Introduction to Scikit-Learn: Machine Learning with Python

Validation and Model Selection

郭耀仁

Validation

About validation

One of the most important pieces of machine learning is **model validation**: that is, checking how well your model fits a given dataset.

Is our model any good?

- Accuracy
- Computation time
- Interpretability

3 Types of Tasks to Bear in Mind

- Classification
- Regression
- Clustering

Classification

- Accuracy and Error
- Accuracy goes up when Error goes down

$$Accuracy = \frac{\text{correctly classified instances}}{\text{total amount of classified instances}}$$

$$Error = 1 - Accuracy$$

Consider the Titanic Kaggle dataset we've introduced previously

In [1]:
```
!kaggle competitions download -c titanic --force
```

```
401 - Unauthorized
```

In [2]:
```
import pandas as pd

train = pd.read_csv("train.csv")
train.head()
```

Out[2]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

About the Titanic Shipwrecks History

The sinking of the RMS Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships.

One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class.

Using a dummy classifier built with our instincts

In [3]:

```
import numpy as np

def dummy_classifier(x):
    if x == "male":
        return 0
    else:
        return 1
```

Now we'll use this classifier to *predict* labels for the data

In [4]:
```
y_pred = np.array(list(map(dummy_classifier, train["Sex"])))
y_train = train["Survived"].values
accuracy = (y_pred == y_train).sum() / y_pred.size
```

How might we check how well our model performs?

In [5]:
```
print("Predicted labels:")
print(y_pred[:30])
print("======")
print("Real labels:")
print(y_train[:30])
print("======")
print("{} / {} correct".format((y_pred == y_train).sum(), y_pred.size))
print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Predicted labels:
[0 1 1 1 0 0 0 0 1 1 1 1 0 0 1 1 0 0 1 1 0 0 1 0 1 1 0 0 1 0]
======
Real labels:
[0 1 1 1 0 0 0 0 1 1 1 1 0 0 0 1 0 1 0 1 0 1 1 1 0 1 0 0 1 0]
======
701 / 891 correct
Accuracy: 78.68%
```

Limits of Accuracy: Classifying very rare heart disease

- Classify all as negative (not sick)
- Predict 99 correct (not sick) and miss 1
- Accuracy: 99%
- Missed every positive case

Confusion Matrix

- Rows and columns contain all available labels
- Each cell contains frequency of instances that are classified in a certain way

		True condition				
	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	
Predicted condition	Predicted condition positive	**True positive**, Power	**False positive**, Type I error	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$	
	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$	
		True positive rate (TPR), Recall, Sensitivity, probability of detection = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR−}}$	F_1 score = $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$
		False negative rate (FNR), Miss rate = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) = $\frac{\text{FNR}}{\text{TNR}}$		

Source: https://en.wikipedia.org/wiki/Confusion_matrix

Important Confusion Matrix Components

- True positive
- True negative
- False positive
- False negative

Classifying very rare heart disease again

- True positive: 0
- True negative: 99
- False positive: 0
- False negative: 1

Using recall and precision for the case now

- Recall

$$Recall = \frac{\text{True positive}}{\text{Condition positive}} = \frac{0}{1} = 0\%$$

- Precision

$$Precision = \frac{\text{True positive}}{\text{Predicted condition positive}} = \frac{0}{0} = \text{Undefined}$$

Getting Confusion Matrix through Scikit-Learn

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html
(https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html)

In [6]:

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_train, y_pred)
tn = cm[0, 0]
tp = cm[1, 1]
fn = cm[1, 0]
fp = cm[0, 1]
print(cm)
print("True positive: {}".format(tp))
print("True negative: {}".format(tn))
print("False negative: {}".format(fn))
print("False positive: {}".format(fp))
```

