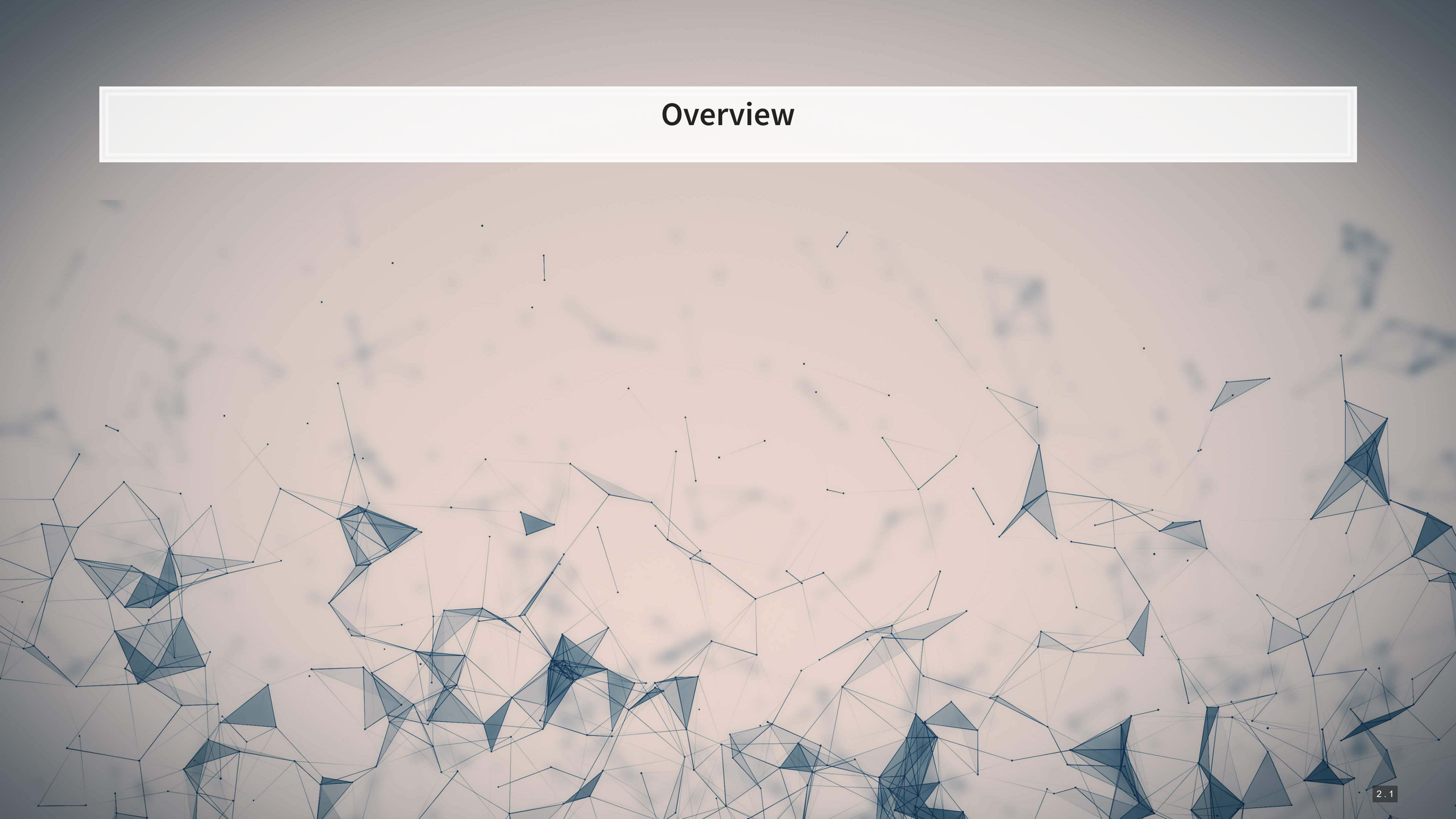
ML for SS: Ensembling and Clustering

Session 3

Dr. Richard M. Crowley
rcrowley@smu.edu.sg
http://rmc.link/

Overview



Papers

Paper 1: Easton et al. 2020

- Introduces a well motivated use for clustering
- Takes a standard approach to the introduction of a new technique
- The points the paper makes are applicable broadly in any archival/empirical discipline

Paper 2: Qiu, Xie and Jun (2020 working)

- A fairly straightforward paper introducing the concept of ensembling

Technical Discussion: Ensembling

Python

- Rolling your own is pretty doable
- `sklearn` is the primary tool for constructing them in python

R

- Rolling your own is pretty doable
- There are some packages for automating ensemble construction:
  - `SuperLearner`
  - `EnsembleML`

> Python is generally a bit stronger for these topics.

> There is a fully worked out solution for using python, data and pretrained models are on eLearn.

Main application: Ensembling

- Idea: Predict instances of intentional misreporting?
- Testing: Predicting 10-K/A irregularities using finance, textual style, and topics

Dependent Variable

Intentional misreporting as stated in 10-K/A filings

Independent Variables

- 17 Financial measures
- 20 Style characteristics
- 31 10-K discussion topics

This test mirrors a subset of Brown, Crowley and Elliott (2020 JAR)

We will **combine** the models from the past two weeks

Technical Discussion: Clustering

Python

- `sklearn` is still good for this
  - k-means and KNN
  - t-SNE
- umap-learn for UMAP

R

- For standard clustering, `caret` is a good choice
- For t-SNE, `Rtsne` works well
- For UMAP, `umap` works

Python is generally a bit stronger for these topics.

