Computer Science

CSC380: Principles of Data Science

Basic machine learning 2

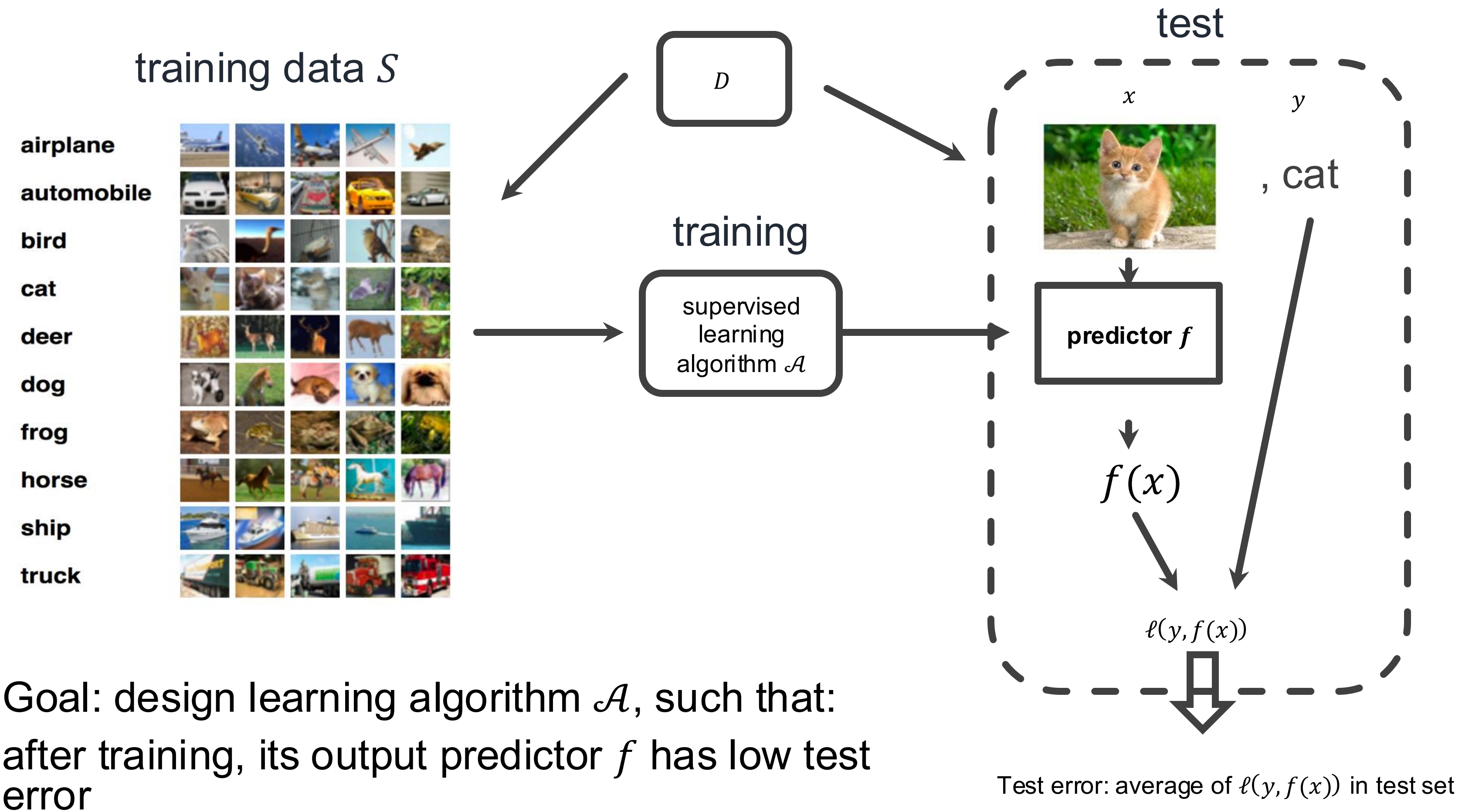
Xinchen Yu

Outline

- Classification basics
- Nearest neighbor Classification
- Logistic regression
- Classification: other considerations
    - Binary classification beyond accuracy
    - Multiclass classification