```
[[468  81]
 [109 233]]
True positive: 233
True negative: 468
False negative: 109
False positive: 81
```

Regression

- Mean Square Error(MSE)
- Mean distance between estimates and regression line

$$MSE = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$

y_i: actual outcome for obs. i

$\hat{y}_i$: predicted outcome for obs. i

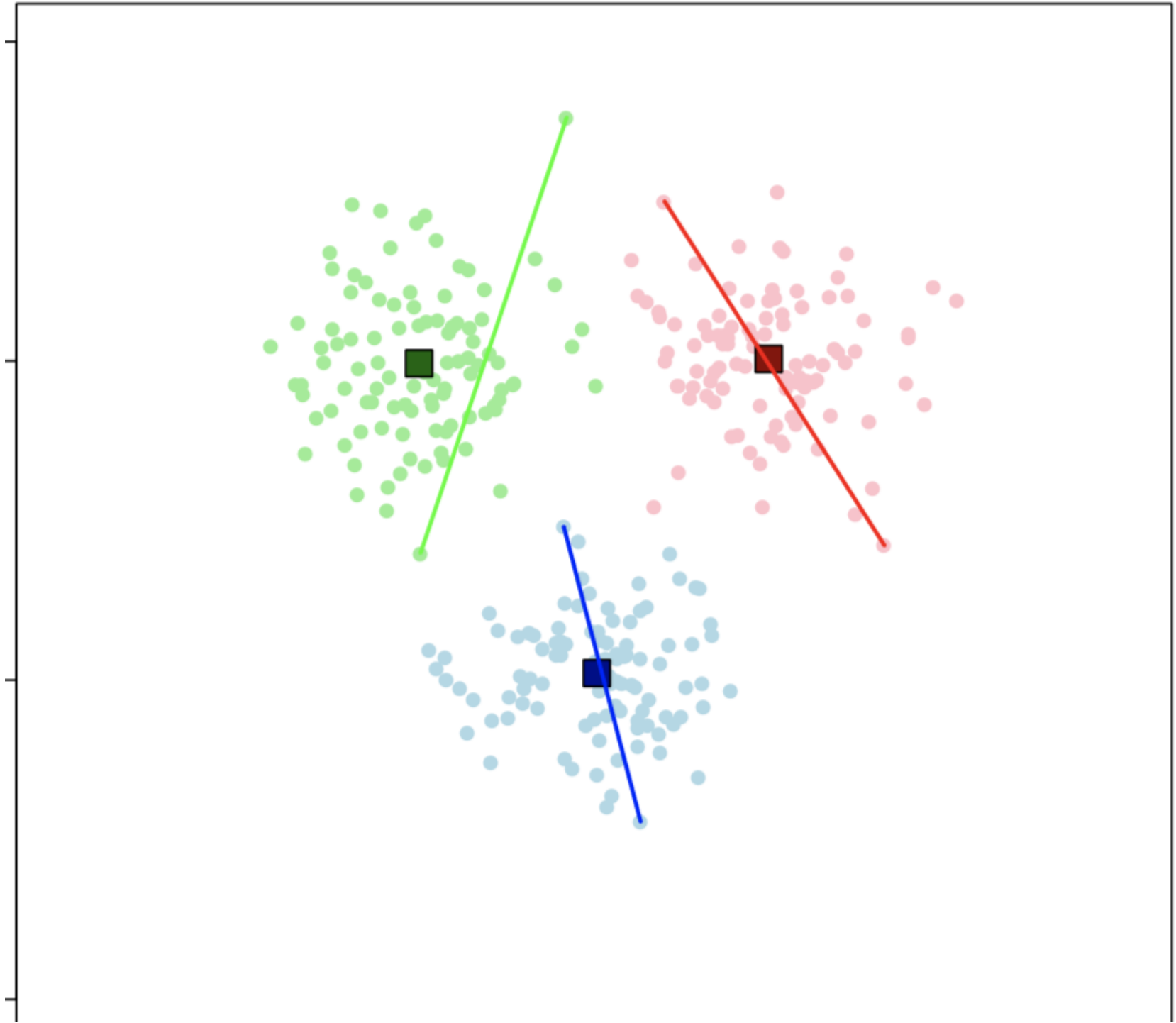
m: Number of obs.

Clustering

- No label information
- Performance measure consists of 2 elements:
    - Similarity within each cluster
    - Similarity between clusters

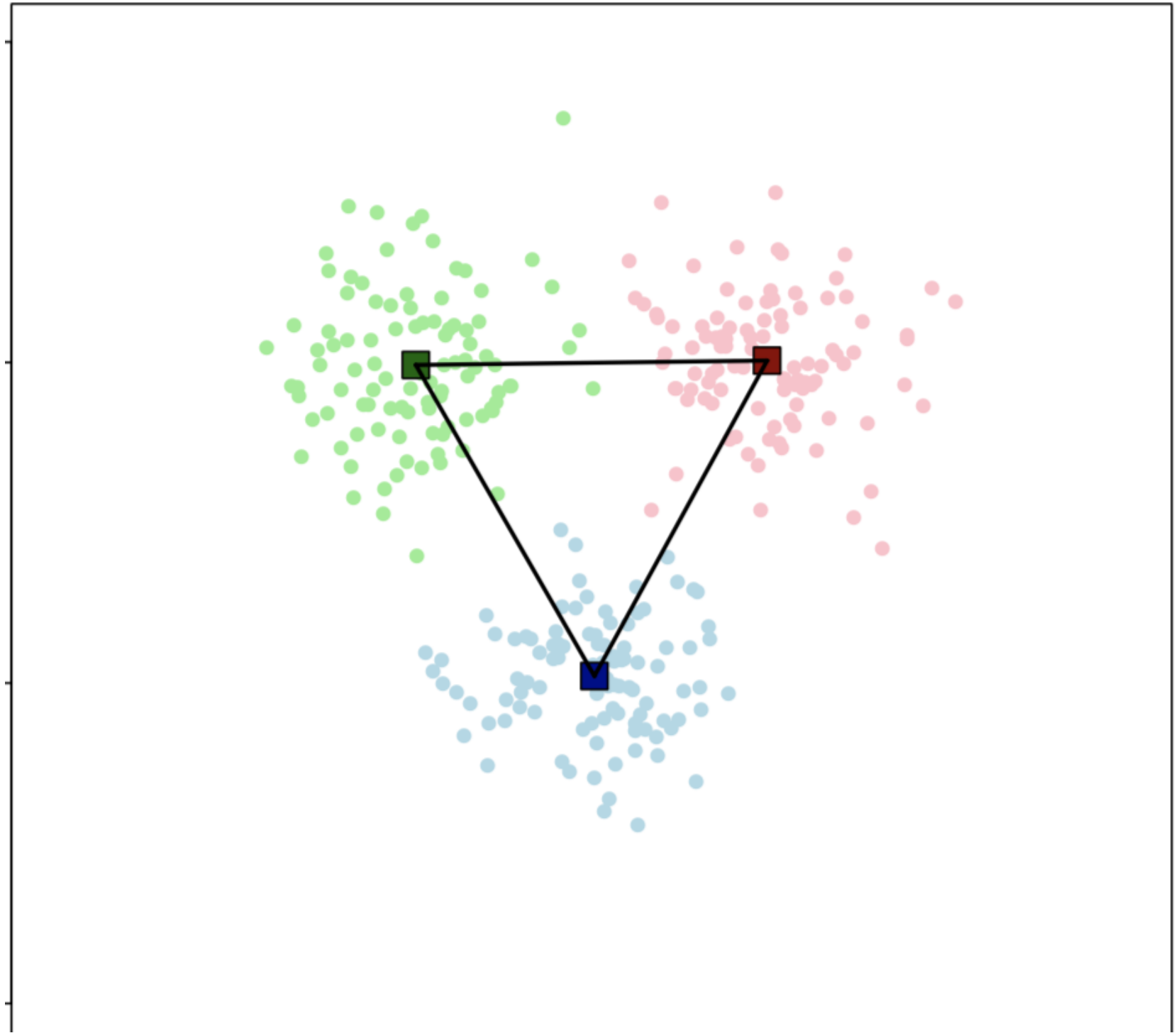
Similarity within each cluster

- Within sum of squares (WSS)
- **Minimize** diameter



Similarity between clusters

- Between cluster sum of squares (BSS)
- **Maximize** intercluster distance



Using Dunn's Index to identify the performance of clustering

$$Dunn's Index = \frac{\text{minimal intercluster distance}}{\text{maximal diameter}}$$

Validation Sets

Machine Learning vs. Statistics

- Predictive power vs. descriptive power
- Supervised learning: model must predict **unseen** observations
- Statistics: model must fit data to explain or describe data

Machine Learning vs. Statistics

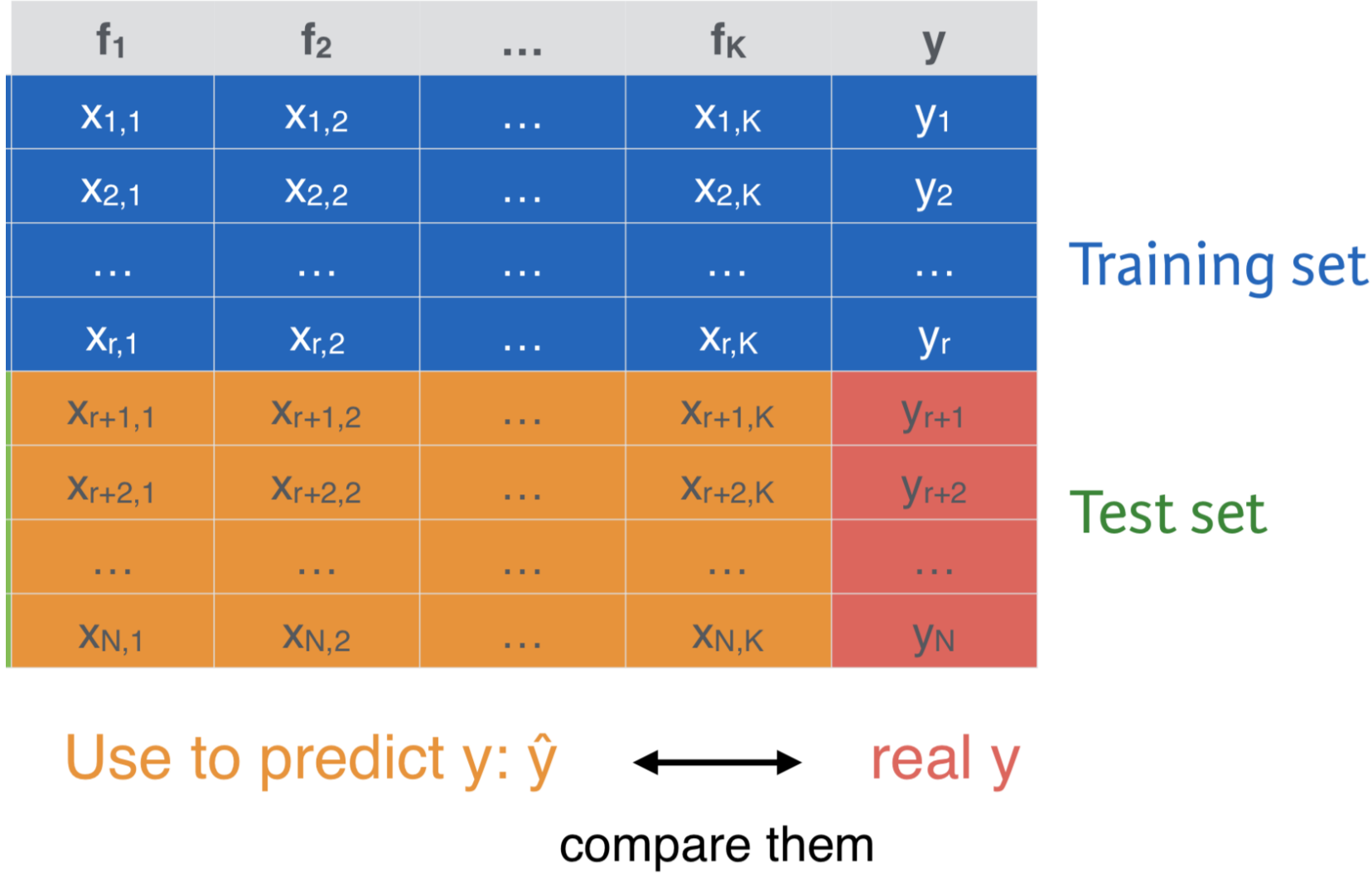
- Predictive power vs. descriptive power
- Supervised learning: model must predict **unseen** observations
- Statistics: model must fit data to explain or describe data

Training on training set, not on complete dataset

- Using test set to validate performance of model
- Sets are **disjoint**
- Model tested on unseen observations in order to be generalized

Training on training set, not on complete dataset

- Using test set to validate performance of model
- Sets are **disjoint**
- Model tested on unseen observations in order to be generalized


f1 f2 ... fK y
x1,1 x1,2 ... x1,K y1
x2,1 x2,2 ... x2,K y2
... ...
xr,1 xr,2 ... xr,K yr
xr+1,1 xr+1,2 ... xr+1,K yr+1
xr+2,1 xr+2,2 ... xr+2,K yr+2
... ...
xN,1 xN,2 ... xN,K yN
Training set
Test set
Use to predict y: ŷ
real y
compare them

Training/test set are mainly used in supervised learning, not for unsupervised due to data not labeled

How to split the sets?

- Which observations go where?
- Training set should be larger than test set
- Typically about 3:1
- Quite arbitrary
- Generally: more data -> better model
- Test set not too small

Distribution of the sets

- For classification:
    - Classes must have similar distributions
    - Avoid a class not being available in a set
- For classification and regression:
    - Shuffling dataset before splitting

In [7]:

