Supervised Learning

Overview/Linear Regression Models

Sources: A. Mueller, ESLR, and R. Tibshirani

Lecture Overview

- Overview with key terms and basic model
- Least Squares walk through
- Ridge and Lasso Regression
- Code introduced along the way

Supervised Learning

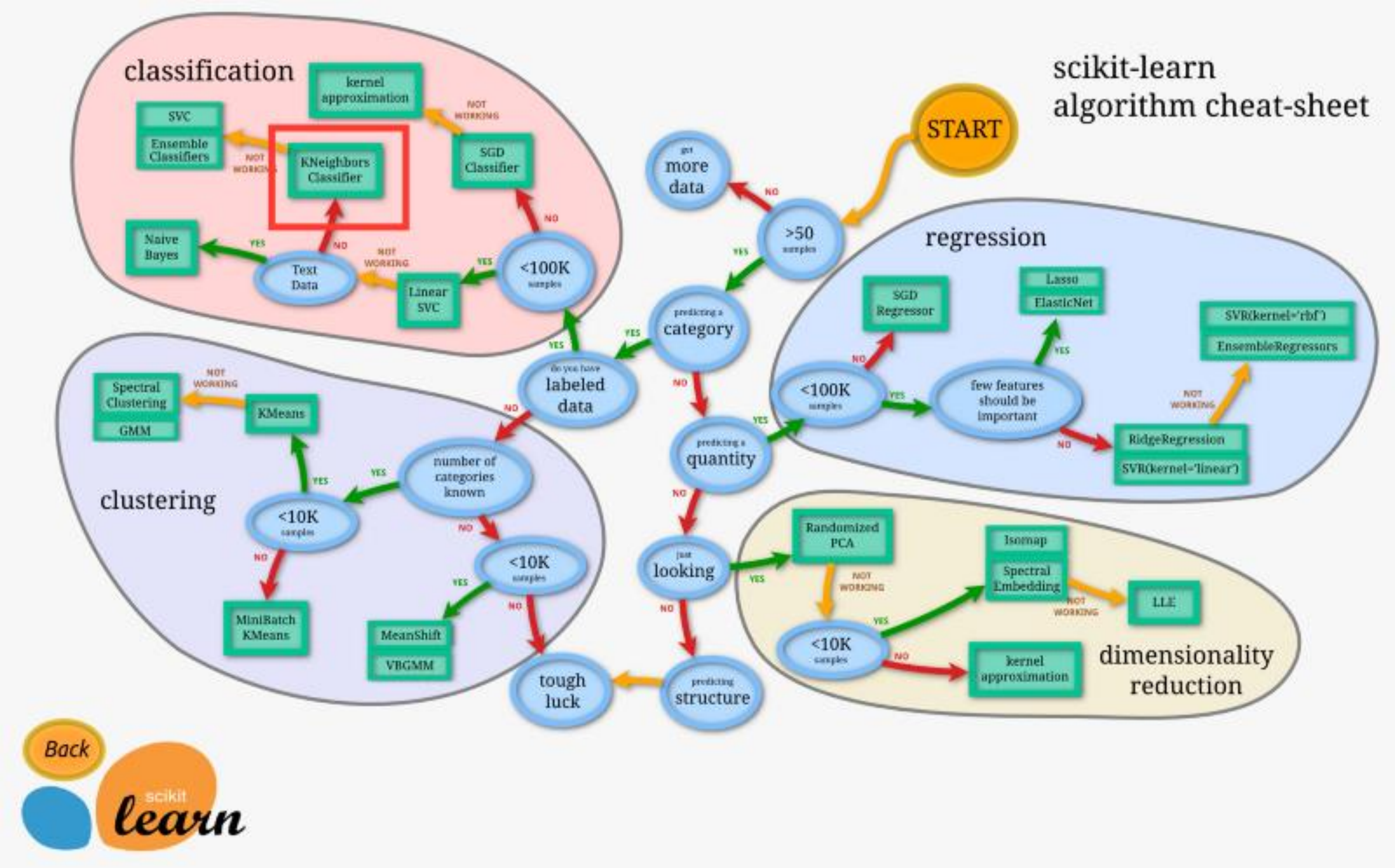
$$(x_i, y_i) \propto p(x, y) \text{ i.i.d.}$$

$$x_i \in \mathbb{R}^p$$

$$y_i \in \mathbb{R}$$

$$f(x_i) \approx y_i$$

$$f(x) \approx y$$


scikit-learn
algorithm cheat-sheet
START
>50 samples
get more data
predicting a category
do you have labeled data
predicting a quantity
just looking
predicting structure
tough luck
classification
kernel approximation
SVC
Ensemble Classifiers
KNeighbors Classifier
SGD Classifier
Naive Bayes
Text Data
Linear SVC
<100K samples
clustering
Spectral Clustering
GMM
KMeans
number of categories known
<10K samples
MiniBatch KMeans
MeanShift
VBGMM
regression
SGD Regressor
Lasso
ElasticNet
SVR(kernel='rbf')
EnsembleRegressors
few features should be important
RidgeRegression
SVR(kernel='linear')
dimensionality reduction
Randomized PCA
Isomap
Spectral Embedding
LLE
YES
NO
NOT WORKING
Back
scikit
learn

K nearest neighbors

- A simple algorithm:

1. Define a measurement of similarity between observations in dataset

   e.g.- Euclidean Distance

2. Define “k” parameter (i.e.- number of nearest neighbors to evaluate in training data)

K nearest neighbors

- A simple algorithm:

Model makes prediction with new test observation by….

3. Calculate observation’s similarity to all observations in training data using distance measurement

4. Find k nearest neighbors in training data.

5. Assign category from majority of these neighbors to observation

(e.g. – take majority vote)

