Supervised Learning
What is it?

Supervised: for possible inputs to the model, you know what you want the outputs to be to achieve the purpose of the project.

Classification

Regression

Curve fitting

Semi-supervised: there are target outputs in the available data, but their relationship to the overall problem may be indirect.

Learned embeddings

Autoencoders

Auto-regressive models
(e.g. time series prediction)

Unsupervised: discover inherent structure within the data without explicit examples of what constitutes a successful outcome.

Clustering

Manifold methods (PCA, etc)

Anomaly detection

Supervised Learning When to use it?

Supervised learning is appropriate when you have some task which is resource intensive, but where you're certain that you can verify or define a **correct answer**.

Supervised learning, done correctly, should generalize to new examples **drawn from the same distribution as the training data**.

However, it will **not** necessarily generalize to new examples that are produced in a different way than the training data.

It will not generally provide explanation of **why** it works, nor will it necessarily capture any **causal** or **process** aspects of what it is used for.

Supervised Learning
When to use it?

Good example: Determining earthquake time in seismic recordings

- Labor-intensive to do by hand
- Hand-labeled data available, can be generated
- Use-case and training-case are very similar
- Only important to obtain accurate timings, doesn't matter how

Bad example: Predicting medical outcomes to design a new medical intervention

- Training data (no-intervention) and use case (intervention) are different distributions
- The model cannot distinguish correlation and causation

E.g., 'to stop someone from smoking, cure their cancer'

Not unsurmountable, but needs more sophistication.

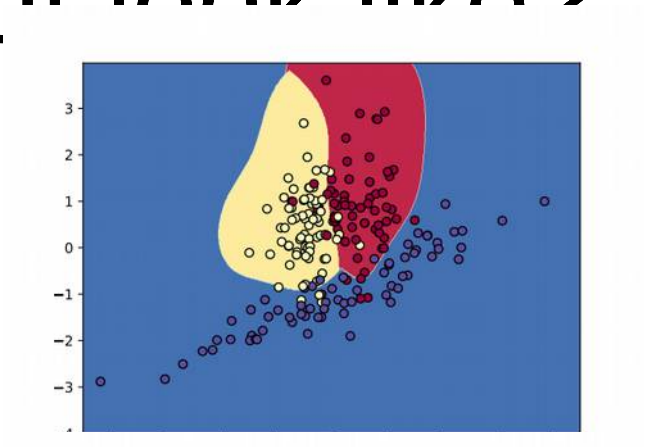
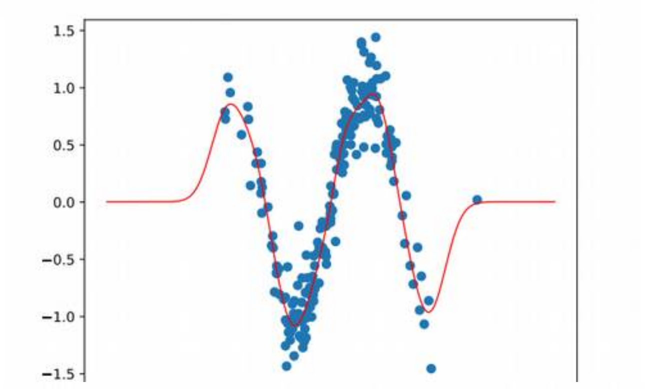
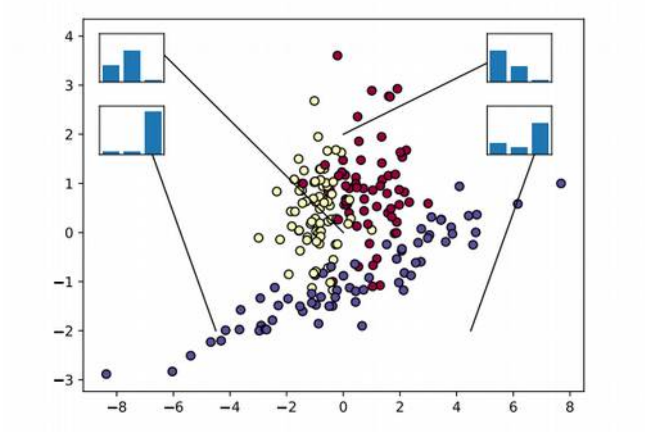
Might involve modeling work to identify which predictions can only be causal, tools from semi-supervised learning, etc.