```
from sklearn.model_selection import train_test_split
import pandas as pd

train = pd.read_csv("train.csv")
X_train = train.drop("Survived", axis=1).values
y_train = train["Survived"].values
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train)
print(train.shape)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(891, 12)
(668, 11)
(223, 11)
(668,)
(223,)
```

Create a `train_test_split` function by ourselves

Sampling can affect performance measures

- Adding robustness to these measures: **cross-validation**
- Core idea of cross-validation: Sampling multiple times, with different separations

What a 4-fold cross-validation looks like?

In [8]:

```
from sklearn.model_selection import KFold
import pandas as pd

train = pd.read_csv("train.csv")
kf = KFold(n_splits=4, shuffle=True, random_state=123)
for train_index, test_index in kf.split(train):
    print("TRAIN:", train_index)
    print("TEST:", test_index)
```

```
TRAIN: [  0   1   2   3   6   7   8   9  10  12  14  15  16  17  18  19  20  2
1
  22  23  24  25  26  27  28  29  30  32  33  34  37  38  39  40  42  44
  45  46  47  49  50  51  53  56  58  60  61  62  63  64  65  66  67  68
  69  70  71  72  73  76  77  79  80  81  82  83  84  85  86  87  88  89
  92  93  94  95  96  98  99 101 104 105 106 108 109 110 111 112 113 114
 115 116 118 119 120 121 122 123 124 125 126 127 128 129 130 132 133 135
 136 137 139 140 141 142 143 144 146 148 149 150 152 153 154 155 157 158
 159 160 163 165 166 168 169 173 174 175 176 178 179 180 181 183 184 186
 187 189 190 191 192 193 194 197 198 199 201 202 204 205 206 207 208 211
 212 213 214 215 216 217 218 219 220 222 223 224 225 228 230 231 232 233
 234 235 238 241 242 243 244 245 246 247 248 250 251 253 254 255 256 257
 258 259 262 264 265 268 271 275 276 277 278 281 282 283 284 286 287 288
 289 290 291 292 293 295 296 297 298 300 301 302 304 305 308 309 311 312
 314 315 319 320 321 322 323 324 325 326 328 330 331 332 333 334 335 337
 339 340 341 342 343 344 347 348 350 352 354 355 357 358 359 360 361 362
 364 365 366 369 371 372 373 376 377 378 379 380 382 383 384 385 386 387
 390 391 392 393 394 395 396 398 399 400 401 402 403 404 405 407 409 410
 411 412 413 414 415 416 417 418 419 420 422 423 424 426 427 428 429 430
 431 432 433 434 436 437 438 439 440 441 442 443 444 445 446 447 449 450
 451 453 454 455 456 457 458 459 460 461 462 464 465 466 467 468 469 470
 471 472 473 474 475 476 477 478 479 480 481 484 485 486 487 488 489 490
 492 493 494 495 496 497 499 500 501 502 503 504 505 506 507 508 510 512
 513 514 515 516 517 518 519 525 526 528 529 530 533 534 535 537 538 539
 540 541 542 544 545 548 549 550 552 553 554 555 556 557 558 559 561 562
 563 564 565 569 570 571 572 573 574 575 576 578 579 581 582 583 584 585
 586 588 589 590 591 592 593 594 595 596 597 598 601 603 604 605 606 607
```