K nearest neighbors

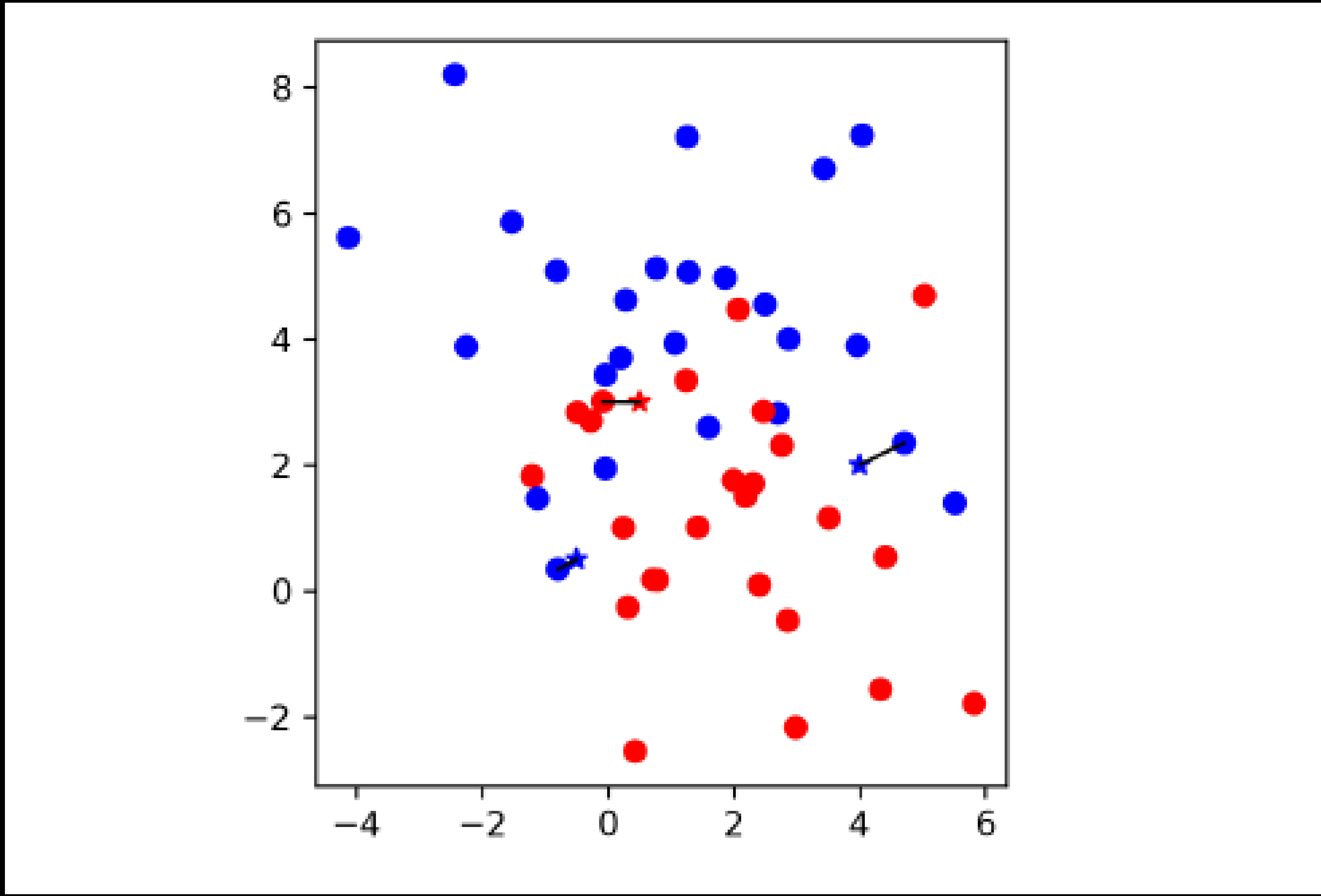
Euclidean distance:

$$d(x, x') = \sqrt{(x_1 - x'_1)^2 + (x_2 - x'_2)^2 + \ldots + (x_n - x'_n)^2}$$

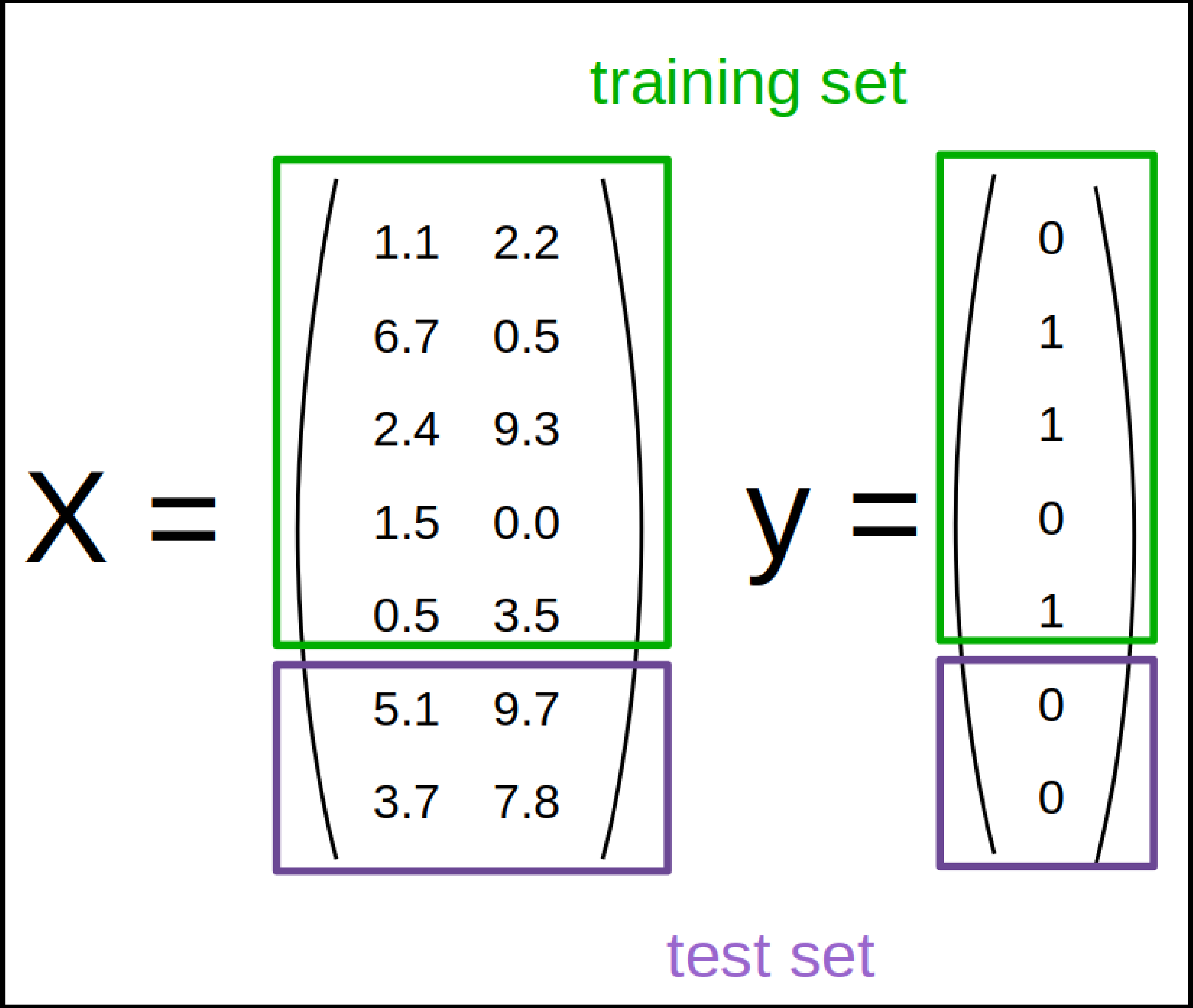
For any two observations in data, subtract one row from the other, square result, add together, then take square root