Supervised Learning
What does an outpu

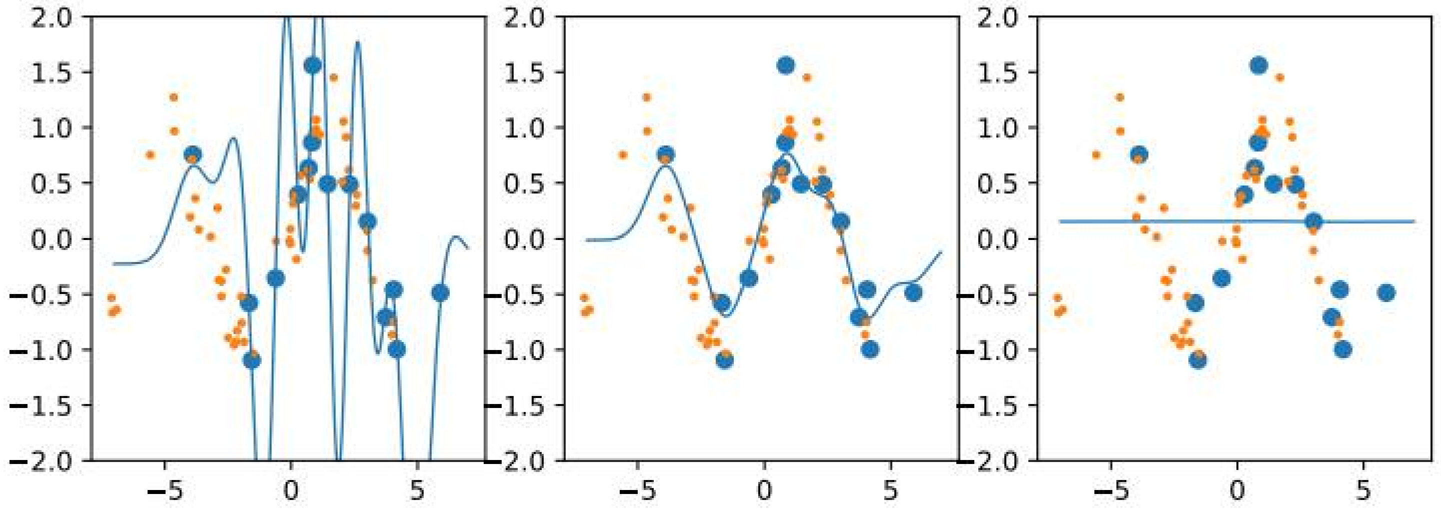
Once you’ve gotten data and trained a supervised model, what can you use it to do?

Given (observation) ...

- Classification: What category?
- Regression: What is the expected value of some other variable?
- Probabilistic models: What is the relative probability of (possible answers to a particular question)?

Supervised Learning Overfitting



A perfect fit passes through every point, but does not **generalize**.

Therefore achieving a good error rate on the **training data** is not evidence of a good model.

Need independent **test data** drawn from the same distribution to determine if a model will generalize.

Supervised Learning
Training, Validation, Testing

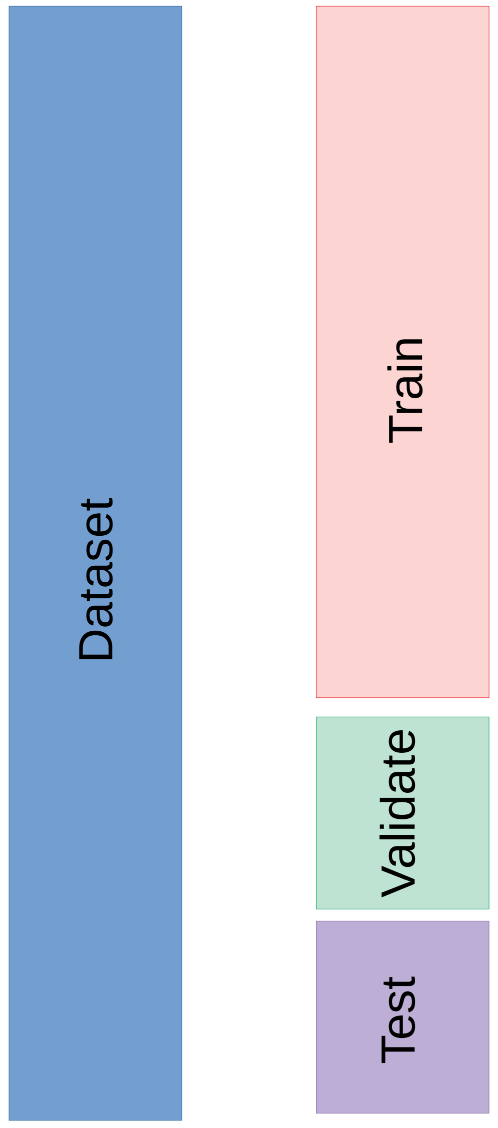
Best practice: use different data when making decisions about the model than when evaluating the model.

