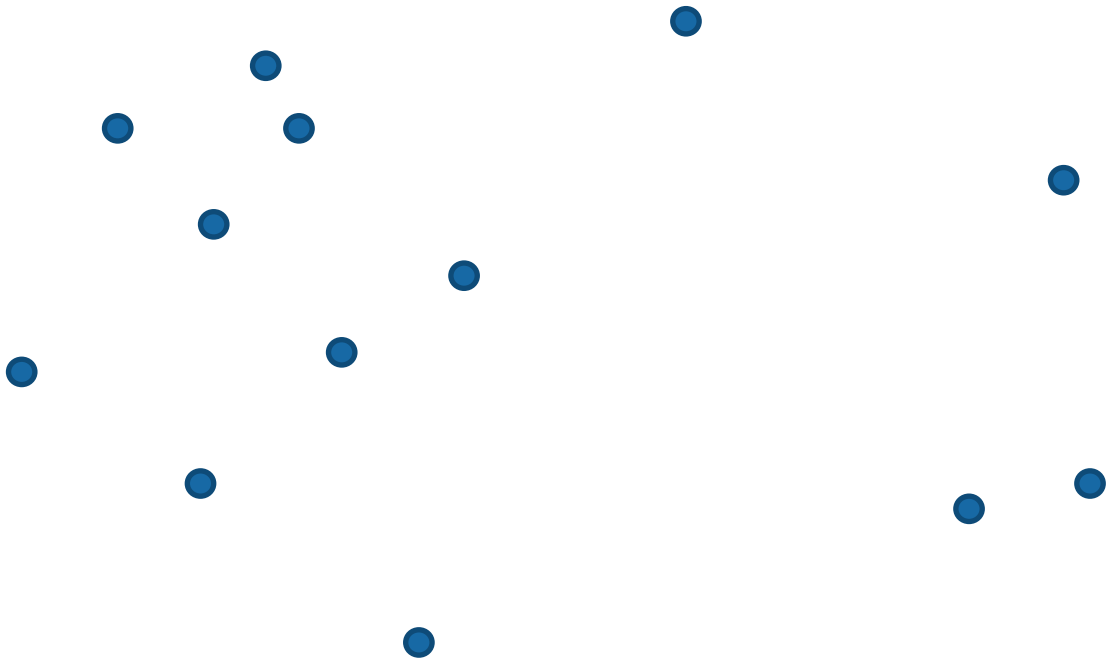
- Define a graph where the nodes are the examples in the dataset (labeled or unlabeled).
- Edges (maybe weighted) reflect the pairwise similarity of examples.
- Assumption: labels vary ‘smoothly’ over the graph:
  nodes that are ‘close’ in the graph distance have similar labels.
- Label propagation can be constrained by observed class proportions or other problem knowledge.



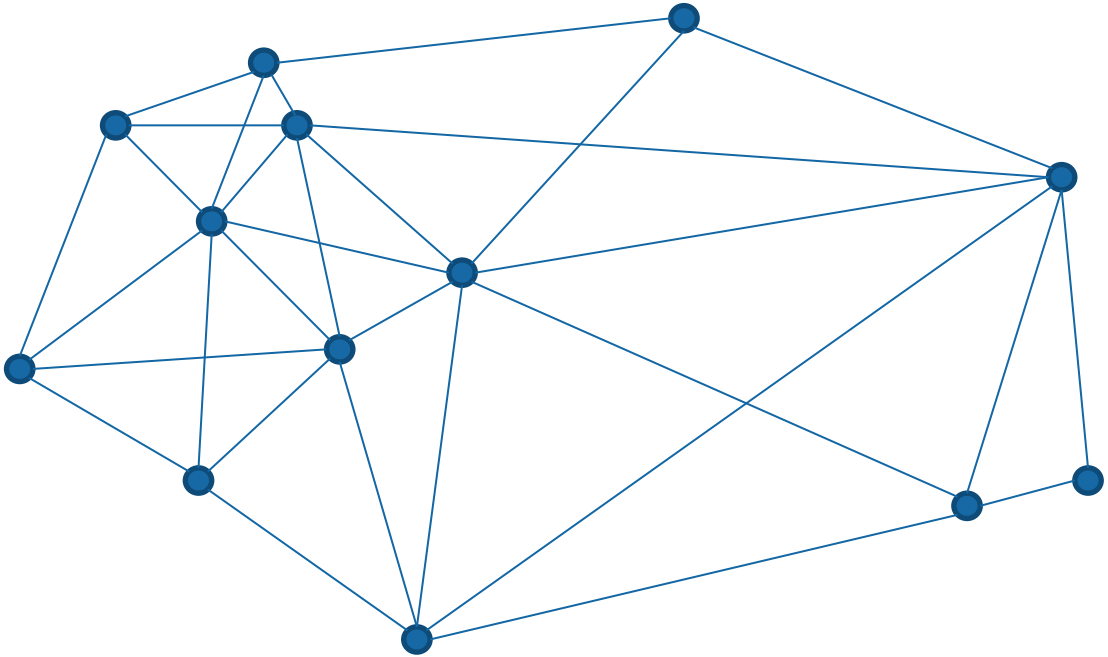
Constructing the graph

- Use domain knowledge
  - Edges reflect similarity in location, time, activity, etc.
- Use the k-NN algorithm
  - Sparse graph: each example is connected to its k-nearest neighbor (k is usually small), given a distance measure.
  - Edges can be weighted by a kernel function
- Use a kernel function (e.g. radial basis function)
  - Dense graph: edges weighted by kernel function
- Learn an embedding on examples
  - Create the graph based on distance in the embedding space

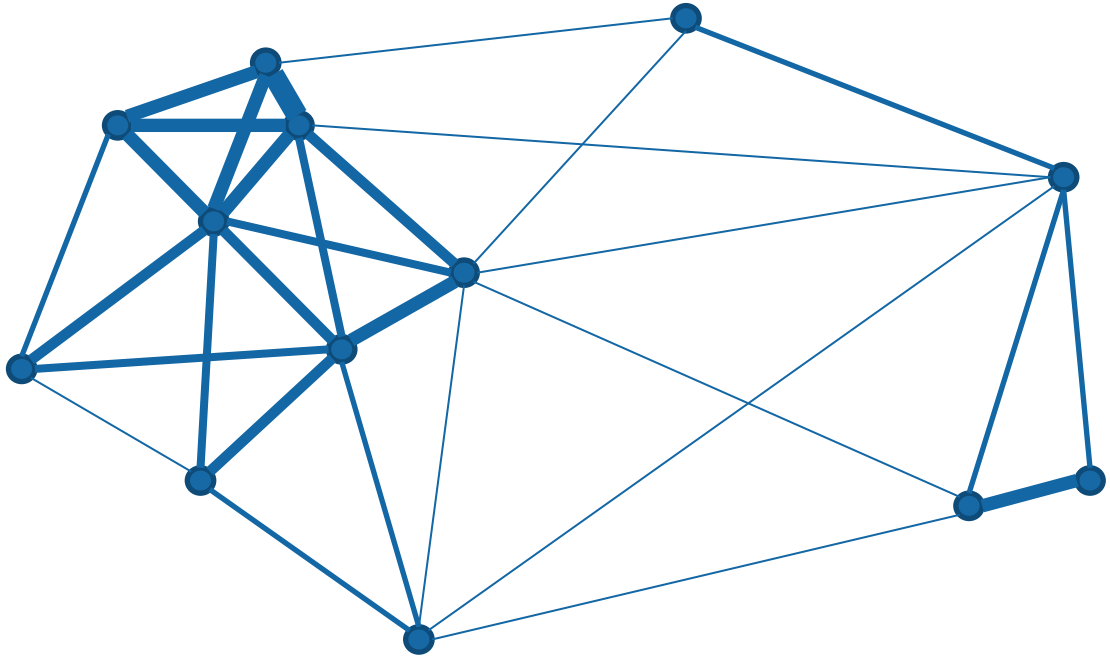
Example: distance-based graph



Example: distance-based graph

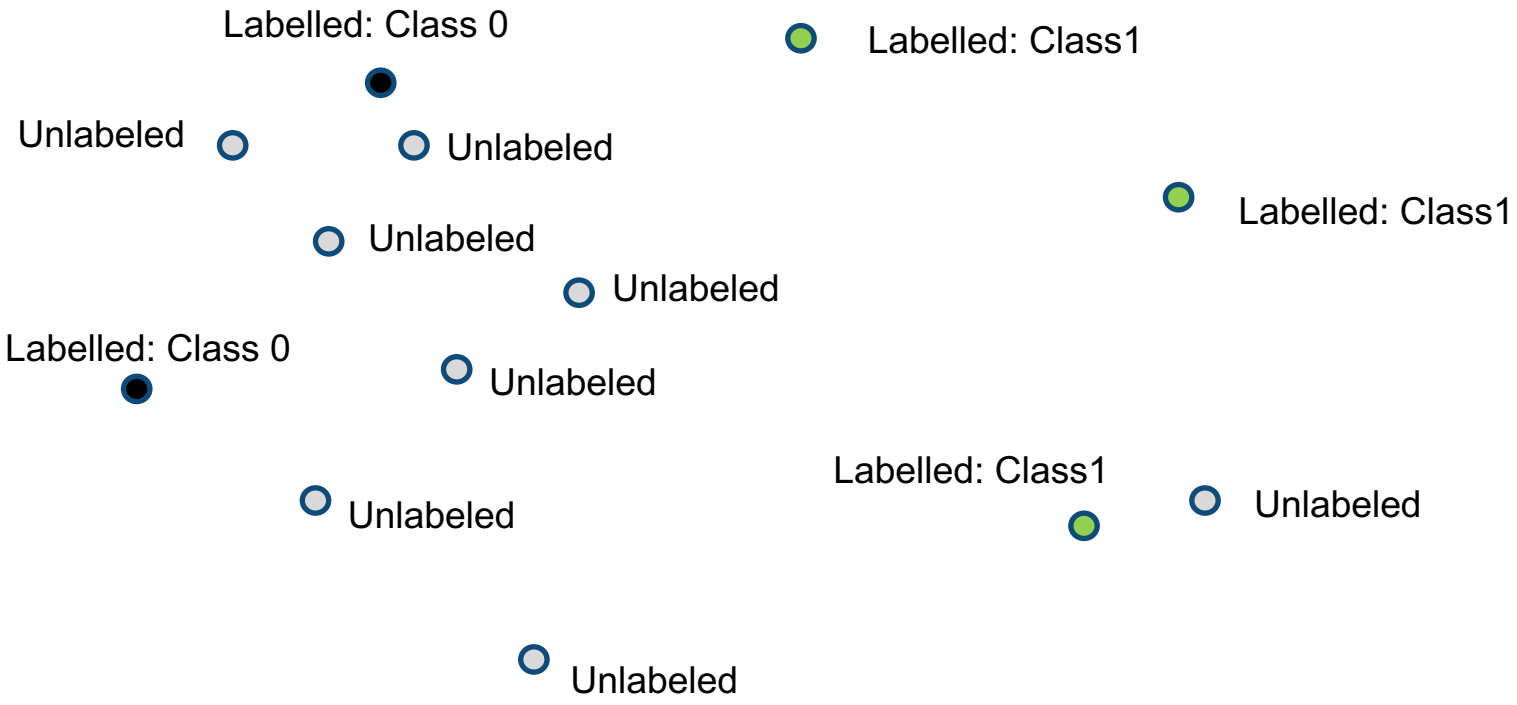


Example: dense graph with edges weighted by kernel function (Gaussian)