```
 608 609 610 611 612 613 615 618 619 622 623 624 625 626 627 628 629 630
 631 634 635 636 637 638 639 643 645 647 648 649 650 651 653 654 655 656
 657 658 660 661 662 663 664 665 666 667 669 672 673 674 679 680 681 682
 684 685 686 687 689 691 692 695 696 697 698 701 703 704 705 706 708 710
 711 712 716 717 718 719 720 721 724 725 728 729 730 732 733 734 736 737
 739 741 742 743 744 745 746 747 750 751 752 754 755 756 757 758 759 760
 761 762 763 765 766 770 771 772 776 778 779 780 781 782 783 785 786 787
 788 789 790 791 792 793 794 797 801 803 804 805 806 807 808 811 812 815
 817 818 819 823 825 826 828 829 830 832 833 834 835 836 837 838 839 840
 842 843 844 845 846 847 849 850 851 852 854 855 856 857 858 859 863 866
 867 868 869 870 871 872 873 874 876 878 879 881 882 883 884 885 886 887
 889 890]
TEST: [  4   5  11  13  31  35  36  41  43  48  52  54  55  57  59  74  75  78
  90  91  97 100 102 103 107 117 131 134 138 145 147 151 156 161 162 164
 167 170 171 172 177 182 185 188 195 196 200 203 209 210 221 226 227 229
 236 237 239 240 249 252 260 261 263 266 267 269 270 272 273 274 279 280
 285 294 299 303 306 307 310 313 316 317 318 327 329 336 338 345 346 349
 351 353 356 363 367 368 370 374 375 381 388 389 397 406 408 421 425 435
 448 452 463 482 483 491 498 509 511 520 521 522 523 524 527 531 532 536
 543 546 547 551 560 566 567 568 577 580 587 599 600 602 614 616 617 620
 621 632 633 640 641 642 644 646 652 659 668 670 671 675 676 677 678 683
 688 690 693 694 699 700 702 707 709 713 714 715 722 723 726 727 731 735
 738 740 748 749 753 764 767 768 769 773 774 775 777 784 795 796 798 799
 800 802 809 810 813 814 816 820 821 822 824 827 831 841 848 853 860 861
 862 864 865 875 877 880 888]
TRAIN: [  1   2   3   4   5   6   8  10  11  12  13  14  16  17  19  20  25  2
7
  28  29  31  32  35  36  37  39  41  43  44  46  47  48  49  51  52  53
  54  55  56  57  58  59  60  61  62  65  67  68  69  70  71  73  74  75
  76  77  78  81  83  86  87  88  89  90  91  92  95  96  97  98  99 100
 102 103 104 105 106 107 108 109 110 111 112 113 114 117 123 125 126 128
 129 130 131 133 134 135 136 137 138 139 140 141 143 144 145 146 147 148
 149 151 152 154 156 157 158 161 162 164 165 167 168 170 171 172 173 175
 176 177 179 180 182 184 185 186 187 188 189 191 193 194 195 196 197 198
 200 201 203 204 206 207 208 209 210 211 213 214 215 216 218 219 220 221
 222 223 224 225 226 227 229 231 233 234 235 236 237 238 239 240 241 243
 244 247 248 249 251 252 253 255 256 257 259 260 261 262 263 265 266 267
 269 270 271 272 273 274 277 278 279 280 281 284 285 286 288 290 293 294
```

296 297 298 299 301 302 303 304 305 306 307 310 312 313 315 316 317 318
319 321 322 323 324 325 327 329 330 333 334 336 337 338 339 340 341 342
345 346 347 349 350 351 352 353 354 355 356 357 358 359 360 361 363 364
365 366 367 368 370 371 374 375 377 378 380 381 382 383 385 386 388 389
390 391 393 394 397 398 400 401 402 403 404 405 406 407 408 409 410 411
412 413 414 418 419 420 421 422 423 424 425 428 432 433 434 435 436 438
439 440 441 442 444 445 448 450 451 452 453 454 459 460 463 464 465 466
467 471 472 473 474 475 477 479 480 481 482 483 484 486 488 489 490 491
493 494 495 496 497 498 501 502 503 504 506 507 509 510 511 513 514 515
516 518 519 520 521 522 523 524 526 527 528 530 531 532 533 535 536 537
538 539 540 541 543 544 545 546 547 549 550 551 552 554 555 557 558 559
560 561 562 564 566 567 568 569 570 571 572 573 574 575 576 577 578 579
580 581 582 583 585 587 588 589 590 592 593 595 596 598 599 600 601 602
604 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 624 626
627 628 629 630 631 632 633 634 635 636 638 639 640 641 642 643 644 645
646 647 649 650 651 652 655 658 659 661 663 665 668 670 671 673 674 675
676 677 678 680 681 683 685 686 687 688 689 690 692 693 694 695 696 697
698 699 700 701 702 705 707 708 709 710 711 712 713 714 715 717 720 722
723 724 726 727 728 730 731 733 734 735 736 737 738 739 740 742 743 744
745 748 749 752 753 755 758 759 761 763 764 765 767 768 769 771 772 773
774 775 776 777 778 780 781 782 784 786 787 788 789 790 791 792 794 795
796 797 798 799 800 802 805 806 807 808 809 810 811 812 813 814 815 816
817 818 819 820 821 822 823 824 826 827 829 830 831 833 835 837 838 839
840 841 842 843 844 846 847 848 849 850 853 854 855 856 857 859 860 861
862 864 865 868 869 870 871 872 873 875 876 877 878 880 882 884 887 888
889 890]
TEST: [0 7 9 15 18 21 22 23 24 26 30 33 34 38 40 42 45 50
63 64 66 72 79 80 82 84 85 93 94 101 115 116 118 119 120 121
122 124 127 132 142 150 153 155 159 160 163 166 169 174 178 181 183 190
192 199 202 205 212 217 228 230 232 242 245 246 250 254 258 264 268 275
276 282 283 287 289 291 292 295 300 308 309 311 314 320 326 328 331 332
335 343 344 348 362 369 372 373 376 379 384 387 392 395 396 399 415 416
417 426 427 429 430 431 437 443 446 447 449 455 456 457 458 461 462 468
469 470 476 478 485 487 492 499 500 505 508 512 517 525 529 534 542 548
553 556 563 565 584 586 591 594 597 603 605 606 607 623 625 637 648 653
654 656 657 660 662 664 666 667 669 672 679 682 684 691 703 704 706 716
718 719 721 725 729 732 741 746 747 750 751 754 756 757 760 762 766 770
779 783 785 793 801 803 804 825 828 832 834 836 845 851 852 858 863 866