There is a fully worked out solution for using python, data is on elearn

Main application: Clustering

- Idea: Industry classification based on the text of annual reports

Dependent Variable

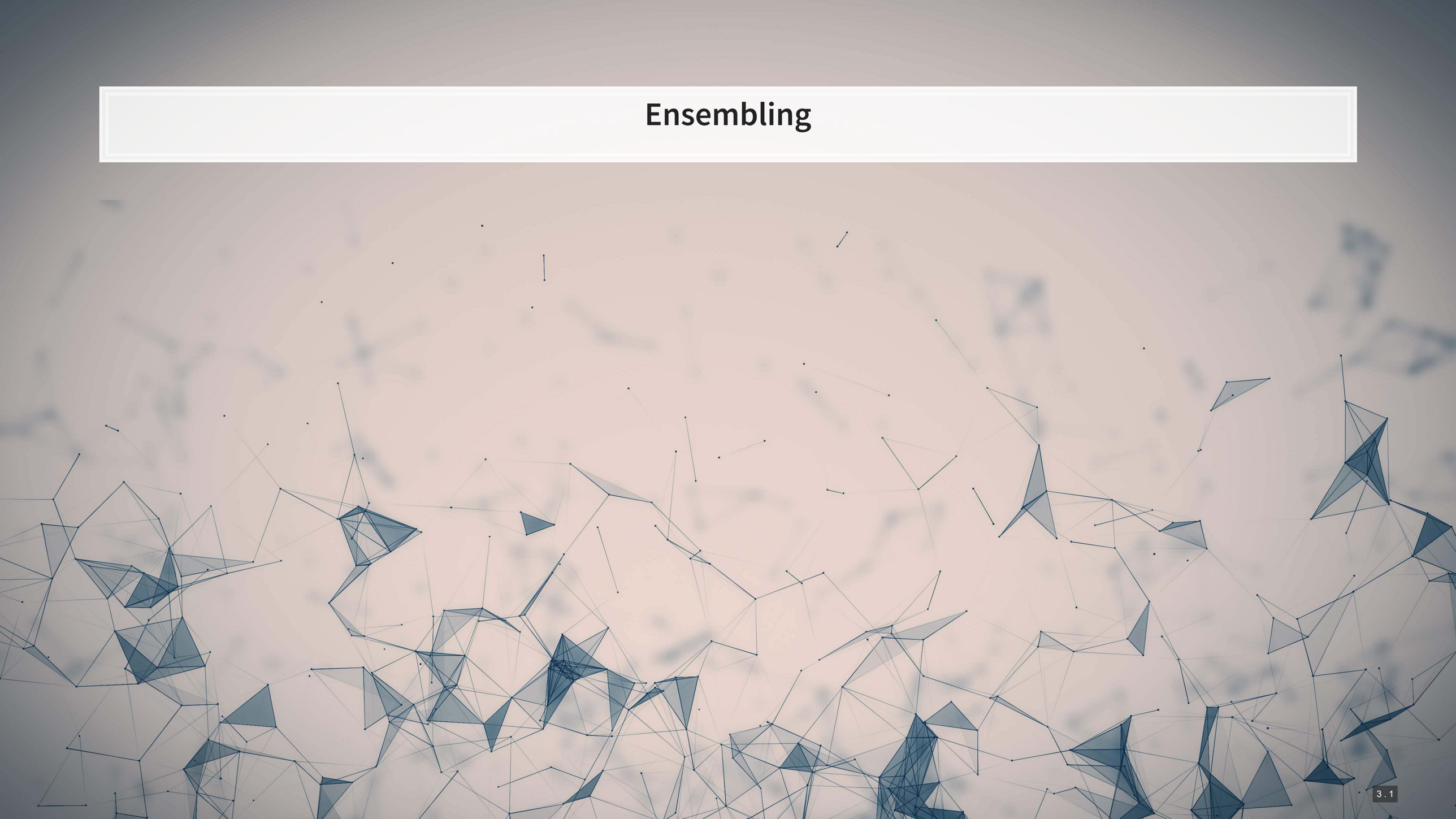
SIC Codes

Independent Variables

- 31 10-K discussion topics

> Somewhat in the vein of Hoberg and Phillips (2016 JPE), though less precise

Ensembling



What are ensembles?

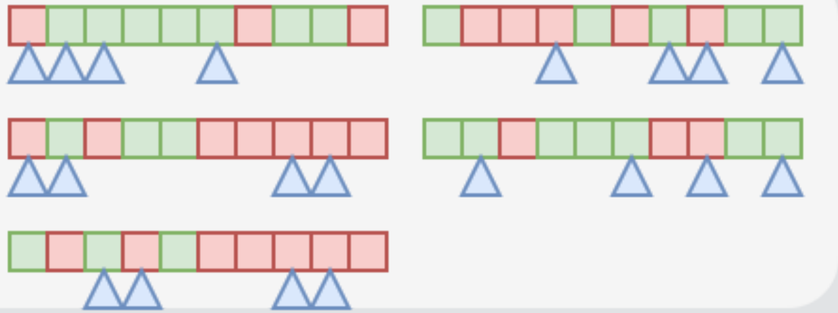
- Ensembles are models made out of models
- Ex.: You train 3 models using different techniques, and each seems to work well in certain cases and poorly in others
  - If you use the models in isolation, then any of them would do an OK (but not great) job
  - If you make a model using all three, you can get better performance if their strengths all shine through
- Ensembles range from simple to complex
  - Simple: a (weighted) average of a few model's predictions

When are ensembles useful?

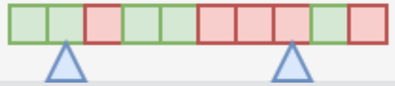
1. You have multiple models that are all decent, but none are great
   - And, ideally, the models' predictions are not highly correlated

Suppose this is the vector to be predicted

Each of these is 60% accurate, and the average correlation between them is 17%
(Errors are marked by blue triangles)



The most voted has 80% accuracy



When are ensembles useful?

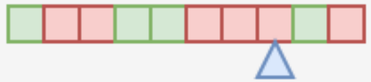
2. You have a really good model and a bunch of mediocre models
   - And, ideally the mediocre models are not highly correlated

Suppose this is the vector to be predicted

The first model is 80% accurate, the others are 60% accurate and 32% correlated

(Errors are marked by blue triangles)

Requiring unanimity to overpower the great model: 90%

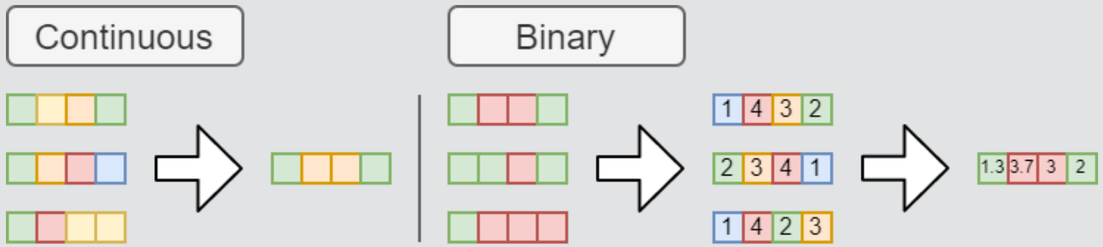


When are ensembles useful?

3. You really need to get just a bit more accuracy/less error out of the model, and you have some other models lying around
4. You want a more stable model
   - It helps to stabilize predictions by limiting the effect of errors or outliers produced by any one model on your prediction
   - Think: Diversification

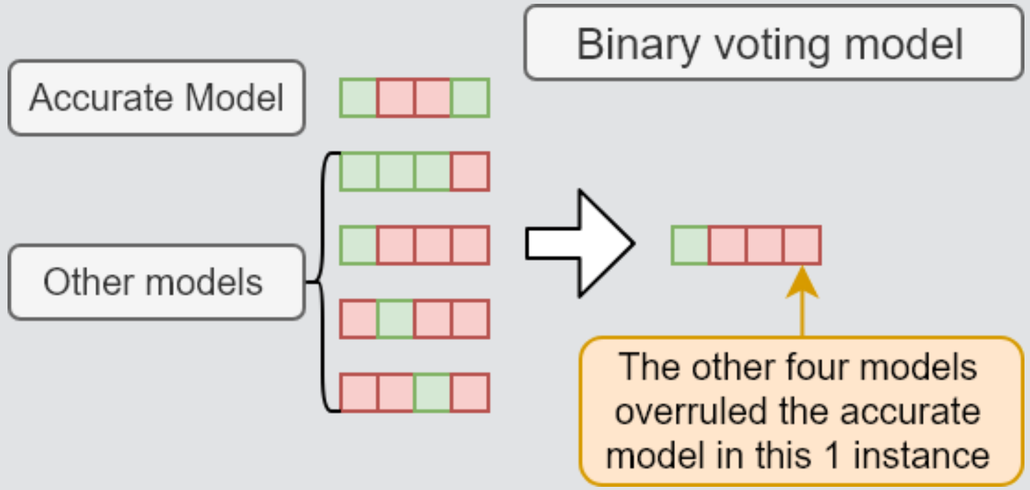
A simple ensemble (averaging)

- For continuous predictions, simple averaging is viable
  - Often you may want to weight the best model a bit higher
- For binary or categorical predictions, consider averaging *ranks*
  - i.e., instead of using a probability from a logit, use ranks 1, 2, 3, etc.
  - Ranks average a bit better, as scores on binary models (particularly when evaluated with measures like AUC) can have extremely different variances across models
    - In which case the ensemble is really just the most volatile model's prediction…
      - Not much of an ensemble