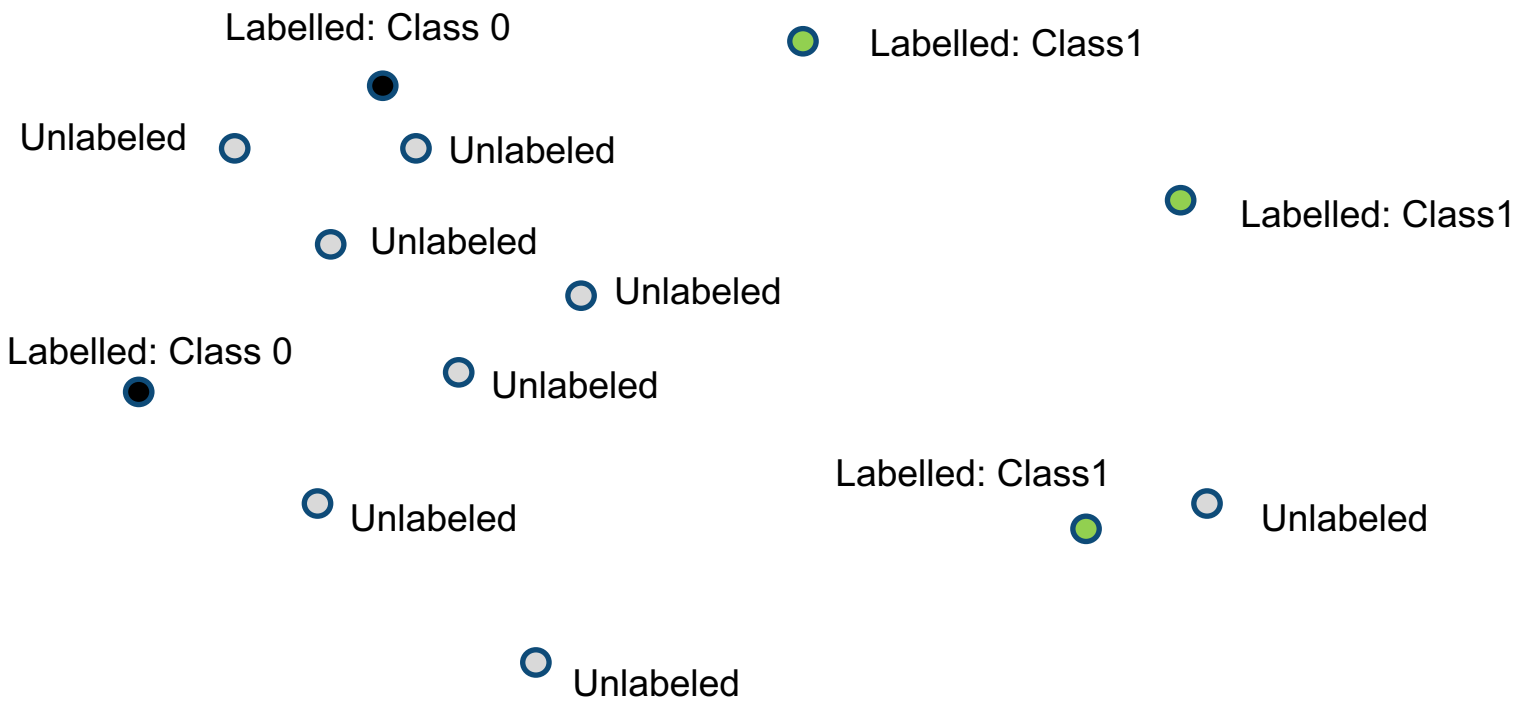


Let’s look at classification: two classes with unlabeled data (getting labeled data can be time-consuming & expensive)

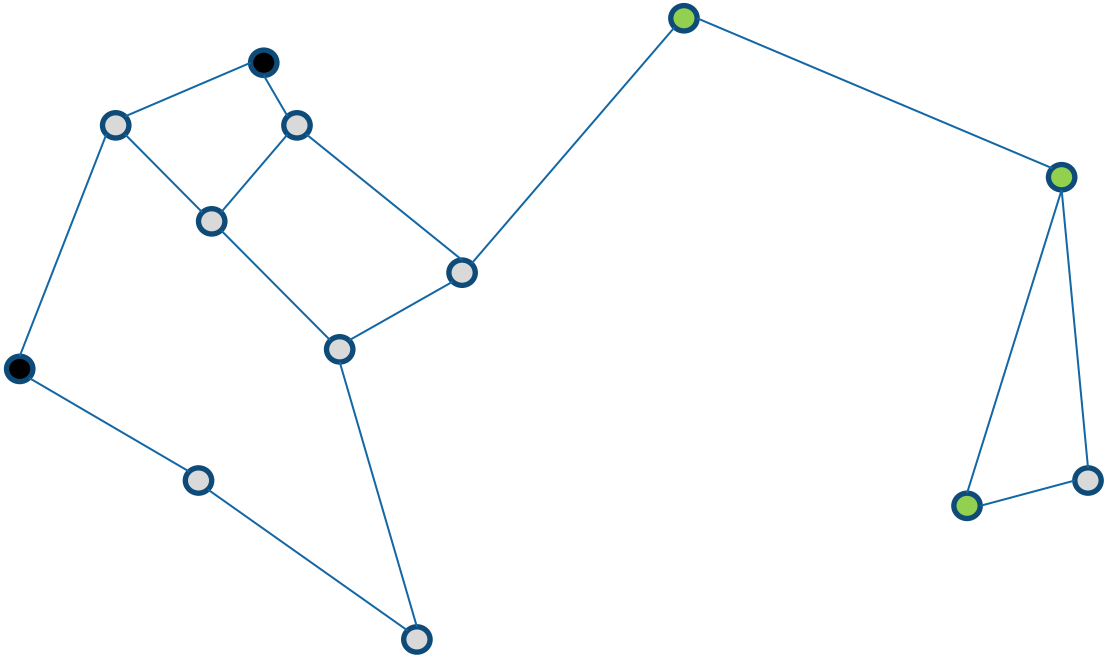
Let’s look at classification: two classes with unlabeled data (getting labeled data can be time-consuming & expensive)



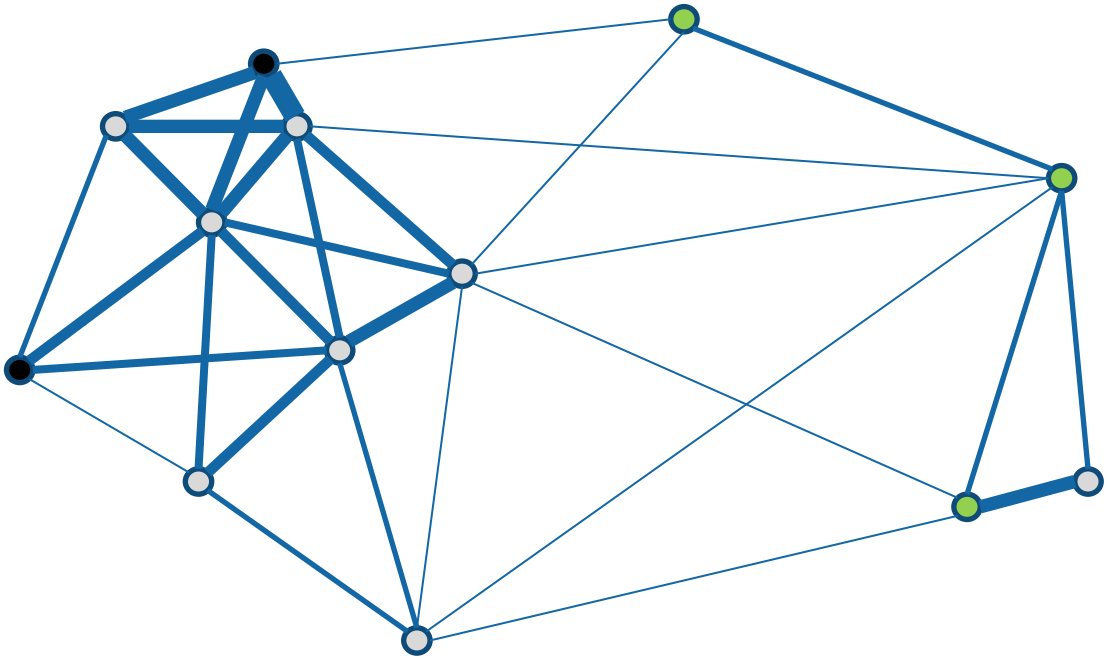
How should the unlabeled points be classified?



We could use k-NN.. But not enough labeled neighbors for some points in the corresponding graph.



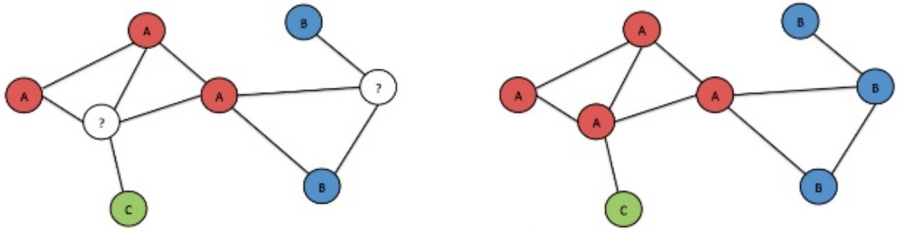
Let’s think about relaxing the idea of ‘nearest neighbor’ so that label values ‘flow’ along adjacent graph edges (‘pipes’) to nearby nodes.



This is label propagation

To do label propagation, estimate a function *f* on the graph:

1. F should agree well with the labeled nodes (loss function)
2. F should be smooth over the whole graph (regularizer)

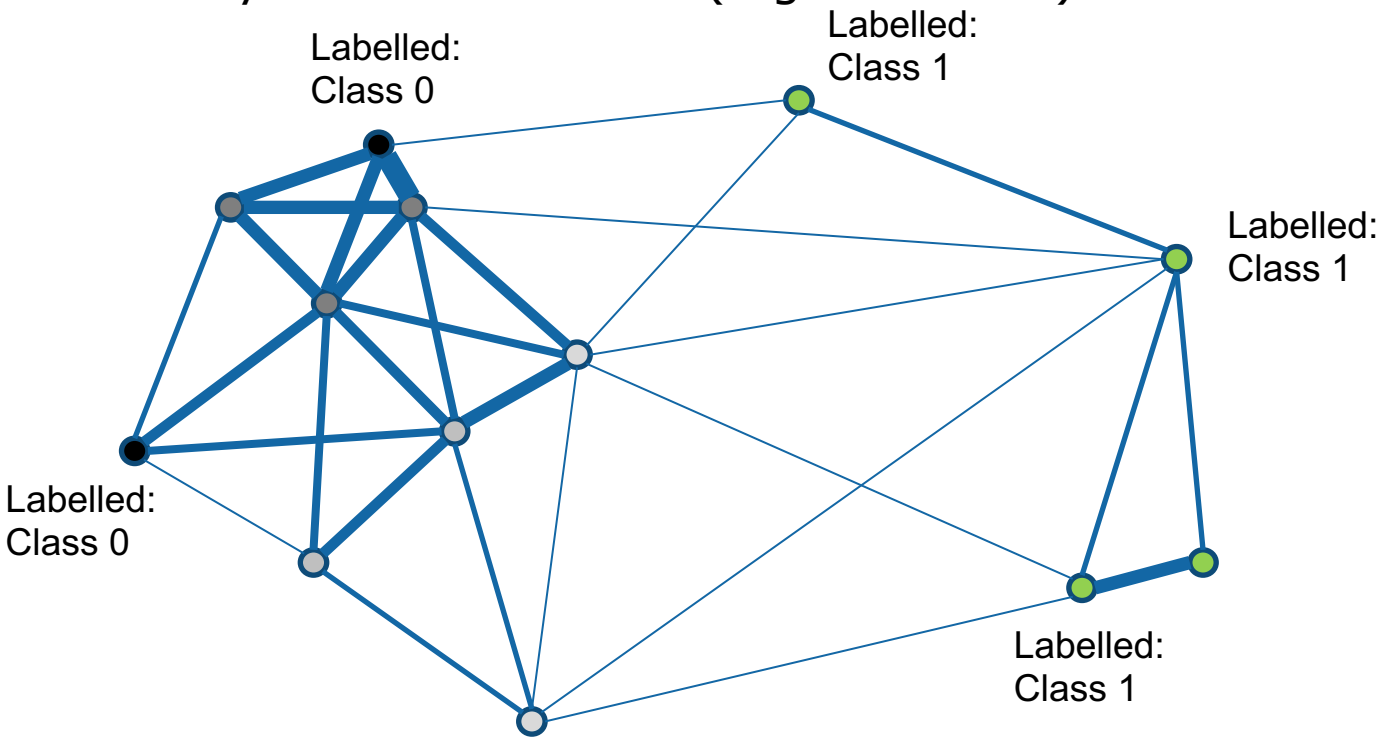


What is the 'best' label propagation? We optimize a learning objective of this form:

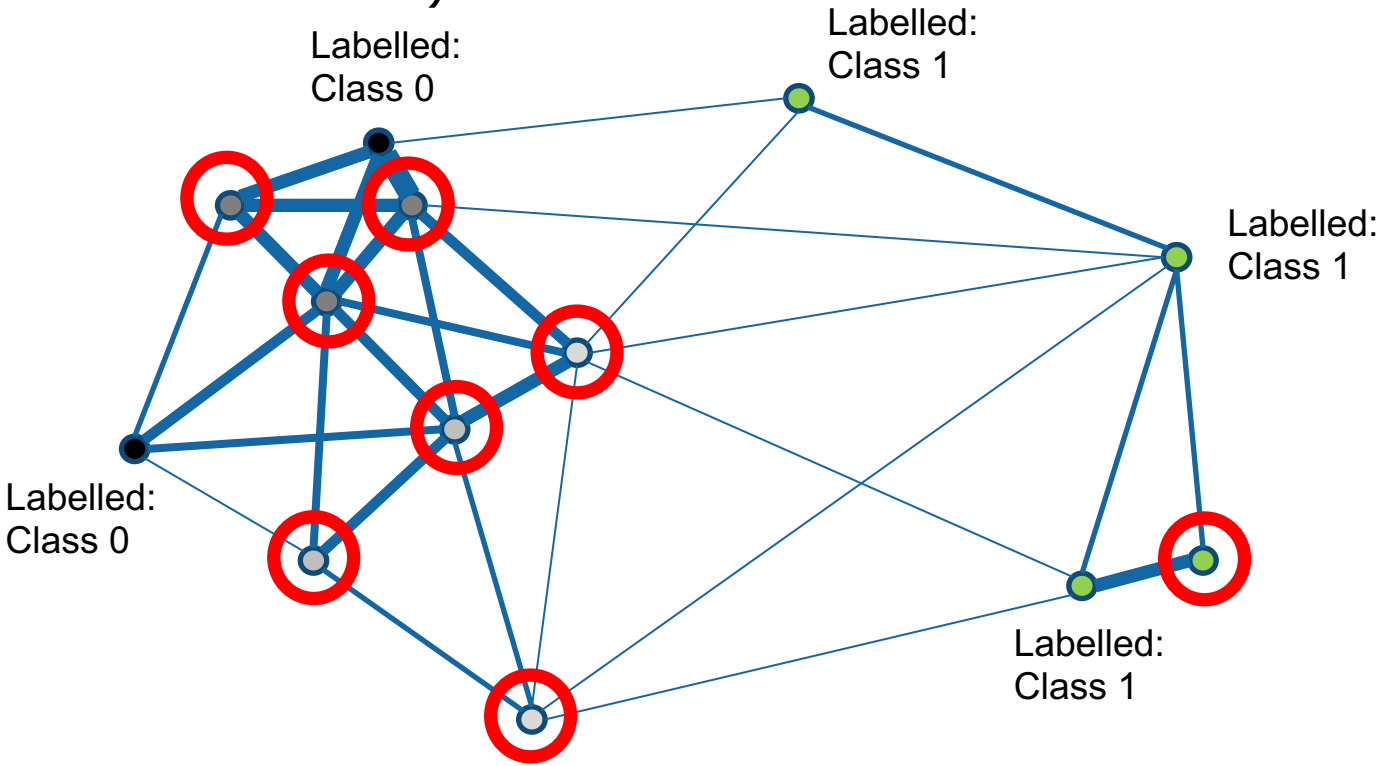
Loss function (1) + regularizer (2)

The underlying graph quality is more important than the specific method.

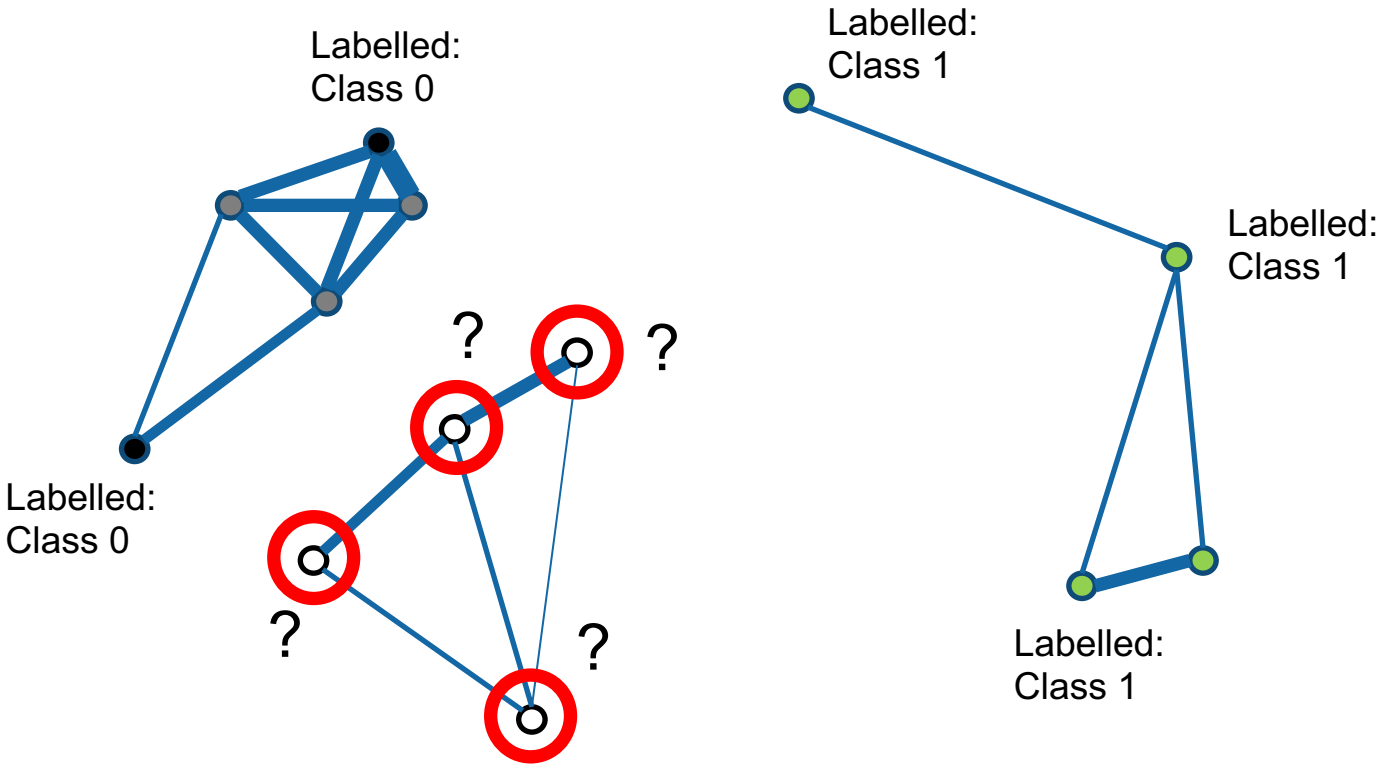
Simple label propagation along weighted edges. Edge weights determined by a kernel function (e.g. Gaussian)



Inferred labels using label propagation (using greyscale to indicate confidence)



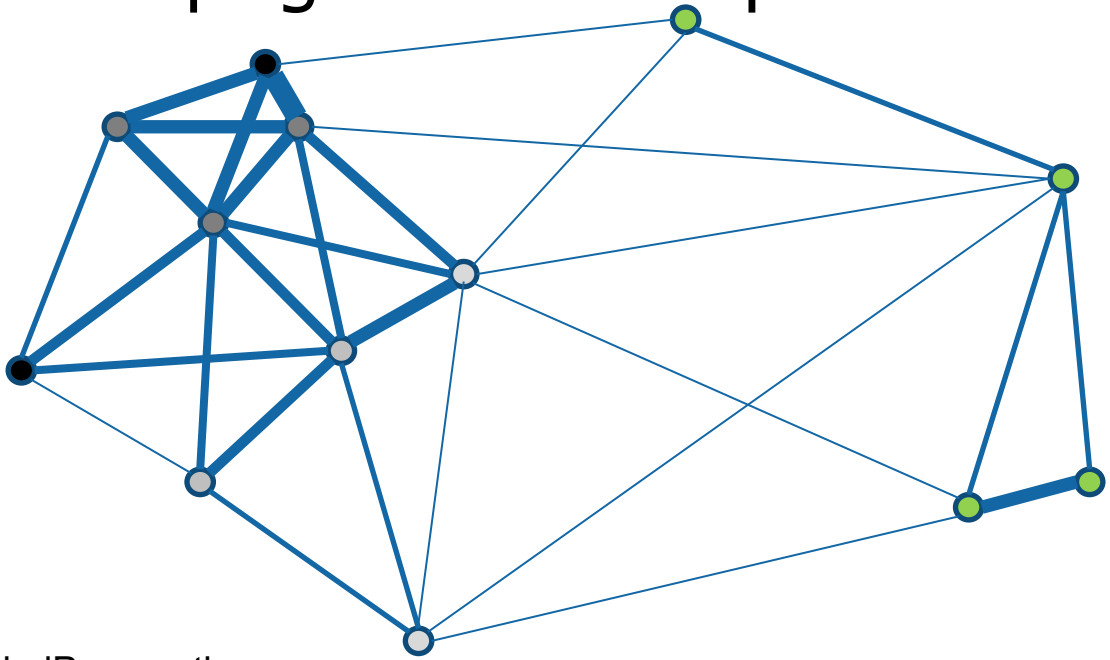
Edges must have non-zero weight to propagate labels between nodes.



scikit-learn LabelPropagation example

X

X1	X2	y_miss	y_pred
		0	0
		0	0
		-1	0
		-1	0
		1	1
		-1	1



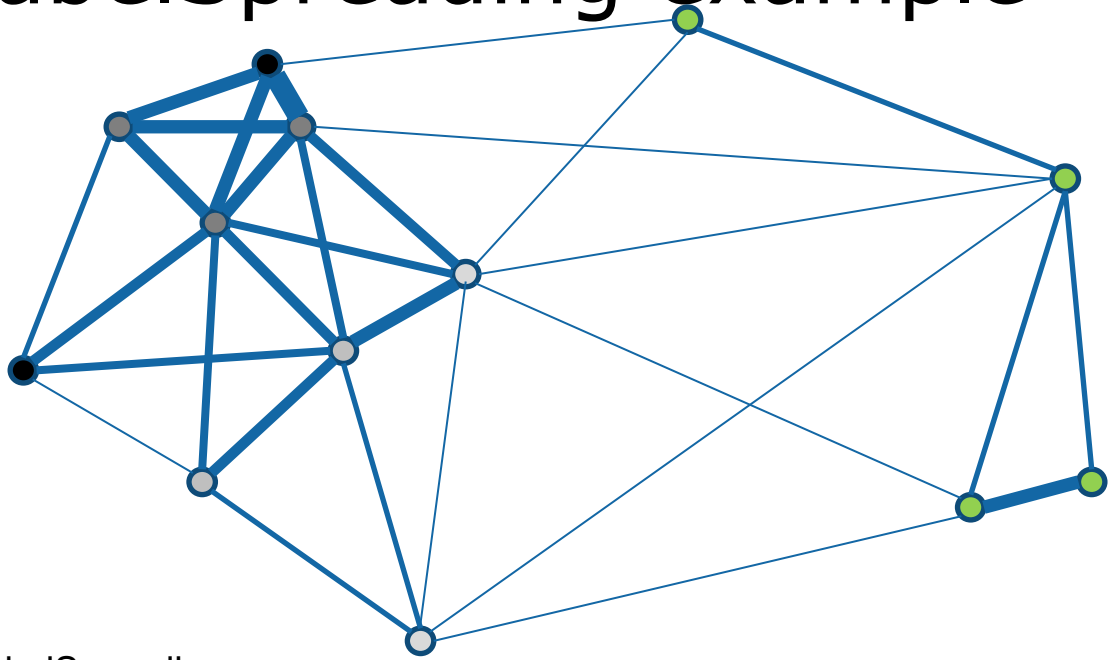
from sklearn.semi_supervised import LabelPropagation

```
label_prop_model = LabelPropagation(kernel='rbf', gamma = 0.05,
                                    max_iter=30, tol=0.001)
label_prop_model.fit(X, y_miss)
y_pred = label_prop_model.predict(X)
```

scikit-learn LabelSpreading example

X

X1	X2	y_miss	y_pred
		0	0
		0	0
		-1	0
		-1	0
		1	1
		-1	1

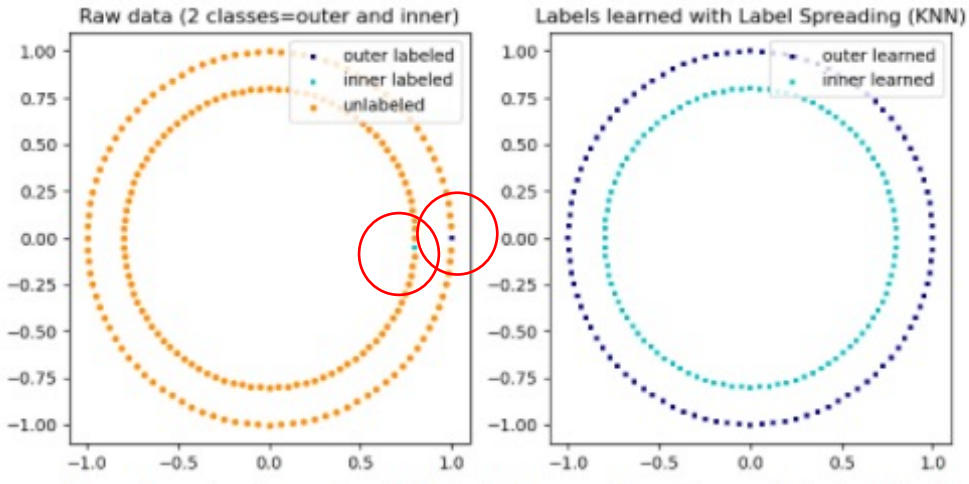


from sklearn.semi_supervised import LabelSpreading

```
label_prop_model = LabelSpreading(kernel='rbf', gamma = .05,
                                  alpha=0.1, max_iter=30, tol=0.001)
label_prop_model.fit(X, y_miss)
y_pred = label_prop_model.predict(X)
```