```
 867 874 879 881 883 885 886]
TRAIN: [  0   2   3   4   5   7   8   9  11  13  14  15  16  17  18  21  22  2
3
  24  26  30  31  33  34  35  36  38  39  40  41  42  43  45  46  47  48
  50  51  52  54  55  56  57  58  59  60  63  64  65  66  67  68  69  72
  73  74  75  76  77  78  79  80  82  84  85  86  87  88  90  91  92  93
  94  96  97  98  99 100 101 102 103 106 107 109 111 112 113 115 116 117
 118 119 120 121 122 123 124 127 129 130 131 132 134 135 138 139 140 141
 142 145 146 147 150 151 153 154 155 156 158 159 160 161 162 163 164 166
 167 169 170 171 172 174 176 177 178 180 181 182 183 185 186 187 188 190
 192 193 194 195 196 197 198 199 200 202 203 205 206 207 208 209 210 212
 213 214 215 217 221 222 224 226 227 228 229 230 232 236 237 239 240 242
 244 245 246 249 250 252 253 254 255 257 258 259 260 261 262 263 264 265
 266 267 268 269 270 271 272 273 274 275 276 279 280 281 282 283 285 287
 289 290 291 292 294 295 296 299 300 301 303 304 305 306 307 308 309 310
 311 312 313 314 315 316 317 318 319 320 321 322 323 325 326 327 328 329
 331 332 335 336 338 339 340 341 342 343 344 345 346 348 349 351 353 354
 356 357 358 359 360 362 363 364 365 367 368 369 370 371 372 373 374 375
 376 377 379 380 381 382 384 385 387 388 389 390 392 393 394 395 396 397
 399 401 406 407 408 409 410 411 412 415 416 417 418 419 420 421 424 425
 426 427 428 429 430 431 432 433 434 435 437 438 440 441 442 443 446 447
 448 449 451 452 455 456 457 458 459 460 461 462 463 465 467 468 469 470
 471 472 473 475 476 478 480 481 482 483 484 485 486 487 489 490 491 492
 493 496 497 498 499 500 502 503 504 505 507 508 509 510 511 512 513 515
 517 520 521 522 523 524 525 527 529 530 531 532 534 536 537 539 542 543
 544 546 547 548 551 552 553 555 556 557 559 560 562 563 565 566 567 568
 569 572 574 575 576 577 580 582 584 586 587 588 591 594 595 596 597 599
 600 602 603 605 606 607 608 609 611 612 614 615 616 617 620 621 622 623
 625 628 630 631 632 633 634 635 636 637 638 640 641 642 644 645 646 647
 648 649 651 652 653 654 656 657 658 659 660 661 662 664 665 666 667 668
 669 670 671 672 675 676 677 678 679 680 681 682 683 684 686 688 690 691
 693 694 695 699 700 702 703 704 705 706 707 709 713 714 715 716 717 718
 719 720 721 722 723 725 726 727 729 731 732 735 736 737 738 740 741 742
 745 746 747 748 749 750 751 752 753 754 755 756 757 760 762 764 765 766
 767 768 769 770 772 773 774 775 777 779 780 782 783 784 785 786 788 790
 793 795 796 797 798 799 800 801 802 803 804 805 806 809 810 811 813 814
 816 817 819 820 821 822 823 824 825 826 827 828 829 831 832 834 836 837
 840 841 843 845 846 847 848 849 850 851 852 853 857 858 860 861 862 863
```

```
 864 865 866 867 869 873 874 875 877 879 880 881 882 883 884 885 886 887
 888 889]
TEST: [  1   6  10  12  19  20  25  27  28  29  32  37  44  49  53  61  62  70
  71  81  83  89  95 104 105 108 110 114 125 126 128 133 136 137 143 144
 148 149 152 157 165 168 173 175 179 184 189 191 201 204 211 216 218 219
 220 223 225 231 233 234 235 238 241 243 247 248 251 256 277 278 284 286
 288 293 297 298 302 324 330 333 334 337 347 350 352 355 361 366 378 383
 386 391 398 400 402 403 404 405 413 414 422 423 436 439 444 445 450 453
 454 464 466 474 477 479 488 494 495 501 506 514 516 518 519 526 528 533
 535 538 540 541 545 549 550 554 558 561 564 570 571 573 578 579 581 583
 585 589 590 592 593 598 601 604 610 613 618 619 624 626 627 629 639 643
 650 655 663 673 674 685 687 689 692 696 697 698 701 708 710 711 712 724
 728 730 733 734 739 743 744 758 759 761 763 771 776 778 781 787 789 791
 792 794 807 808 812 815 818 830 833 835 838 839 842 844 854 855 856 859
 868 870 871 872 876 878 890]
TRAIN: [  0   1   4   5   6   7   9  10  11  12  13  15  18  19  20  21  22  2
3
  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  40  41  42
  43  44  45  48  49  50  52  53  54  55  57  59  61  62  63  64  66  70
  71  72  74  75  78  79  80  81  82  83  84  85  89  90  91  93  94  95
  97 100 101 102 103 104 105 107 108 110 114 115 116 117 118 119 120 121
 122 124 125 126 127 128 131 132 133 134 136 137 138 142 143 144 145 147
 148 149 150 151 152 153 155 156 157 159 160 161 162 163 164 165 166 167
 168 169 170 171 172 173 174 175 177 178 179 181 182 183 184 185 188 189
 190 191 192 195 196 199 200 201 202 203 204 205 209 210 211 212 216 217
 218 219 220 221 223 225 226 227 228 229 230 231 232 233 234 235 236 237
 238 239 240 241 242 243 245 246 247 248 249 250 251 252 254 256 258 260
 261 263 264 266 267 268 269 270 272 273 274 275 276 277 278 279 280 282
 283 284 285 286 287 288 289 291 292 293 294 295 297 298 299 300 302 303
 306 307 308 309 310 311 313 314 316 317 318 320 324 326 327 328 329 330
 331 332 333 334 335 336 337 338 343 344 345 346 347 348 349 350 351 352
 353 355 356 361 362 363 366 367 368 369 370 372 373 374 375 376 378 379
 381 383 384 386 387 388 389 391 392 395 396 397 398 399 400 402 403 404
 405 406 408 413 414 415 416 417 421 422 423 425 426 427 429 430 431 435
 436 437 439 443 444 445 446 447 448 449 450 452 453 454 455 456 457 458
 461 462 463 464 466 468 469 470 474 476 477 478 479 482 483 485 487 488
 491 492 494 495 498 499 500 501 505 506 508 509 511 512 514 516 517 518
 519 520 521 522 523 524 525 526 527 528 529 531 532 533 534 535 536 538
```

540 541 542 543 545 546 547 548 549 550 551 553 554 556 558 560 561 563
564 565 566 567 568 570 571 573 577 578 579 580 581 583 584 585 586 587
589 590 591 592 593 594 597 598 599 600 601 602 603 604 605 606 607 610
613 614 616 617 618 619 620 621 623 624 625 626 627 629 632 633 637 639
640 641 642 643 644 646 648 650 652 653 654 655 656 657 659 660 662 663
664 666 667 668 669 670 671 672 673 674 675 676 677 678 679 682 683 684
685 687 688 689 690 691 692 693 694 696 697 698 699 700 701 702 703 704
706 707 708 709 710 711 712 713 714 715 716 718 719 721 722 723 724 725
726 727 728 729 730 731 732 733 734 735 738 739 740 741 743 744 746 747
748 749 750 751 753 754 756 757 758 759 760 761 762 763 764 766 767 768
769 770 771 773 774 775 776 777 778 779 781 783 784 785 787 789 791 792
793 794 795 796 798 799 800 801 802 803 804 807 808 809 810 812 813 814
815 816 818 820 821 822 824 825 827 828 830 831 832 833 834 835 836 838
839 841 842 844 845 848 851 852 853 854 855 856 858 859 860 861 862 863
864 865 866 867 868 870 871 872 874 875 876 877 878 879 880 881 883 885
886 888 890]
TEST: [2 3 8 14 16 17 39 46 47 51 56 58 60 65 67 68 69 73
76 77 86 87 88 92 96 98 99 106 109 111 112 113 123 129 130 135
139 140 141 146 154 158 176 180 186 187 193 194 197 198 206 207 208 213
214 215 222 224 244 253 255 257 259 262 265 271 281 290 296 301 304 305
312 315 319 321 322 323 325 339 340 341 342 354 357 358 359 360 364 365
371 377 380 382 385 390 393 394 401 407 409 410 411 412 418 419 420 424
428 432 433 434 438 440 441 442 451 459 460 465 467 471 472 473 475 480
481 484 486 489 490 493 496 497 502 503 504 507 510 513 515 530 537 539
544 552 555 557 559 562 569 572 574 575 576 582 588 595 596 608 609 611
612 615 622 628 630 631 634 635 636 638 645 647 649 651 658 661 665 680
681 686 695 705 717 720 736 737 742 745 752 755 765 772 780 782 786 788
790 797 805 806 811 817 819 823 826 829 837 840 843 846 847 849 850 857
869 873 882 884 887 889]

k-fold cross-validation

- Fold test set over dataset n times
- Each test set is 1/k size of total dataset

In [9]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold

def dummy_classifier(x):
    if x == "male":
        return 0
    else:
        return 1

train = pd.read_csv("train.csv")
X_train = train["Sex"].values
y_train = train["Survived"].values
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train)
kf = KFold(n_splits=10, shuffle=True, random_state=123)
for train_index, valid_index in kf.split(X_train):
    X_valid_cv = X_train[valid_index]
    y_valid_cv = y_train[valid_index]
    y_pred_cv = np.array(list(map(dummy_classifier, X_valid_cv)))
    accuracy = (y_pred_cv == y_valid_cv).sum() / y_pred_cv.size
    print("Accuracy: {:.2f}%".format(accuracy*100))
```

```
Accuracy: 73.13%
Accuracy: 73.13%
Accuracy: 82.09%
Accuracy: 74.63%
Accuracy: 80.60%
Accuracy: 86.57%
Accuracy: 80.60%
Accuracy: 76.12%
Accuracy: 74.24%
Accuracy: 84.85%
```

