SI 670: Applied Machine Learning
Linear Regression
Overfitting

Grant Schoenebeck

Last time

- **Work flow: features, estimator, evaluation, repeat**
- **k-NN classifiers**
- **Feature normalization**

Class plan

- Supervises Learning Assumptions
- k-NN regression
- Linear Regression
- Cross-validation
- Feature Expansion
- Overfitting and underfitting

Supervised Learning

Supervised Learning

Goal:

1) take data with labels and train a classifier / regressor
2) Using classifier on data with no labels.

Type of Supervised Learning

Three Types (for this class):

1) **Classification: output is predicted class**
2) **Soft Classification: output is distribution over classes**
   - *A combination of the other two*
3) **Regression: Output is a real number**
4) **More general types possible:**
   1) *sentence diagram,*
   2) *list of search results*
   3) *Etc.*

Supervised Learning

Goal:

1) take data with labels and train a classifier / regressor
2) Using classifier on data with no labels.

BIG ASSUMPTION: the training data looks like the testing data.

BIG ASSUMPTION
training data testing data

When does this hold?

When does this not hold?

Big (Assumption) Failures

- **Financial Crisis**
  - *Assumed housing prices would always go up.*
  - *Assumed independence in CDOs.*
- **Google Flu Trends**
- **Adversarial Settings:**
  - *Flash Crashes on Stock Market*
- **Feedback Loop:**
  - *Facebook Engagement*

Think, Pair, Share

- You are interested machine learning to grade student's problem sets (a dubious proposition to begin with). You grade a random 50% of the submissions. You train a learner on 50% of the graded assignments. Its mean squared error is X on the remaining 50% of graded assignments. You expect the error on the 50% ungraded assignments to be:

- A) More than X
- B) Equal to X
- C) Less than X

Follow-Up

- Now say you train your ML grader on the first 5 problem sets. It only had error X on the held out data. You then would like to run it on future problem sets. You expect the error to be:

- A) More than X
- B) Equal to X
- C) Less than X

Supervised Learning Justifications

- **Empirical Risk Minimization:**
  - *Fix loss function*
  - *Chose the predictor from a set the minimizes error on training data (actual).*
  - *Hope that it generalizes to test data (ideal).*
- **Maximum Likelihood Estimator:**
  - *In Bayesian Environment, chose the predictor that is most likely given the training data.*
- **Regularization:**
  - *Explicitly prefer "simpler" classifiers.*

2 Line Segment Title

Maximum Likelihood Estimation

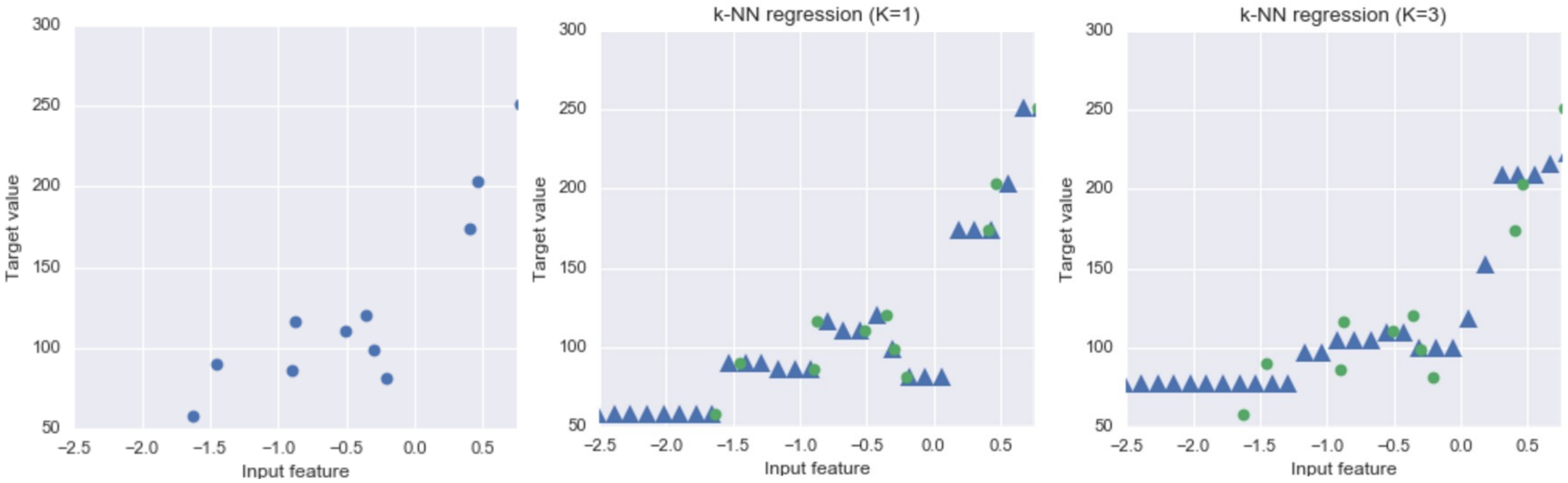
- **Given p($y|w,x$)**
- **Compute p(w | x, y) via Bayes rule**
  - $p(w|x,y) = p(w|x)p(y|w,x)/p(y|x)$
  - $p(w|x,y) \quad p(y|w,x)\ p(w|x)$
- **Gives distribution for w**
- **Take mode = MLE**
  - $p(w|x,y) \quad p(y|w,x)$ *assuming all w equally likely*