Using the LabelPropagation and LabelSpreading classes

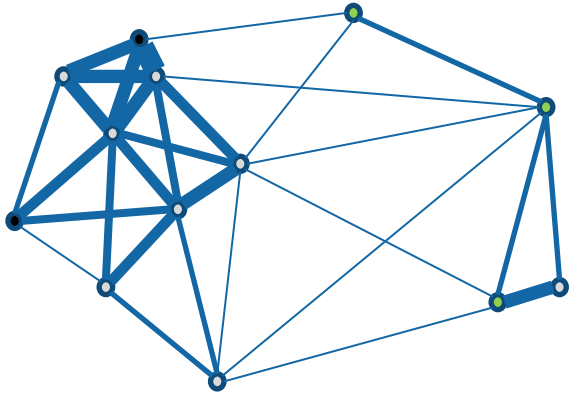
- Reminder: when training the label propagation class with the ‘fit’ method, the integer -1 is used for the ‘label’ of unlabeled entries in y.
- Can be used for both classification and regression.
- Kernel methods can be applied to project the data into new (implicit) feature spaces.
- Works best when the structure of the unlabeled points is consistent with the class structure (recall our early clustering example)



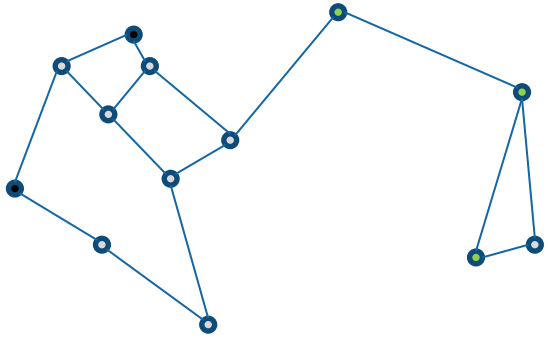
https://scikit-learn.org/stable/modules/label_propagation.html

Both label propagation methods support rbf and k-NN kernels

- Label propagation models have two built-in kernel methods.
- rbf: radial basis function. Parameter: gamma.
- knn: k-nearest neighbors. Parameter: n_neighbors.
- The RBF kernel will produce a **fully connected** graph which is represented in memory by a dense matrix. For very large datasets, the resulting matrix is also large: the cost of performing a full matrix multiplication calculation for each iteration of the algorithm can lead to prohibitively high computational cost.
- The KNN kernel will produce a more memory-friendly sparse matrix which can drastically reduce running times.. but possibly give a lower-quality labeling.



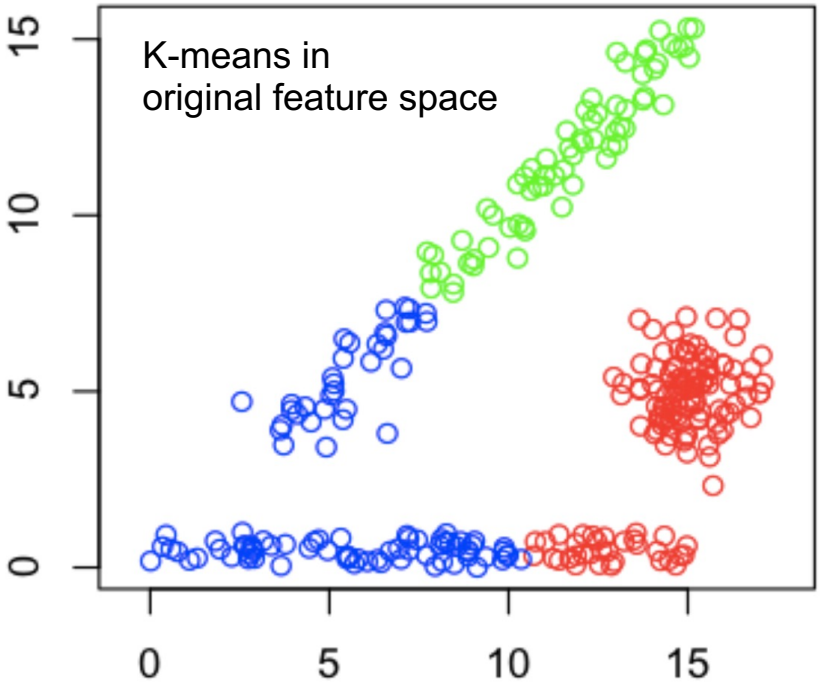
rbf

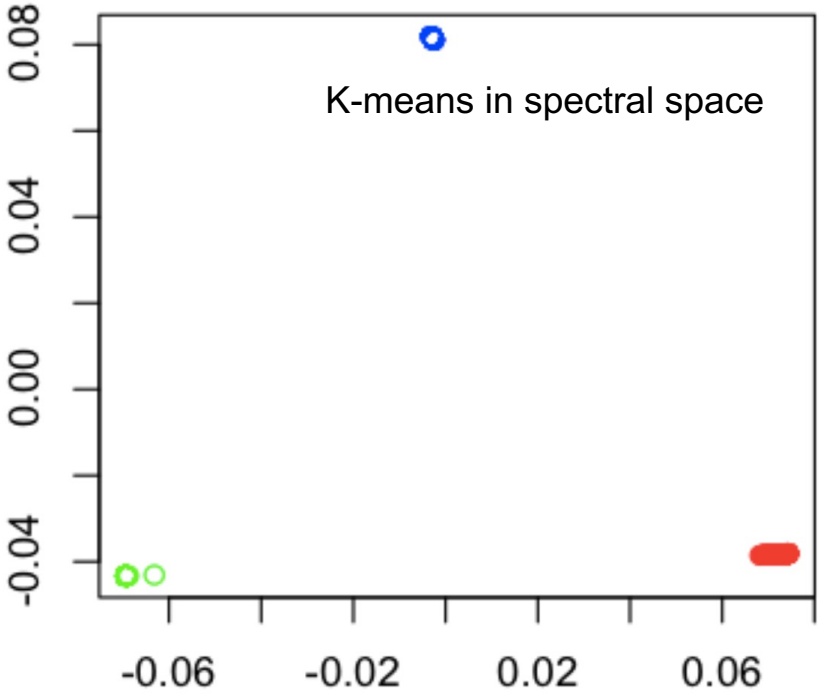


k-NN
n=2

A closer look at the LabelSpreading algorithm: spectral embedding

- Take the adjacency matrix W of a graph (W[i,j] has distance or weight between node i and node j)
- Define a matrix called the graph Laplacian
  - L = G – W where
  - G is a diagonal matrix with each node's degree (number of connected neighbors)
- Compute the eigenvectors of the graph Laplacian matrix and take the ones with 2^{nd}, 3^{rd}, 4^{th}, … smallest eigenvalues.
- This is basically like PCA for graphs.
- Nodes in the graph that are closely connected via the adjacency matrix will be close when projected to this "Graph PCA" space.





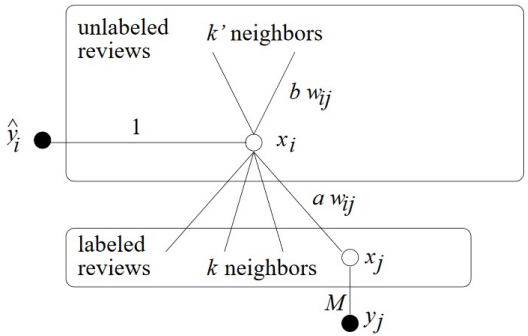
Spectral clustering:

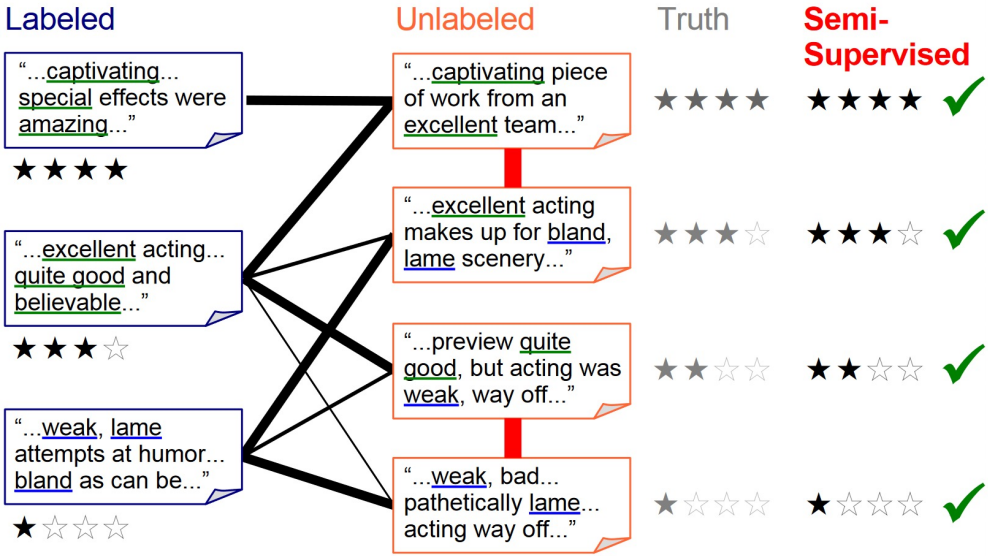
- A way to separate clustering by connectedness
- Do k-means (or other method) on the spectral embedding.
- DBSCAN is a special case of spectral clustering.

Graph-based semi-supervised learning for sentiment categorization

Problem: we want to infer a rating score for a movie (or other reviewing item). But in some scenarios many reviews may not come with ratings.

Main assumption encoded in graph:
Similar reviews should have similar ratings.





Source: Goldberg & Zhu. **Seeing stars when there aren't many stars: Graph-based semi-supervised learning for sentiment categorization.** In *HLT-NAACL 2006 Workshop on Textgraphs: Graph-based Algorithms for Natural Language Processing*, New York, NY, 2006.
http://pages.cs.wisc.edu/~jerryzhu/pub/sslsa.pdf

Summary

- Semi-supervised learning can help supervised learning in some situations
  - Generative models
  - Self-training
  - Graph-based methods
- scikit-learn support: LabelPropagation and LabelSpreading
- Awareness of spectral embedding and clustering.

Kevyn Collins-Thompson

kevynct@umich.edu

School of Information

Self-Supervised Learning

Kevyn Collins-Thompson

Associate Professor of Information and Computer Science
School of Information, University of Michigan

Problem: labeled training data is expensive

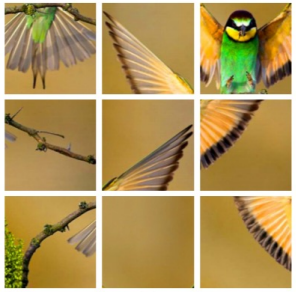
- That's partly why we're exploring the usefulness of unlabeled data…
- Labeled training data is especially needed for deep learning methods… which guzzle training data.
- How could we get large quantities of labeled training data very cheaply?