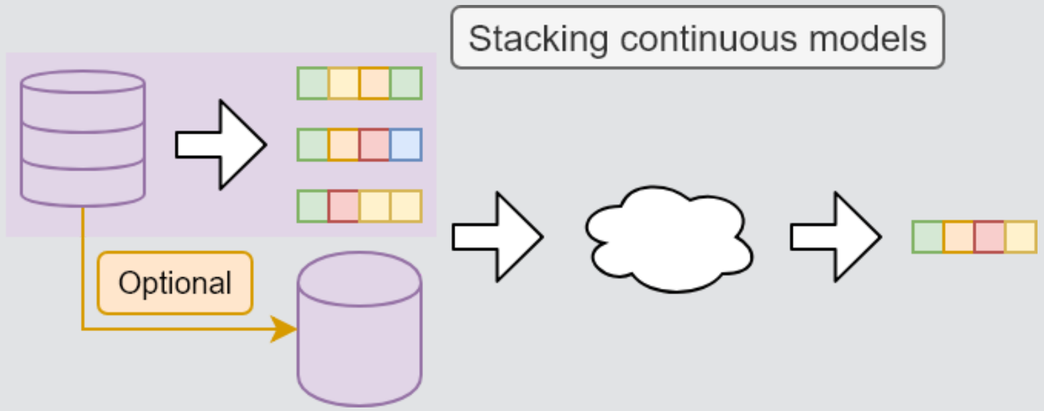
A more complex ensemble (voting model)

- If you have a model the is very good at predicting a binary outcome, ensembling can still help
  - This is particularly true when you have other models that capture different aspects of the problem
- Let the other models vote against the best model, and use their prediction if they are above some threshold of agreement



A lot more complex ensemble

- Stacking models (2 layers)
  1. Train models on subsets (folds) of the training data
  2. Make predictions for each model on the folds it wasn't applied to
  3. Train a new model that takes those predictions as inputs (and optionally the data set as well)
- Blending (similar to stacking)
  - Like stacking, but using predictions on a hold out sample instead of folds (and thus all models are using the same data for predictions)



A simple averaging ensemble of our models

```
test_X_ens =  pd.DataFrame({'XGBoost':   models['XGBoost'].predict_proba(models['test_X_ML'])[:,1],
                            'SVC': logistic(models['SVC'].decision_function(models['test_X_ML'])),
                            'ElasticNet':        models['ElasticNet'].predict_proba(models['test_X_ML'])[:,1],
                            'LASSO':     models['LASSO'].predict_proba(models['test_X_ML'])[:,1],
                            'logit':  models['logit'].predict(models['test_pd'][models['vars']])})

rank_X_ens = test_X_ens.rank()
arank_X_ens = rank_X_ens.XGBoost + rank_X_ens.SVC + rank_X_ens.ElasticNet + rank_X_ens.LASSO + rank_X_ens.logit
auc = metrics.roc_auc_score(models['test_pd'].Restate_Int, arank_X_ens)
fpr, tpr, thresholds = metrics.roc_curve(models['test_pd'].Restate_Int, arank_X_ens)
display = metrics.RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=auc)
display.plot()
```

Practicalities

- Methods like stacking or blending are much more complex than a simple averaging or voting based ensemble
  - But in practice they perform slightly better

> Recall the tradeoff between complexity and accuracy!

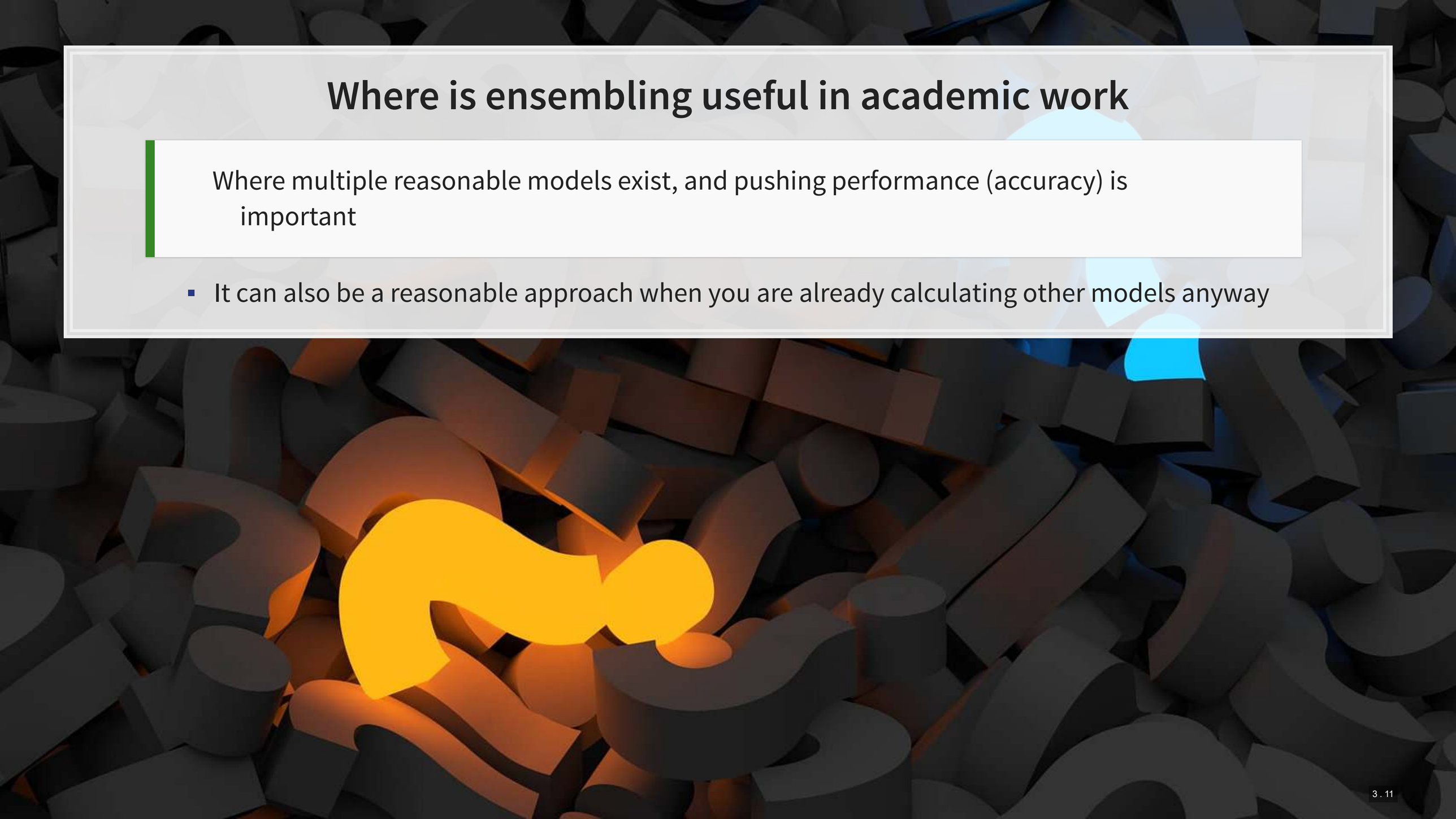
  - As such, we may not prefer the complex ensemble in practice, unless we only care about accuracy

> Example: In 2009, Netflix awarded a $1M prize to the BellKor's Pragmatic Chaos team for beating Netflix's own user preference algorithm by >10%. The alogorithm was so complex that Netflix never used it. It instead used a simpler algorithm with an 8% improvement.