K-NN for Regression

K-NN for Regression

- **What if we wanted to predict a continuous quantity given a fruit representation?**
  - *Say the height given the width.*
- **Could we somehow apply k-nearest neighbors to regression instead of classification?**
- **Yes! How?**

k-Nearest Neighbors Regression



Linear Regression

Linear Models

- **Linear model: predicts target with weighted sum of features**
- **Example: *predicting housing prices***
  - *House features: taxes per year (), age in years ()*

    =

  - *A house with feature values (,) of (10 000, 75) would have a predicted selling price of:*

    = **1,152,000**

Linear Regression is an Example of a Linear Model

Input instance – feature vector: =

Predicted output: =

Parameters to estimate:

: feature weights/ model coefficients

/ intercept

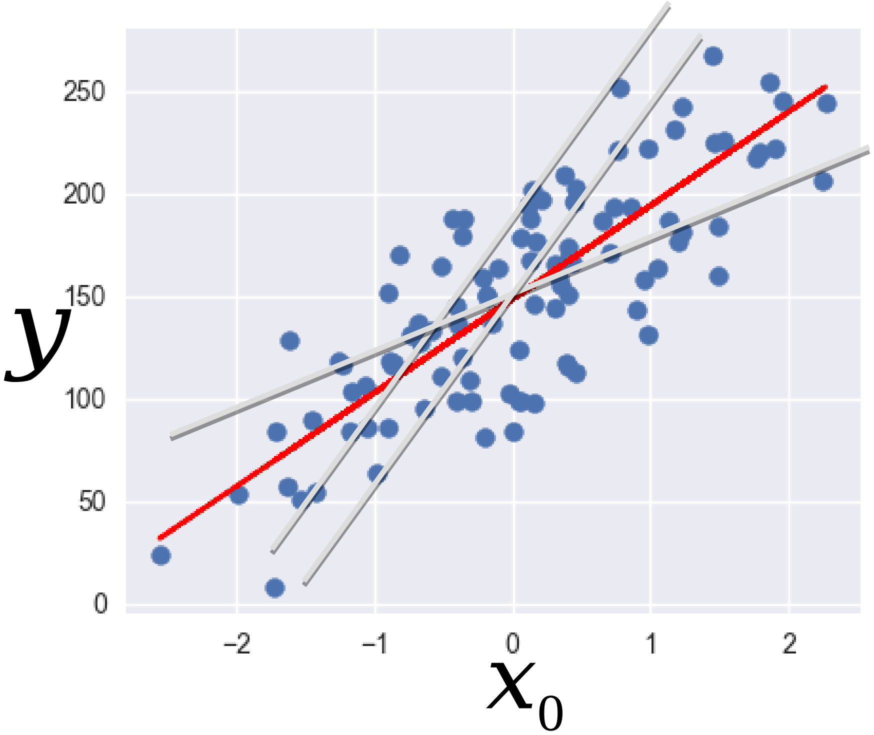
A Linear Regression Model with one Variable (Feature)

Input instance: =

Predicted output: =

Parameters to estimate: $\widehat{w}_0$ (slope)

(y-intercept)



Mean Squared Error

- residual sum of squares: the sum of squared differences between predicted target and actual target values.

- Goal: Minimize RSS
- Side note: Residuals are helpful in evaluation!