What do this tasks have in common?

What goes in the blank?

He put the freshly-baked _____ in the toaster.

What was the original unscrambled image?



What was the original orientation of this object?

What will this scene look like in 5 seconds?

What will this scene look like in 5 seconds?

What goes in the missing rectangle?

Source: https://www.scientificamerican.com/article/when-it-comes-to-safety-autonomous-cars-are-still-teen-drivers1/
https://arxiv.org/abs/1604.07379

They are all self-supervised learning tasks

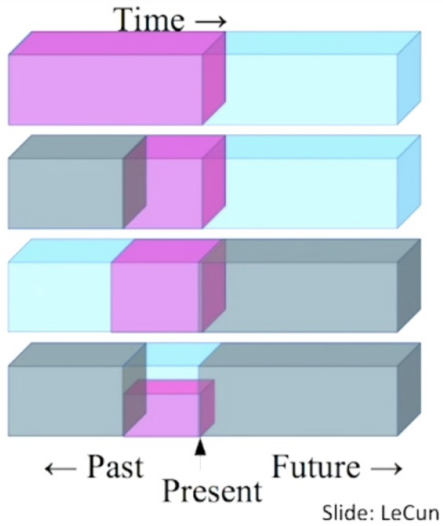
- Machines need predictive models of the world.
- Think back to our description of ‘structure’: low-dimensional manifolds in a high dimensional space.
- Enter **self-supervised** predictive learning:
  - Take something as input – an object A - with interesting structure.
  - Remove a part (say R) of A.
  - R becomes our prediction target.
  - We now have a supervised learning problem:

    Can we predict R from the other parts of A?

  - But didn’t have to exert any effort to get the labeled example since it was just part of our observed input.
  - We’re doing supervised learning, but with an unlabeled dataset!

Self-supervised learning tasks predict any part of the input from any other part.

- Predict any part of the input from any other part.
- Predict the future from the past.
- Predict the future from the recent past.
- Predict the past from the present.
- Predict the top from the bottom.
- Predict the occluded from the visible
- Pretend there is a part of the input you don't know and predict that.



Source: Yann LeCun

Brief digression: deep learning summary

- Deep learning architectures combine a sophisticated automatic feature extraction phase with a supervised learning phase.
- The feature extraction phase uses a hierarchy of multiple feature extraction layers.
- Starting from primitive, low-level features in the initial layer, each feature layer's output provides the input features to the next higher feature layer.
- All features are used in the final supervised learning model.

An example of a simple deep learning architecture

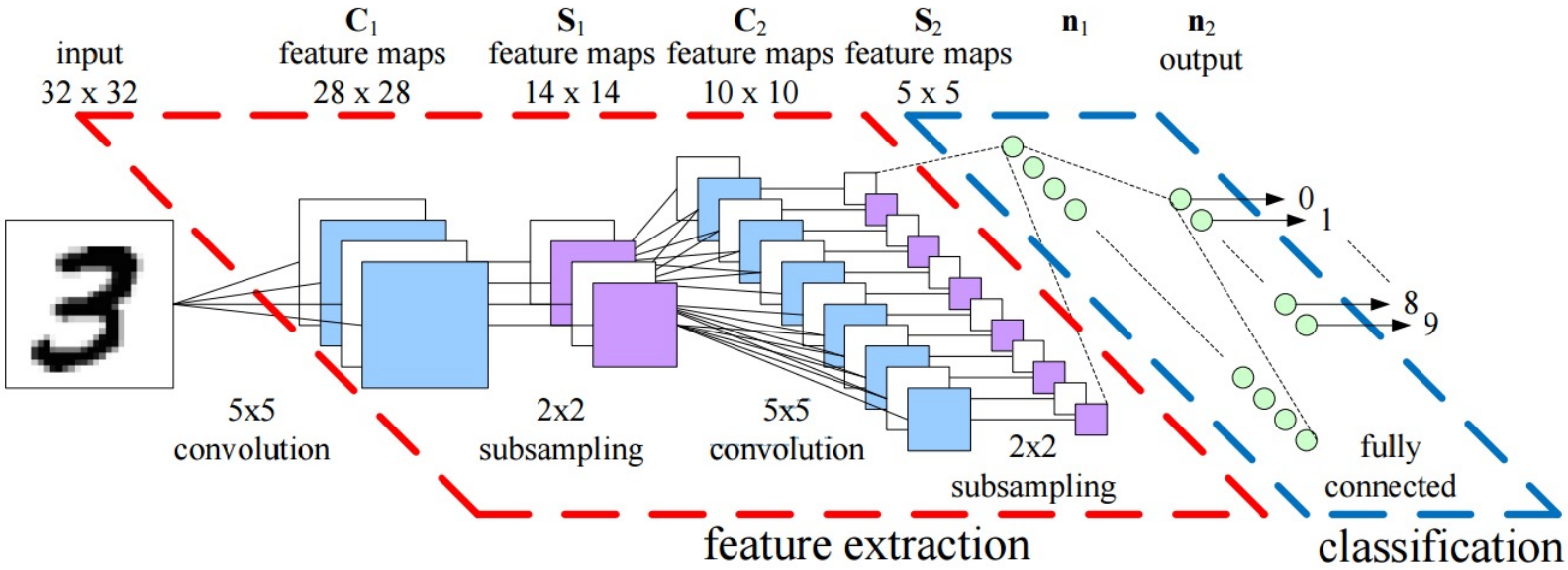
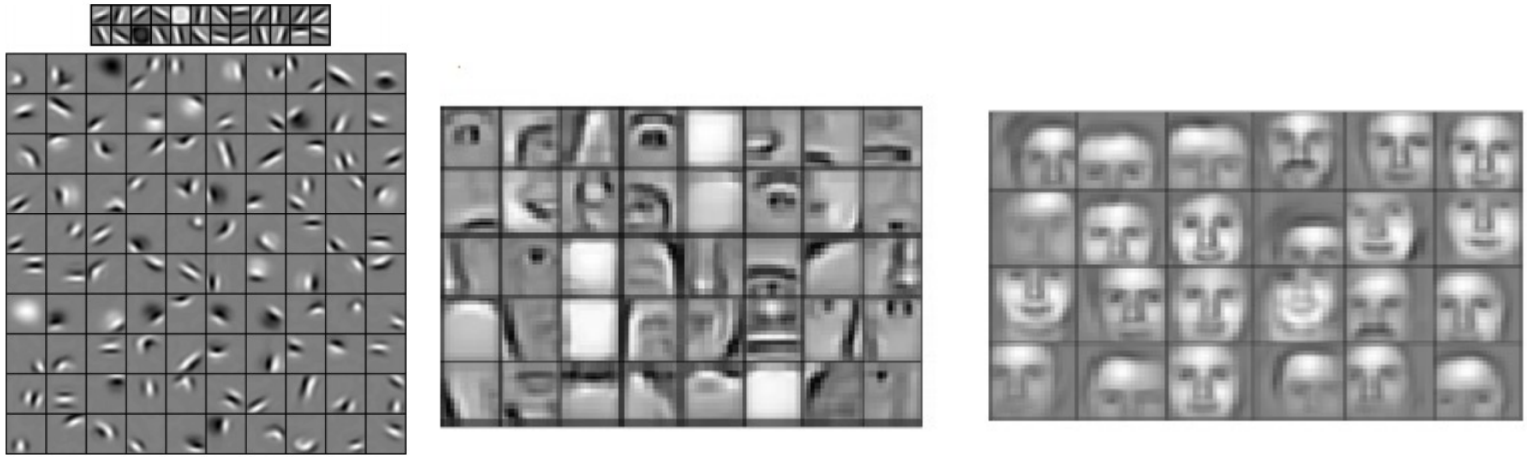


Image: M. Peemen, B. Mesman, and H. Corporaal. Efficiency Optimization of Trainable Feature Extractors for a Consumer Platform. Proceedings of the 13th International Conference on Advanced Concepts for Intelligent Vision Systems, 2011.

Feature extraction: intermediate representations for faces learned by the DL model



First, second, and third feature layer bases learned for faces

Image: Honglak Lee and colleagues (2011) from “Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks”. Communications of the ACM, Vol. 54 No. 10, Pages 95-103

Intermediate representations learned for other categories

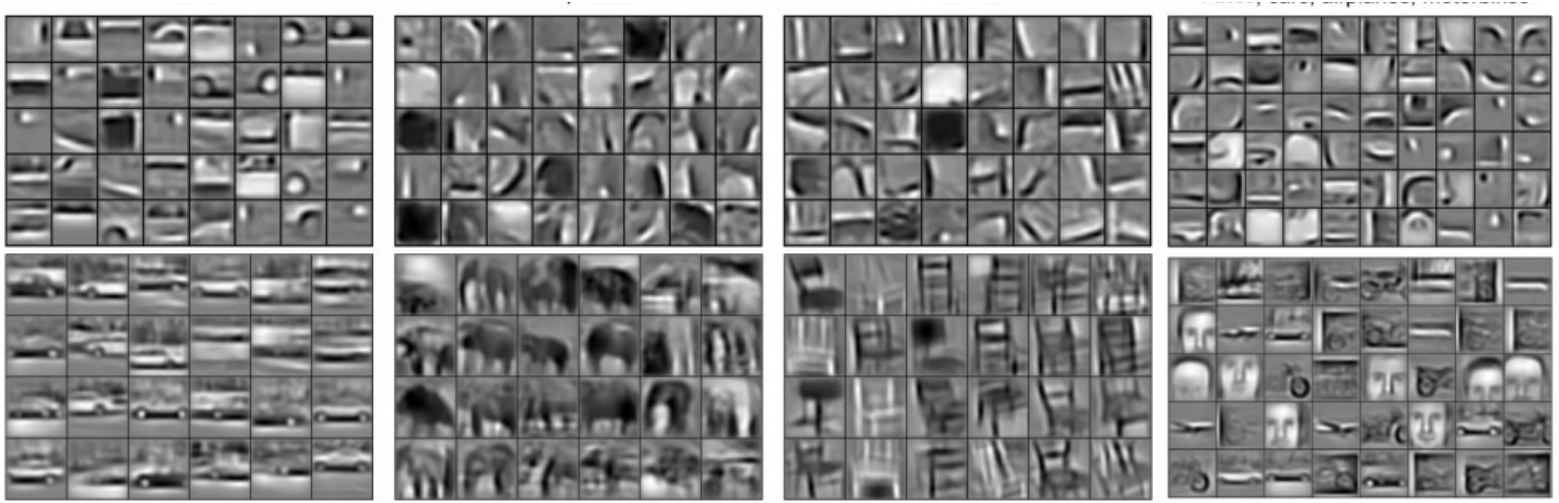
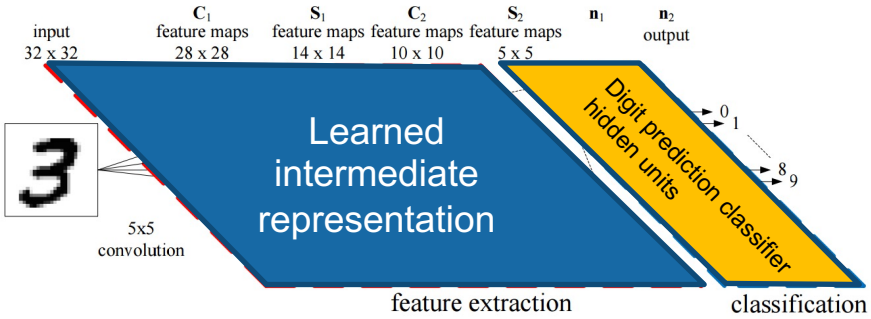


Image: Honglak Lee and colleagues (2011) from “Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks”. Communications of the ACM, Vol. 54 No. 10, Pages 95-103

Using pretext (self-supervised) prediction tasks to learn high-quality intermediate representations

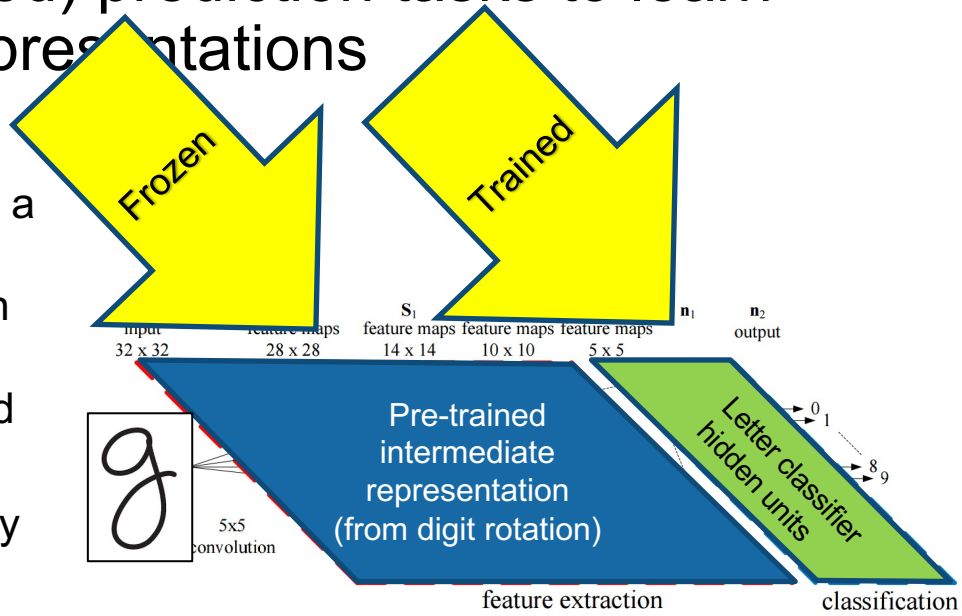
- Pick a ‘pretext’ prediction task appropriate to your domain, such as: a rotated image.
  - We don’t actually care too much about perfect prediction.
  - We care more about the learned intermediate representation
- Expect the model to learn high-quality latent variable models for real-world tasks
- Use the same pre-trained representation for many different prediction tasks.