Classification recap

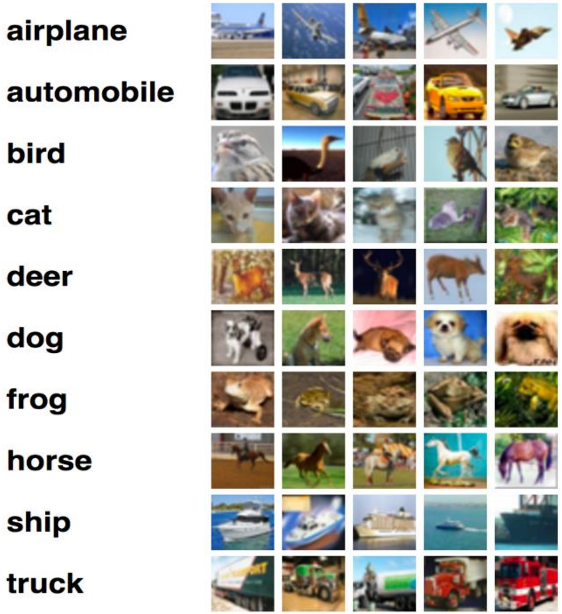
Supervised learning setup in one figure



- Goal: design learning algorithm $\mathcal{A}$, such that:
- after training, its output predictor f has low test error

Classification

- The labels are categorical
- Loss function ℓ: measures the quality of prediction $\hat{y}$ respect to true label y
  - $\ell(y, \hat{y}) = I(y \neq \hat{y})$
  - I: indicator of predicate; 1 if true; 0 if false



- A classifier f's error on a dataset S is the fraction of examples in S that it predicts incorrectly.
  - f's training / test error is its error on training / test set
  - Accuracy = 1 – error

In-class activity: finding test error

A company develops a simple **spam classifier** f that predicts whether an email is **spam (1)** or **not spam (0)** based on the number of capital letters in the subject line.

f outputs **Spam** if the number of capital letters ≥ 5, and **Not Spam** otherwise.

Suppose the test dataset is as follows. Find f's test error.

Subject	True label	Predicted label
"WIN A FREE VACATION NOW!!!"	1	1
Meeting rescheduled to 3 PM	0	0
"HUGE DISCOUNT ON ALL ITEMS!!!"	1	1
URGENT: Please submit your report	0	1
Can you review this document?	0	0

f's test error = 1/5 = 20%

Nearest Neighbor Classification

Example: Course Recommendation

Rating	Easy?	AI?	Sys?	Thy?	Morning?
+2	y	y	n	y	n
+2	y	y	n	y	n
+2	n	y	n	n	n
+2	n	n	n	y	n
+2	n	y	y	n	y
+1	y	y	n	n	n
+1	y	y	n	y	n
+1	n	y	n	y	n
0	n	n	n	n	y
0	y	n	n	y	y
0	n	y	n	y	n
0	y	y	y	y	y
-1	y	y	y	n	y
-1	n	n	y	y	n
-1	n	n	y	n	y
-1	y	n	y	n	y
-2	n	n	y	y	n
-2	n	y	y	n	y
-2	y	n	y	n	n
-2	y	n	y	n	y

Label: "like"

Label: "dislike"

Features

Suppose we'd like to build a recommendation system for classes

We've collected information about many past classes

We can frame this as a classification problem:

Predict like/dislike from class features

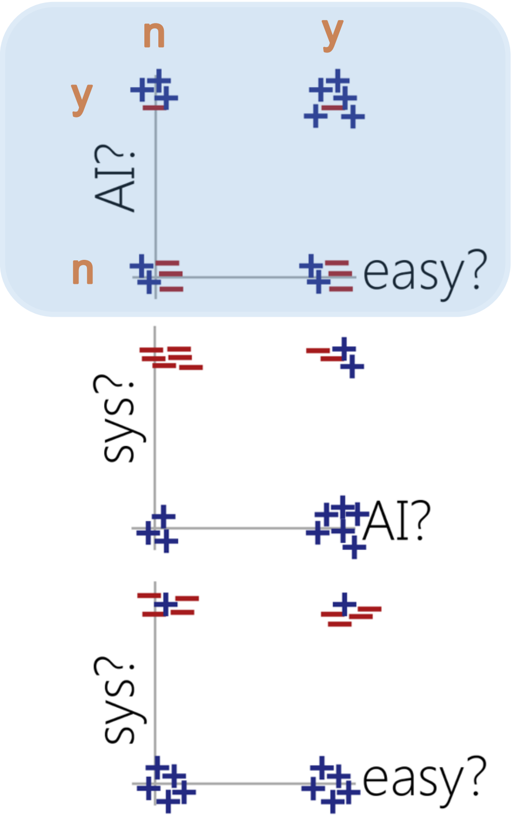
Example: Course Recommendation

Label: +

Label: -

Rating	Easy?	AI?	Sys?	Thy?	Morning?
+2	y	y	n	y	n
+2	y	y	n	y	n
+2	n	y	n	n	n
+2	n	n	n	y	n
+2	n	y	y	n	y
+1	y	y	n	n	n
+1	y	y	n	y	n
+1	n	y	n	y	n
0	n	n	n	n	y
0	y	n	n	y	y
0	n	y	n	y	n
0	y	y	y	y	y
-1	y	y	y	n	y
-1	n	n	y	y	n
-1	n	n	y	n	y
-1	y	n	y	n	y
-2	n	n	y	y	n
-2	n	y	y	n	y
-2	y	n	y	n	n
-2	y	n	y	n	y

Features



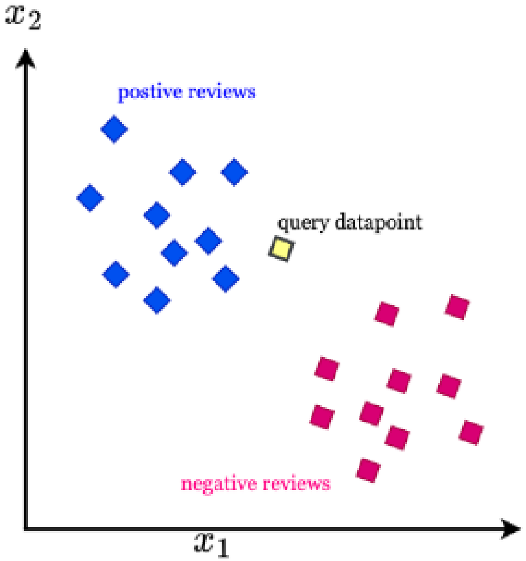
Each course's feature is Represented as points in 5-dimensional space

That's too many dimensions to plot…so we look at 2D projections…

Observation: examples with same labels tend to be closer!

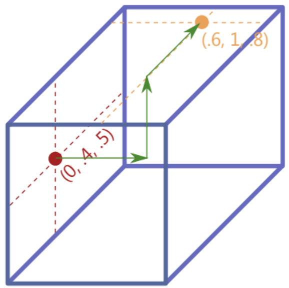
Nearest neighbor classification

- Given a new course, would like to predict its label (+/-)
- Idea: Find its most similar course in the training set, and use that course's label to predict



