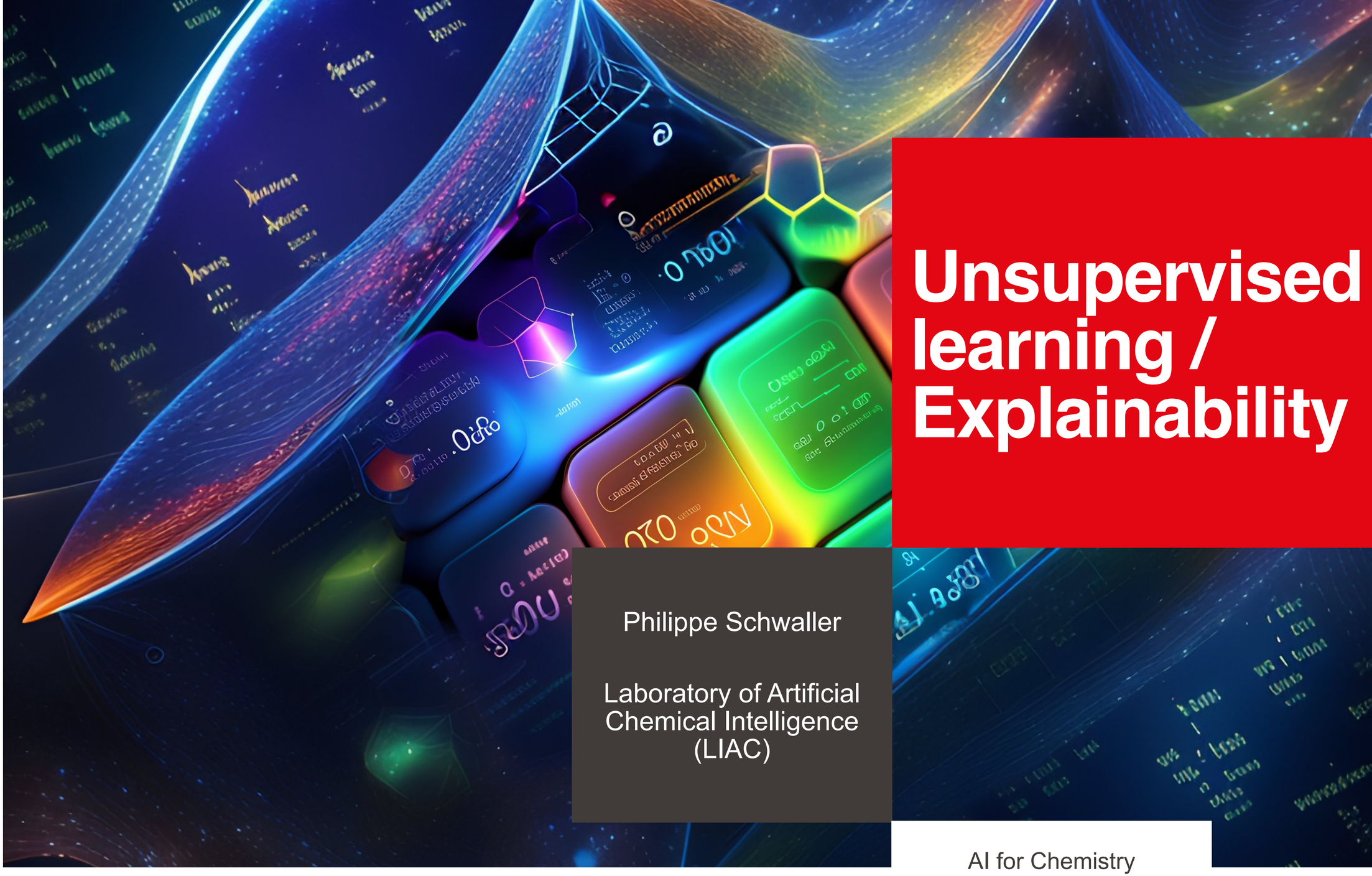


Unsupervised learning / Explainability

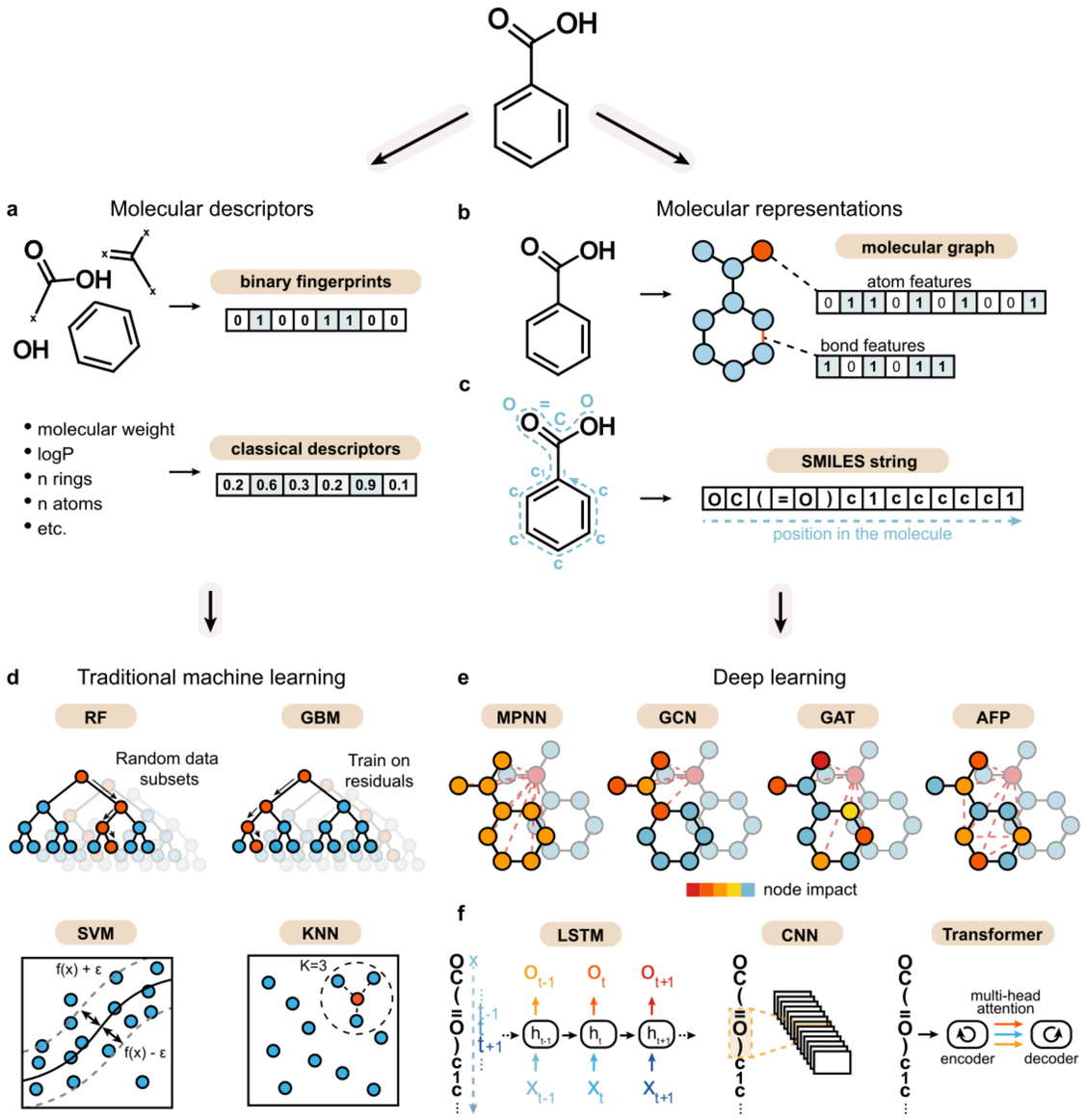
Philippe Schwaller

Laboratory of Artificial Chemical Intelligence (LIAC)

AI for Chemistry

https://image.lexica.art/full_jpg/566a2db6-7b59-4fc3-a38d-7608bc3009fb

Recap



scikit-learn
Machine Learning in Python

chemprop

PyTorch Lightning

Hugging Face

Exposing the Limitations of Molecular Machine Learning with Activity Cliffs

Derek van Tilborg, Alisa Alenicheva, and Francesca Grisoni*

Cite this: *J. Chem. Inf. Model.* 2022, 62, 23, 5938–5951
Publication Date: December 1, 2022
https://doi.org/10.1021/acs.jcim.2c01073

Article Views	Altmetric	Citations
7333	60	-

LEARN ABOUT THESE METRICS

Key concepts — PyTorch/Pytorch lightning

- Three things needed to train a deep learning model in PyTorch

1. **Data**: Methods to load your data and make it available for PyTorch to consume. This includes loading the data from disk or memory, performing any necessary preprocessing (e.g., converting NumPy arrays to PyTorch tensors), batching, shuffling, etc.
2. **Model**: A class that describes your model structure and forward pass. PyTorch makes it convenient by requiring you to define only the forward pass while relying on its built-in gradient tracking for the backward pass. However, if you need to implement a custom function, you'll need to define both forward and backward pass logic for that function. Most of the time, the functionalities you need are already implemented in PyTorch.
3. **Training Loop**: This is where everything is stitched together. The training loop involves loading data, using it to train the model, calculating the loss function, propagating back the gradients, and updating the model parameters. PyTorch's flexibility allows you to implement the training loop yourself, instead of relying on higher-level abstractions like `model.fit`. This hands-on approach deepens your understanding of deep learning.