Training set: used directly for model updates/training.

Validation set: evaluate algorithm choice, hyperparameters.

Test set: evaluate the expected performance for where the model will be used.

For example, a 70/15/15 split. May not be feasible for small datasets.

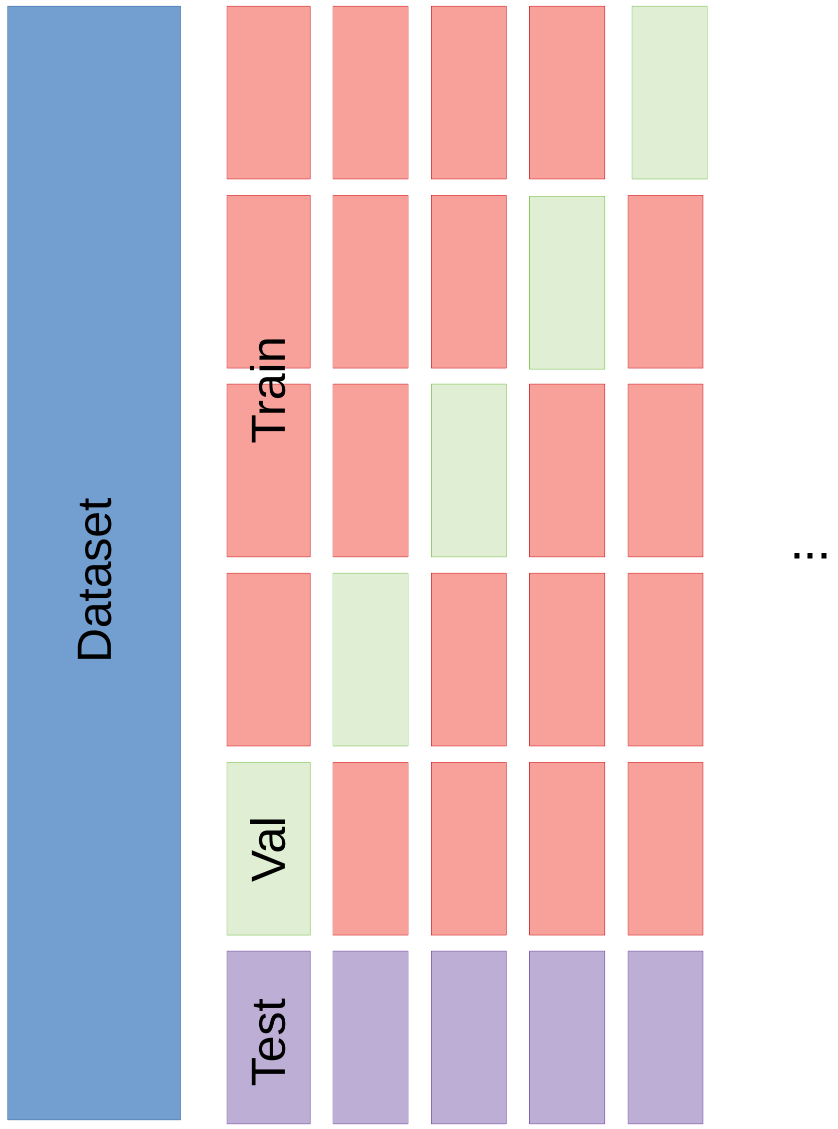


Supervised Learning
Insufficient data? Cross-validation!