Similar observations have smaller distances

Nearest Neighbors



Nearest Neighbors



Nearest Neighbors

KNN with scikit-learn

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
print("accuracy: {:.2f}".format(knn.score(X_test, y_test)))
y_pred = knn.predict(X_test)
```

accuracy: 0.77

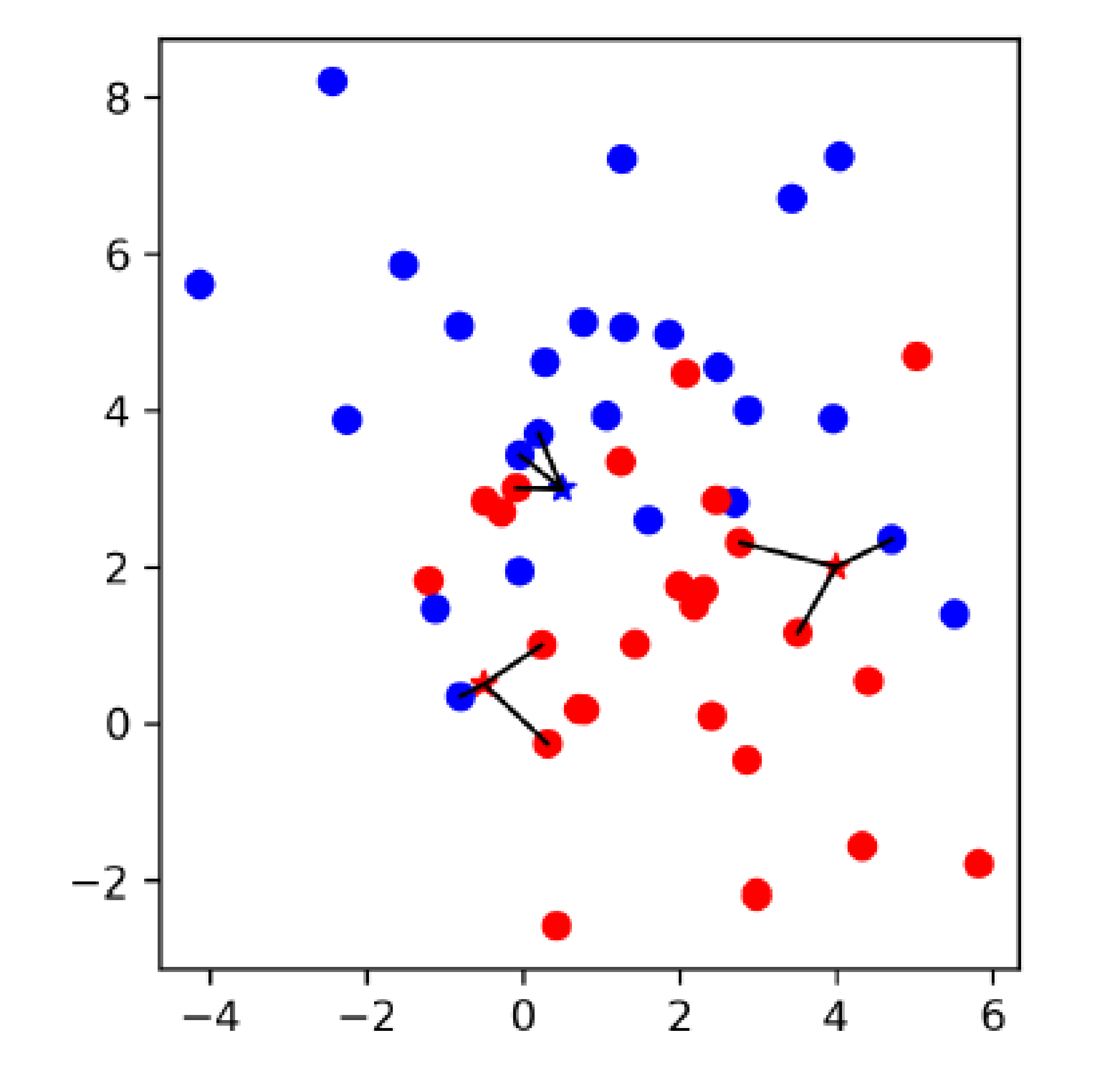
Nearest Neighbors

KNN with scikit-learn

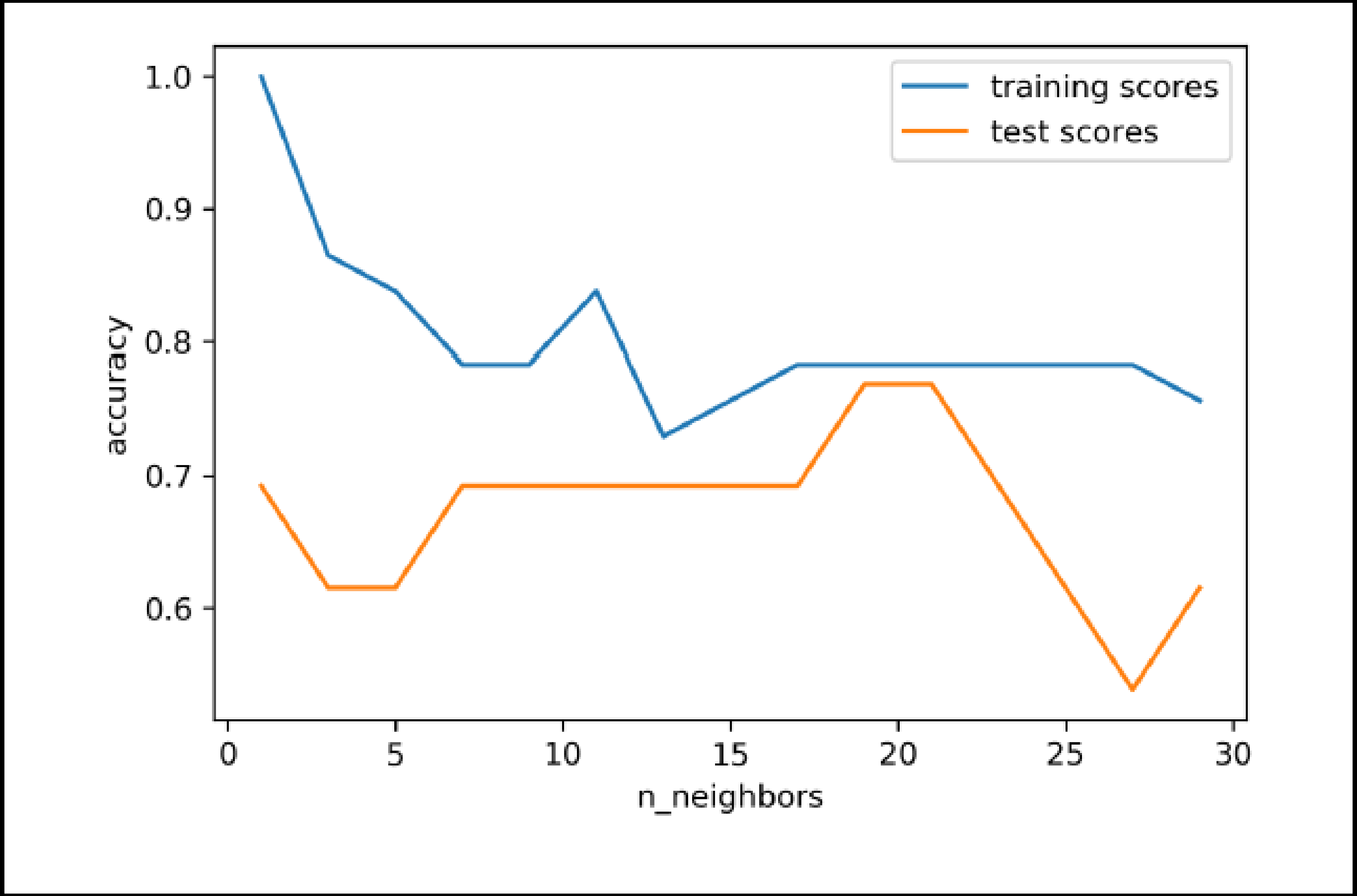
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
print("accuracy: {:.2f}".format(knn.score(X_test, y_test)))
y_pred = knn.predict(X_test)
```