Pair of pairs

- **Two justifications:**
  - *Empirical Risk Minimization*
  - *Maximum Likelihood Estimator*
- **Two ways to compute**

2 Line Segment Title

Maximum Likelihood Estimation

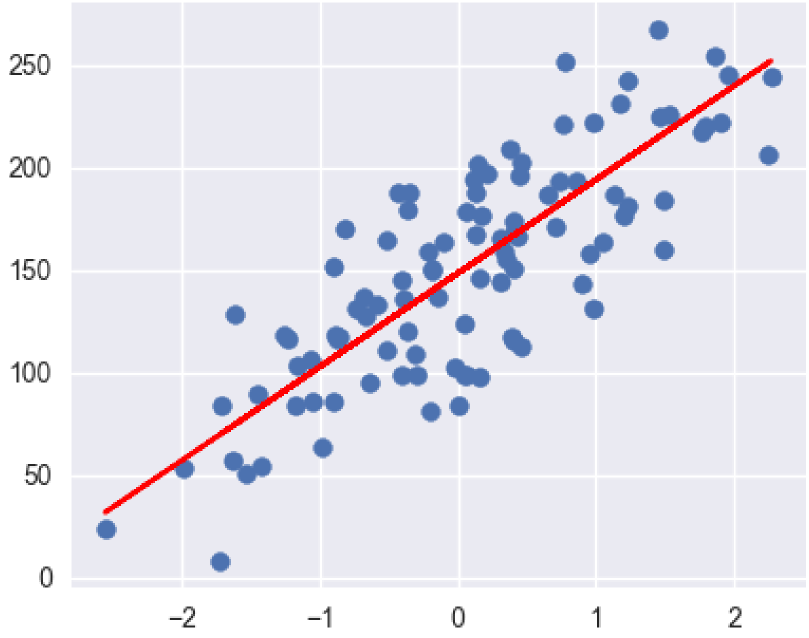
- **Given p($y|w,x$)**
  - *Example:* $p(y|w,x)$ = *Normal (mean* wx, *var* σ^2*)*
- **Compute p(w | x,y) via Bayes rule**
  - $p(w|x,y) = p(w|x)p(y|w,x)/p(y|x)$
  - $p(w|x,y) \quad p(y|w,x)$
- **Gives distribution for w**
- **Take mode = MLE**
- **MLE minimizes RSS on above example (regardless of σ^2).**

Computation

- $\boldsymbol{w = (X^T X)^{-1}(X^T Y)}$
  - *X, Y are inputs and outputs*
  - *"Min" Xw-Y*
  - *No constant term, just use input 1*
- **Can do matrix multiplication**
- **Can just use stochastic gradient descent**
  - *More on this later in the course*
  - *Typically faster*

Least-Squares Linear Regression in Scikit-Learn

```
from sklearn.linear_model import LinearRegression

linreg = LinearRegression().fit(X_train, y_train)
```



Linear Regression

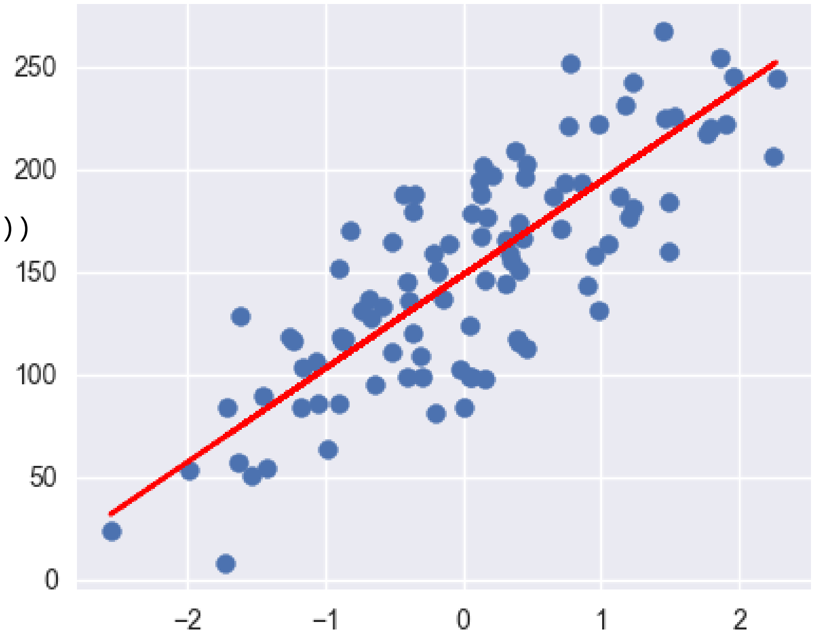
- **Intuition**
  - *Linear!*
- **Parameters**
  - *Fit bias term (if so, then can normalize)*
- **Evaluation**
  - *What are ways you can see what happening?*