Where is ensembling useful in academic work

> Where multiple reasonable models exist, and pushing performance (accuracy) is important

- It can also be a reasonable approach when you are already calculating other models anyway



[Geoff Hinton's] Dark knowledge

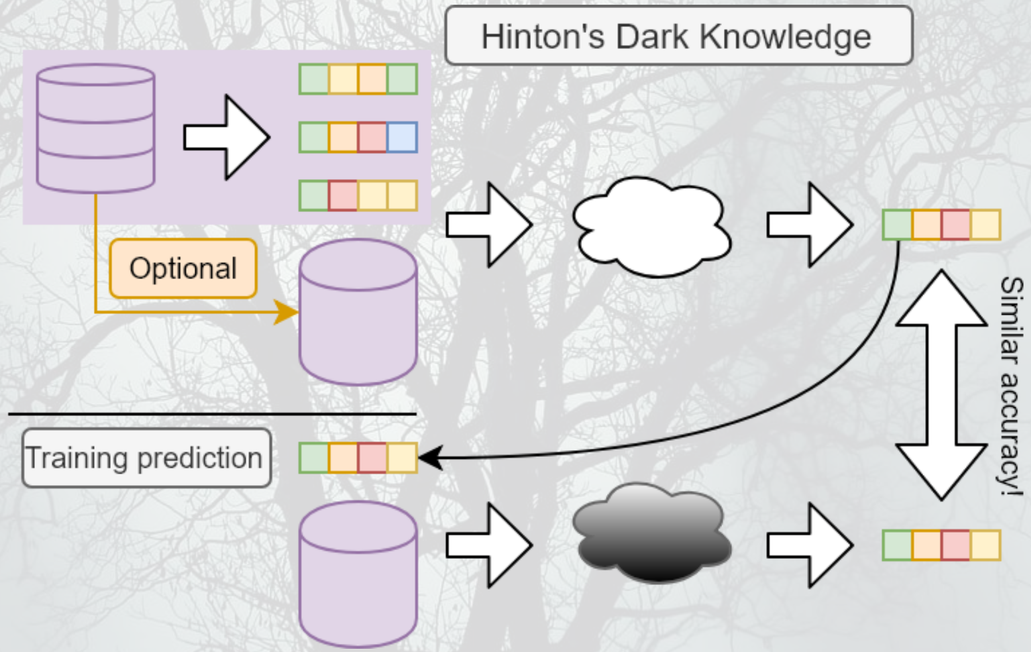
- Complex ensembles work well
- Complex ensembles are exceedingly computationally intensive
  - This is bad for running on small or constrained devices (like phones)

> Dark knowledge

- We can (almost) always create a simple model that approximates the complex model
  - Interpret the above literally – we can train a model to fit the model

Dark knowledge

- Train the simple model not on the actual DV from the training data, but on the best algorithm's (softened) prediction for the training data
- Somewhat surprisingly, this new, simple algorithm can work almost as well as the full thing!



An example of this dark knowledge

- Google's full model for interpreting human speech is >100GB
  - As of October 2019
- In Google's Pixel 4 phone, they have human speech interpretation running locally *on the phone*
  - Not in the cloud like it works on any other Android phone

> How did they do this?

  - They can approximate the output of the complex speech model using a 0.5GB model
  - 0.5GB isn't small, but it's small enough to run on a phone

Learning more about Ensembling

- Scikit-learn's documentation on ensemble methods it supports
- Geoff Hinton's Dark Knowledge slides
  - For more details on *dark knowledge*, applications, and the softening transform
  - His interesting (though highly technical) Reddit AMA
- A Kaggler's Guide to Model Stacking in Practice
  - A short guide on stacking with nice visualizations
- Kaggle Ensembling Guide
  - A comprehensive list of ensembling methods with some code samples and applications discussed
- Ensemble Learning to Improve Machine Learning Results
  - Nicely covers bagging and boosting (two other techniques)

> There are many ways to ensemble, and there is no specific guide as to what is best. It may prove useful in the group project, however.

Addendum: Using R

- There are a couple interesting packages in R for ensembling:
  - The `Superlearner` package aims to automate building ensembles
    - Think of it like an automated cross-validation for ensemble construction
  - The `EnsembleML` package allows you to specify an ensemble and train the underlying models together
- You can also roll your own ensemble as we did in the example earlier

Clustering: k-means

What is k-means?

$$\min_{C_k} \sum_{k=1}^{K} \sum_{x_i \in C_k} (x_i - \mu_k)^2$$

- Minimizes the sum of squared distance between points within groups
- Technically this is a machine learning algorithm, despite its simplicity
- You need to specify the number of groups you want

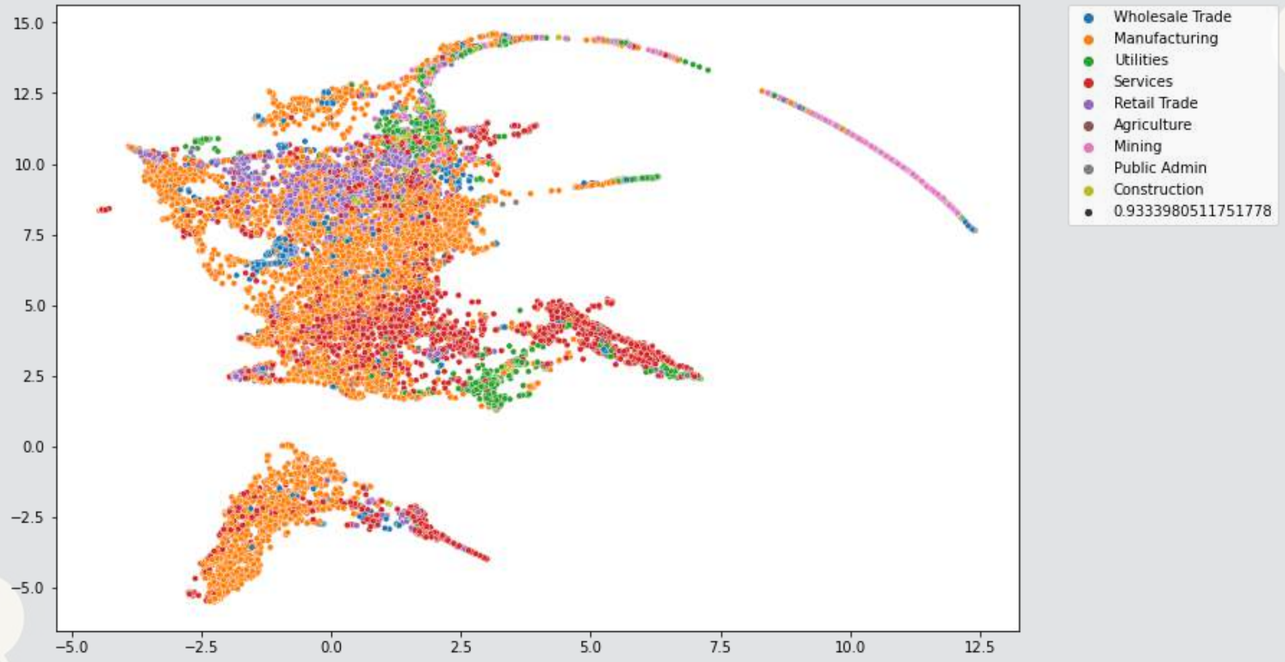
- Pros:
  - Very fast to run
  - Simple interpretation