accuracy: 0.77

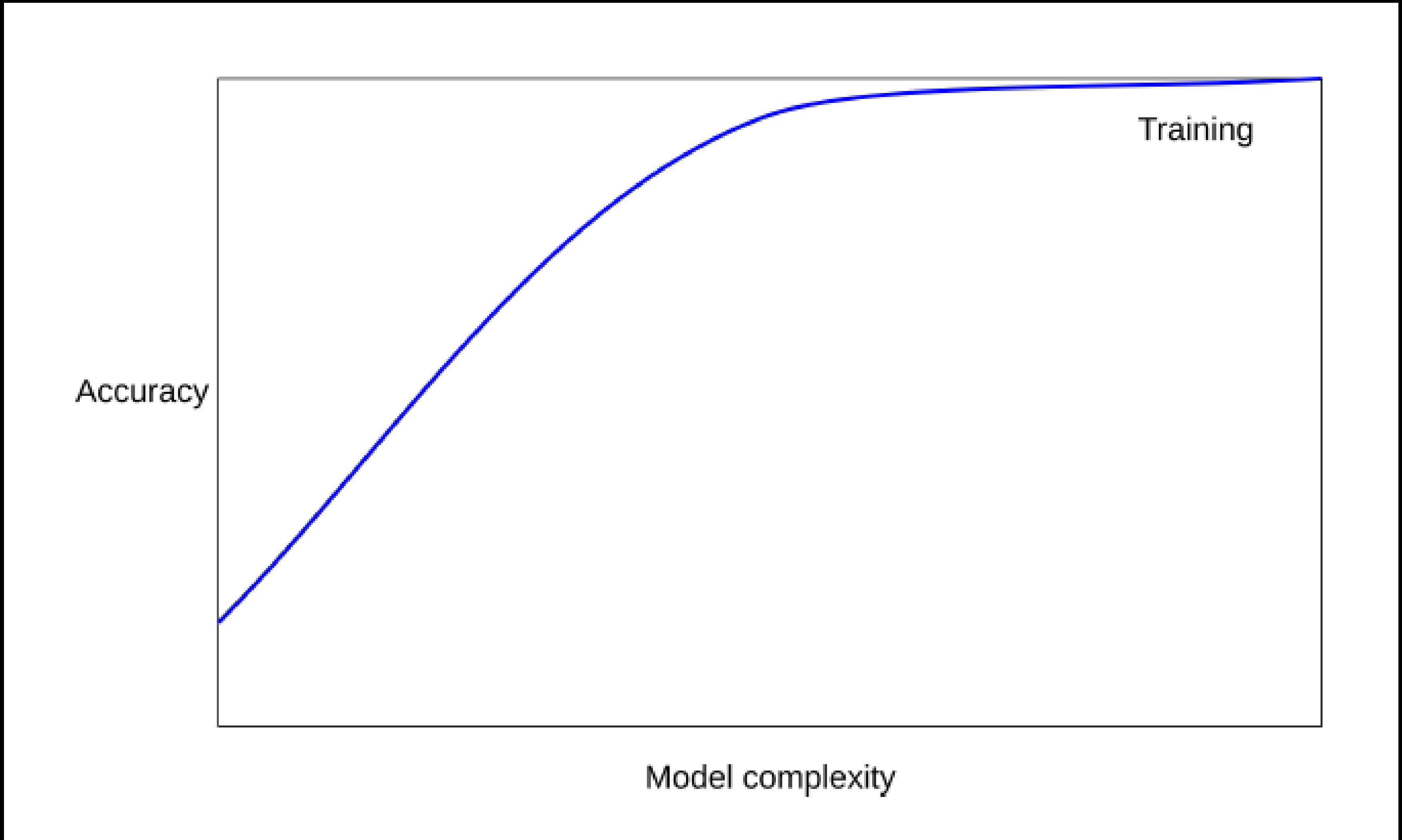
More Neighbors



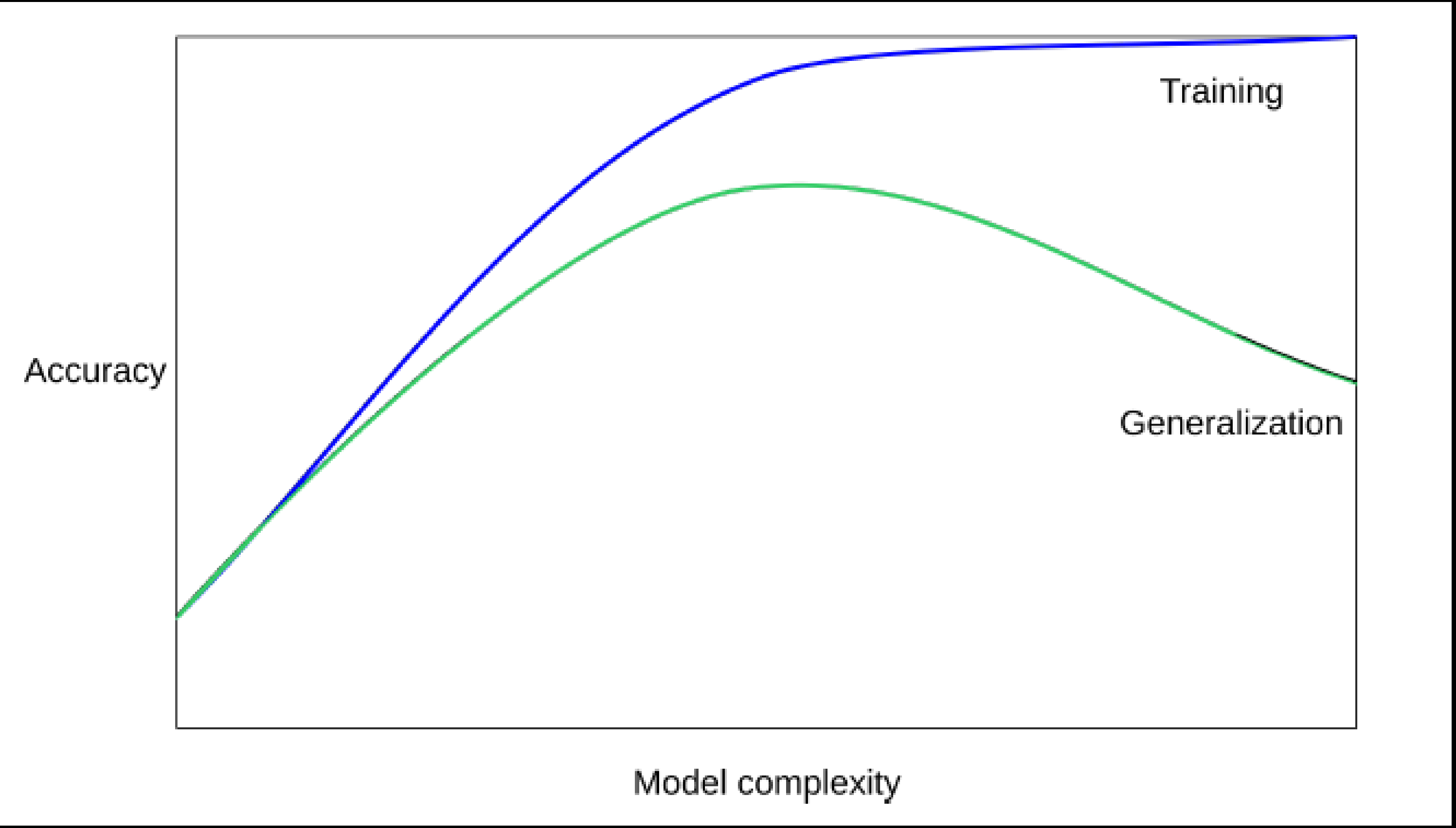
More Neighbors



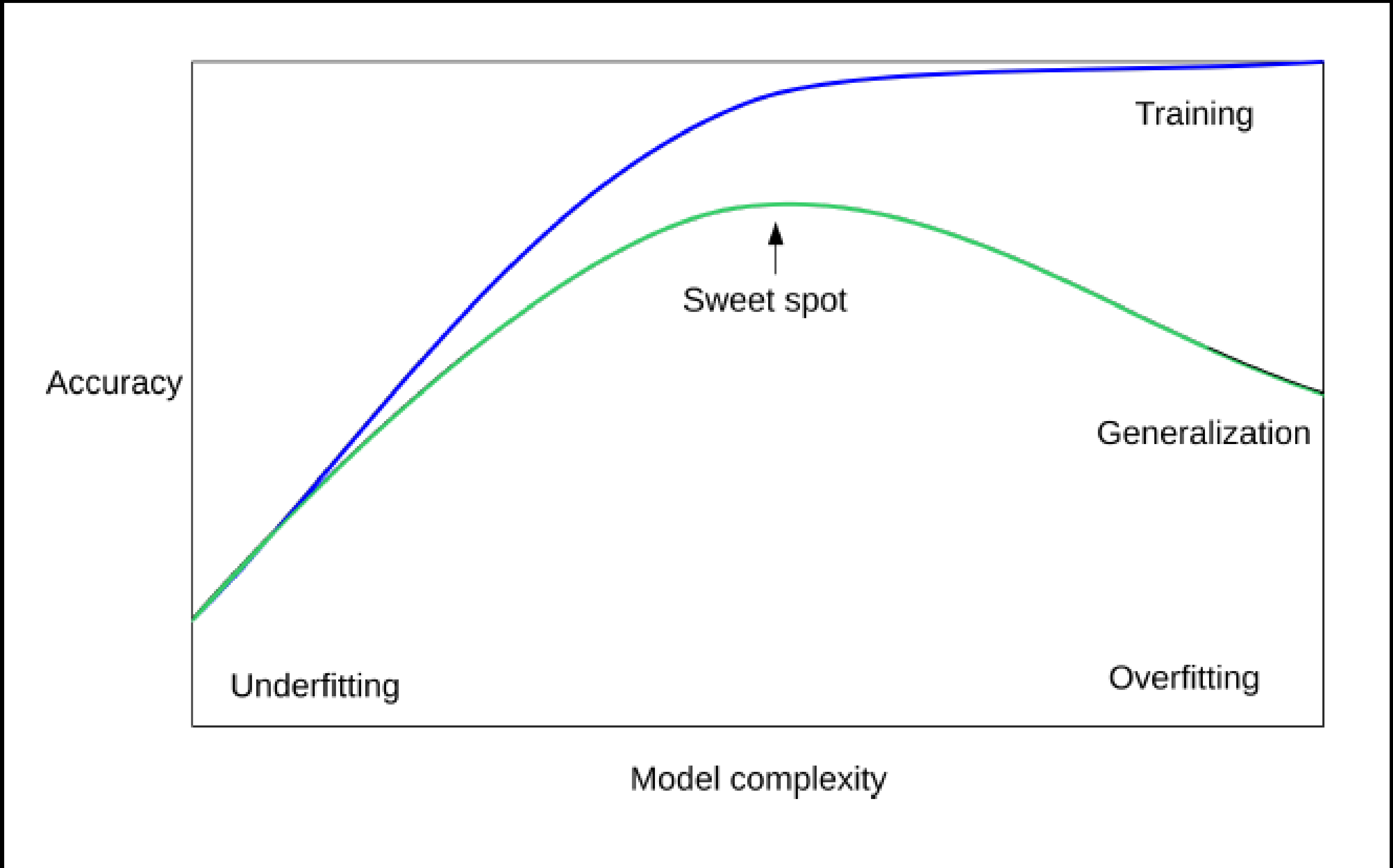
Overfitting and Underfitting



Overfitting and Underfitting



Overfitting and Underfitting



Other considerations for KNN Models

- Practically speaking, it's better to…
  - Scale your IVs before taking distance calculations
    - Use z scores for example
  - Also good to use odd number for k to break any ties in majority votes

Other considerations for KNN Models

- Minimal training but expensive testing.
  - Training model is fast/easy
  - Testing model, however, requires large training dataset to draw from each time for model predictions