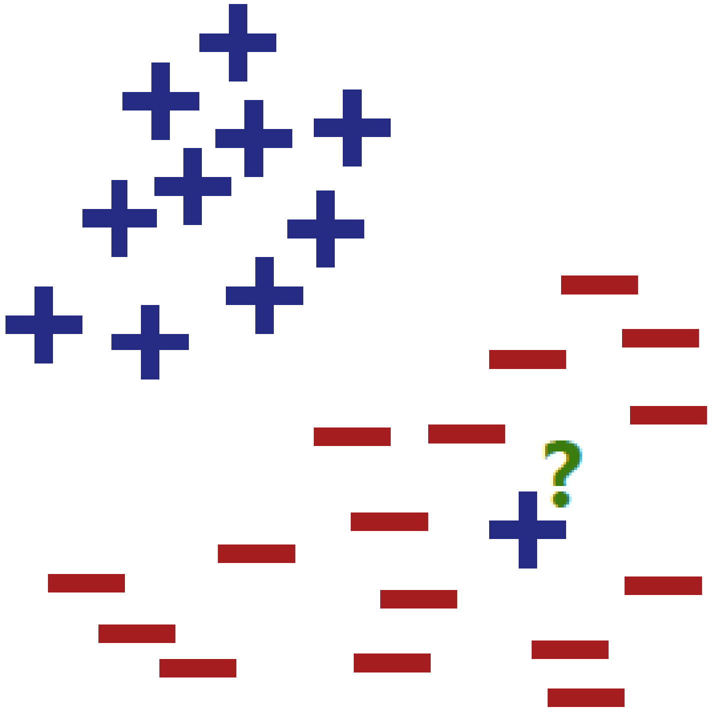
Measuring nearest neighbors

- Oftentimes convenient to work with feature $x \in \mathrm{R}^d$
- Distances in R^d: notation $x(f)$: $x = (x(1), \dots, x(d))$
  - (popular) Euclidean distance $d_2(x, x') = \sqrt{\sum_{f=1}^{d} \left(x(f) - x'(f)\right)^2}$
  - Manhattan distance $d_1(x, x') = \sum_{f=1}^{d} |x(f) - x'(f)|$



- How to extract features as **real values**?
  - Boolean features: {Y, N} -> {0,1}
  - Categorical features: {Red, Blue, Green, Black}
    - Convert to {1, 2, 3, 4}?
    - Better one-hot encoding: (1,0,0,0), .., (0,0,0,1) (IsRed?/isGreen?/isBlue?/IsBlack?)

Robustify Nearest Neighbor Classification



Q: Can we predict using 1 nearest neighbor's?

Query point ? Will be classified as + but should be -

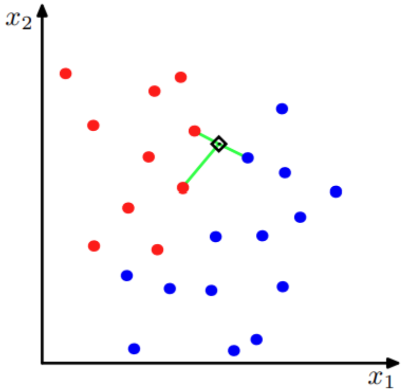
Problem: predicting using 1 nearest neighbor's label can be sensitive to noisy data

How to mitigate this?

k-nearest neighbors (k-NN): main concept

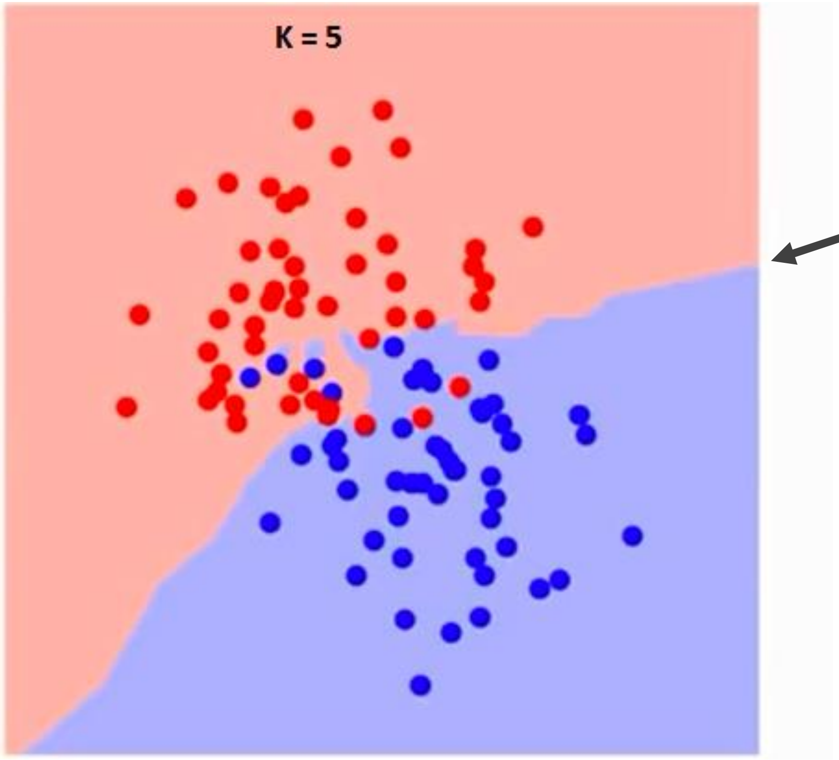
- Training set: $S = \{ (x_1, y_1), \dots, (x_m, y_m) \}$