- Cons
  - Simple algorithm
  - Need to specify k, the number of clusters

Since the algorithm is unsupervised, optimizing k can be tricky

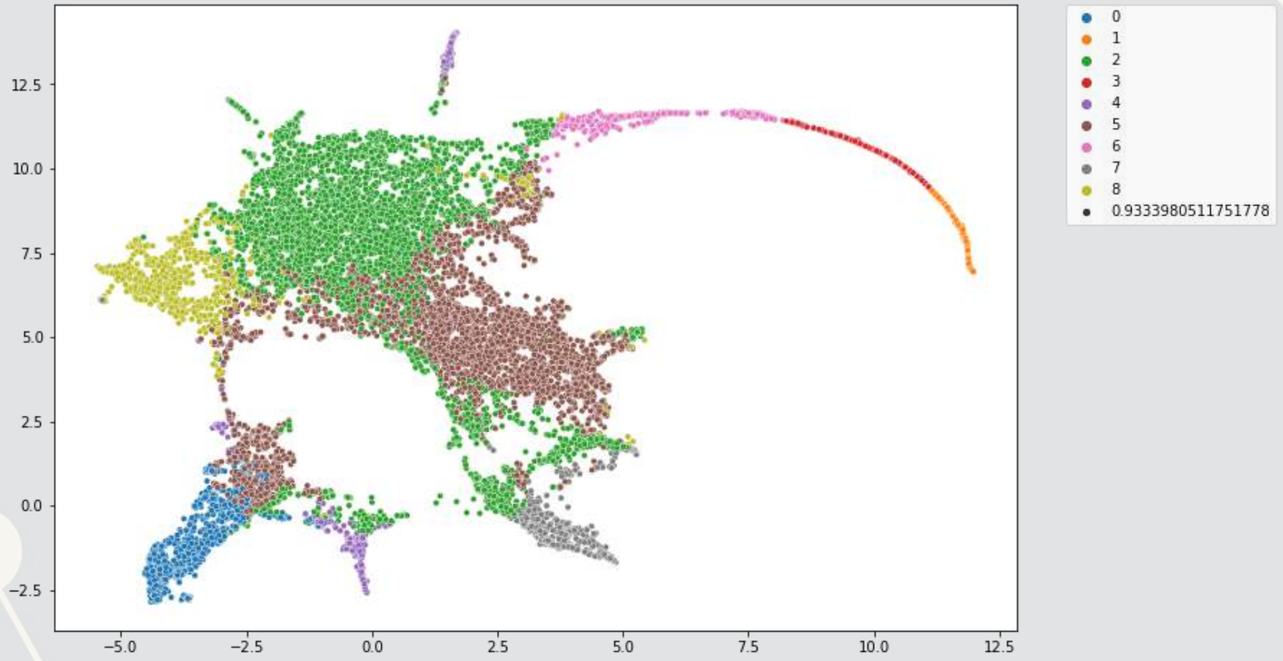
Projecting to 2D with UMAP

- Like last session, we will use UMAP to get a sense of how well topics line up with SIC industries



Projecting to 2D with UMAP

- It is also interesting to see how well the topics can be clustered
  - The below colors UMAP by a `k=9` kmeans algorithm applied to the LDA output



Why are these graphs different?

- Possibly due to…
  - Data: 10-K disclosure content doesn’t fully capture industry inclusion
  - Topic modeling: The measure may be noisy
  - SIC code: The measure doesn’t cleanly capture industry inclusion
    - Some firms are essentially misclassified
- Recall, SIC covers Agriculture, Forestry and Fishing; Mining; Construction; Manufacturing; Transportation, Communications, Electric, Gas, and Sanitary Services; Wholesale Trade; Retail Trade; Finance, Insurance, and Real Estate; Services; Public Administration

Optimizing K-means clustering

- K-means clustering is very fast to run, but suffers from the same issue as LDA:

> You need to specify the number of clusters!

  - Often times the solutions to this are similar to what we discussed for LDA
    - Hand tuning
    - In sample performance
  - However, there is a statistics-based, researcher-bias-free method

> The Gap Statistic

How does the Gap statistic work?

- Let…
  - k be the number of clusters,
  - B the number of simulated samples
  - W_k be the K-Means inertia score on actual data
  - $W^*_{k,r}$ be the K-Means inertia score for iteration r with synthetic data
  - $\bar{l}$ be the average of the $W^*_{k,r}$ s

$$Gap(k) = \left(\frac{1}{B}\right)\sum_{r=1}^{B}\log\left(W^*_{k,r}\right) - \log\left(W_k\right) \text{ and}$$

$$s_k = sd_k\sqrt{1+\frac{1}{B}},\ \text{where } sd_k = \sqrt{\left(\frac{1}{B}\right)\sum_{r=1}^{B}\left\{\log\left(W^*_{k,r} - \bar{l}\right)\right\}^2}$$

  - Select the lowest k such that $Gap(k) \geq Gap(k+1) - s_{k+1}$

> I.e., select the lowest k s.t. the log-scaled error removed by clustering on real data at k is no worse than 1 SD below the log-scaled error removed at $k+1$

Implementation in python

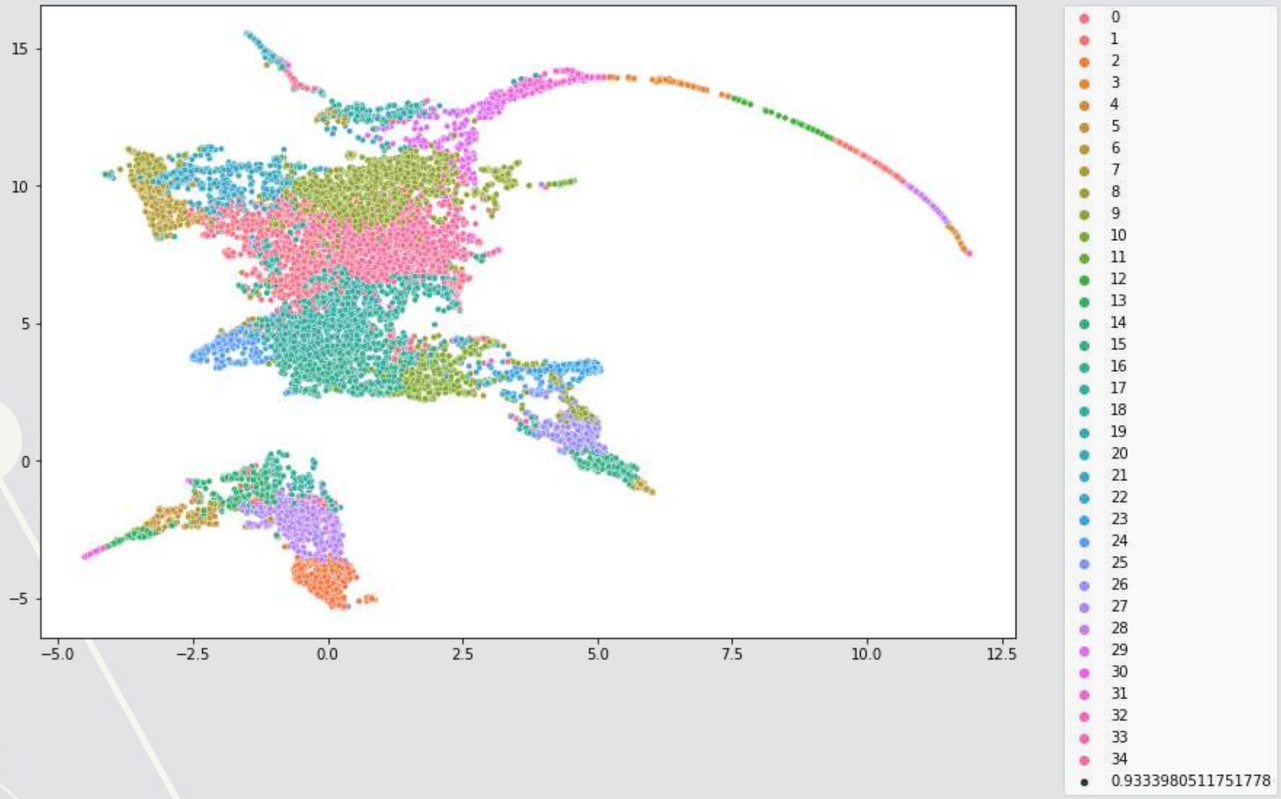
- The code is too long to put in the slides, but it is in the code file
- Sketch of the code:
  1. Iterate through k values starting at 2
  2. Determine performance (inertia) at k with real data
  3. Determine performance (inertia) at k with simulated (random) data 10 times
  4. Calculate the standard deviation of the log of performance on random data
  5. See if the 2x2 difference in log inertia between k and $k+1$ on real and random data is less than the standard deviation
     - If so, k is optimal, stop iterating
     - If not, $k = k + 1$ and start again