When validation/test set sizes are too small to get good enough statistics to compare model choices, use CV:

Cut dataset in multiple ways, train separate model on each, use statistics of results to estimate performance for validation.

Important consideration is statistical similarity of the cuts to the original dataset. Particularly an issue when # of labels ~ # of points.



Supervised Learning
Standard Problem Setup

Input data x

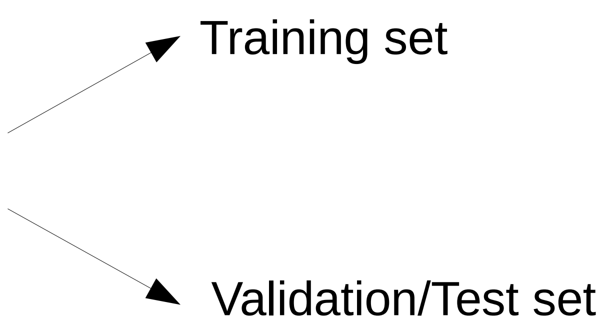
Feat 1	Feat 2	Feat 3	...	Feat M
Data 1				
Data 2				
Data 3				
...				
Data N				

NumPy array (N,M)

Target values y

Labels
Label 1
Label 2
Label 3
...
Label N

NumPy array (N,)



```
import numpy as np
import pandas as pd

data = pd.read_csv(“datafile.csv”)
feature_fields = [ “Feat 1”, “Feat 2” ] # For example
label_field = “Label” # For example

x = np.array(data[feature_fields])
y = np.array(data[label_field])
```

```
# 25% test set, 75% training set example

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x,y,0.25)

...
```

```
# Cross-validation example – tries to balance labels

from sklearn.model_selection import StratifiedShuffleSplit

indexer = StratifiedShuffleSplit(n_splits=5, test_size = 0.25)

for train_idx, test_idx in indexer.split(x,y):
      x_train = x[train_idx]
      y_train = y[train_idx]
      x_test = x[test_idx]
      y_test = y[test_idx]

      ...
```

Supervised Learning
Interchangeable Methods

Data that fits into the standard form can be used with a variety of pre-packaged methods.

```
models = [
sklearn.linear_model.ElasticNet,
sklearn.linear_model.LogisticRegression,
sklearn.linear_model.RidgeClassifier,
sklearn.svm.LinearSVC,
sklearn.svm.SVC,
sklearn.tree.DecisionTreeClassifier,
sklearn.ensemble.RandomForestClassifier,
sklearn.ensemble.GradientBoostingClassifier,
... ]
```

```
for model_type in models:
      clf = model_type()
      clf.fit(x_train,y_train)
      p = clf.predict(x_test)
      score = metric(p, y_test)

      print(model_type, score)
```

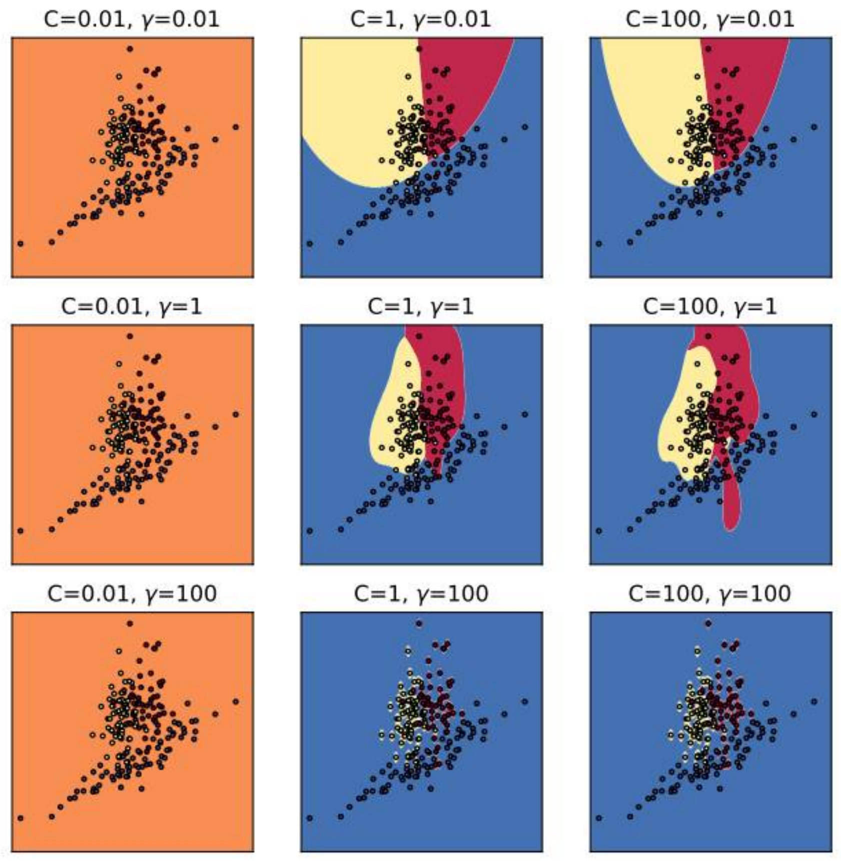
Compare model performance without needing detailed knowledge of how each type of model works.