- **Key insight**: given test example x, its label should resemble the labels of *nearby points*



- Function
  - input: x
  - find the k nearest points to x from S; call their indices $N(x)$
  - output:
    - (classification) the majority vote of $\{y_i : i \in N(x)\}$
    - (regression) the average of $\{y_i : i \in N(x)\}$

k-NN classification example



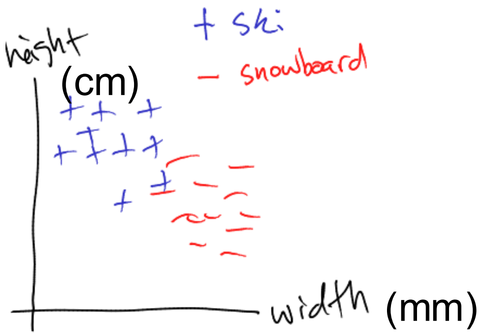
decision boundary

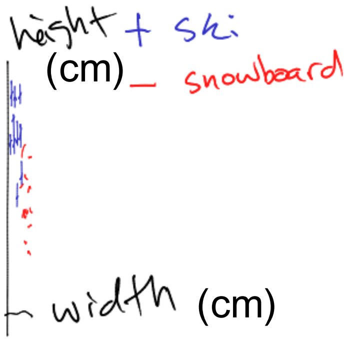
Issue 1: scaling

- Features having different scales can be problematic.

- Ex: ski vs. snowboard classification

$$d = \sqrt{(height_1 - height_2)^2 + (weight_1 - weight_2)^2}$$





- One solution: feature standardization

Make sure features are scaled fairly

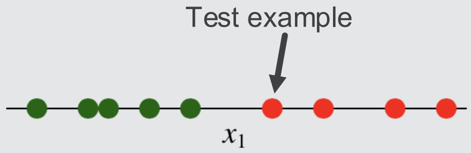
- Features having different scale can be problematic
- [Definition] **Standardization**
  - For each feature f, compute $\mu_f = \frac{1}{m}\sum_{i=1}^{m} x_f^{(i)}$, $\sigma_f = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(x_f^{(i)} - \mu_f\right)^2}$
  - Then, transform the data by $\forall f \in \{1, \dots, d\}, \forall i \in \{1, \dots, m\}$, $x_f^{(i)} \leftarrow \frac{x_f^{(i)} - \mu_f}{\sigma_f}$

after transformation, each feature has mean 0 and variance 1

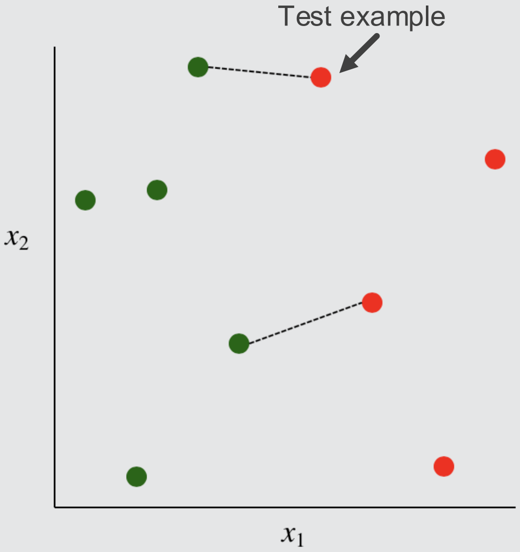
- Be sure to keep the "standardize" function and apply it to the test points.
  - Save $\{(\mu_f, \sigma_f)\}_{f=1}^{d}$
  - For test point x^*, apply $x_f^* \leftarrow \frac{x_f^* - \mu_f}{\sigma_f}, \forall f$