Bias and Variance

The main goal of supervised learning is prediction

Prediction error can be decomposed as **reducible** and **irreducible** error

Irreducible error is commonly referred as noise, cannot be minimized

Reducible error is commonly caused by model unfit

- We can try to minize it!
- Reducible error is split into **bias** and **variance**

A quick recap

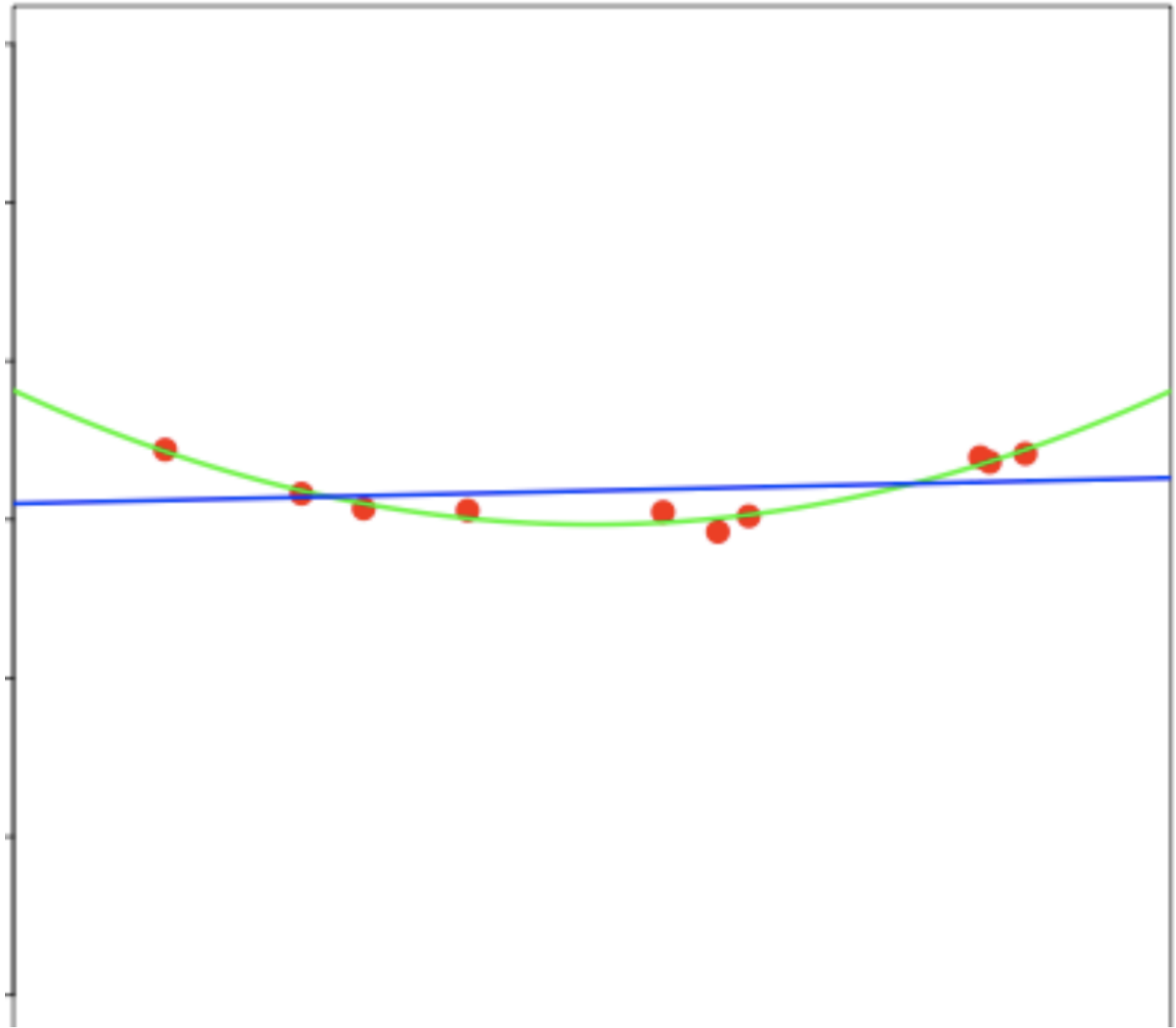
- Prediction errors
  - Irreducible error
  - Reducible error
    - Bias
    - Variance

Bias: wrong assumptions

- Difference between predictions and truth
- Using models trained by a specific learning algorithm