> $k = 35$ for the model presented here

Optimal clustering

```
model = cluster.KMeans(n_clusters=30)
kmeans = model.fit(df[topic_names])
df['cluster_opt'] = kmeans.labels_

umap_color(df[topic_names], df.cluster_opt.astype("category"))
```



Example companies in the optimized clusters

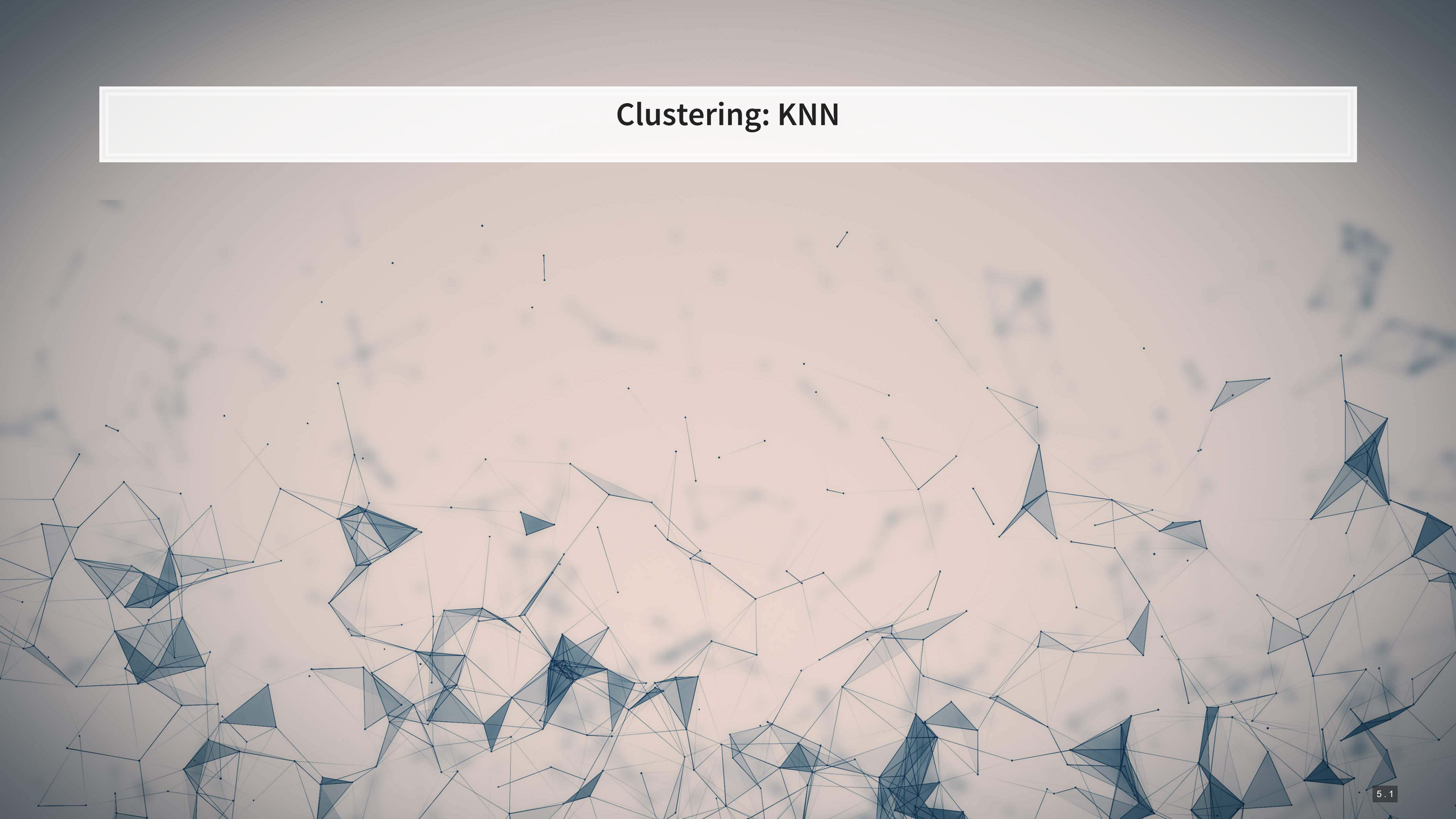
```python
df[df.cluster_opt==0][['industry']].sample(n=10)
```

```
##                industry
## 9770           Services
## 8622    Wholesale Trade
## 11401          Services
## 1882       Retail Trade
## 696       Manufacturing
## 222       Manufacturing
## 6752      Manufacturing
## 10218          Services
## 189       Manufacturing
## 2021       Retail Trade
```

```python
df[df.cluster_opt==2][['industry']].sample(n=10)
```

```
##              industry
## 13779  Manufacturing
## 7898   Manufacturing
## 9036   Manufacturing
## 1500         Services
## 12279  Manufacturing
## 12383  Manufacturing
## 5800   Manufacturing
## 11568  Manufacturing
## 7256   Manufacturing
## 7181   Manufacturing
```

Clustering: KNN



Using k-means for filling in data

- One possible approach we could use is to fill based on the category assigned by k-means
- However, as we saw, k-means and SIC code don’t line up perfectly…
  - So using this classification will definitely be noisy

A better approach with KNN

- KNN, or **K**-**N**earest **N**eighbors is a *supervised* approach to clustering
- Since we already have industry classifications for most of our data, we can use that structure to inform our assignment of the missing industry codes
- The way the model uses the information is by letting the nearest labeled points “vote” on what the point should be
  - Points are defined by 10-K content in our case
  - Voting can be weighted by distance or done uniformly

Implementing KNN

- Scikit-learn has a KNN implementation in its `neighbors` module
- The primary parameter in the model is `k`: how many points get to vote
  - `k` is `n_neighbors` in Scikit-learn

```
knn = neighbors.KNeighborsClassifier(n_neighbors=5)
knn.fit(df[topic_names], df['Industry'])
```

The above is sufficient to fit a simple model

Checking performance

- First, we need to get predictions
- Since this is a multiclass problem, we will not output probabilities, but instead the top guess