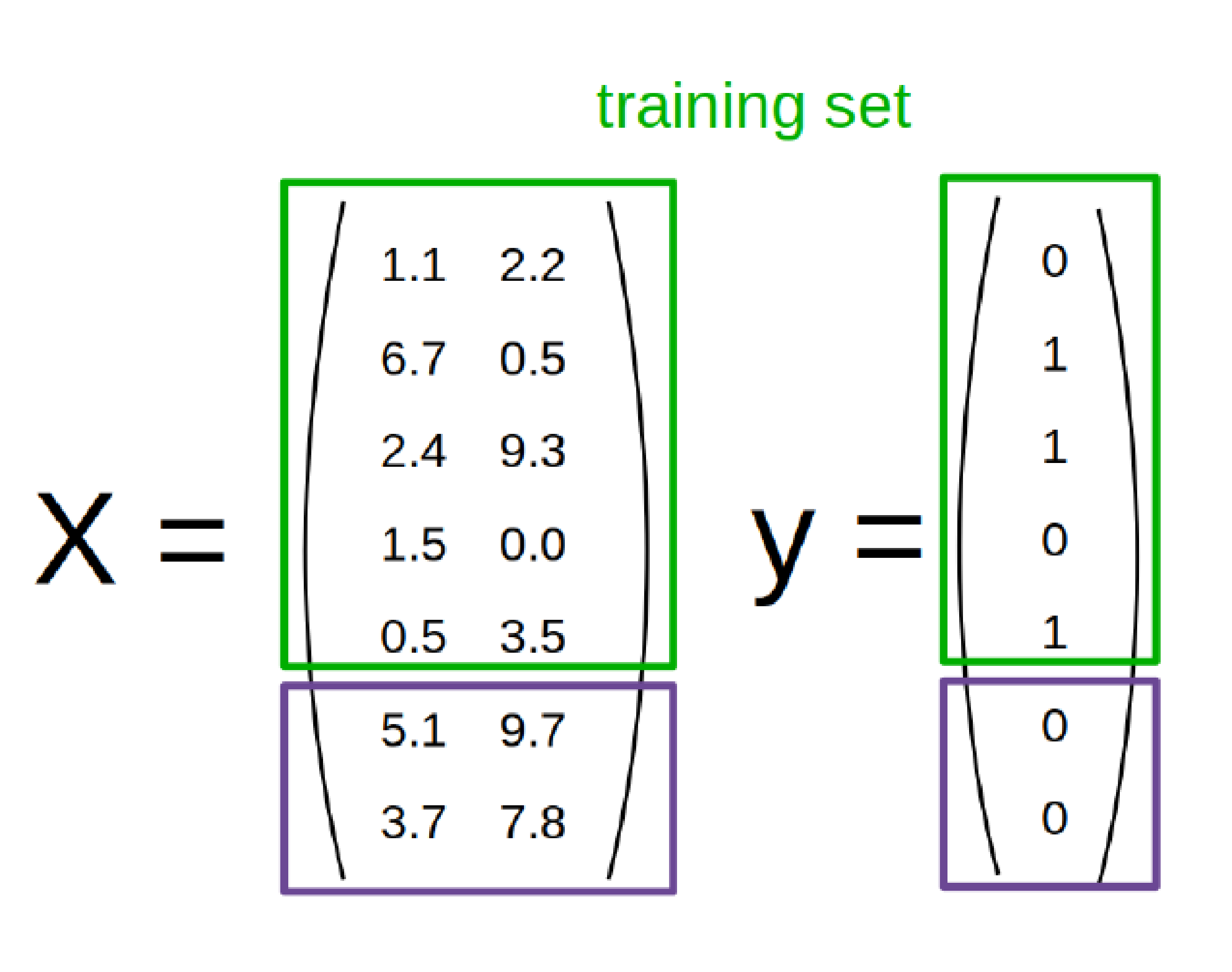
KNN regression model

- Same algorithm as classification, except..
  - Take average value of y for nearest neighbors
  - **Use diff't code:**
    - KNeighborsRegressor(n_neighbors=3)

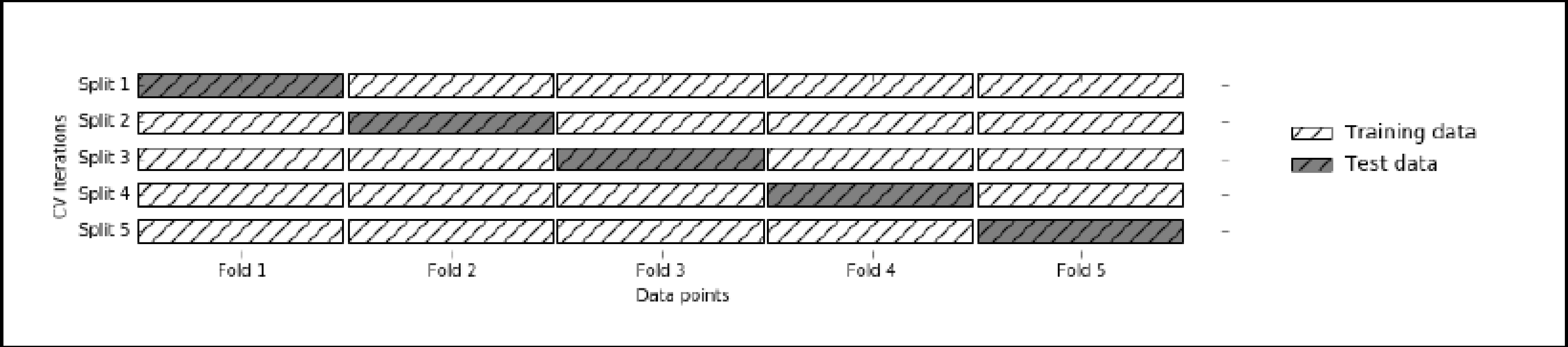
Parametric and non-parametric models

- Parametric models
  - Assume functional form of population parameters model is trying to examine
    - E.g. Gaussian/Poisson/Binomial/etc.
  - Prediction relies on parameters learned from training data
- Non-parametric models
  - No explicit assumptions about the functional form
  - Assume test data has same functional form as training data, however
  - Prediction can lean heavily on training data