- https://www.eletreby.me/blog/getting-started-with-pytorch-dataset-and-dataloader

Key concepts — Dataset and Loader

- The Dataset class defines your dataset and how to fetch a single row.
- The DataLoader class handles batching and shuffling

Toy Dataset: FashionMNIST

```python
from torchvision import datasets
from torchvision.transforms import ToTensor

# Download training data from open datasets.
training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor(),
)

# Download test data from open datasets.
test_data = datasets.FashionMNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor(),
)
```

```python
from torch.utils.data import DataLoader

batch_size = 64

# Create data loaders.
train_dataloader = DataLoader(training_data, batch_size=batch_size, shuffle=True)
test_dataloader = DataLoader(test_data, batch_size=batch_size)

for X, y in test_dataloader:
    print(f"Shape of X [N, C, H, W]: {X.shape}")
    print(f"Shape of y: {y.shape} {y.dtype}")
    break
```

Output:

```
Shape of X [N, C, H, W]: torch.Size([64, 1, 28, 28])
Shape of y: torch.Size([64]) torch.int64
```

More information: https://www.eletreby.me/blog/getting-started-with-pytorch-dataset-and-dataloader

Key concepts — PyTorch model

1. **Inherit from `torch.nn.Module`** : This is the base class for all neural network modules in PyTorch.
2. **Implement an `__init__` method**: This method defines the model's components, such as layers and activation functions.
3. **Implement a `forward` method**: This method defines the forward pass, specifying how input data propagates through the model components.

```python
import torch
from torch import nn

class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.l1 = nn.Linear(28 * 28, 512)
        self.l2 = nn.Linear(512, 512)
        self.l3 = nn.Linear(512, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.flatten(x)
        x = self.l1(x)
        x = self.relu(x)
        x = self.l2(x)
        x = self.relu(x)
        logits = self.l3(x)
        return logits

model = NeuralNetwork()
print(model)
```

```
NeuralNetwork(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (l1): Linear(in_features=784, out_features=512, bias=True)
  (l2): Linear(in_features=512, out_features=512, bias=True)
  (l3): Linear(in_features=512, out_features=10, bias=True)
  (relu): ReLU()
)
```

```python
total_params = sum(p.numel() for p in model.parameters())
print(f"Total parameters: {total_params}")
```

Output:

```
Total parameters: 669706
```

https://www.eletreby.me/blog/building-a-pytorch-model-class

Key concepts — Training loop


```python
def train(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    model.train()
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)

        # Compute prediction error
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        if batch % 100 == 0:
            loss, current = loss.item(), (batch + 1) * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")
```

1. **Cross-Entropy Loss** (`nn.CrossEntropyLoss`): Used for classification problems. It combines `nn.LogSoftmax` and `nn.NLLLoss` in a single class, making it ideal for multi-class classification tasks like our FashionMNIST example.
2. **Mean Squared Error** (`nn.MSELoss`): Used for regression problems. It measures the average squared difference between the estimated values and the actual value.
3. **Binary Cross-Entropy** (`nn.BCELoss`): Used for binary classification problems where each output is independently classified.
4. **L1 Loss** (`nn.L1Loss`): Also known as Mean Absolute Error, it's less sensitive to outliers than MSE.

```python
epochs = 5
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train(train_dataloader, model, loss_fn, optimizer)
    test(test_dataloader, model, loss_fn)
print("Done!")
```

```python
def test(dataloader, model, loss_fn):
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    model.eval()
    test_loss, correct = 0, 0
    with torch.no_grad():
        for X, y in dataloader:
            X, y = X.to(device), y.to(device)
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()
    test_loss /= num_batches
    correct /= size
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f}
```

Recommendation: Start with code that worked for a similar problem, and adapt it to your new dataset.

https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html

https://www.eletreby.me/blog/training-loop

PyTorch Lightning

Original PyTorch:

- Components like the model, training loop, and data loading are separate pieces
- You manually wrote functions for training and testing
- You explicitly managed device placement, optimization steps, and logging

PyTorch Lightning:

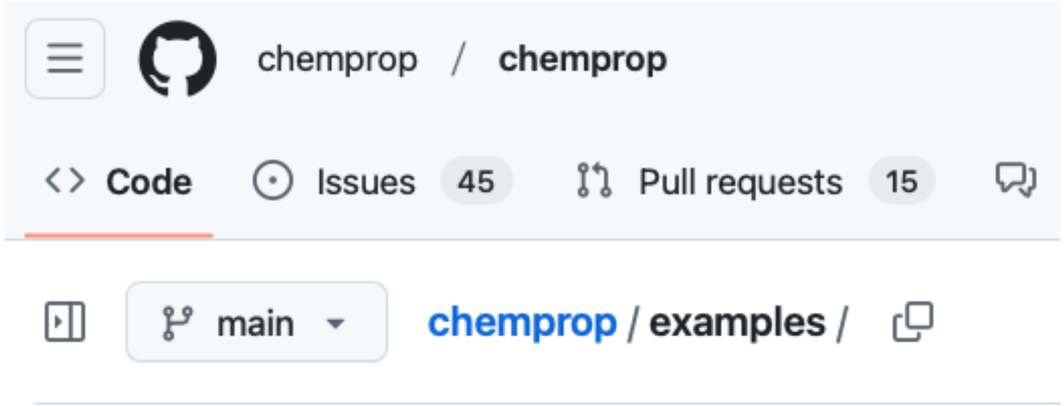
- Everything is organized into two main classes:
  - `LightningModule` (model + training logic)
  - `LightningDataModule` (data preparation and loading)
- The training loop is abstracted away by the `Trainer` class

1. **Less boilerplate code**: The Lightning version eliminates repetitive code patterns
2. **Better organization**: Clear separation of concerns between model, data, and training
3. **Built-in best practices**: Lightning implements ML engineering best practices by default
4. **Easier scaling**: The same code can run on CPU, single GPU, or multiple GPUs with minimal changes

```python
class FashionMNISTModel(pl.LightningModule):
    def __init__(self, learning_rate=1e-3):
        super().__init__()
        # Model architecture - same as your original model
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10)
        )
        self.learning_rate = learning_rate
        self.loss_fn = nn.CrossEntropyLoss()

    def forward(self, x):
        x = self.flatten(x)
        return self.linear_relu_stack(x)

    def training_step(self, batch, batch_idx):
        x, y = batch
        logits = self(x)
        loss = self.loss_fn(logits, y)
        self.log('train_loss', loss)
        return loss

    def validation_step(self, batch, batch_idx):
        x, y = batch
        logits = self(x)
        loss = self.loss_fn(logits, y)
        preds = torch.argmax(logits, dim=1)
        acc = (preds == y).float().mean()
        # Log metrics
        self.log('val_loss', loss, prog_bar=True)
        self.log('val_acc', acc, prog_bar=True)
        return loss

    def test_step(self, batch, batch_idx):
        # Similar to validation_step
        x, y = batch
        logits = self(x)
        loss = self.loss_fn(logits, y)
        preds = torch.argmax(logits, dim=1)
        acc = (preds == y).float().mean()
        self.log('test_loss', loss, prog_bar=True)
        self.log('test_acc', acc, prog_bar=True)
        return loss

    def configure_optimizers(self):
        return torch.optim.SGD(self.parameters(), lr=self.learning_rate
```

```python
pl.seed_everything(42)

# Initialize model and data
model = FashionMNISTModel()
data_module = FashionMNISTDataModule()

# Set up logger and checkpoint callback
checkpoint_callback = ModelCheckpoint(
    monitor='val_acc',
    mode='max',
    save_top_k=1,
    filename='best-checkpoint'
)

# Initialize trainer
trainer = pl.Trainer(
    max_epochs=5,
    accelerator='auto',  # Automatically use GPU if available
    callbacks=[checkpoint_callback],
)

# Train the model
trainer.fit(model, data_module)

# Test the model
trainer.test(model, data_module)
```

Chemprop



- mpnn_fingerprints.ipynb
- multi_task.ipynb
- predicting.ipynb
- predicting_regression_multicomponent.ipynb
- predicting_regression_reaction.ipynb
- rigr_featurizer.ipynb
- shapley_value_with_customized_featurizers.ipynb
- training.ipynb
- training_classification.ipynb
- training_regression_multicomponent.ipynb
- training_regression_reaction.ipynb

- https://github.com/chemprop/chemprop/tree/main/examples

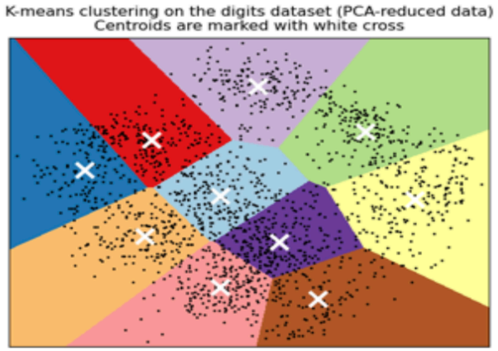
Unsupervised machine learning

Categories of unsupervised learning

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, and more...

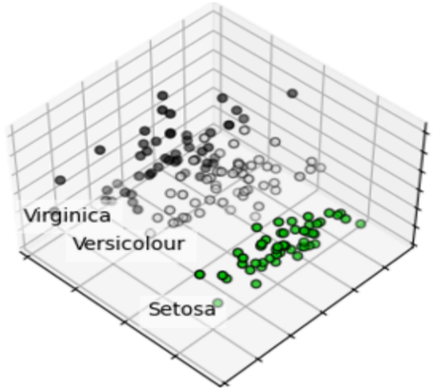


Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization, and more...

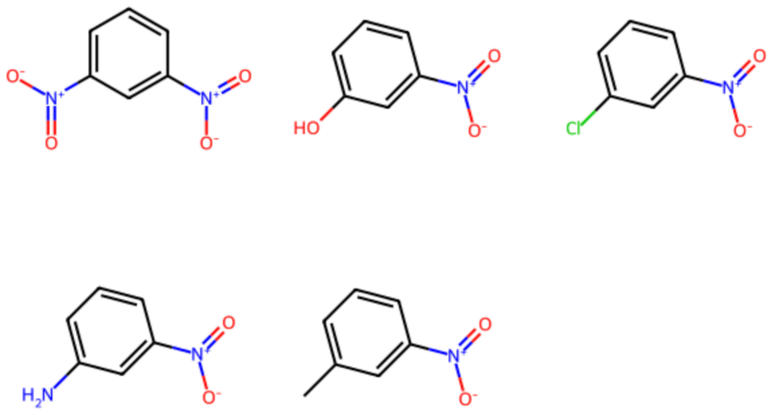


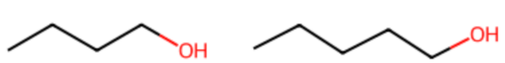
Examples

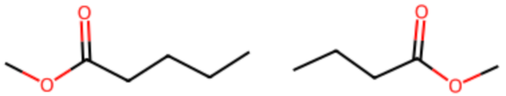
- https://scikit-learn.org/stable/

Clustering — what is it?