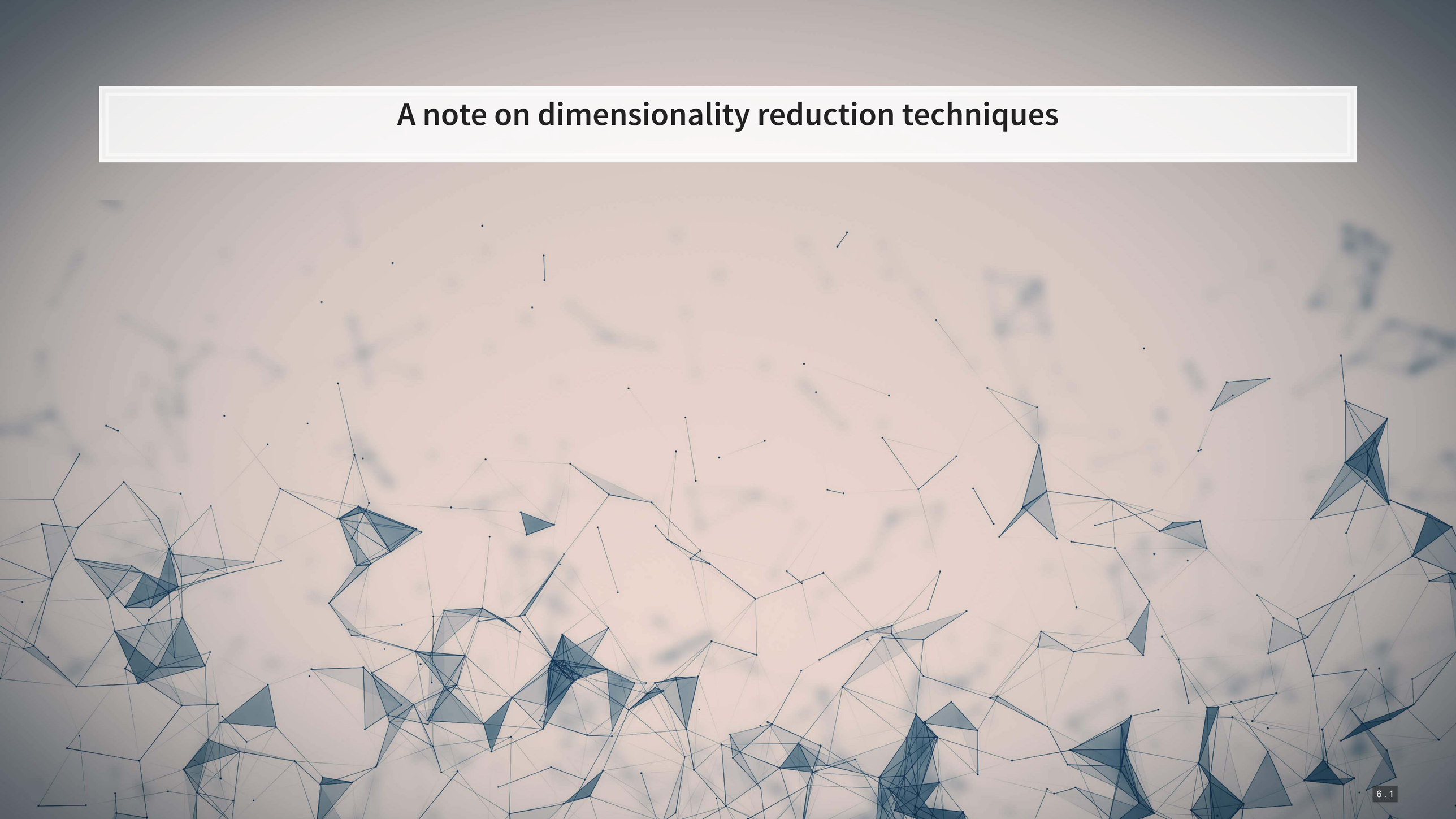
```
in_pred = knn.predict(df[topic_names])
out_pred = knn.predict(testing[topic_names])
```

- We can quickly check multiclass performance using Scikit-learn as well

```
print('In sample: {},\nOut of sample: {}'.format(
  metrics.accuracy_score(df['industry'], in_pred),
  metrics.accuracy_score(testing['industry'], out_pred)))
```

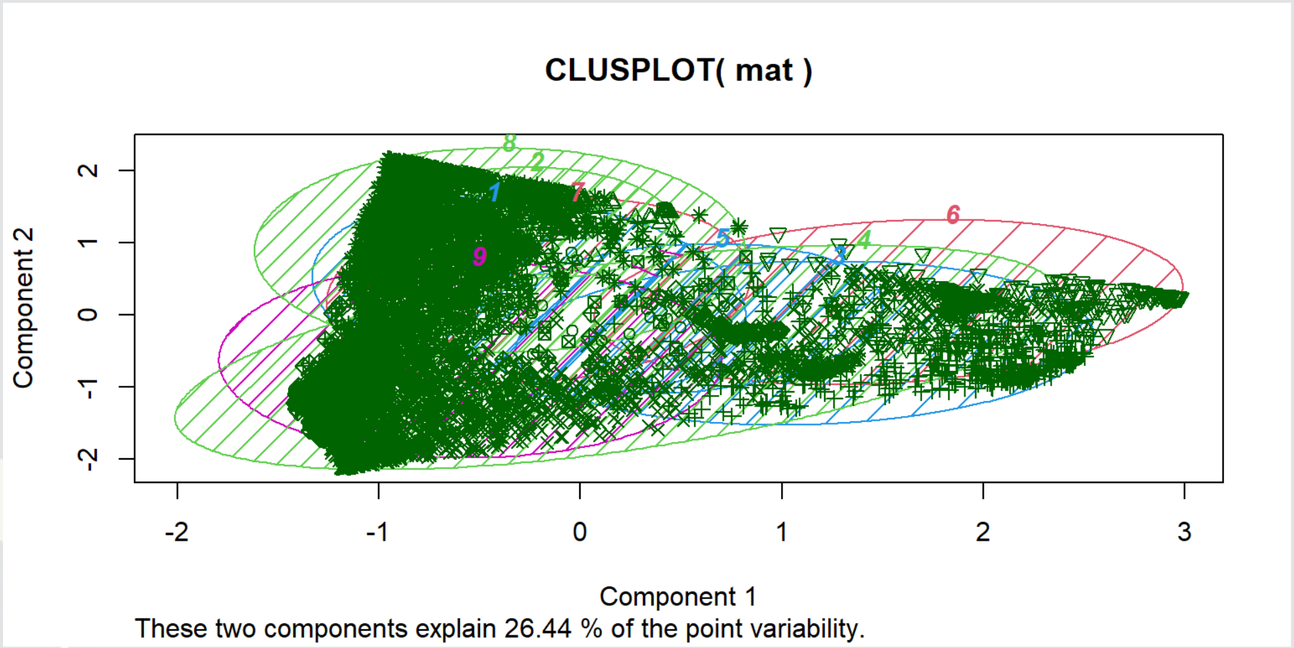
```
## In sample: 1.0,
## Out of sample: 0.922422954303932
```

A note on dimensionality reduction techniques



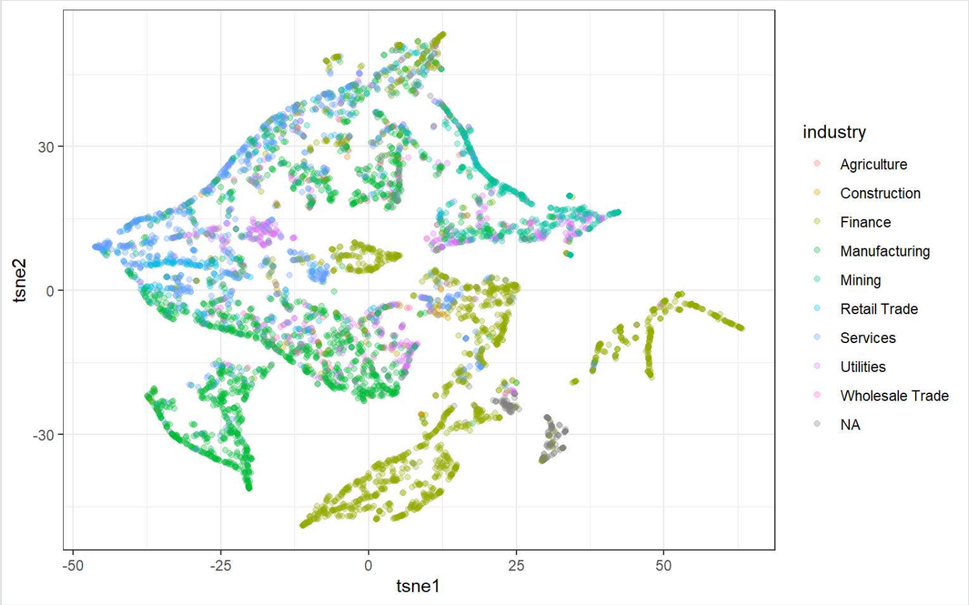
Principle Component Analysis

- PCA is a common technique to see in older studies
- It is reasonably efficient at identifying a lower dimensional representation of a relationship
- It is not good at maintaining relationships in the lower dimensional space