Issue 2: irrelevant features

here's a case in which there is one relevant feature x_1 and a 1-NN rule classifies each instance correctly



consider the effect of an irrelevant feature x_2 on distances and nearest neighbors

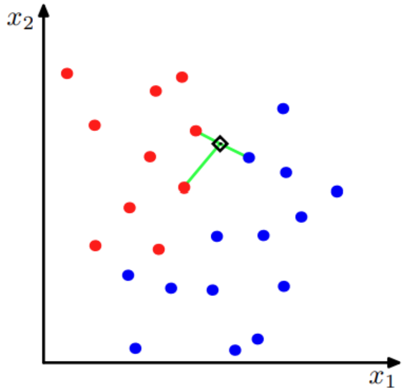


- Mitigation: feature selection

Issue 3: choosing k

- Q: How would a k-NN classifier predict when k=training set size?
  - Predict majority label everywhere
  - Underfitting

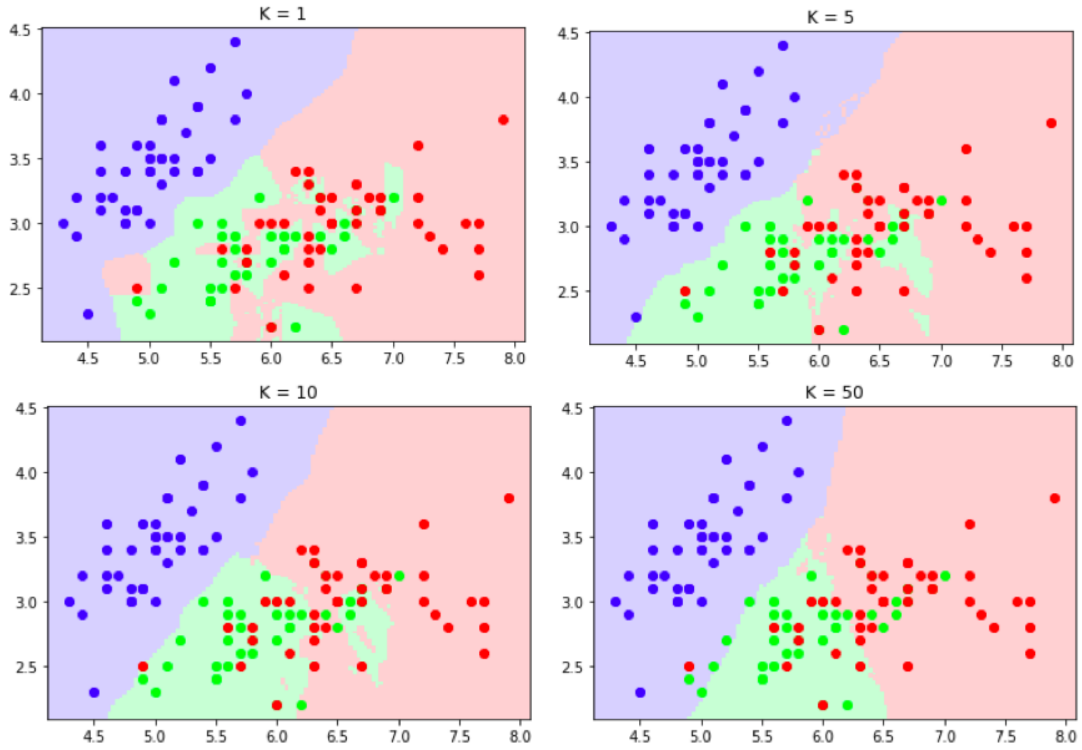
- Q: What is the training error of a 1-NN classifier?
  - 0
  - Overfitting