- Grouping unlabelled examples
- Typically you have a *large collection of molecules*, and you would like to divide them into *small groups of similar molecules* (clusters)
- This can help you *analyse outcomes* of high-throughput screening or virtual screening, but also *pick diverse molecules* (1 from each cluster)



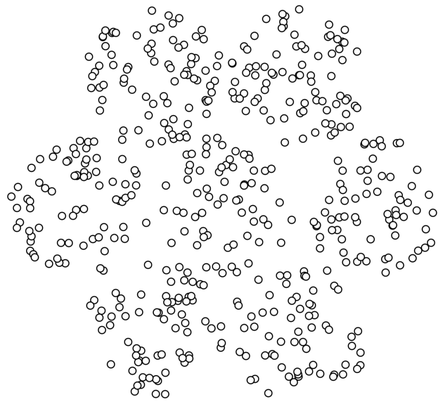




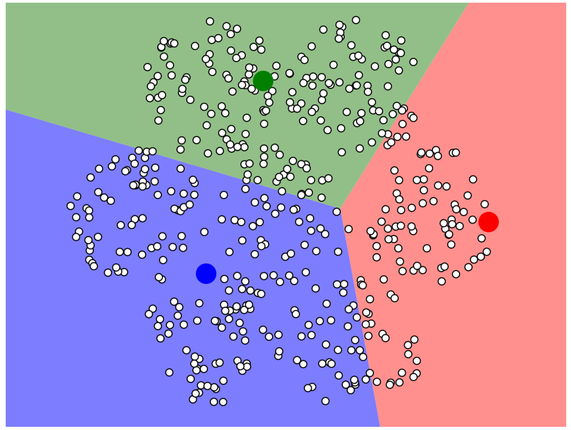
K-Means clustering (centroid-based)

https://www.naftaliharris.com/blog/visualizing-k-means-clustering/

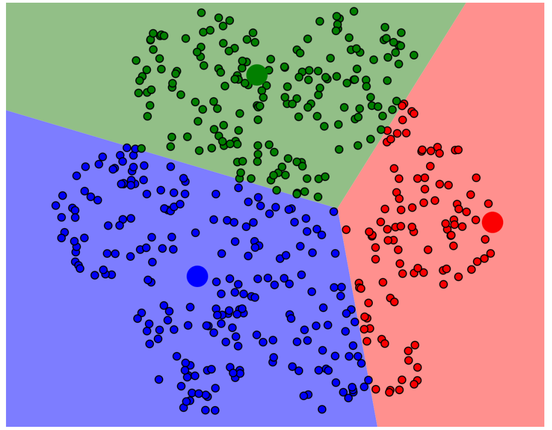
- Centroid == center of the cluster



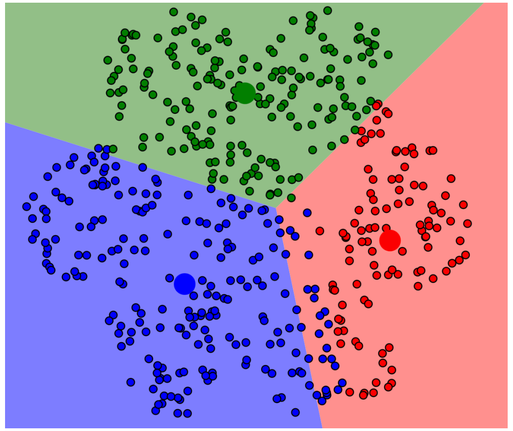
Initial data



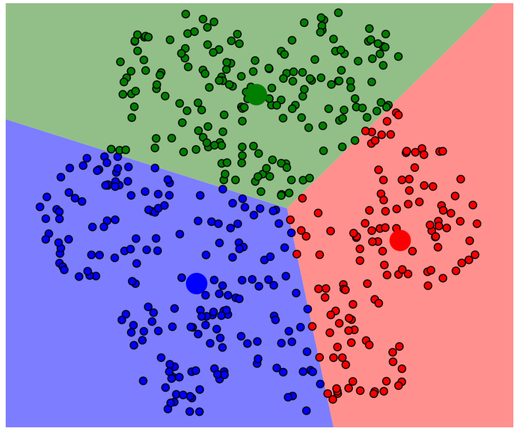
K randomly chosen centroids (C)



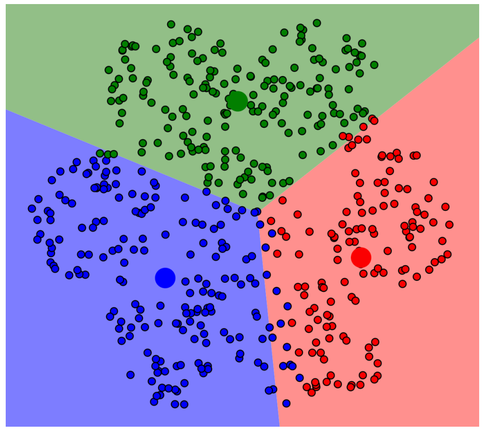
Assign points to Cs



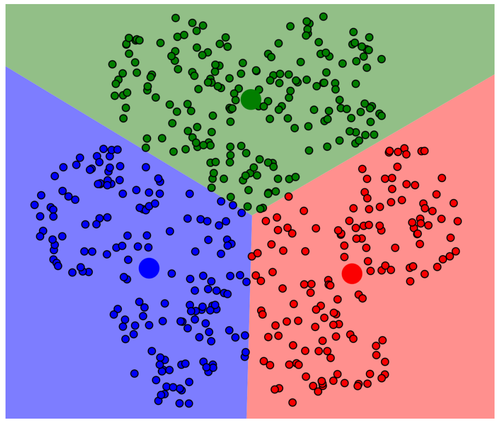
Assign new Cs



- Reassign points to closest C



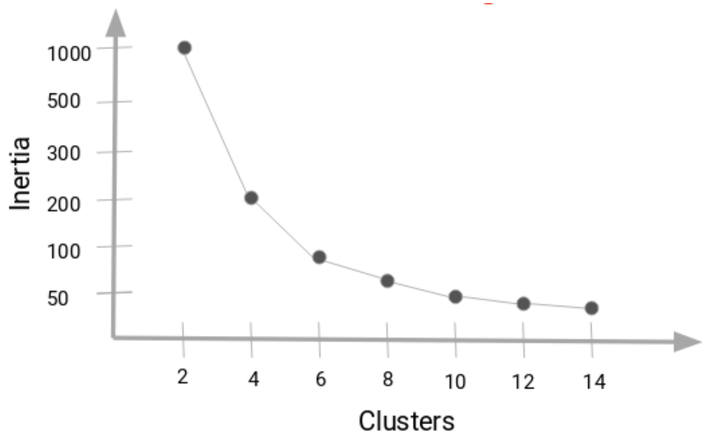
Reassign Cs

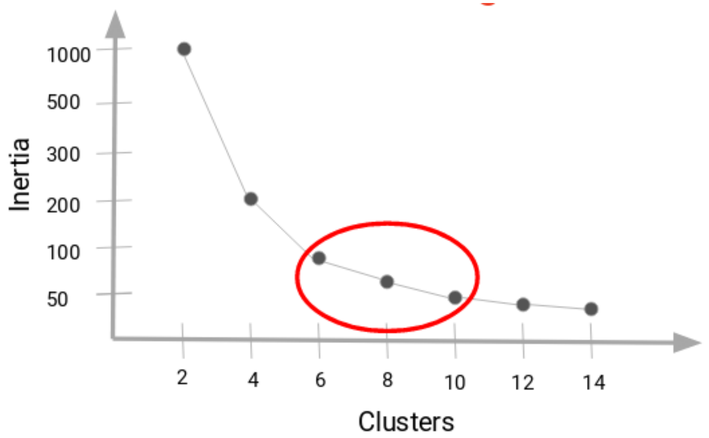


Iterate until centroids do not change anymore

How do you define K (the number of clusters)?

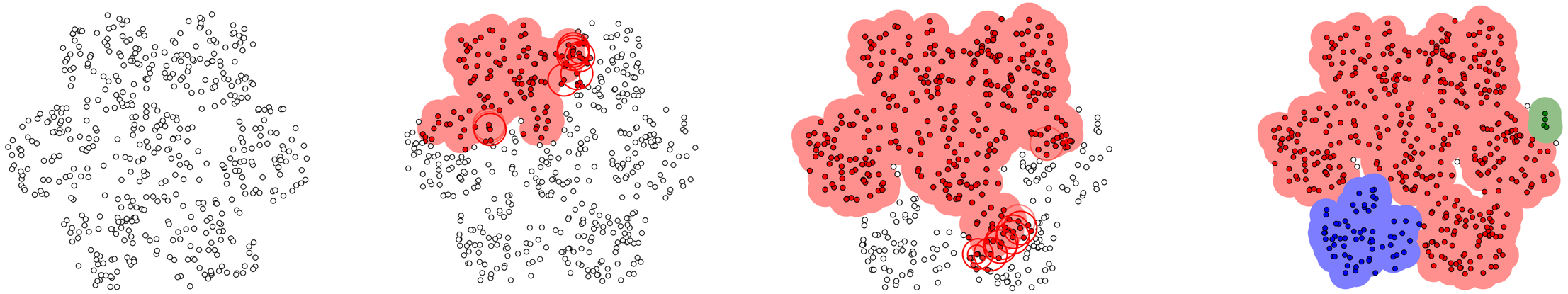
- A good model has low inertia and low K.





Inertia measures how well a dataset was clustered by K-Means. It is calculated by measuring the distance between each data point and its centroid, squaring this distance, and summing these squares across one cluster.

Density-based spatial clustering of applications with noise (DBSCAN)

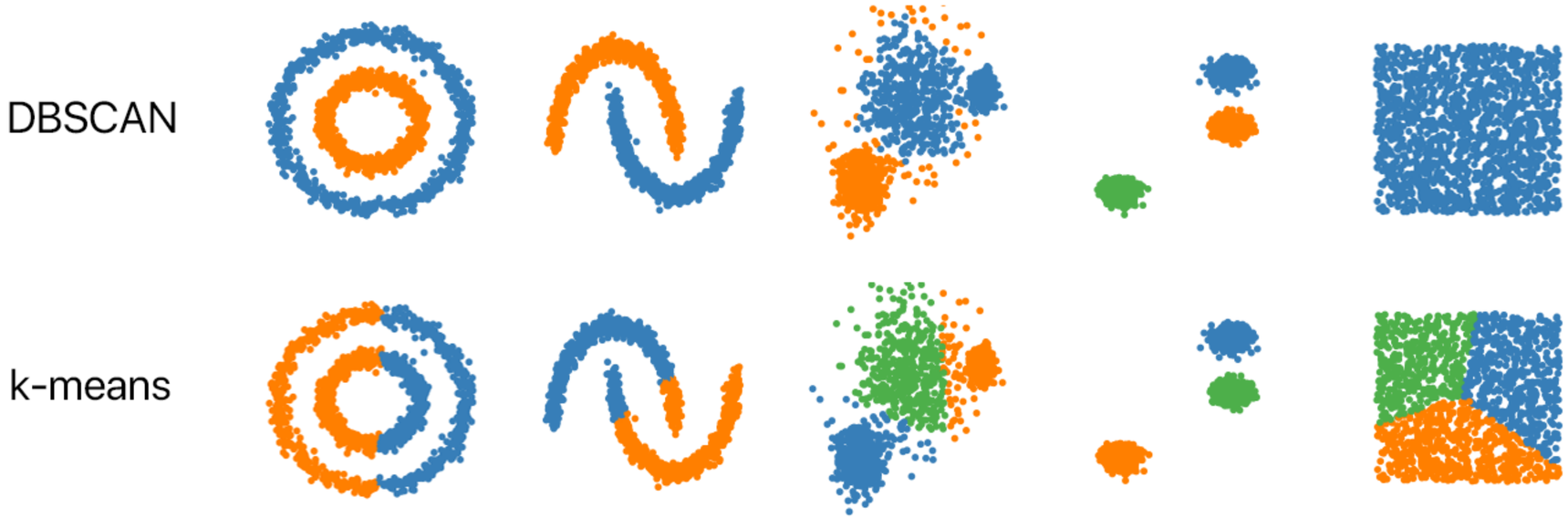


- Point belonging to cluster is near lots of points in that cluster
- Start by picking random point, min points in distance => add to cluster
- When no more point can be added, pick the next arbitrary point