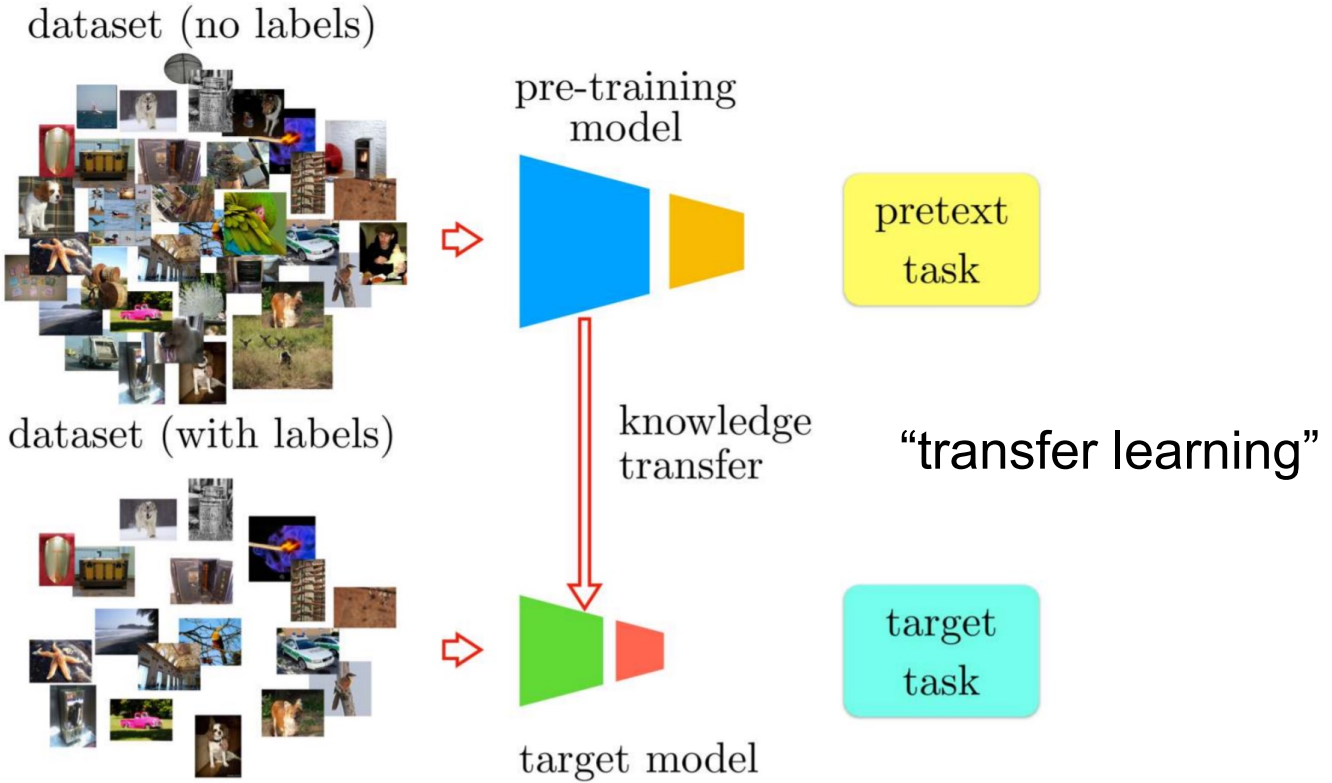
Pretext task: predict class of a rotated image

Using pretext (self-supervised) prediction tasks to learn high-quality intermediate representations

- Pick a ‘pretext’ prediction task appropriate to your domain, such as: a rotated image.
  - We don’t actually care too much about perfect prediction.
  - We care more about the learned intermediate representation
- Expect the model to learn high-quality latent variable models for real-world tasks
- Use the same pre-trained representation for many different prediction tasks.



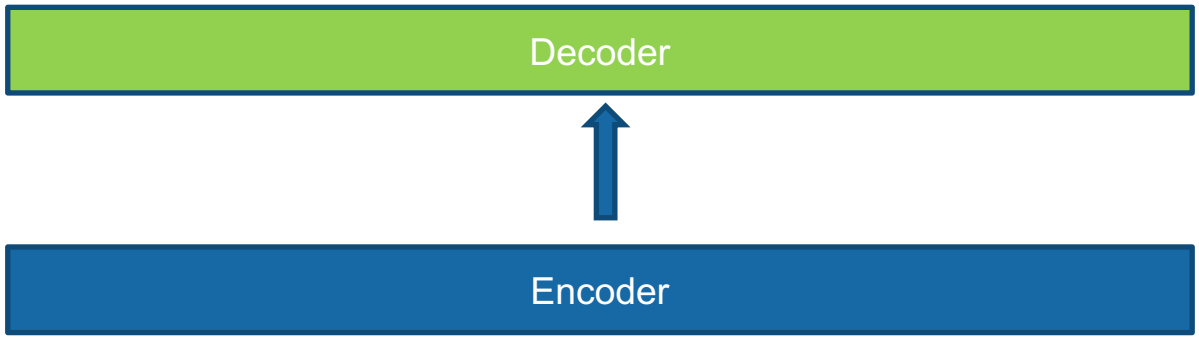
<u>New</u> classification task: predict handwritten letters.
Uses <u>pretrained model from the pretext task *for feature extraction.*</u>



Source: https://www.csee.umbc.edu/~hpirsiav/papers/transfer_cvpr18.pdf

Language models and word embeddings as self-supervised learning

Output: This is a **piece** of text extracted **from** a large set of **news** articles.



Input: **This is a** […] **of text extracted** […] **a large set of** […] **articles**.

When to consider self-supervised learning tasks

- When getting labels is expensive.
- When the input data is unstructured and requires a complex, automatically derived feature representation.
  - Images, video, natural language, sensor data..
- .. which implies large volumes of data are needed.
- When the intermediate representation learned by the pretext task could benefit a variety of downstream prediction tasks.
  - Embeddings: part of speech tagging, analogy prediction, semantic matching, many other NLP tasks.

Kevyn Collins-Thompson

kevynct@umich.edu

School of Information

Imputation of Missing Data

Kevyn Collins-Thompson

Associate Professor of Information and Computer Science
School of Information, University of Michigan

Imputation of Missing Data

- Authentic datasets contain missing values
  - NaN, empty cells or strings, -1, NA, -999, etc.
- Particularly evident in survey questionnaires.
- But scikit learn estimators assume complete data!
- What to do?
- (Very) basic strategy: discard entire rows and/or columns containing missing values.
  - Price: lose valuable (if incomplete) data
- Better: *impute* the missing values. Infer them from the known data.

User	Device	OS	Transactions
A	Mobile	Android	5
B	Mobile	Android	3
C	NA	iOS	-1
D	Tablet		1
E	Mobile	iOS	4
F	NA	Android	2
G	Tablet		4

Ozone data set: how to do regression with missing values?

	maxO3	T9	T12	T15	Ne9	Ne12	Ne15	Vx9	Vx12	Vx15	maxO3v
0601	NA	15.6	18.5	18.4	4	4	8	NA	-1.7101	-0.6946	84
0602	82	17	18.4	17.7	5	5	7	NA	NA	NA	87
0603	92	NA	17.6	19.5	2	5	4	2.9544	1.8794	0.5209	82
0604	114	16.2	NA	NA	1	1	0	NA	NA	NA	92
0605	94	17.4	20.5	NA	8	8	7	-0.5	NA	-4.3301	114
0606	80	17.7	NA	18.3	NA	NA	NA	-5.6382	-5	-6	94
0607	NA	16.8	15.6	14.9	7	8	8	-4.3301	-1.8794	-3.7588	80
0610	79	14.9	17.5	18.9	5	5	4	0	-1.0419	-1.3892	NA
0611	101	NA	19.6	21.4	2	4	4	-0.766	NA	-2.2981	79
0612	NA	18.3	21.9	22.9	5	6	8	1.2856	-2.2981	-3.9392	101
0613	101	17.3	19.3	20.2	NA	NA	NA	-1.5	-1.5	-0.8682	NA
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	
0919	NA	14.8	16.3	15.9	7	7	7	-4.3301	-6.0622	-5.1962	42
0920	71	15.5	18	17.4	7	7	6	-3.9392	-3.0642	0	NA
0921	96	NA	NA	NA	3	3	3	NA	NA	NA	71
0922	98	NA	NA	NA	2	2	2	4	5	4.3301	96
0923	92	14.7	17.6	18.2	1	4	6	5.1962	5.1423	3.5	98
0924	NA	13.3	17.7	17.7	NA	NA	NA	-0.9397	-0.766	-0.5	92
0925	84	13.3	17.7	17.8	3	5	6	0	-1	-1.2856	NA
0927	NA	16.2	20.8	22.1	6	5	5	-0.6946	-2	-1.3681	71
0928	99	16.9	23	22.6	NA	4	7	1.5	0.8682	0.8682	NA
0929	NA	16.9	19.8	22.1	6	5	3	-4	-3.7588	-4	99
0930	70	15.7	18.6	20.7	NA	NA	NA	0	-1.0419	-4	NA

Source: http://www.airbreizh.assoc.fr

Three categories of missing data

1. MCAR ‘Missing Completely At Random’
    - Missingness not related to any specific variable: missing data are just a completely random subset.
    - e.g. you use a weighing scale to gather weights of specimens, but afterward some cells in your data file are randomly corrupted due to a software glitch.
    - A fairly unrealistic assumption.
2. MAR ‘Missing At Random (conditionally)’
    - Propensity for a data point to be missing is related to some of the observed data.
    - e.g. when placed on a soft surface, your weighing scale may produce more missing values than when placed on a hard surface.
    - Most missing data is in this category.