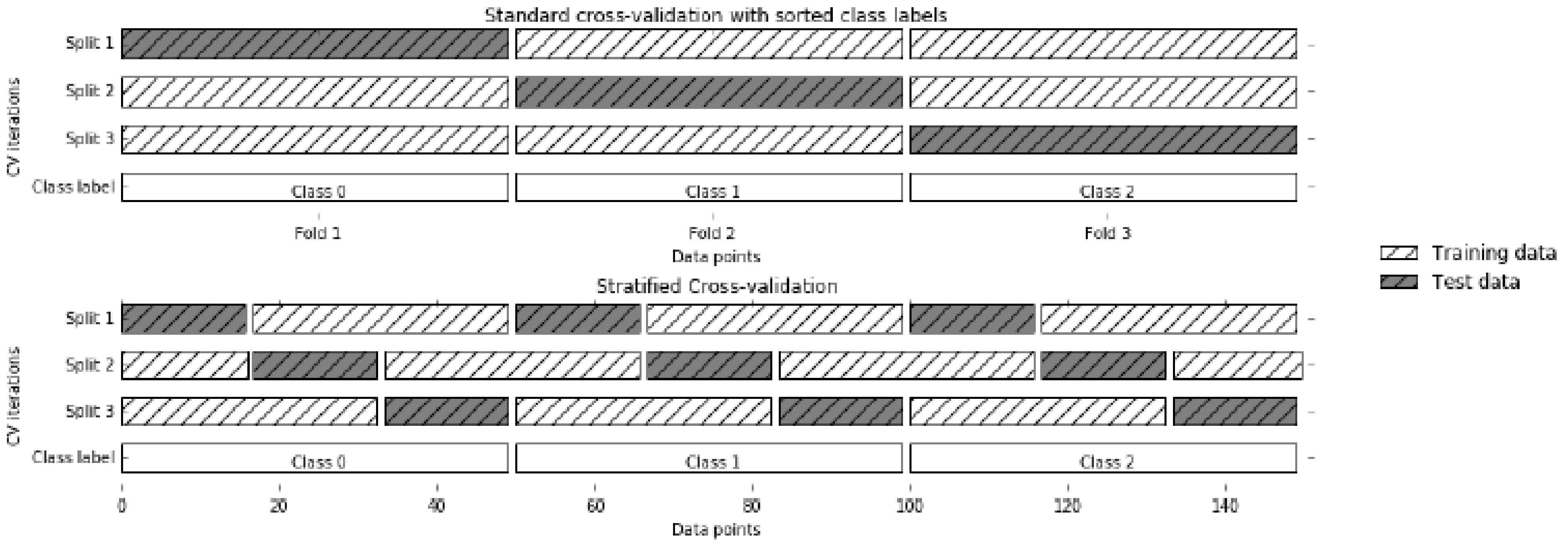
Model Evaluation



K-fold Cross Validation



Stratified K-fold Cross Validation



Stratified: Ensure relative class frequencies in each fold reflect relative class frequencies on the whole dataset.

Stratified K-fold Cross Validation

- LeaveOneOut : KFold(n_folds=n_samples) Highvariance, takes a long time
- Better: RepeatedKFold. Apply KFold or StratifiedKFold multiple times with shuffled data. Reduces variance!
  - E.g. – You do 5 fold CV 10 times with data randomly shuffled before each 5 fold CV

Defaults for CV in scikit-learn

- Three-fold is default number of folds
- For classification cross-validation is stratified
- train_test_split has stratify option: train_test_split(X, y,
- stratify=y)
- No shuffle for repeat sampling by default!

Tuning model hyper-parameters with CV

- For KNN we can choose many values for k

- How do we choose a reasonable value?

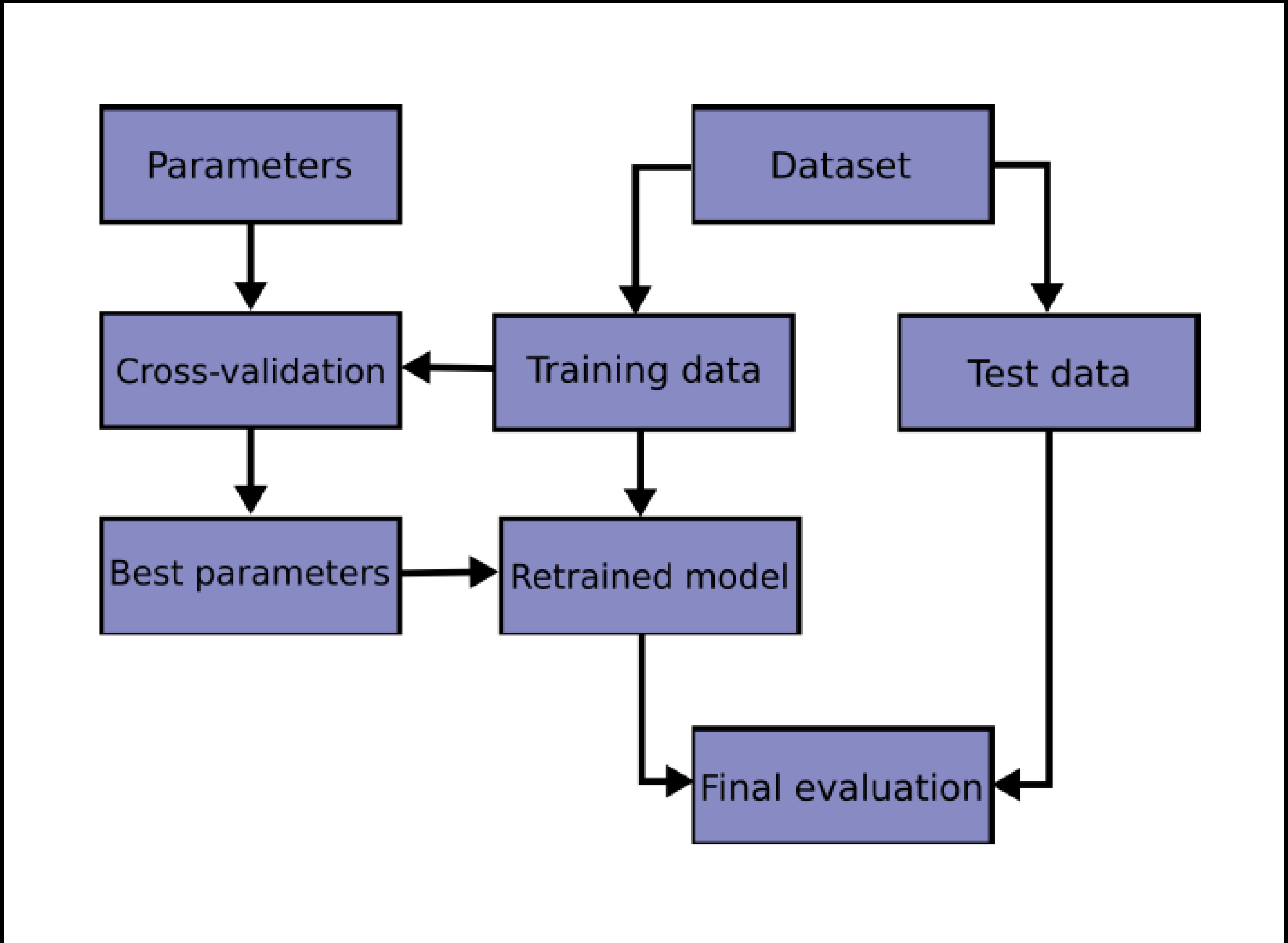
    - Generate models for lots of different parameters
    - Evaluate them using CV
    - Choose model that predicts new data best