Generally still need to optimize hyperparameters (validation set or cross-validation needed to avoid overfitting).

Supervised Learning Hyperparameter Search

Hyperparameters: extra constants associated with a model type, not learned directly from data.

Dominant effect is in controlling trade-off between overfitting and underfitting.



Supervised Learning
Hyperparameter Search

Cross-validation can be used to automatically search hyperparameters.

Grid search: try all combinations exhaustively

Random search: try random combinations

Fancier methods exist, but beyond the scope of this.

Grid search for 1-2 hyperparameters

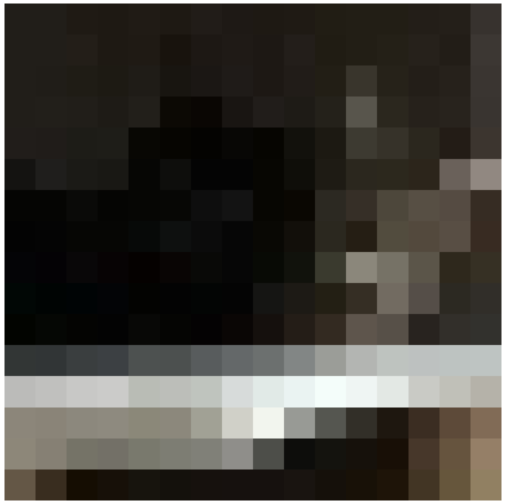
```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

clf = SVC()
params = { ‘C’: [1e-4,1e-3,1e-2,1e-1,1,10,1e2,1e3,1e4],
           ‘gamma’: [0.01,0.1,1,10,100]}

search = GridSearchCV(clf, params, cv=5)

search.fit(train_x, train_y)
print(search.best_params_, search.best_score_)
```

Random search for 3+ hyperparameters

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier()
params = { ‘n_estimators’: [5,10,20,40,80,200],
           ‘max_depth’: [2,3,4,5,6,7],
           ‘min_samples_split’: [2,4,8,16],
           ‘max_features’: [0.1,0.2,0.4,0.6,0.8,1] }

search = RandomizedSearchCV(clf, params,
                            n_iter = 20, cv = 5)