An example of bias

Quadratic data but using linear model

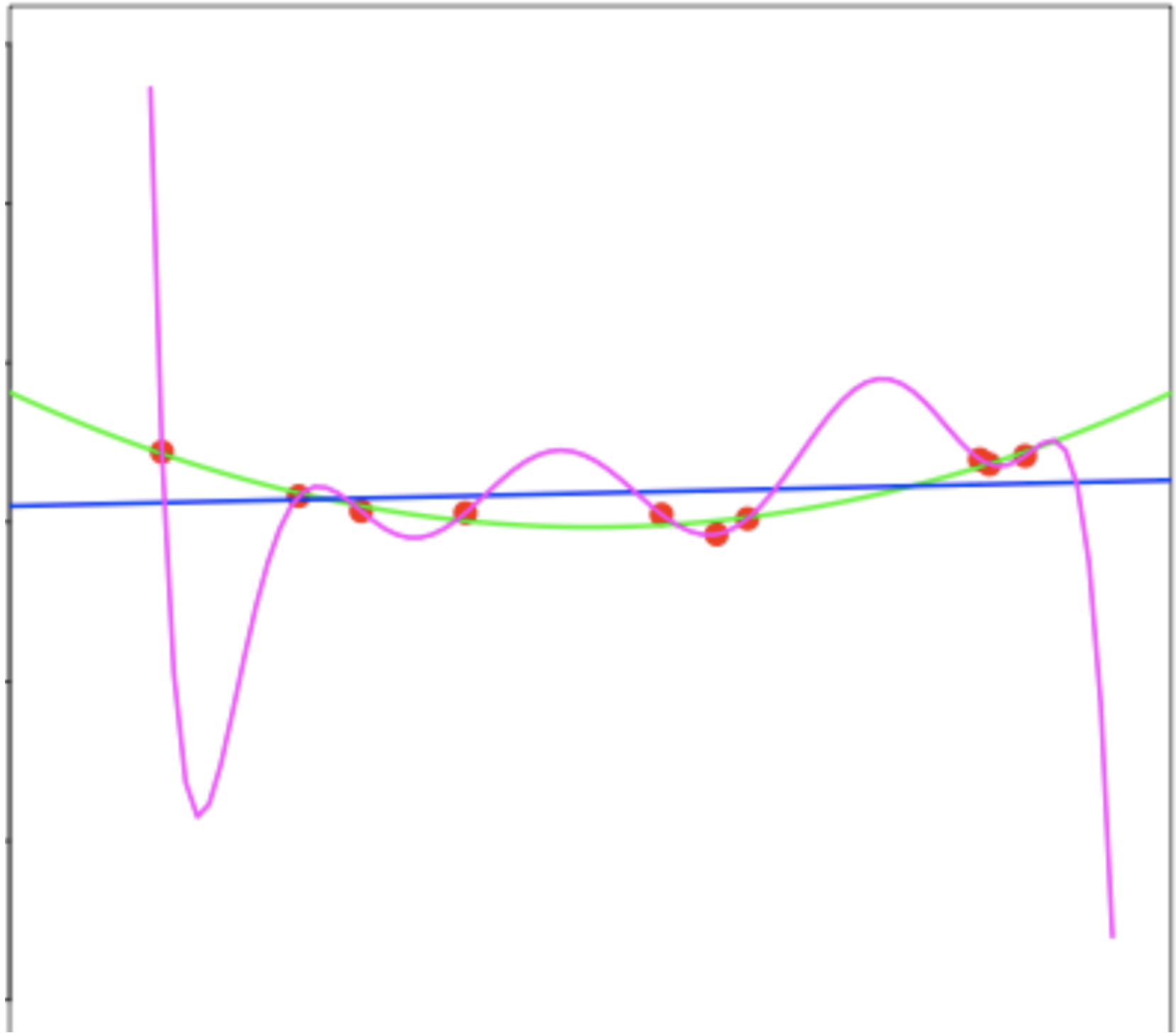


Variance: sampling of the training set

- Model with high variance fits **training set** closely

An example of variance

- Quadratic data using polynomial model fits perfectly through training set
- High variance leads model generalizing bad to test set



Bias and Variance Tradeoff

- low bias leads to high variance
- low variance leads to high bias

Overfitting

- Model fits **training set** a lot better than test set
- High variance
- Too specific

Underfitting

- Model does not fit **training set**, nor does **test set**
- High bias
- Too general

The importance

The issues associated with validation and cross-validation are some of the most important aspects of the practice of machine learning.

If our estimator is underperforming, how should we move forward?

- Use simpler or more complicated model?
- Add more features to each observed data point?
- Add more training samples?

The answer is often counter-intuitive

In particular, **Sometimes using a more complicated model will give *worse* results.** Also, **Sometimes adding training data will not improve your results.** The ability to determine what steps will improve your model is what separates the successful machine learning practitioners from the unsuccessful.

Illustration of the Bias-Variance Tradeoff

In [10]:
```
def test_func(x, err=0.5):
    y = 10 - 1. / (x + 0.1)
    if err > 0:
        y = np.random.normal(y, err)
    return y
```

Now let's create a realization of this dataset:

In [11]:
```
def make_data(N=40, error=1.0, random_seed=1):
    # randomly sample the data
    np.random.seed(1)
    X = np.random.random(N)[:, np.newaxis]
    y = test_func(X.ravel(), error)

    return X, y
```

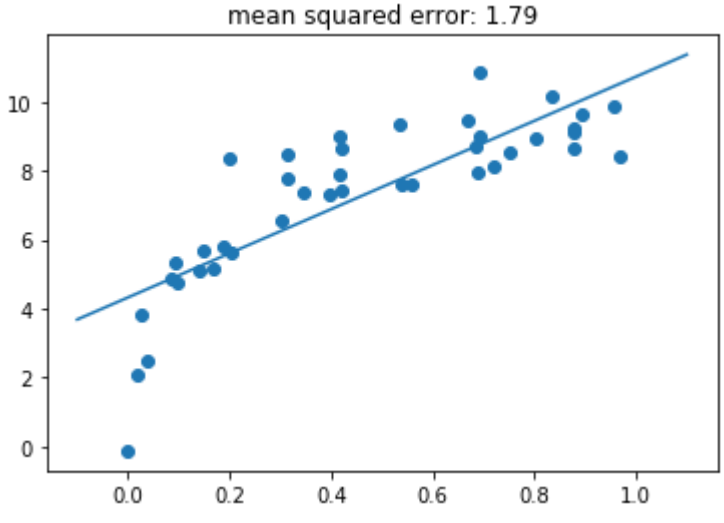
In [12]:
```
import matplotlib.pyplot as plt

X, y = make_data(40, error=1)
plt.scatter(X.ravel(), y)
plt.show()
```

```
<Figure size 640x480 with 1 Axes>
```

Now say we want to perform a regression on this data. Let's use the built-in linear regression function to compute a fit:

In [13]:
```
X_test = np.linspace(-0.1, 1.1, 500)[:, None]

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
model = LinearRegression()
model.fit(X, y)
y_test = model.predict(X_test)

plt.scatter(X.ravel(), y)
plt.plot(X_test.ravel(), y_test)
plt.title("mean squared error: {0:.3g}".format(mean_squared_error(model.predict(X
), y)))
```

Out[13]:
```
Text(0.5, 1.0, 'mean squared error: 1.79')
```



In [14]:
```
plt.show()
```

We have fit a straight line to the data, but clearly this model is not a good choice

We say that this model is **biased**, or that it **under-fits** the data.

Let's try to improve this by creating a more complicated model

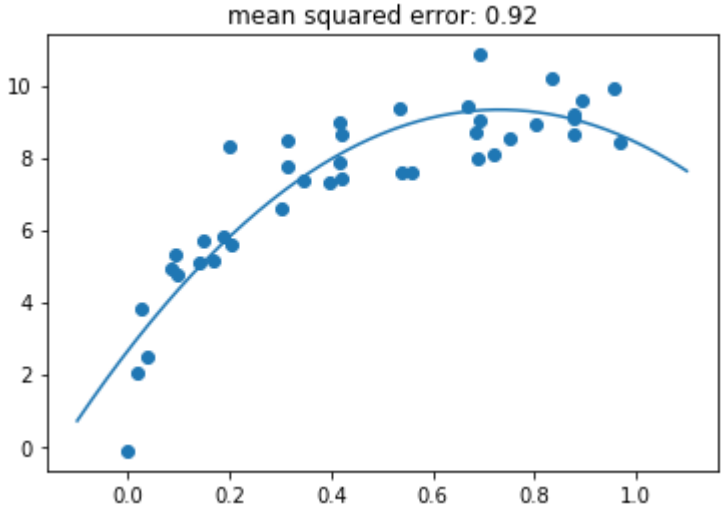
We can do this by adding degrees of freedom, and computing a polynomial regression over the inputs.