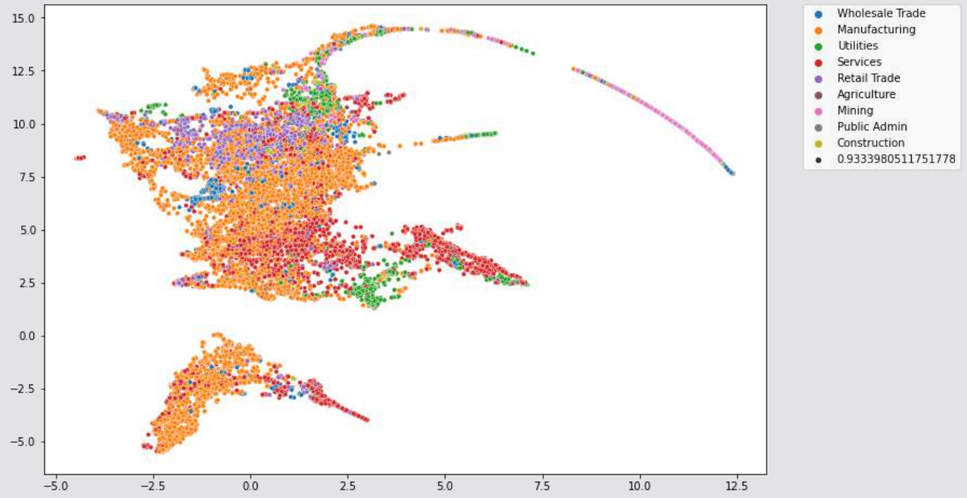
t-distributed Stochastic Neighbor Embedding

- t-SNE is focused on keeping distances relatively similar between the full dimensional input space and the projected output space
  - If 2 points are close to each other in N dimensions, they will be close to each other in 2 or 3 dimensions as well!
- t-SNE *does not* maintain distances over longer distances!
  - Should not be used as input to a regression

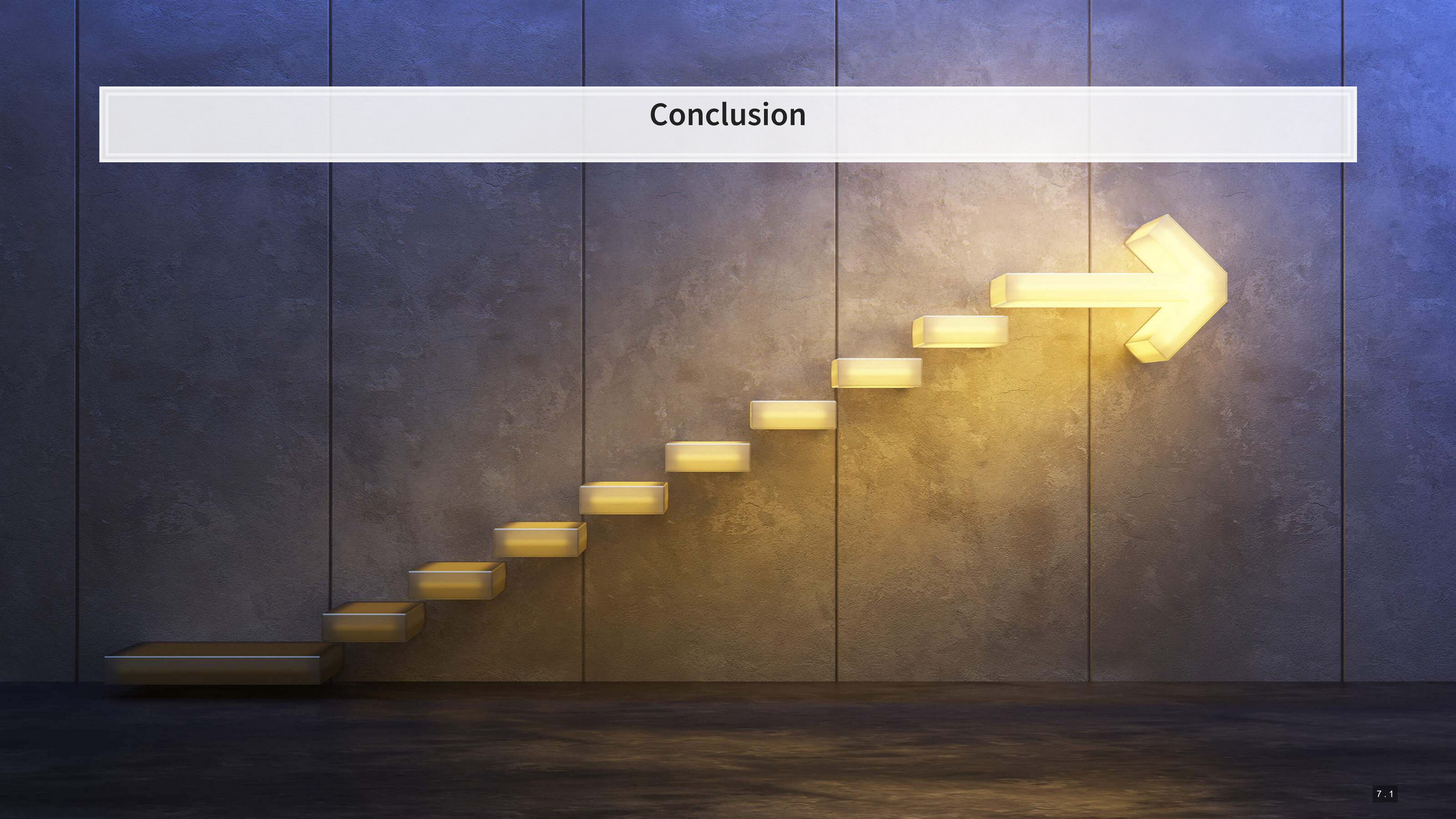


Uniform Manifold Approximation and Projection

- UMAP maintains local distances like t-SNE
- UMAP also maintains global distances, mostly
  - As such, it can be used for isolating data components for regression like PCA



Conclusion



Wrap-up

Ensembling

- Good for pushing forecasting ability
- Easy to do when you already made a bunch of models

Clustering

- Can be done as an unsupervised or supervised algorithm
- Can be used for dimensionality reduction
- Many possible uses

Packages used for these slides

Python

- matplotlib
- numpy
- pandas
- scikit-learn
- seaborn
- umap-learn

R

- caret
- cluster

References

- Easton, Peter D., Martin Kapons, Steven J. Monahan, Harm H. Schütt, and Eric H. Weisbrod. “Forecasting Earnings Using k-Nearest Neighbor Matching.” Available at SSRN (2020).
- Qiu, Yue, Tian Xie, and Y. U. Jun. “Forecast combinations in machine learning.” (2020).
- Wang, Bingling, Min-Bin Lin, and Wolfgang Karl Hardle, “Non-fungible Tokens & VizTech.” (2021).

Custom code

```python
# From umap.plot source code on Github
def _get_embedding(umap_object):
    if hasattr(umap_object, "embedding_"):
        return umap_object.embedding_
    elif hasattr(umap_object, "embedding"):
        return umap_object.embedding
    else:
        raise ValueError("Could not find embedding attribute of umap_object")


# Cut down version of umap.plot.points to remove dependencies on  datashader, bokeh, holoviews, scikit-image, and colorcet
# Introduces a dependency on seaborn though
def umap_color(data_map, data_color, cmap='viridis', subset=None, title=None):
    reducer = umap.UMAP()
    umap_object = reducer.fit(data_map)
    embed = _get_embedding(umap_object)

    if subset is not None:
        embed_X = embed[subset,0]
        embed_Y = embed[subset,1]
        data_color = np.array(data_color[subset])
    else:
        embed_X = embed[:, 0]
        embed_Y = embed[:, 1]

    point_size = 100.0 / np.sqrt(len(embed_X))

    # color by values
    fig, ax = plt.subplots(figsize=(12,8))
    g = sns.scatterplot(ax=ax, x=embed_X, y=embed_Y, hue=data_color, size=point_size)
    _ = plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
    return g
```