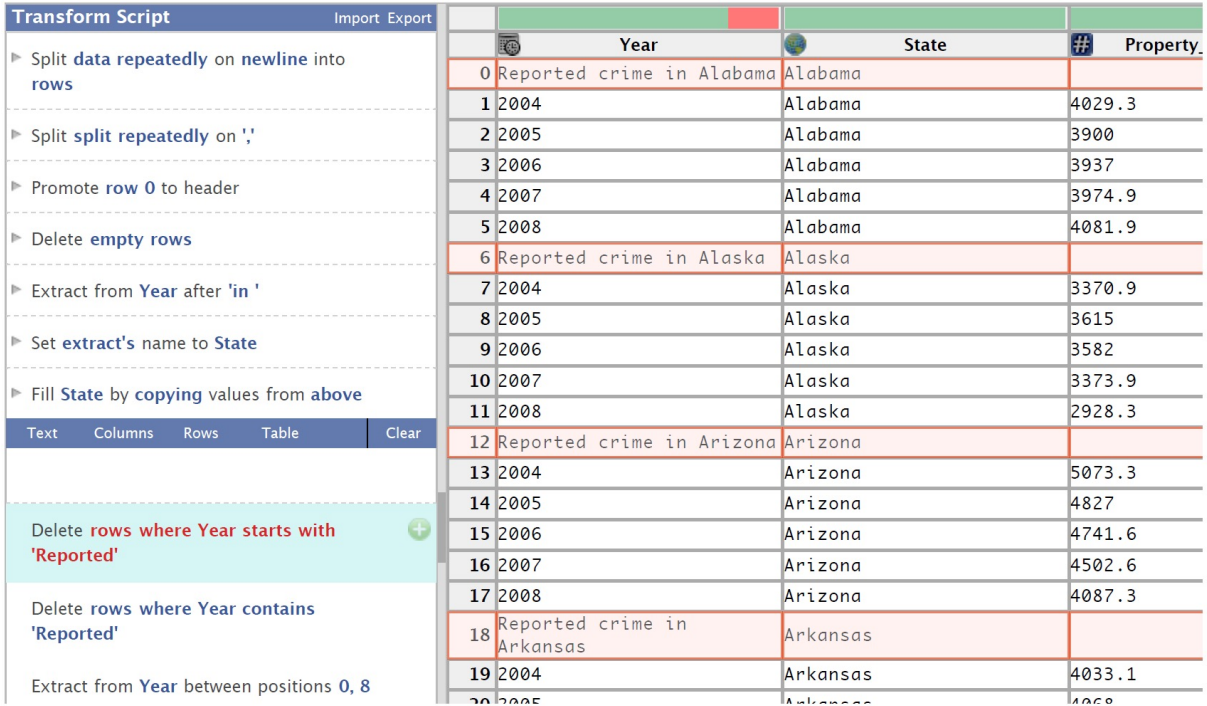
Three categories of missing data

- 3. MNAR/NMAR 'Missing Not At Random'
  - Probability of missing value varies for unknown reasons
  - e.g. weighing scale may wear out over time but we didn't notice.
  - e.g. surveys: people with weaker opinions (or other individual factors) respond less often.
  - This is the most complex case: need more data about missingness causes, what-if analysis to see how sensitive results are under different scenarios (e.g. if there are associated feature value ranges)
- Most simple imputation methods (replace with mean) assume MCAR (completely random is a restrictive and usually unrealistic assumption).
- Data imputation is a significant sub-area of statistical research.
  - There exist a vast array of techniques…
  - The goal of this module is to make you aware of key concepts and methods.

Option 1: complete case analysis

- First look at the proportion of rows with missing values in your data. Just ignore rows with missing features and use only complete cases (those rows with no missing features).
  - `df.dropna(axis = 0)`
- Typically only consider this if % of missing values is small (< 5%)
- In the MCAR case (completely random), using complete cases is statistically unbiased, but MCAR is not that realistic. Instead…
- Do an analysis (e.g. PCA/MDS): are the complete cases systematically different from the whole dataset? If so, beware bias from focusing on only the complete cases.
  - e.g. you send out a survey by email. Only a fraction return the questionnaire – which factors might be associated with completed responses? Time? Income? Age?

Option 2: Data ‘wrangling’ and cleaning tools if human effort is available.



	Year	State	Property_
0	Reported crime in Alabama	Alabama	
1	2004	Alabama	4029.3
2	2005	Alabama	3900
3	2006	Alabama	3937
4	2007	Alabama	3974.9
5	2008	Alabama	4081.9
6	Reported crime in Alaska	Alaska	
7	2004	Alaska	3370.9
8	2005	Alaska	3615
9	2006	Alaska	3582
10	2007	Alaska	3373.9
11	2008	Alaska	2928.3
12	Reported crime in Arizona	Arizona	
13	2004	Arizona	5073.3
14	2005	Arizona	4827
15	2006	Arizona	4741.6
16	2007	Arizona	4502.6
17	2008	Arizona	4087.3
18	Reported crime in Arkansas	Arkansas	
19	2004	Arkansas	4033.1

Source: http://vis.stanford.edu/files/2011-Wrangler-CHI.pdf

Option 3: Learning algorithm handles the missing values

- Decision trees and related tree-based algorithms handle missing data naturally (classification and regression)
  - One approach: distribute the missing value instance to the child node with the most instances OR to a random child node.
- Incorporate a missing value weight into the objective function (recall MDS)

Add weights to the optimization objective 'stress' function:

$$\sigma(\hat{d}, X, \theta) = \sum_{i<j} w_{ij} (f_\theta(d_{ij}) - \|x_i - x_j\|_2)^2$$

w_{ij} =1 if the dissimilarity between points x_i, x_j is defined

w_{ij} =0 if the dissimilarity between points x_i, x_j is undefined

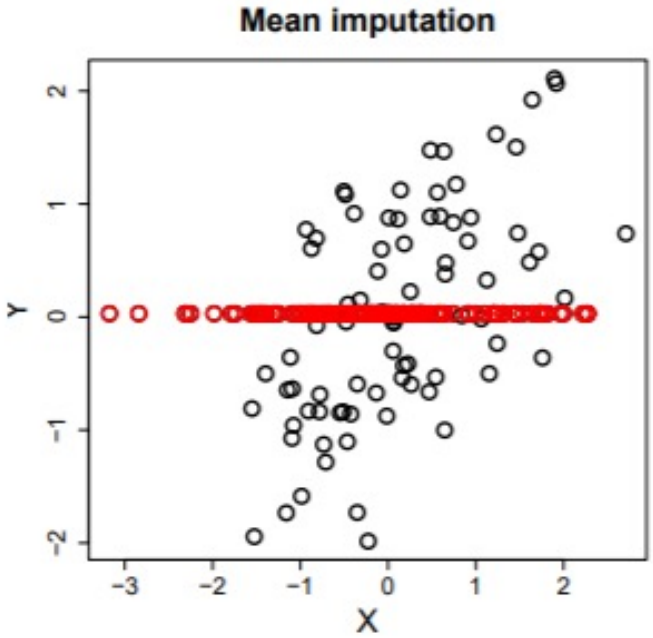
In theory, this is a more subtle criterion than just skipping the data point.

Option 4: impute missing values (carefully)

Univariate imputation:

- Focus on the *i*-th feature column only.
- Impute missing values in that column using the known values in same column.
- A ‘simple’ strategy.

The danger of simple imputation



before	after
$\mu_y = 0$	0.01
$\sigma_y = 1$	0.5
$\rho = 0.6$	0.30

Source: http://juliejosse.com/wp-content/uploads/2018/07/ChairBio.pdf

Simple univariate imputation in scikit-learn: SimpleImputer

```
def get_scores_using_imputer(regressor, imputer,
                             X_missing, y_missing, score_type = score_type):
    estimator = make_pipeline(imputer, regressor)
    impute_scores = cross_val_score(estimator, X_missing, y_missing,
                                    scoring=score_type,
                                    cv=CV_SPLITS)
    return impute_scores
```

```
def get_impute_zero_score(regressor, X_missing, y_missing):

    imputer = SimpleImputer(missing_values=np.nan, add_indicator=True,
                            strategy='constant', fill_value=0)
    zero_impute_scores = get_scores_using_imputer(regressor, imputer, X_missing, y_missing)
    return zero_impute_scores.mean(), zero_impute_scores.std()


mses_dataset[0], stds_dataset[0] = get_impute_zero_score(regressor, X_missing, y_missing)
```

Supports dense or sparse input matrices

Possible choices for the 'strategy' property:

- "mean": replace missing values using the mean along each column. Can only be used with numeric data.
- "median": replace missing values using the median along each column. Can only be used with numeric data.
- "most_frequent": replace missing using the most frequent value along each column. Can be used with strings or numeric data.
- "constant": replace missing values with fill_value. Can be used with strings or numeric data.

Applying k-NN to imputation

- impute.KNNImputer

```
def get_scores_using_imputer(regressor, imputer,
                             X_missing, y_missing, score_type = score_type):
    estimator = make_pipeline(imputer, regressor)
    impute_scores = cross_val_score(estimator, X_missing, y_missing,
                                    scoring=score_type,
                                    cv=CV_SPLITS)
    return impute_scores
```

```
def get_impute_knn_score(regressor, X_missing, y_missing):
    imputer = KNNImputer(missing_values=np.nan, add_indicator=True)
    knn_impute_scores = get_scores_using_imputer(regressor, imputer, X_missing, y_missing)
    return knn_impute_scores.mean(), knn_impute_scores.std()

mses_dataset[4], stds_dataset[4] = get_impute_knn_score(regressor, X_missing, y_missing)
```

Could pass in n_neighbors here

The MissingIndicator class

```
import numpy as np
from sklearn.impute import MissingIndicator
X1 = np.array([[np.nan, 1, np.nan, 7],
               [4, 0, 1., np.nan],
               [8, 1, 2., 0]])
X2 = np.array([[5, 1, np.nan, -1.],
               [np.nan, 2, 3, -3.],
               [2, 10., 4, 0]])
indicator = MissingIndicator()
indicator.fit(X1)
X2_tr = indicator.transform(X2)
print(X2_tr)
print(indicator.features_)
```

```
[[False  True False]
 [ True False False]
 [False False False]]
[0 2 3]
```

- The fit method flags all columns that have missing data.
- The transform method produces an indicator vector for each instance, showing which of the columns originally flagged during ‘fit’ as having missing data also have missing data in the transform input.

The EM algorithm can be used for imputation

- Recall that the whole point of the EM algorithm was to deal with incomplete data situations.
- Example: missing responses to survey data. We want to estimate probabilities from the responses using a particular statistical model whose parameters we want to estimate.
  - If the data were complete, it would be easy.
  - But some observations aren't complete…
  - So with EM we guess some initial values (e.g. mean imputation), then iterate:
    - E-step: fill the missing data with pseudo-data
    - M-step: use the now "pseudo-complete" dataset to re-estimate parameters.

Using EM with incomplete survey data (two-way tables)

Each respondent is polled twice, six months apart about a topic: in this case, whether they have been the victim of a crime.

	Second Interview		
First Interview	Crime-Free	Victims	Nonrespondents
Crime-Free	392	55	33
Victims	76	38	9
Nonrespondents	31	7	115

We have two binary variables *I* and *J*
I = crime-free first interview
J = crime-free second interview

$n_{ij}^{(1)}$ complete observations for both *I* and *J*
$n_{ij}^{(2)}$ observations only for *I*
$n_{ij}^{(3)}$ observations only for *J.*

Where is the missing data here?

- Nonrespondents in either first or second interview.

What are the parameters we want to estimate?

- p_{ij} : probability of the conditions on I, J
- e.g. fraction of respondents that reported crime victim status (or not) in 1st/2nd interview.

Source/details: http://www.stats.ox.ac.uk/~steffen/teaching/fsmHT07/fsm407c.pdf

Using EM with incomplete survey data (two-way tables)

Each respondent is polled twice, six months apart about a topic: in this case, whether they have been the victim of a crime.

	Second Interview		
First Interview	Crime-Free	Victims	Nonrespondents
Crime-Free	392	55	33
Victims	76	38	9
Nonrespondents	31	7	115

We have two binary variables *I* and *J*
I = crime-free first interview
J = crime-free second interview

$n_{ij}^{(1)}$ complete observations for both *I* and *J*
$n_{ij}^{(2)}$ observations only for *I*
$n_{ij}^{(3)}$ observations only for *J.*

Where is the missing data here?

- Nonrespondents in either first or second interview.

What are the parameters we want to estimate?