- https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/

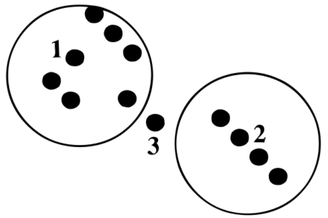
Comparison between k-Means and DBSCAN

- Advantage: You don’t have to specify the number of clusters in advance in DBSCAN



Butina Clustering method (centroid-based, like K-Means)

- Generation of Fingerprints.
- Identifying Potential Cluster Centroids
- Cluster Algorithm Based on Exclusion Spheres at a Given Tanimoto Level
  - Hence, cluster will only include molecules that are above a similarity threshold



Unsupervised data base clustering based on daylight's fingerprint and Tanimoto similarity: A fast and automated way to cluster small and large data sets

D Butina - Journal of Chemical Information and Computer …, 1999 - ACS Publications

One of the most commonly used clustering algorithms within the worldwide pharmaceutical industry is Jarvis− Patrick's (J− P)(Jarvis, RA IEEE Trans. Comput. 1973, C-22, 1025− 1034). The implementation of J− P under Daylight software, using Daylight's fingerprints and the Tanimoto similarity index, can deal with sets of 100 k molecules in a matter of a few hours. However, the J− P clustering algorithm has several associated problems which make it difficult to cluster large data sets in a consistent and timely manner. The clusters produced …

☆ Save 99 Cite Cited by 424 Related articles All 3 versions

- https://projects.volkamerlab.org/teachopencadd/talktorials/T005_compound_clustering.html

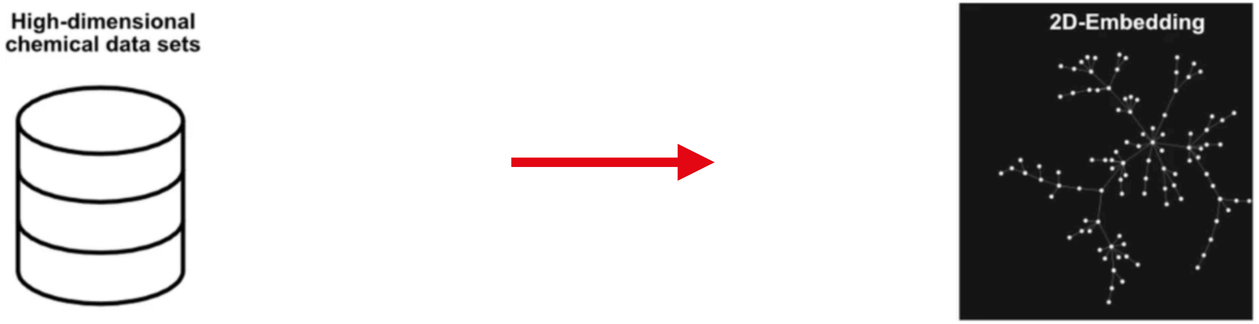
Code examples


```python
class ButinaClustering:
    def __init__(self, cutoff=0.8, metric='jaccard'):
        self.cutoff = cutoff
        self.metric = metric

    def fit(self, x):
        """
        Perform Butina clustering on a set of fingerprints.

        :param x: A numpy array of binary data
        :return: self
        """
        # Calculate the distance matrix
        distance_matrix = []
        x = x.astype(bool)
        for i in range(1, len(x)):
            distances = pairwise_distances(x[i,:].reshape(1, -1), x[:i,:], metric=self.metric)
            distance_matrix.extend(distances.flatten().tolist())

        # Perform Butina clustering
        clusters = Butina.ClusterData(distance_matrix, len(x), self.cutoff, isDistData=True)
        self.clusters = clusters

        # Assign cluster labels to each data point
        cluster_labels = np.full(len(x), -1, dtype=int)
        for label, cluster in enumerate(clusters):
            for index in cluster:
                cluster_labels[index] = label

        self.labels_ = cluster_labels


    def fit_predict(self, x):
      self.fit(x)
      return self.labels_
```

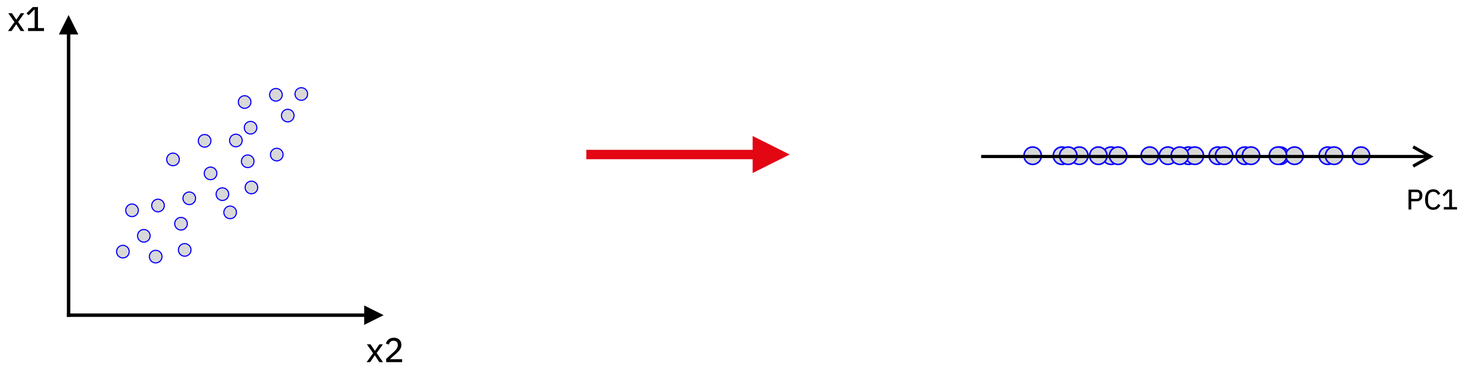
Dimensionality reduction

- high-dimensional space into a low-dimensional space
- How to display molecules described with a 1024-dimensional fingerprint in 2D?

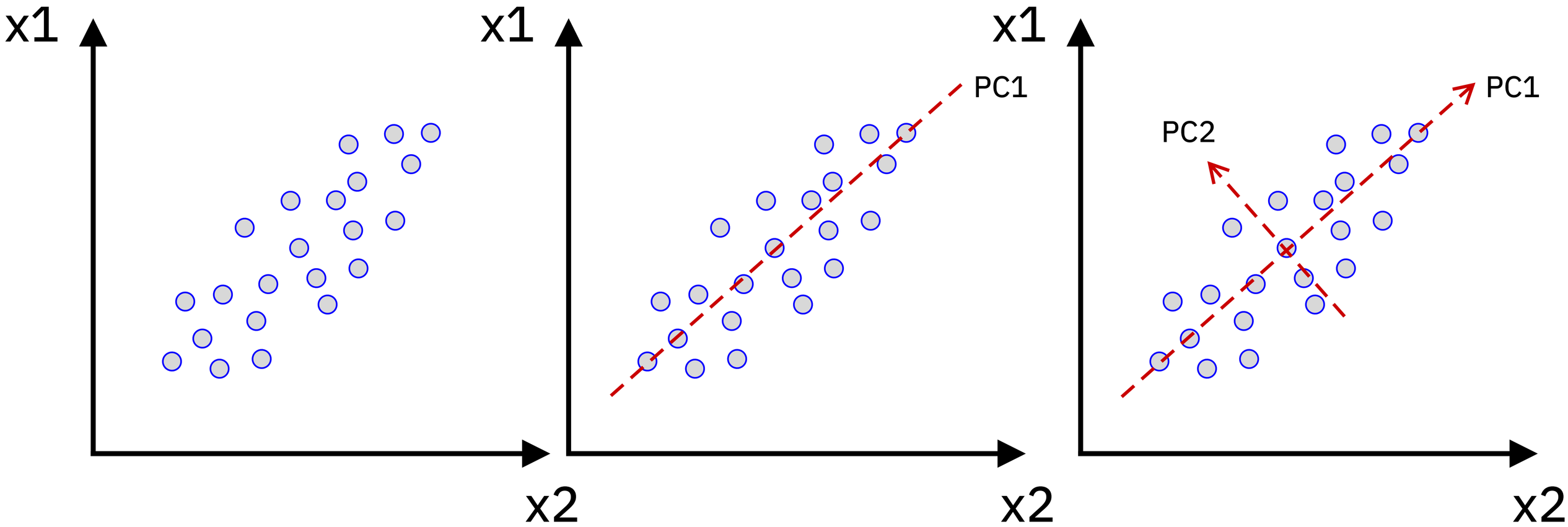
Reality:



Examples
(for simplicity)