In [15]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline

def PolynomialRegression(degree=2, **kwargs):
    return make_pipeline(PolynomialFeatures(degree),
                         LinearRegression(**kwargs))
```

Now we'll use this to fit a quadratic curve to the data.

In [16]:
```
model = PolynomialRegression(2)
model.fit(X, y)
y_test = model.predict(X_test)

plt.scatter(X.ravel(), y)
plt.plot(X_test.ravel(), y_test)
plt.title("mean squared error: {0:.3g}".format(mean_squared_error(model.predict(X
), y)))
```

Out[16]:
```
Text(0.5, 1.0, 'mean squared error: 0.92')
```

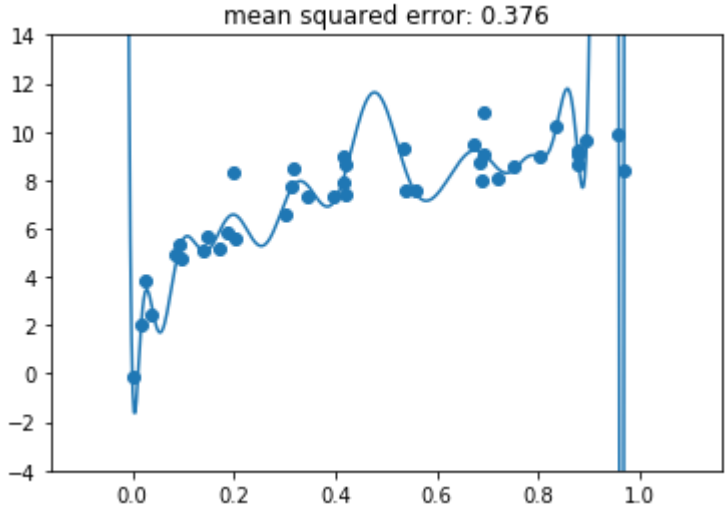


In [17]:
```
plt.show()
```

This reduces the mean squared error, and makes a much better fit. What happens if we use an even higher-degree polynomial?

In [18]:
```
model = PolynomialRegression(30)
model.fit(X, y)
y_test = model.predict(X_test)

plt.scatter(X.ravel(), y)
plt.plot(X_test.ravel(), y_test)
plt.title("mean squared error: {0:.3g}".format(mean_squared_error(model.predict(X
), y)))
plt.ylim(-4, 14)
```

Out[18]:
```
(-4, 14)
```



In [19]:
```
plt.show()
```

When we increase the degree to this extent, it's clear that the resulting fit is no longer reflecting the true underlying distribution, but is more sensitive to the noise in the training data. For this reason, we call it a **high-variance model**, and we say that it **over-fits** the data.

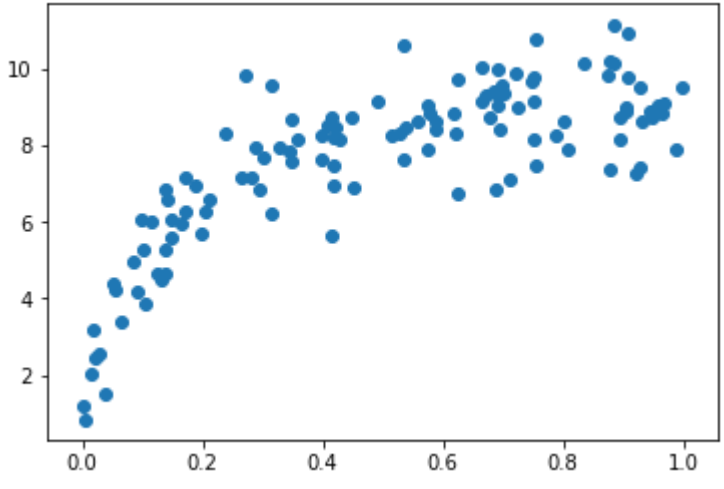
Detecting Over-fitting with Validation Curves

Computing the error on the training data is not enough

As above, we can use **cross-validation** to get a better handle on how the model fit is working.

In [20]:

```
X, y = make_data(120, error=1.0)
plt.scatter(X, y)
plt.show()
```



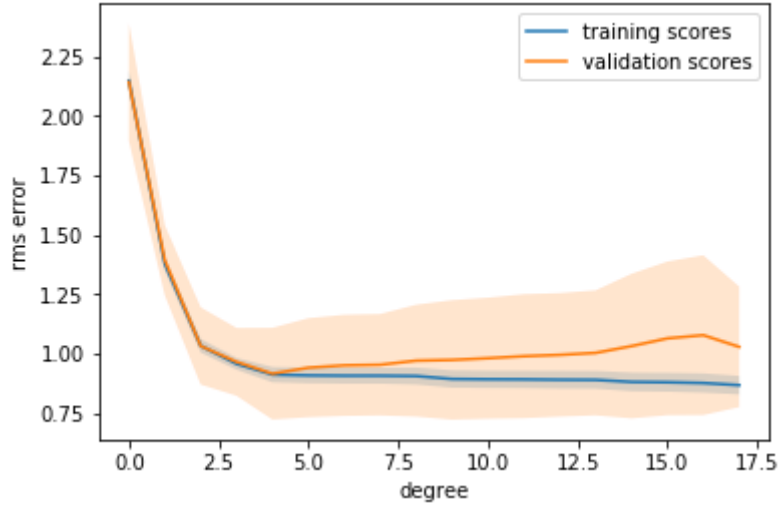
In [21]:

```
from sklearn.model_selection import validation_curve

def rms_error(model, X, y):
    y_pred = model.predict(X)
    return np.sqrt(np.mean((y - y_pred) ** 2))

degree = np.arange(0, 18)
val_train, val_test = validation_curve(PolynomialRegression(), X, y, 'polynomialfe
atures__degree', degree, cv=7, scoring=rms_error)
```

Now let's plot the validation curves:

In [22]:
```
def plot_with_err(x, data, **kwargs):
    mu, std = data.mean(1), data.std(1)
    lines = plt.plot(x, mu, '-', **kwargs)
    plt.fill_between(x, mu - std, mu + std, edgecolor='none',
                     facecolor=lines[0].get_color(), alpha=0.2)

plot_with_err(degree, val_train, label='training scores')
plot_with_err(degree, val_test, label='validation scores')
plt.xlabel('degree'); plt.ylabel('rms error')
plt.legend()
```

Out[22]:
```
<matplotlib.legend.Legend at 0x1a172240f0>
```



In [23]:
```
plt.show()
```

Notice the trend here

1. For a small model complexity, the training error and validation error are very similar. This indicates that the model is **under-fitting** the data: it doesn't have enough complexity to represent the data. Another way of putting it is that this is a **high-bias** model.

2. As the model complexity grows, the training and validation scores diverge. This indicates that the model is **over-fitting** the data: it has so much flexibility, that it fits the noise rather than the underlying trend. Another way of putting it is that this is a **high-variance** model.

3. Note that the training score (nearly) always improves with model complexity. This is because a more complicated model can fit the noise better, so the model improves. The validation data generally has a sweet spot, which here is around 5 terms.

Here's our best-fit model according to the cross-validation:

In [24]:
```
model = PolynomialRegression(4).fit(X, y)
plt.scatter(X, y)
plt.plot(X_test, model.predict(X_test))
plt.show()
```