- p_{ij} : probability of the conditions on I, J
- e.g. fraction of respondents that reported crime victim status in second interview.

crimefree@1 and crimefree@2: p_{00} = 0.69
crimefree@1 and victim@2: p_{01} = 0.09
victim@1 and crimefree@2: p_{10} = 0.13
victim@1 and victim@2 p_{11} = 0.06

After 5-10 iterations

$$p_{ij} = \left(n_{ij}^{(1)} + n_{ij}^{(2*)} + n_{ij}^{(3*)}\right)/n$$

Source/details: http://www.stats.ox.ac.uk/~steffen/teaching/fsmHT07/fsm407c.pdf

Level 2 strategy: Multivariate imputation in scikit-learn

- impute.IterativeImputer
- Can model each feature with missing values as a function of other features (columns) and use that estimate.
- Iterative round-robin:
  1. Pick a feature column to be output *y.*
  2. The other feature columns are the input *X.*
  3. Fit a regressor R on (X, y) for known values of *y.*
  4. Use the trained R to predict the missing values of *y.*
  5. Go back to step 1 with a different feature column until max_iter rounds are complete.

	maxO3	T9	T12	T15	Ne9
0601	NA	15.6	18.5	18.4	4
0602	82	17	18.4	17.7	5
0603	92	NA	17.6	19.5	2
0604	114	16.2	NA	NA	1
0605	94	17.4	20.5	NA	8
0606	80	17.7	NA	18.3	NA
0607	NA	16.8	15.6	14.9	7
0610	79	14.9	17.5	18.9	5
0611	101	NA	19.6	21.4	2
0612	NA	18.3	21.9	22.9	5
0613	101	17.3	19.3	20.2	NA
⋮	⋮	⋮	⋮	⋮	⋮
0919	NA	14.8	16.3	15.9	7
0920	71	15.5	18	17.4	7
0921	96	NA	NA	NA	3
0922	98	NA	NA	NA	2
0923	92	14.7	17.6	18.2	1
0924	NA	13.3	17.7	17.7	NA
0925	84	13.3	17.7	17.8	3
0927	NA	16.2	20.8	22.1	6
0928	99	16.9	23	22.6	NA
0929	NA	16.9	19.8	22.1	6
0930	70	15.7	18.6	20.7	NA

Level 2 strategy: Multivariate imputation in scikit-learn

- impute.IterativeImputer
- Can model each feature with missing values as a function of other features (columns) and use that estimate.
- Iterative round-robin:
  1. Pick a feature column to be output *y.*
  2. The other feature columns are the input *X.*
  3. Fit a regressor R on (X, y) for known values of *y.*
  4. Use the trained R to predict the missing values of *y.*
  5. Go back to step 1 with a different feature column until max_iter rounds are complete.

	maxO3	T9	T12	T15	Ne9
0601	NA	15.6	18.5	18.4	4
0602	82	17	18.4	17.7	5
0603	92	NA	17.6	19.5	2
0604	114	16.2	NA	NA	1
0605	94	17.4	20.5	NA	8
0606	80	17.7	NA	18.3	NA
0607	NA	16.8	15.6	14.9	7
0610	79	14.9	17.5	18.9	5
0611	101	NA	19.6	21.4	2
0612	NA	18.3	21.9	22.9	5
0613	101	17.3	19.3	20.2	NA
⋮	⋮	⋮	⋮	⋮	⋮
0919	NA	14.8	16.3	15.9	7
0920	71	15.5	18	17.4	7
0921	96	NA	NA	NA	3
0922	98	NA	NA	NA	2
0923	92	14.7	17.6	18.2	1
0924	NA	13.3	17.7	17.7	NA
0925	84	13.3	17.7	17.8	3
0927	NA	16.2	20.8	22.1	6
0928	99	16.9	23	22.6	NA
0929	NA	16.9	19.8	22.1	6
0930	70	15.7	18.6	20.7	NA

training

y X

Level 2 strategy: Multivariate imputation in scikit-learn

- impute.IterativeImputer
- Can model each feature with missing values as a function of other features (columns) and use that estimate.
- Iterative round-robin:
  1. Pick a feature column to be output *y.*
  2. The other feature columns are the input *X.*
  3. Fit a regressor R on (X, y) for known values of *y.*
  4. Use the trained R to predict the missing values of *y.*
  5. Go back to step 1 with a different feature column until max_iter rounds are complete.

	maxO3	T9	T12	T15	Ne9
0601	NA	15.6	18.5	18.4	4
0602	82	17	18.4	17.7	5
0603	92	NA	17.6	19.5	2
0604	114	16.2	NA	NA	1
0605	94	17.4	20.5	NA	8
0606	80	17.7	NA	18.3	NA
0607	NA	16.8	15.6	14.9	7
0610	79	14.9	17.5	18.9	5
0611	101	NA	19.6	21.4	2
0612	NA	18.3	21.9	22.9	5
0613	101	17.3	19.3	20.2	NA
⋮	⋮	⋮	⋮	⋮	⋮
0919	NA	14.8	16.3	15.9	7
0920	71	15.5	18	17.4	7
0921	96	NA	NA	NA	3
0922	98	NA	NA	NA	2
0923	92	14.7	17.6	18.2	1
0924	NA	13.3	17.7	17.7	NA
0925	84	13.3	17.7	17.8	3
0927	NA	16.2	20.8	22.1	6
0928	99	16.9	23	22.6	NA
0929	NA	16.9	19.8	22.1	6
0930	70	15.7	18.6	20.7	NA

prediction

y X

scikit-learn example of IterativeImputer

- You need at least scikit-learn version (0.22), still experimental.

```
from sklearn.experimental import enable_iterative_imputer  # IMPORTANT: you need this extra flag!
from sklearn.impute import IterativeImputer
```

```
def get_impute_iterative(regressor, X_missing, y_missing):
    imputer = IterativeImputer(missing_values=np.nan, add_indicator=True,
                               random_state=0, n_nearest_features=5,
                               sample_posterior=True)
    iterative_impute_scores = get_scores_using_imputer(regressor, imputer,
                                                       X_missing,
                                                       y_missing)
    return iterative_impute_scores.mean(), iterative_impute_scores.std()

mses_dataset[3], stds_dataset[3] = get_impute_iterative(regressor, X_missing, y_missing)
```

- `n_nearest_features`: # of other features to use to impute missing values. Drawn randomly with probability proportional to absolute correlation
- `add_indicator`: add MissingIndicator to flag columns with missing values (just in case it's useful to downstream estimator)
- `estimator`: USEFUL! (See next slide) You can pass in a custom estimator, e.g. random forest regressor.

Passing a custom estimator to IterativeImputer using the estimator property

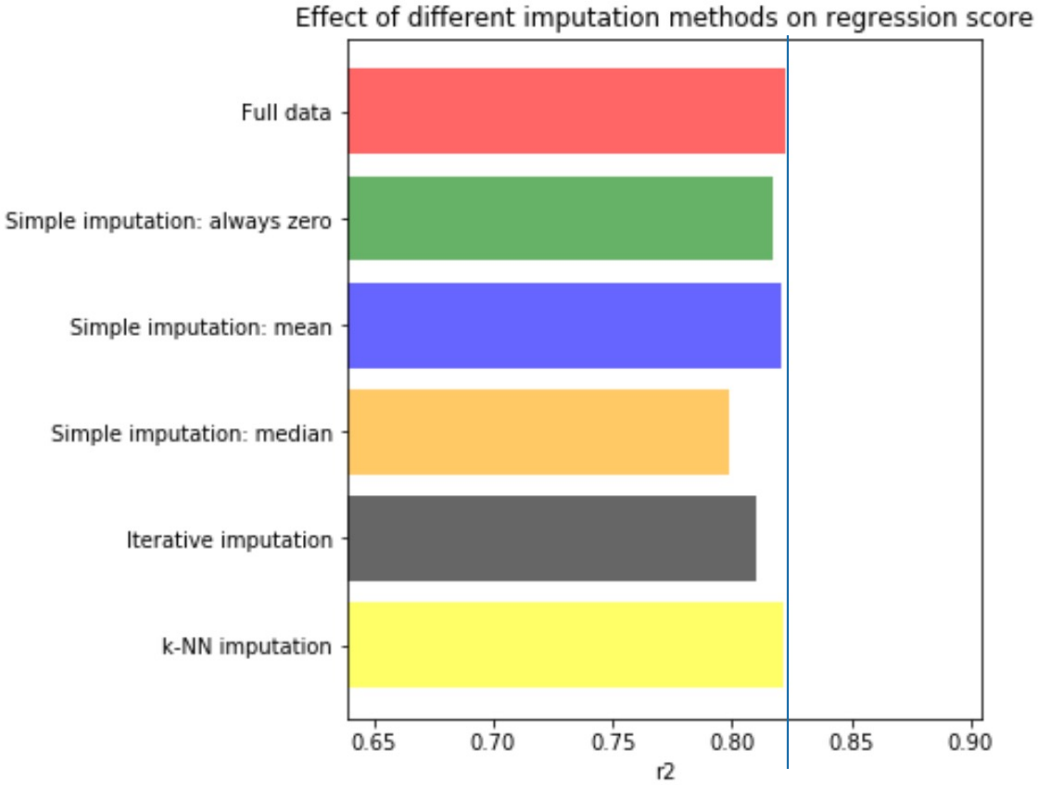
```
from sklearn.experimental import enable_iterative_imputer  # noq
from sklearn.impute import IterativeImputer

from sklearn.pipeline import make_pipeline
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import Ridge
from sklearn.tree import DecisionTreeRegressor

r_estimator = Ridge()
estimator = make_pipeline(
    IterativeImputer(random_state=42,
                     estimator=DecisionTreeRegressor(max_features = 'sqrt', random_state=42)),
    r_estimator

cvResults = cross_val_score(
            estimator, X_missing, y_missing, scoring='neg_mean_squared_error',
            cv=CV_SPLITS)
```

See notebook for sample imputer code comparing different types



Summary

- Option 1: Complete case analysis.
  - Consider only doing this if <5% missing.
  - If you do this, build a model of the complete vs incomplete cases and check for systematic differences.
- Option 2: Data wrangling software.
- Option 3: Some learning algorithms can deal with missing values.
- Option 4: Univariate/multivariate imputation
  - scikit-learn: SimpleImputer, KNNImputer, IterativeImputer, etc.

Kevyn Collins-Thompson

kevynct@umich.edu

School of Information

Unsupervised learning: Course conclusion

Kevyn Collins-Thompson

Associate Professor of Information and Computer Science
School of Information, University of Michigan

Unsupervised learning: Course review

1. Transforms
    - Dimensionality Reduction
        - Feature extraction: SVD and PCA
        - Biplots and PCA variants
        - Manifold learning
            - Distance-preserving (MDS)
            - Neighborhood-preserving (t-SNE, UMAP)
    - Density Estimation
        - Non-parametric (Histograms, kernel density)
        - Parametric: Gaussian Mixture Model

Clustering

2. Finding structure: Clustering
    - k-means
    - Heirarchical
    - Neighbor-based: DBSCAN
    - Evaluating and labeling clusters

Finding latent structure

3a. The Expectation-Maximization algorithm

3b. Transforms: Text processing

- Vectorizers
- Typical text pipeline steps

3c. Finding structure: Topic modeling

- Non-probabilistic (LSI, NMF)
- Probabilistic (LDA)

3d. Transforms: Word embeddings

Blending supervised and unsupervised learning

4a. Semi-supervised learning
- Label propagation

4b. Self-supervised learning
- Pretext tasks
- Transfer learning

4c. Unsupervised preprocessing for supervised learning
- Detecting data drift/covariate shift
- Feature extraction (and more..)

4d. Data imputation
- Univariate and multivariate imputation

Deeper ideas

- Many algorithms can be seen as different kinds of constrained matrix factorization.
- The “kernel trick” can be applied to unsupervised learning methods to find non-linear structure.
- Unsupervised learning can be used as a form of regularization.

What you should know

- Be able to correctly apply and interpret results from clustering methods in scikit-learn, including k-means, agglomerative clustering, hierarchical clustering, and DBSCAN.
- Be able to correctly apply and interpret results from manifold learning methods, including multidimensional scaling and t-SNE.
- Understand how to evaluate clustering results using a variety of metrics.
- Be able to correctly prepare text for processing by scikit-learn.
- Understand the use of topic modeling and best practices for its application.

What you should know

- Understand the tradeoffs and assumptions inherent in different clustering techniques.
- Understand how unsupervised learning can be used to improve supervised prediction.
- Know how to perform density estimation using a kernel, with a single random variable.
- Know how to interpret a biplot result from principal components analysis (PCA).

What you should be aware of

- The basic mechanism and use of word embeddings (in preparation for later coverage in deep learning and natural language processing).
- The EM algorithm: what it does, how and why it’s used, and how it relates to clustering.
- Advanced methods like label propagation for semi-supervised learning and Kernel PCA for non-linear dimensionality reduction.
- Self-supervised learning and its importance in finding rich feature representations.
- Issues around cluster quality and interpretation

Next steps in the MADS degree

- Machine Learning Pipelines: unsupervised learning provides a powerful set of tools that can be used before, during, or after the data processing in a machine learning pipeline.
- Deep Learning: prepares you for more insight into embeddings and latent variable representations.
- Applied Natural Language processing.
- Milestone II project: data exploration, transformation, assist with prediction.

Kevyn Collins-Thompson

kevynct@umich.edu

School of Information