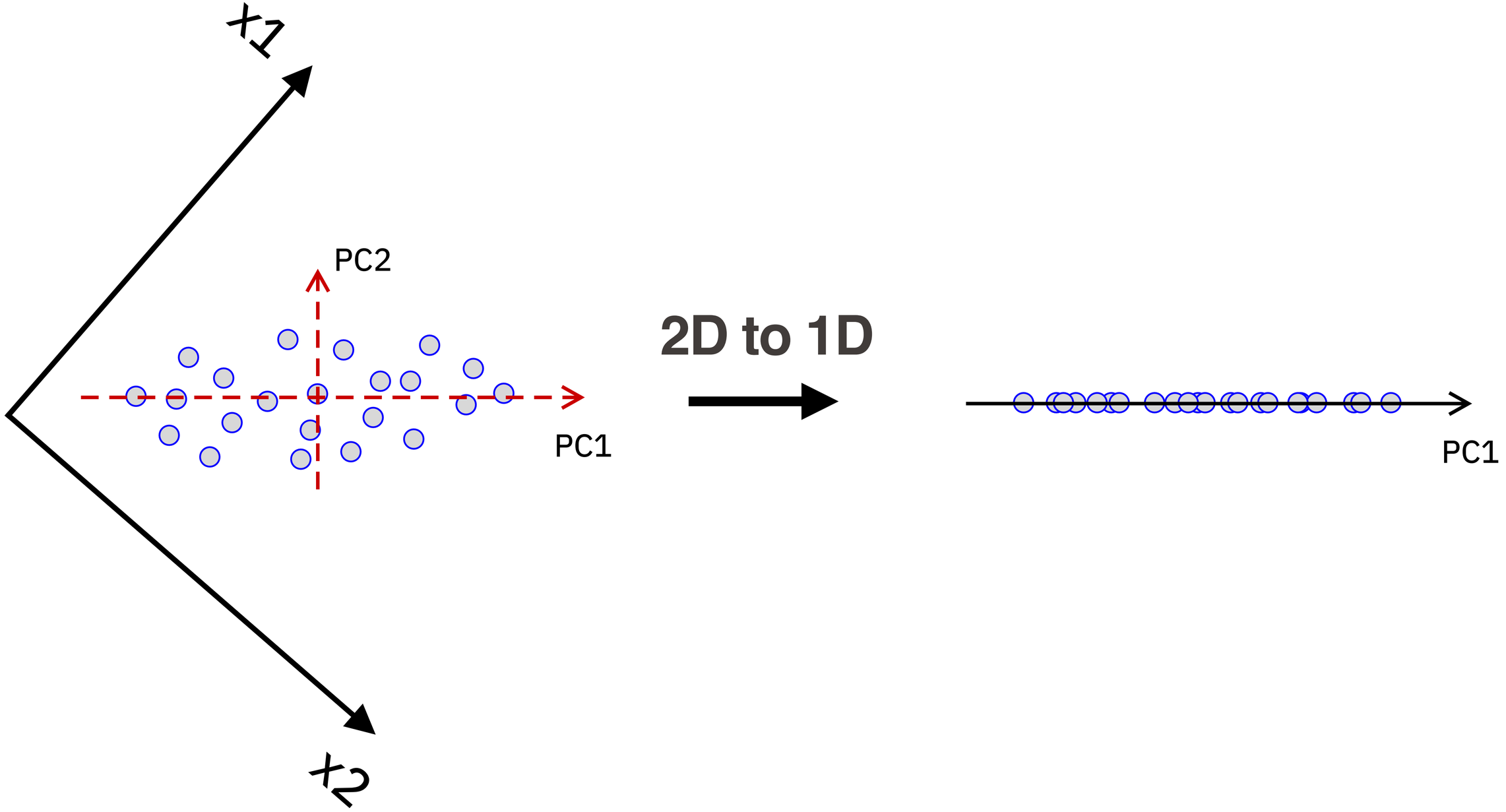


Principal Component Analysis (PCA)



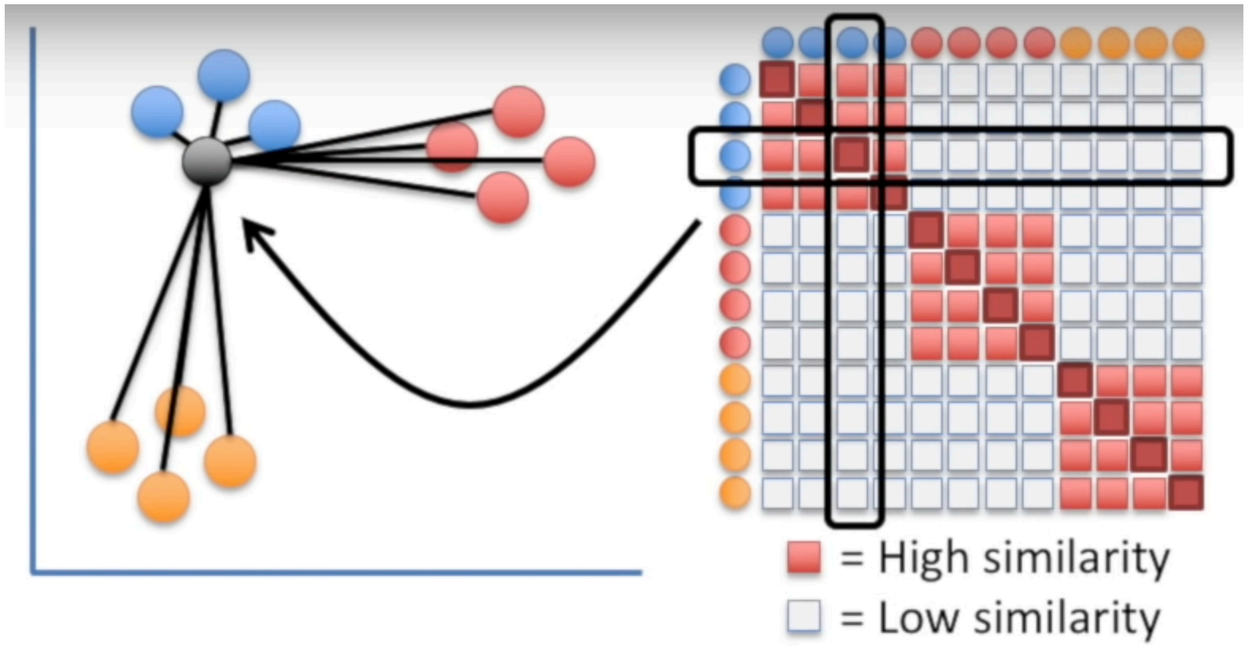
- PCA finds directions of maximum variance
- principal components are orthogonal

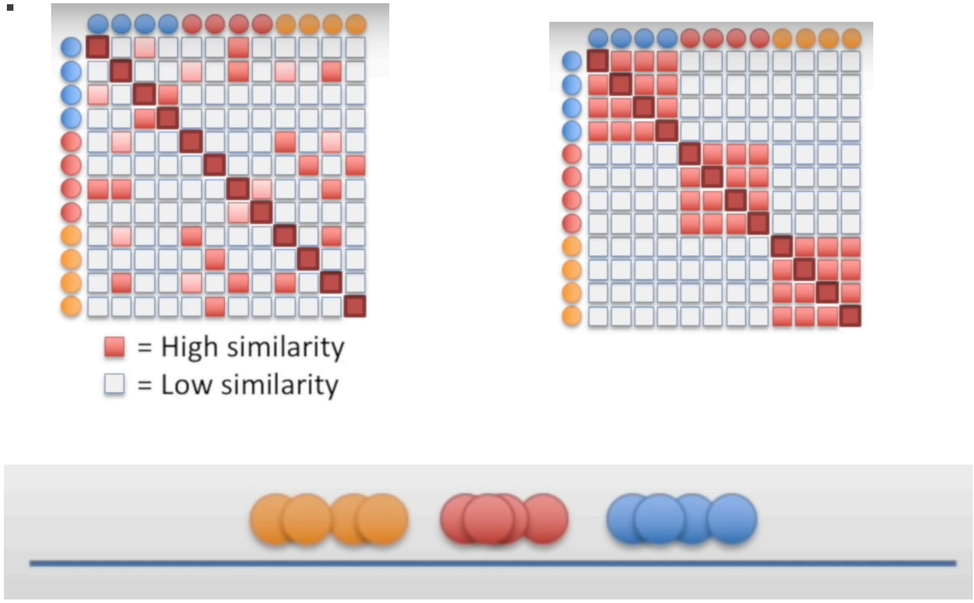
Principal Component Analysis (PCA)



T-Distributed Stochastic Neighbor Embedding (T-SNE)

- For complicated datasets, PCA may not work well.





- Calculate similarity scores in high dimensions (here 2D)
- Randomly place the samples in lower dimensionality (1D)
- Iteratively move the points, until similarity matrix looks same as in high dimensions

- Youtube: v=NEaUSP4YerM

[PDF] **Visualizing data using t-SNE.**
L Van der Maaten, G Hinton - Journal of machine learning research, 2008 - jmlr.org
… For **visualizing** the structure of very large **data** sets, we show how **t-SNE** can **use** random … structure of all of the **data** to influence the way in which a subset of the **data** is displayed. We …
☆ Save 99 Cite Cited by 33868 Related articles All 42 versions »

Uniform manifold approximation and projection (UMAP)

- Main ideas are similar to t-SNE
  - Compute similarity in high dimensions
  - Move points in low dimensions according to high-D similarity
- Two key differences
  - t-SNE starts with random initialisation in low-D (every time you use it on same data, it will be different).
    In contrast, UMAP always starts with the same points.
  - t-SNE moves every point at every iteration. UMAP moves one or a small subset of points (better scaling).

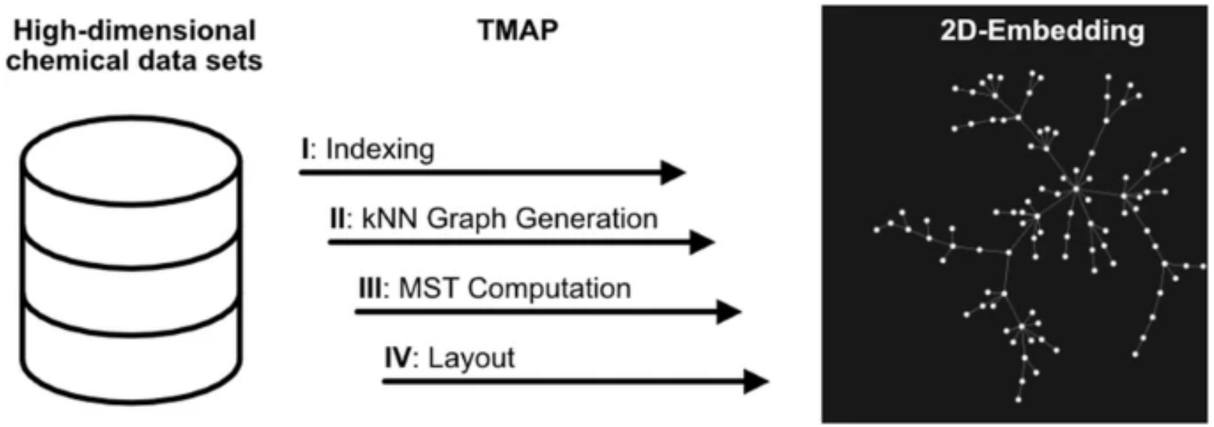
Umap: **Uniform manifold approximation** and **projection** for dimension reduction