search.fit(train_x, train_y)
print(search.best_params_, search.best_score_)
```

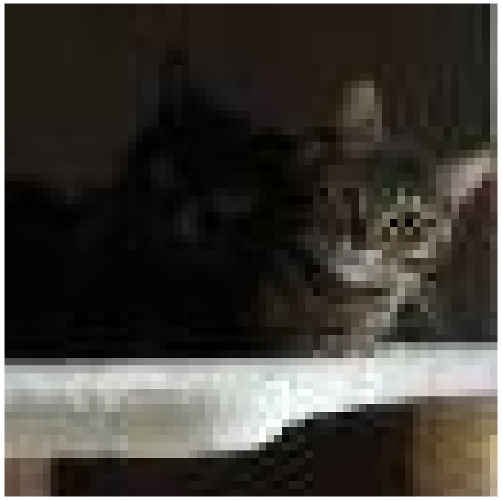
Supervised Learning
Non-tabular data

16x16x3



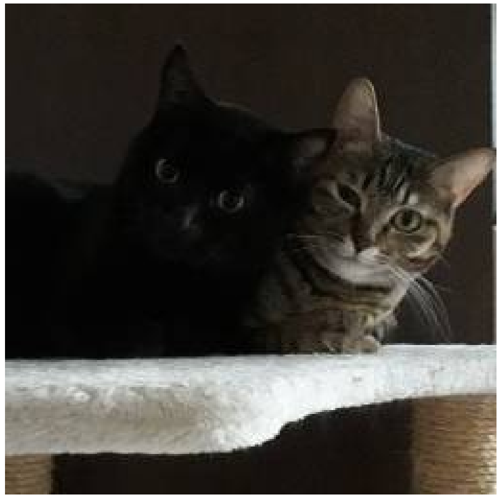
768 features

64x64x3



12288 features

256x256x3



196608 features

- Higher resolution – better quality. But more features – problem becomes paradoxically harder!

- Want one model that can be applied to multiple image resolutions

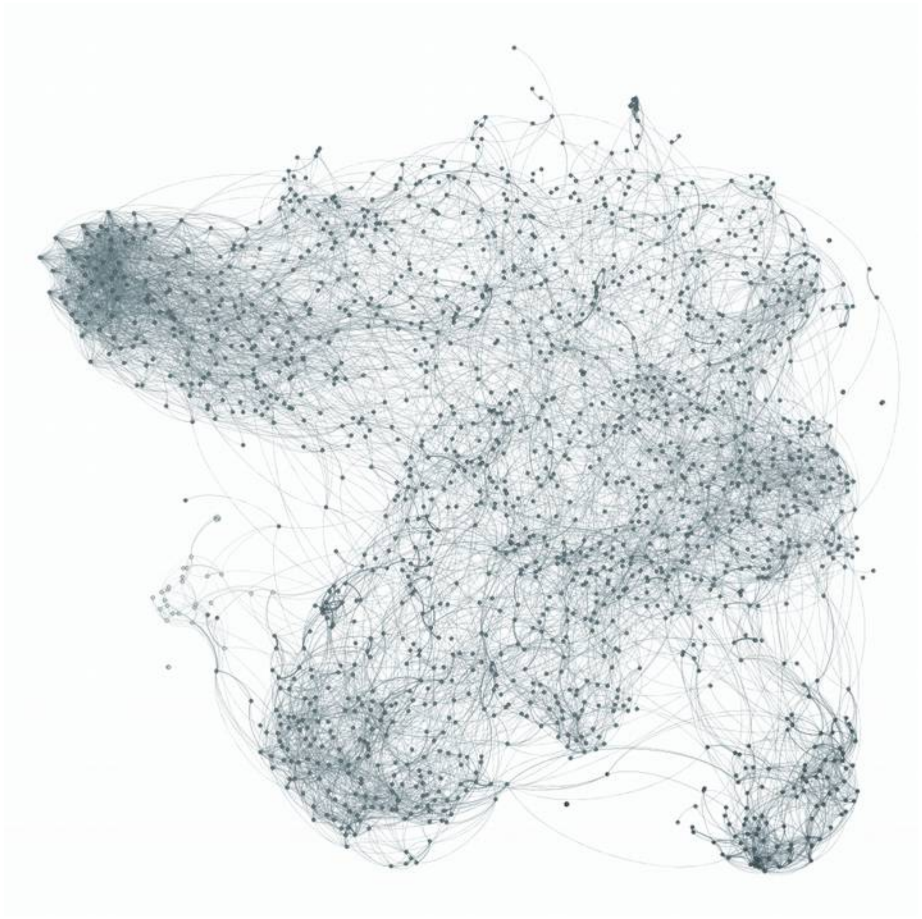
Supervised Learning
Non-tabular data

```
>QCW01532.1 membrane protein YoeI [Salmonella enterica subsp. enterica serovar
1,4,[5],12:i:-]
MGQFFAYATAFAVKENDHVA

>QCW00677.1 protein MgtR [Salmonella enterica subsp. enterica serovar 1,4,
[5],12:i:-]
MNRSPDKIIALIFLLISLLVLCLALWQIVF

>QCV97095.1 glucose uptake inhibitor SgrT [Salmonella enterica subsp. enterica
serovar 1,4,[5],12:i:-]
MRQFWLRYFAATEKTSWLACLSAPQRLKMLAELMQWEATD