Grid Search with CV using Loop:

```
from sklearn.model_selection import cross_val_score

X_train, X_test, y_train, y_test = train_test_split(X, y)
cross_val_scores = []

for i in neighbors:
    knn = KNeighborsClassifier(n_neighbors=i)
    scores = cross_val_score(knn, X_train, y_train, cv=10)
    cross_val_scores.append(np.mean(scores))

print("best cross-validation score: {:.3f}".format(np.max(cross_val_scores)))
best_n_neighbors = neighbors[np.argmax(cross_val_scores)]
print("best n_neighbors: {}".format(best_n_neighbors))

knn = KNeighborsClassifier(n_neighbors=best_n_neighbors)
knn.fit(X_train, y_train)
print("test-set score: {:.3f}".format(knn.score(X_test, y_test)))
```

```
best cross-validation score: 0.967
best n_neighbors: 9
test-set score: 0.965
```

Conceptual Overview



GridSearchCV

```
from sklearn.model_selection import GridSearchCV

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y)


param_grid = {'n_neighbors':  np.arange(1, 15, 2)}

grid = GridSearchCV(KNeighborsClassifier(), param_grid=param_grid, cv=10)
grid.fit(X_train, y_train)

print("best mean cross-validation score: {:.3f}".format(grid.best_score_))
print("best parameters: {}".format(grid.best_params_))

print("test-set score: {:.3f}".format(grid.score(X_test, y_test)))
```

```
best mean cross-validation score: 0.967
best parameters: {'n_neighbors': 9}
test-set score: 0.993
```

GridSearchCV

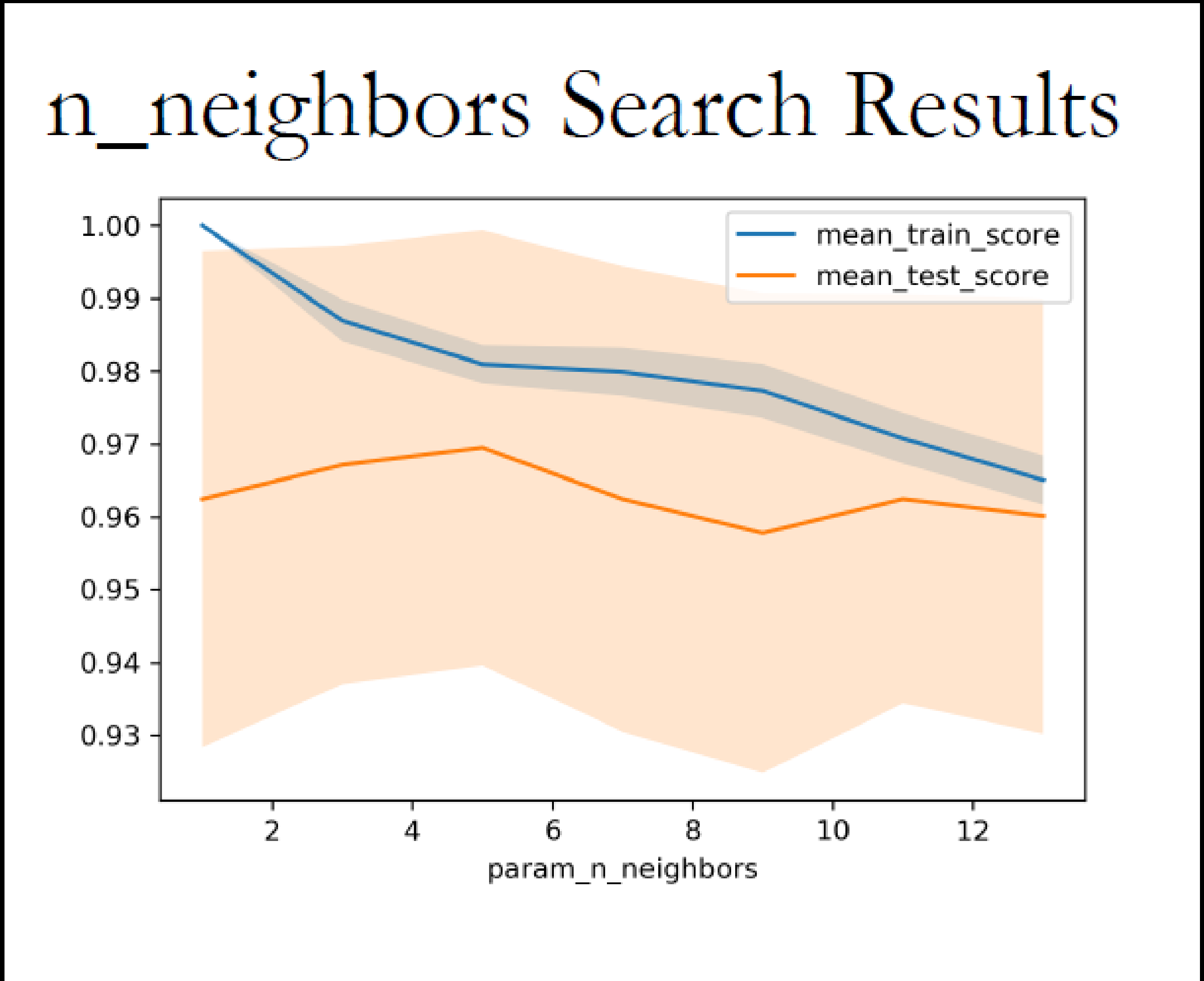
```
import pandas as pd
results = pd.DataFrame(grid.cv_results_)
results.columns
```

```
Index(['mean_fit_time', 'mean_score_time', 'mean_test_score',
       'mean_train_score', 'param_n_neighbors', 'params', 'rank_test_score',
       'split0_test_score', 'split0_train_score', 'split1_test_score',
       'split1_train_score', 'split2_test_score', 'split2_train_score',
       'split3_test_score', 'split3_train_score', 'split4_test_score',
       'split4_train_score', 'split5_test_score', 'split5_train_score',
       'split6_test_score', 'split6_train_score', 'split7_test_score',
       'split7_train_score', 'split8_test_score', 'split8_train_score',
       'split9_test_score', 'split9_train_score', 'std_fit_time',
       'std_score_time', 'std_test_score', 'std_train_score'],
      dtype='object')
```

```
results.params
```

```
0     {'n_neighbors': 1}
1     {'n_neighbors': 3}
2     {'n_neighbors': 5}
3     {'n_neighbors': 7}
4     {'n_neighbors': 9}
5    {'n_neighbors': 11}
6    {'n_neighbors': 13}
Name: params, dtype: object
```

GridSearchCV



Conclusion of Sup. Learning Overview

Questions before we move on?

Part 2 of Lecture: Linear regression models

- Least Squares walk through
- Ridge and Lasso Regression
- Code introduced along the way