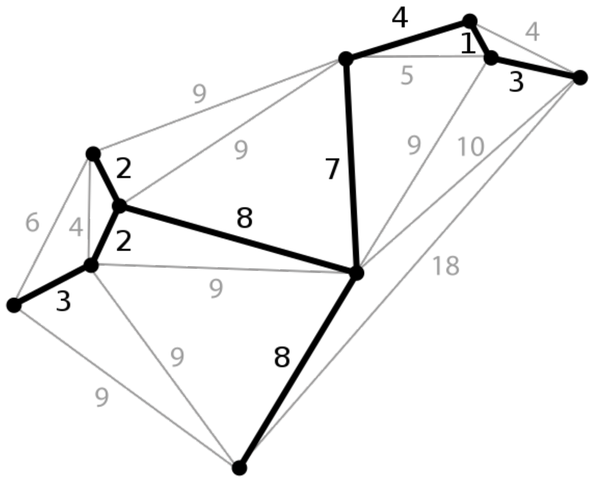
L McInnes, J Healy, J Melville - arXiv preprint arXiv:1802.03426, 2018 - arxiv.org

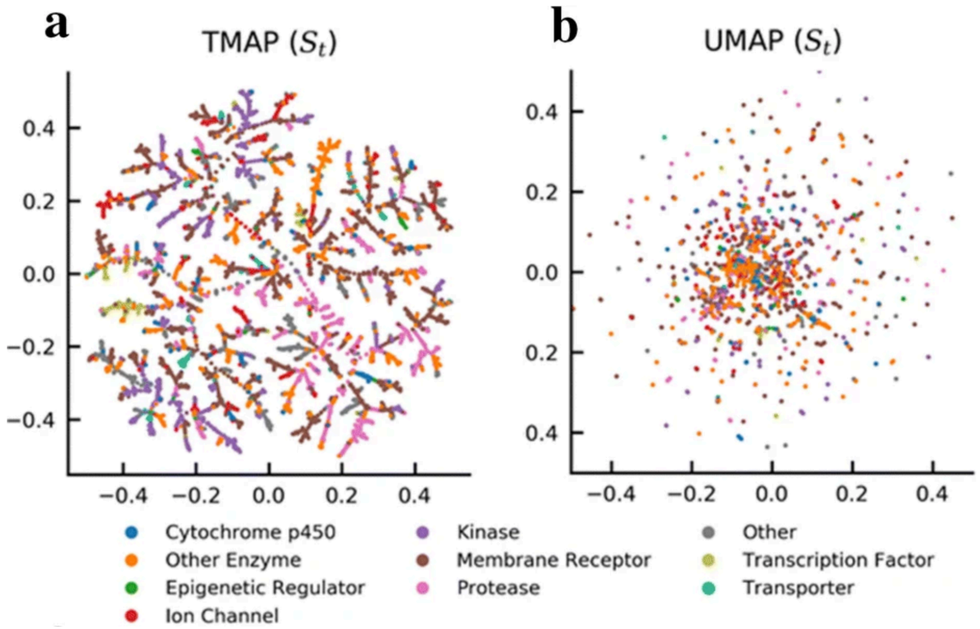
… **approximated manifold**. In explaining the algorithm we will rst discuss the method of **approximating** the **manifold** … simplicial set structure from the **manifold approximation**. Finally, we …

☆ Save Cite Cited by 7605 Related articles All 12 versions »

Tree Map (TMAP)







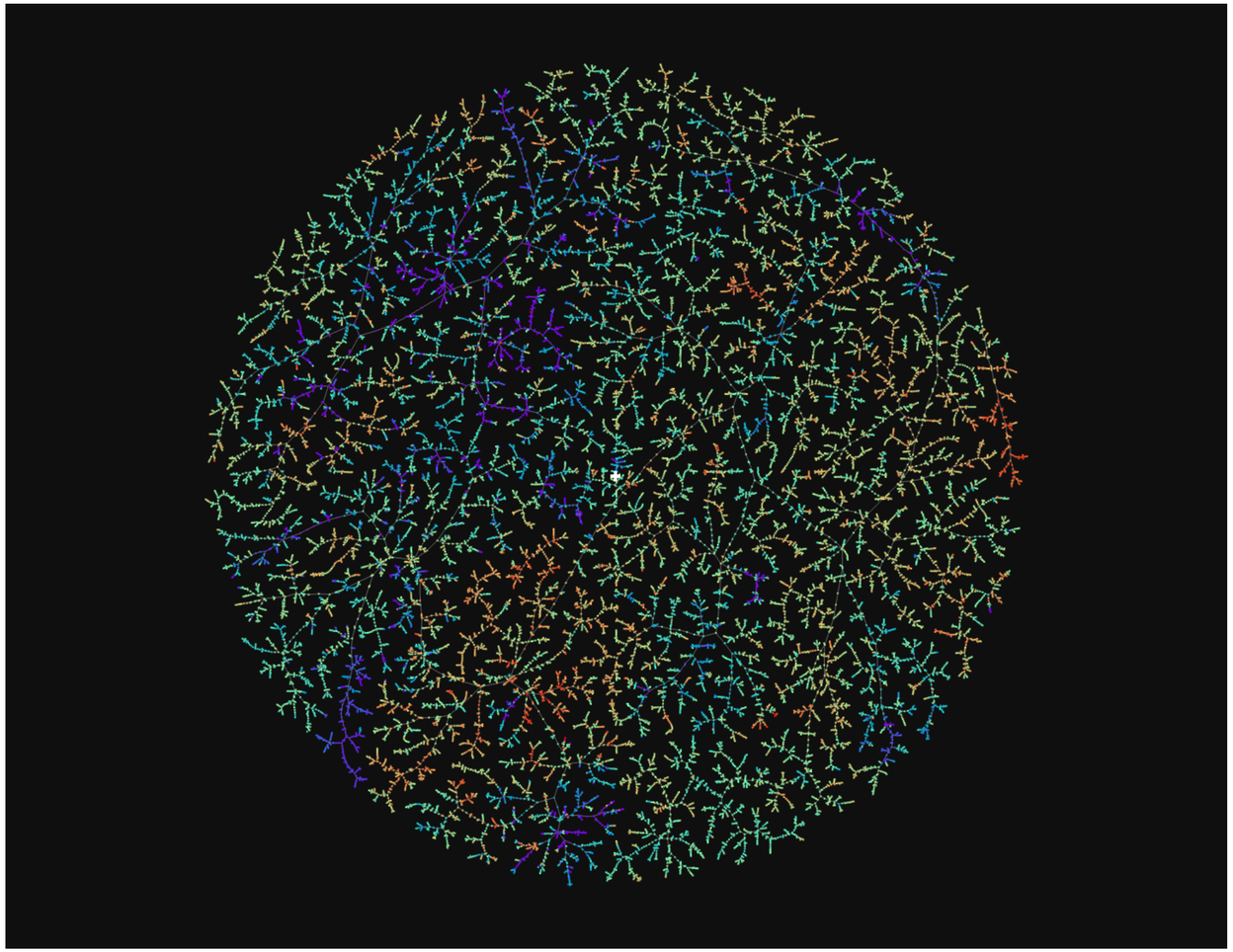
[HTML] Visualization of very large high-dimensional data sets as minimum spanning trees

D **Probst**, JL Reymond - Journal of Cheminformatics, 2020 - jcheminf.biomedcentral.com

… large data sets due to their **tree-like** nature, increased local and … topographic **maps** (GTM) and self-organizing **maps** (SOM), … Here we present an algorithm, named TMAP (Tree **MAP**), to ..

☆ Save 99 Cite Cited by 140 Related articles All 21 versions »

Natural Product Atlas — with TMAP



https://tmap.gdb.tools/src/npatlas/index.html

- Build your own: https://tmap.gdb.tools/

RESEARCH ARTICLE | CATALYSIS

Accelerated dinuclear palladium catalyst identification through unsupervised machine learning

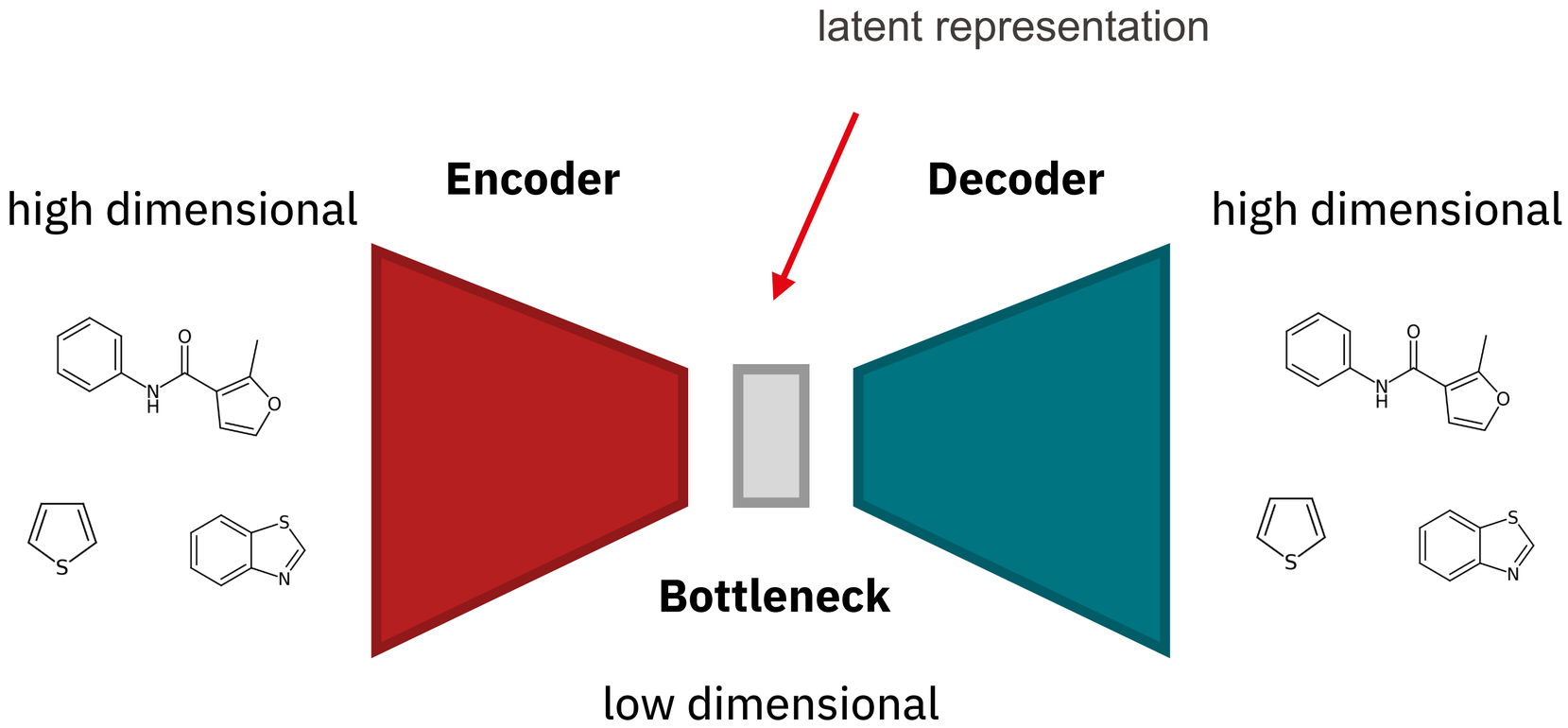
JULIAN A. HUEFFEL, THERESA SPERGER, IGNACIO FUNES-ARDOIZ, JAS S. WARD, KARI RISSANEN, AND FRANZISKA SCHOENEBECK Authors Info & Affiliations