>QCW00541.1 type I toxin-antitoxin system toxin Ldr family protein [Salmonella
enterica subsp. enterica serovar 1,4,[5],12:i:-]
MTLTQLGVVFWHDLAAPIIAGIIASVIVNWLRDRK
```

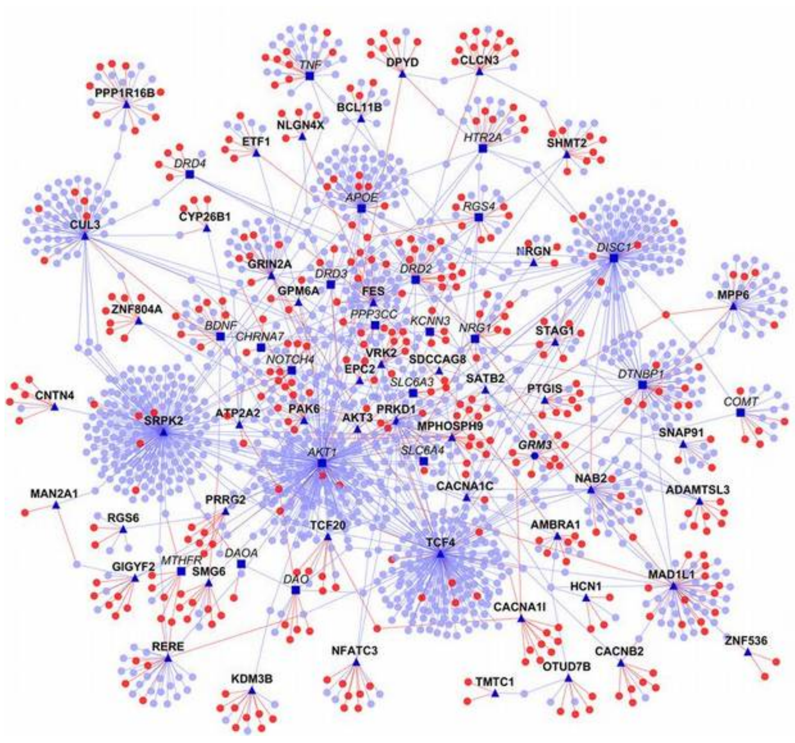
Sequences may have variable length.

Specific indices are not necessarily informative; relative indices. E.g. 87th letter of a book isn’t special.

Supervised Learning
Non-tabular data



http://colah.github.io/posts/2014-07-FFN-Graphs-Vis/



By Madhavicmu - Own work, CC BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?curid=48447204

How do I even...

Supervised Learning
Non-tabular data

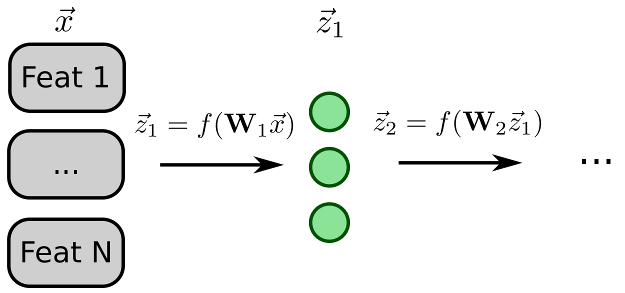
For these kinds of data, there are (roughly) two approaches:

- Come up with a set of statistical features that can be represented in tabular format to characterize each example.