Issue 3: choosing k

k can be viewed as a model complexity measure

Smaller k results in a more complex model

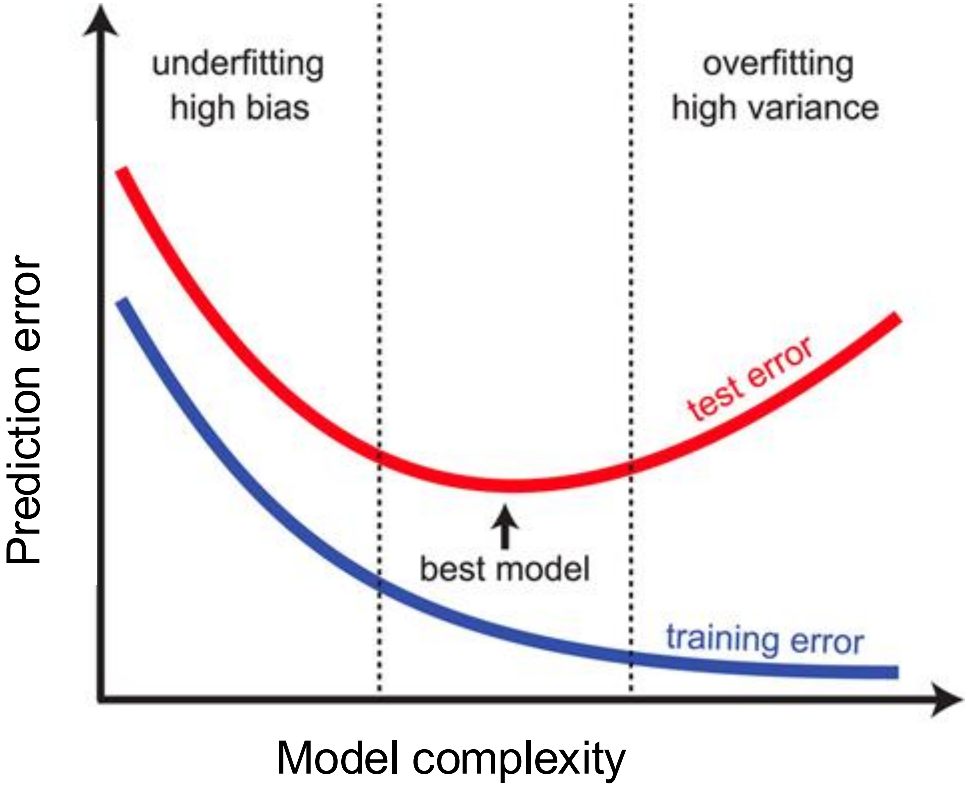


Issue 3: choosing k

We'd like to choose appropriate k to balance model bias and complexity

We can choose k in the same way we chose λ in ridge regression

- Cross validation



Scikit-learn nearest neighbors

class sklearn.neighbors.NearestNeighbors(**, n_neighbors=5, radius=1.0, algorithm='auto', leaf_size=30, metric='minkowski', p=2, metric_params=None, n_jobs=None*) [source]

Unsupervised learner for implementing neighbor searches.

```python
# 1. Load the Iris dataset
iris = load_iris()
X = iris.data  # Features
y = iris.target  # Target labels (species)

# 2. Split the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 3. Create the KNN classifier model
knn = KNeighborsClassifier(n_neighbors=3)  # Use 3 nearest neighbors

# 4. Train the model on the training data
knn.fit(X_train, y_train)
```

Scikit-learn nearest neighbors

```
# 5. Make predictions on the test set
y_pred = knn.predict(X_test)

# 6. Evaluate the model using accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of the KNN model: {accuracy * 100:.2f}%')

# Optionally, display the predictions vs. actual values
print(f'Predictions: {y_pred}')
print(f'Actual: {y_test}')
```

```
Accuracy of the KNN model: 100.00%
Predictions: [1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0]
Actual: [1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0]
```