SCIENCE • 25 Nov 2021 • Vol 374, Issue 6571 • pp. 1134-1140 • DOI: 10.1126/science.abj0999

- Catalyst optimization is often difficult to do rationally.
- very small class of phosphine ligands known for stabilising Pd dimers.
- Hueffel et al. used machine learning to search for patterns in this known class of ligands and thereby guide the discovery of variants that likewise stabilize the dimers. => K-Means clustering
- The authors were able to synthesize eight previously unreported dimers.
- More on that in the exercises!

Unsupervised learning in Deep Learning — representation learning

Autoencoder — dimensionality reduction

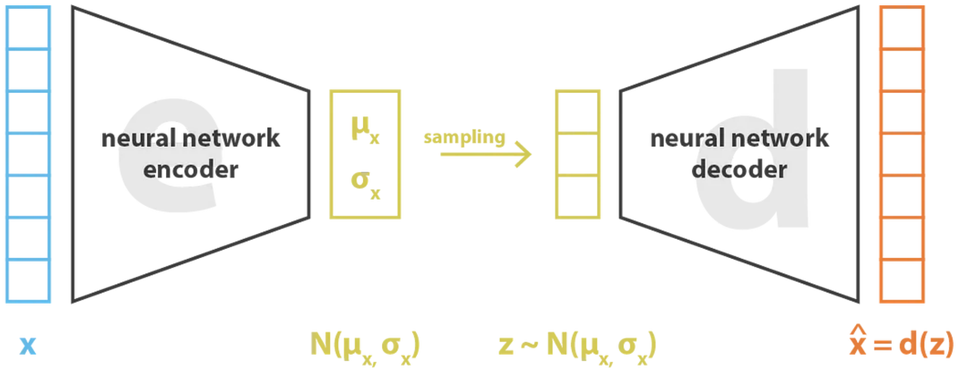


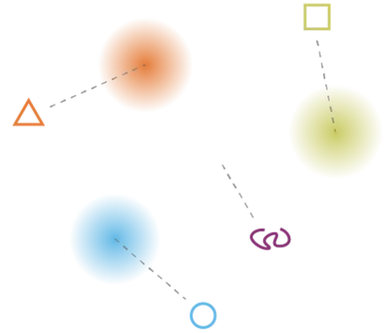
- for molecules: encoder and decoder could be recurrent neural networks/Transformers on SMILES
- There is an information bottleneck, the model has to compress the high dimensional data in a way to be able to reconstruct it.
- Trained no a reconstruction loss
- No guarantee that close points in bottleneck are close points in high dimensional space

Variational Autoencoder

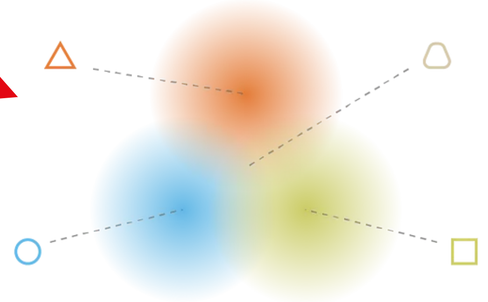
	input	encoding	latent representation	decoding	input reconstruction
simple autoencoders	**x**	→	**z = e(x)**	→	**d(z)**

	input	encoding	latent distribution	sampling	sampled latent representation	decoding	input reconstruction
variational autoencoders	**x**	→	**p(z|x)**	→	**z ~ p(z|x)**	→	**d(z)**





what can happen without regularisation



what we want to obtain with regularisation

$$\text{loss} = \|x - \hat{x}\|^2 + KL[N(\mu_x, \sigma_x), N(0, I)] = \|x - d(z)\|^2 + KL[N(\mu_x, \sigma_x), N(0, I)]$$

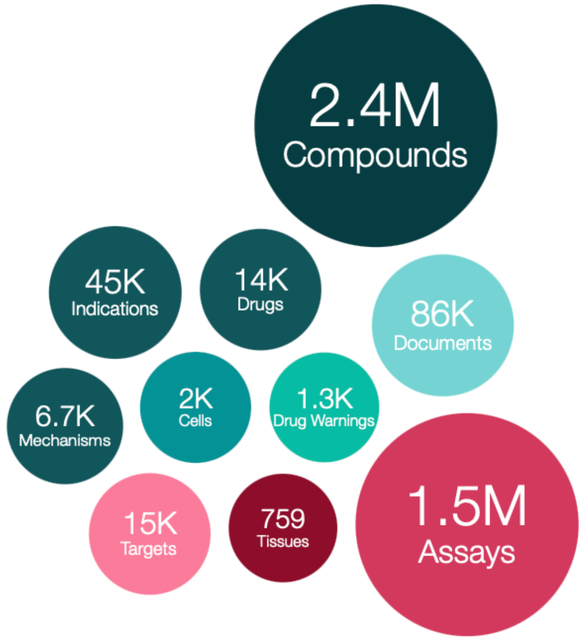
Auto-encoding variational bayes
DP Kingma, M Welling - arXiv preprint arXiv:1312.6114, 2013 - arxiv.org
… **variational** lower bound yields a simple differentiable unbiased estimator of the lower bound; this SGVB (Stochastic Gradient **Variational Bayes**… , we propose the **AutoEncoding** VB (AEVB…
☆ Save 99 Cite Cited by 25476 Related articles All 41 versions »

- https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

How to apply deep learning in the low data regime.

Unsupervised learning in Deep Learning — pretaining

Pretaining



CHEMBL dataset
(Millions of structures — not labeled for your task)

Fine-tuning

Your task

100 labeled datapoints
(Not enough to train a deep learning model)

What can we do?

Self-supervised learning / pretraining

A Systematic Survey of Molecular Pre-trained Models (Chemical Language Models)

awesome stars 194 Fork 18

This is a repository to help all readers who are interested in pre-training on molecules. If you find there are other resources with this topic missing, feel free to let us know via github issues, pull requests or my email: xiajun@westlake.edu.cn. We will update this repository and paper on a regular basis to maintain up-to-date.

> **Last update date: 2023-3-09**

https://github.com/junxia97/awesome-pretrain-on-molecules

Strategies for pre-training graph neural networks

W Hu, B Liu, J Gomes, M Zitnik, P Liang… - arXiv preprint arXiv …, 2019 - arxiv.org

… For Context Prediction illustrated in Figure 2 (a), on **molecular graphs**, we define context **graphs** by setting inner radius r1 = 4. On PPI networks whose diameters are often smaller than 5…

☆ Save 99 Cite Cited by 666 Related articles All 13 versions »

Pre-training molecular graph representation with 3d geometry

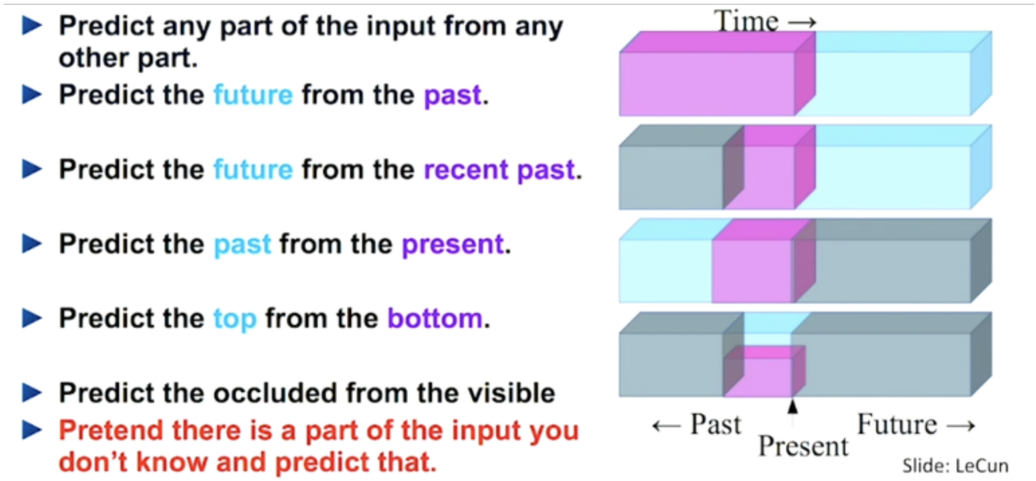
S Liu, H Wang, W Liu, J Lasenby, H Guo… - arXiv preprint arXiv …, 2021 - arxiv.org

… the **Graph** Multi-View **Pre-training** (GraphMVP) framework, where a 2D **molecule** encoder is pre-… Finally, we summarize a broader **graph** SSL family that prevails the 2D **molecular graph** …