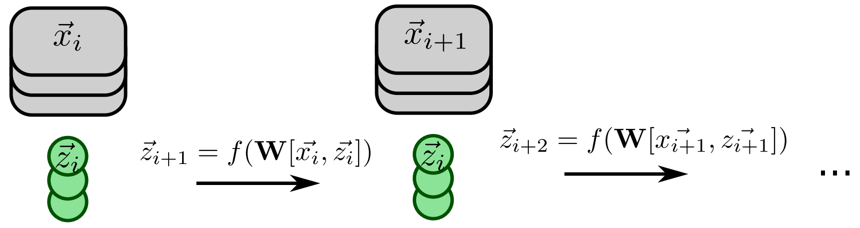
- Use a flexible framework (e.g. neural networks) that can optimize over input and output processing steps designed to accommodate the specific kind of data

Some neural network types

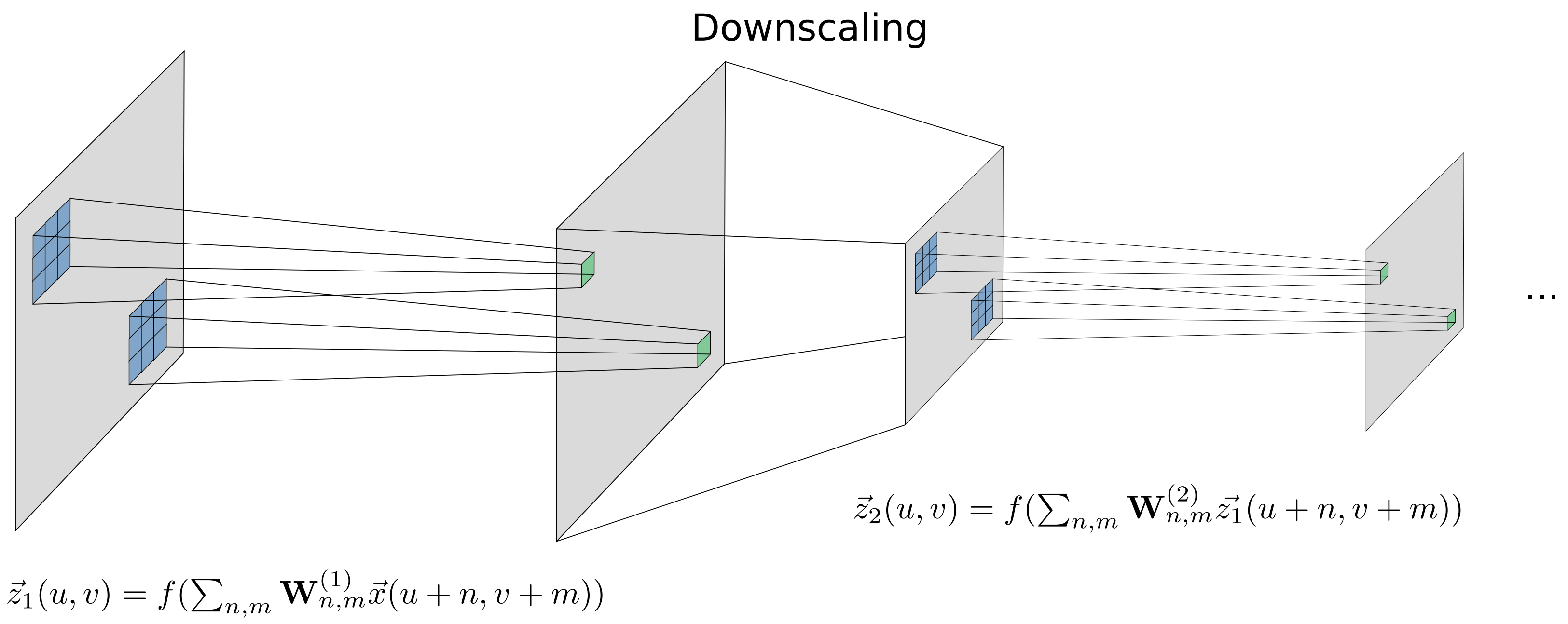
Neural network for tabular data ("dense")


$\vec{x}$
$\vec{z}_1$
Feat 1
...
Feat N
$\vec{z}_1 = f(\mathbf{W}_1 \vec{x})$
$\vec{z}_2 = f(\mathbf{W}_2 \vec{z}_1)$
...

Neural network for sequence data ("recurrent")


$\vec{x}_i$
$\vec{x}_{i+1}$
$\vec{z}_i$
$\vec{z}_{i+1} = f(\mathbf{W}[\vec{x_i}, \vec{z_i}])$
$\vec{z}_i$
$\vec{z}_{i+2} = f(\mathbf{W}[\vec{x_{i+1}}, \vec{z_{i+1}}])$
...

Neural network for image data ("convolutional")


Downscaling
...
$\vec{z}_2(u,v) = f(\sum_{n,m} \mathbf{W}^{(2)}_{n,m} \vec{z}_1(u+n, v+m))$
$\vec{z}_1(u,v) = f(\sum_{n,m} \mathbf{W}^{(1)}_{n,m} \vec{x}(u+n, v+m))$