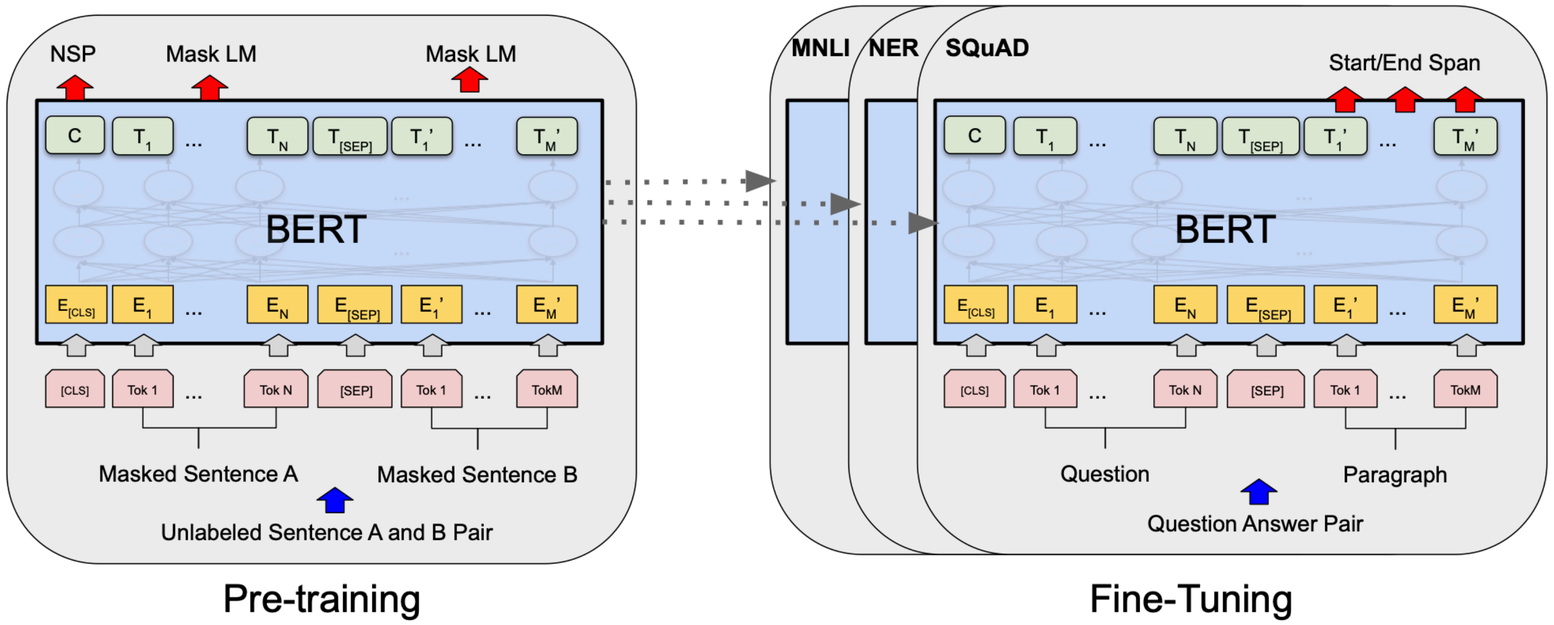
☆ Save 99 Cite Cited by 68 Related articles All 5 versions »

What does self-supervised learning mean?

- We have a lot of unlabelled data (molecules without corresponding activity), and we invent a task that we can train on without labels.
  - Corrupt the graph / hide an atom, and let the model predict it
  - For SMILES-based language models, predict the next SMILES token
  - …



Masked Language Modelling (MLM) and next sentence prediction



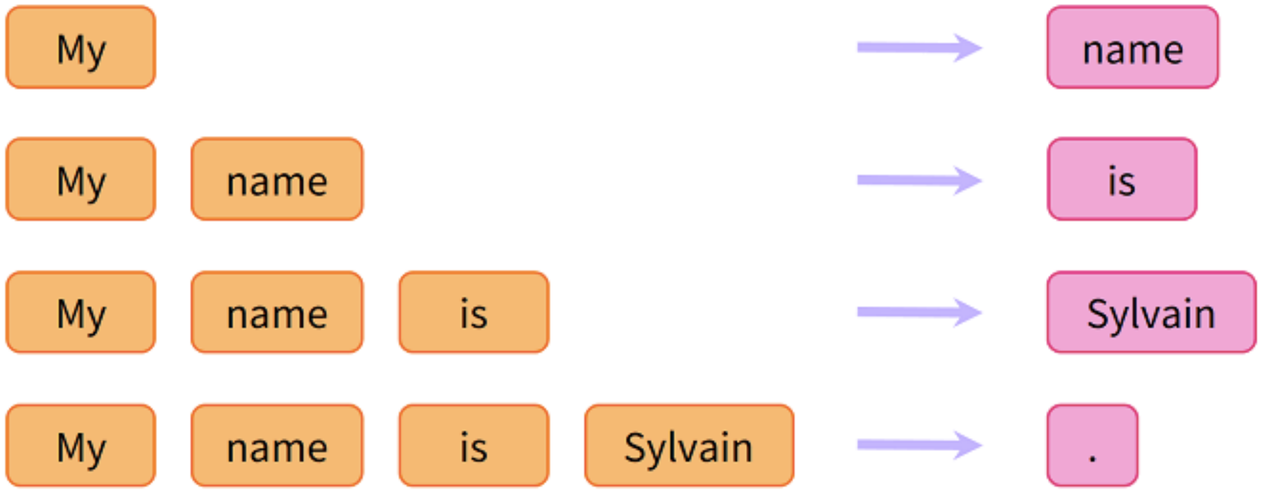
Input = [CLS] the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP].
Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP] penguin [MASK] are flight ##less birds [SEP]
Label = NotNext

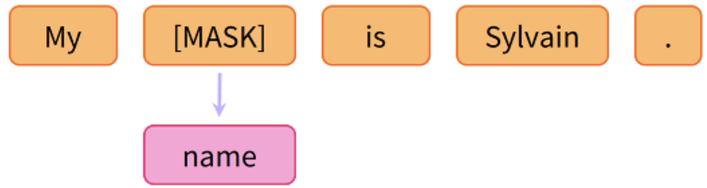
BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

Auto-regressive language modelling

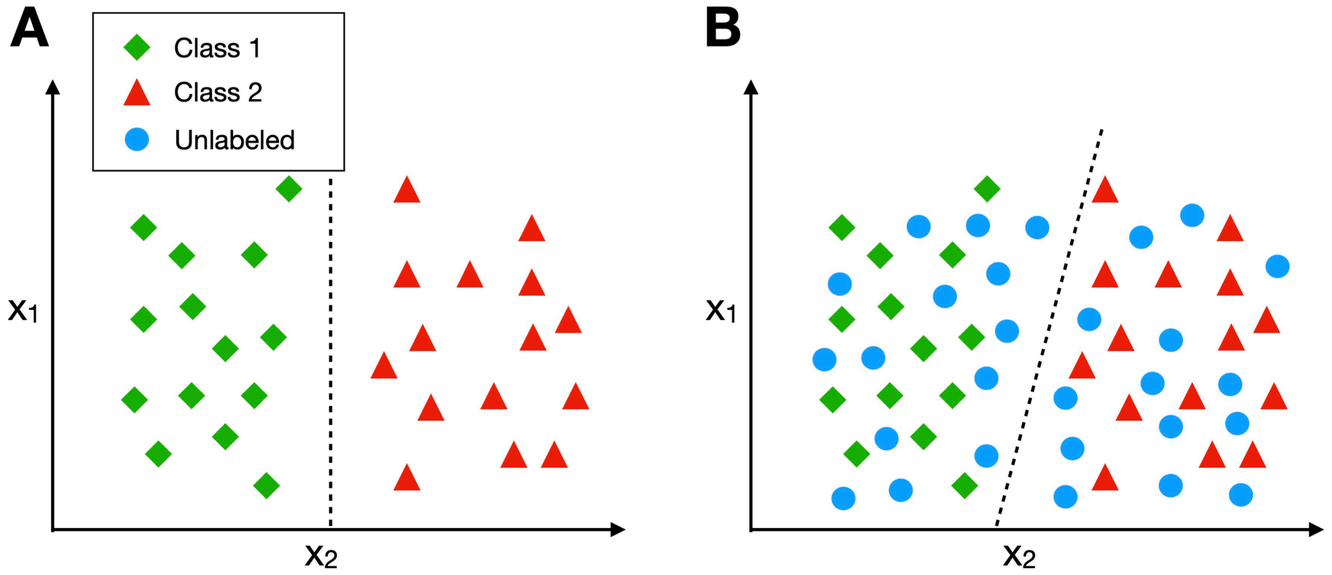


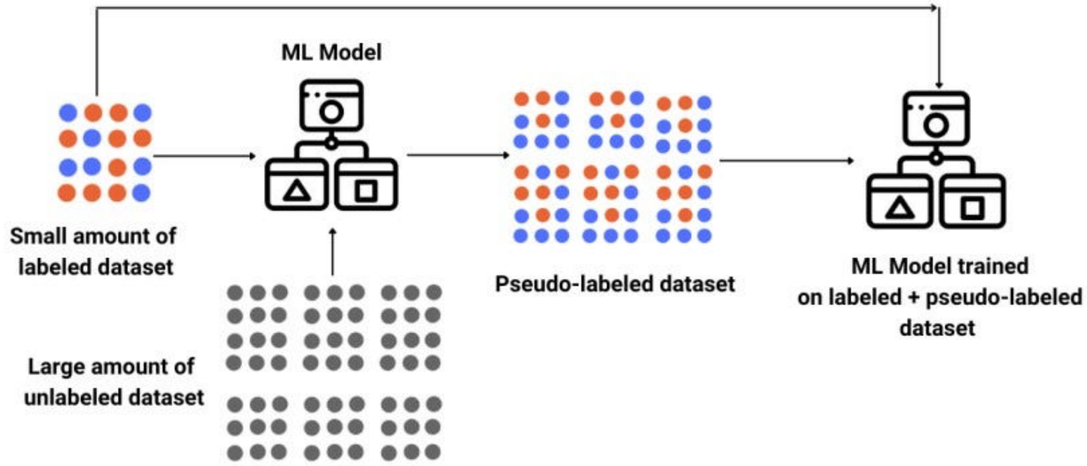
Here the model predicts a masked word in the sentence.



- https://thinkingneuron.com/sentiment-analysis-of-tweets-using-bert/

Semi-supervised learning





- https://www.enjoyalgorithms.com/blogs/supervised-unsupervised-and-semisupervised-learning

Projects

- Groups of up to 3-4
- ML project in chemistry
  - Collect data from literature / online / from a project you have worked on
  - Train ML models (GitHub repo, 30%)
  - Write a 4-page report including intro/task/data/methods/results (40%)
  - Present outcomes in the last course session (15-20 min per group, 30%)

- Sign up on Google Form
- Fill it in by next week

Potential topics — select a use case

- Molecular / reaction property prediction
- Reaction/process condition optimization
- De novo molecular optimization
- Free topic

- It’s fine to start from an open-source codebase and build on top.
- Please let us know what you are most interest in, in the google form, and we can provide useful pointers to data, baseline models, etc.