Least-Squares Linear Regression in Scikit-Learn

```
from sklearn.linear_model import LinearRegression

X_train, X_test, y_train, y_test =
      train_test_split(X_R1, y_R1, random_state = 0)

linreg = LinearRegression().fit(X_train, y_train)

print("linear model intercept (b): {}".format(linreg.intercept_))
print("linear model coeff (w): {}".format(linreg.coef_))
```

`linreg.coef_`   `linreg.intercept_`

=



The R^2 ('r-squared') Regression Score

- **Measures how well a prediction model for regression fits the given data.**
- **The score is between 0 and 1:**
  - *A value of 0 corresponds to a constant model that predicts the mean value of all training target values.*
  - *A value of 1 corresponds to perfect prediction*
- **Also known as 'coefficient of determination“**
  - *Fraction of variance explained*

Regression metrics

- **Typically r2_score is enough**
  - *Reminder: computes how well future instances will be predicted*
  - *Best possible score is 1.0*
  - *Constant prediction score is 0.0*
- **Alternative metrics include:**
  - *mean_absolute_error (absolute difference of target & predicted values)*
  - *mean_squared_error (squared difference of target & predicted values)*
  - *median_absolute_error (robust to outliers)*

Residuals

- **Look at Xw-Y**
  - *Distribution*
  - *Outliers*
  - *Is there a pattern? Is your prediction systematically off?*

Cross-validation

Interlude

Cross-validation

- **Uses multiple train-test splits, not just a single one**
- **Each split used to train & evaluate a separate model**
- **Why is this better?**

random_state	Test set accuracy
0	1.00
1	0.93
5	0.93
7	0.67
10	0.87

Accuracy of k-NN classifier (k=5) on fruit data test set for different random_state values in train_test_split.

Cross-validation Example (5-fold)

Original dataset		Model 1	Model 2	Model 3	Model 4	Model 5
	Fold 1	Test	Train	Train	Train	Train
	Fold 2	Train	Test	Train	Train	Train
	Fold 3	Train	Train	Test	Train	Train
	Fold 4	Train	Train	Train	Test	Train
	Fold 5	Train	Train	Train	Train	Test

Stratified Cross-validation

fruit_label	fruit_name
1	Apple
1	Apple
1	Apple
1	Apple
1	Apple
2	Mandarin
...	...
3	Orange
...	...
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon

(Folds and dataset shortened for illustration purposes.)

Example has 20 data samples
= 4 classes with 5 samples each.

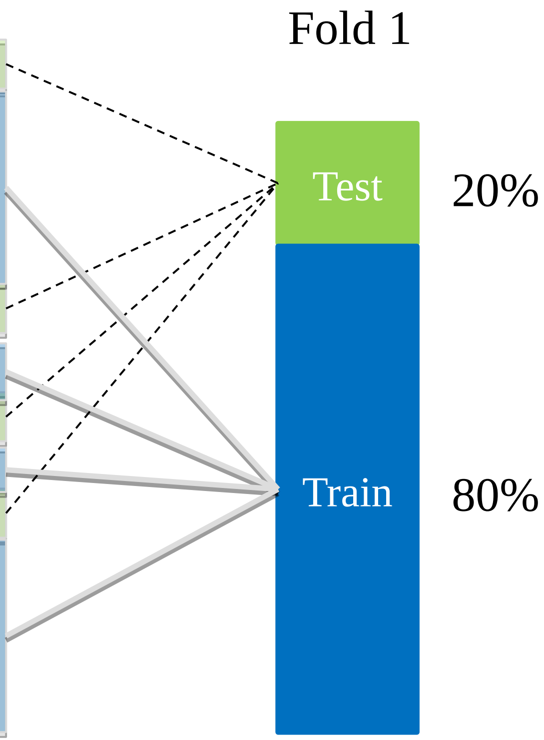
5-fold CV: 5 folds of 4 samples each.

Fold 1 uses the first 20% of the dataset as the test set, which only contains samples from class 1.

Classes 2, 3, 4 are missing entirely from test set and so will be missing from the evaluation.

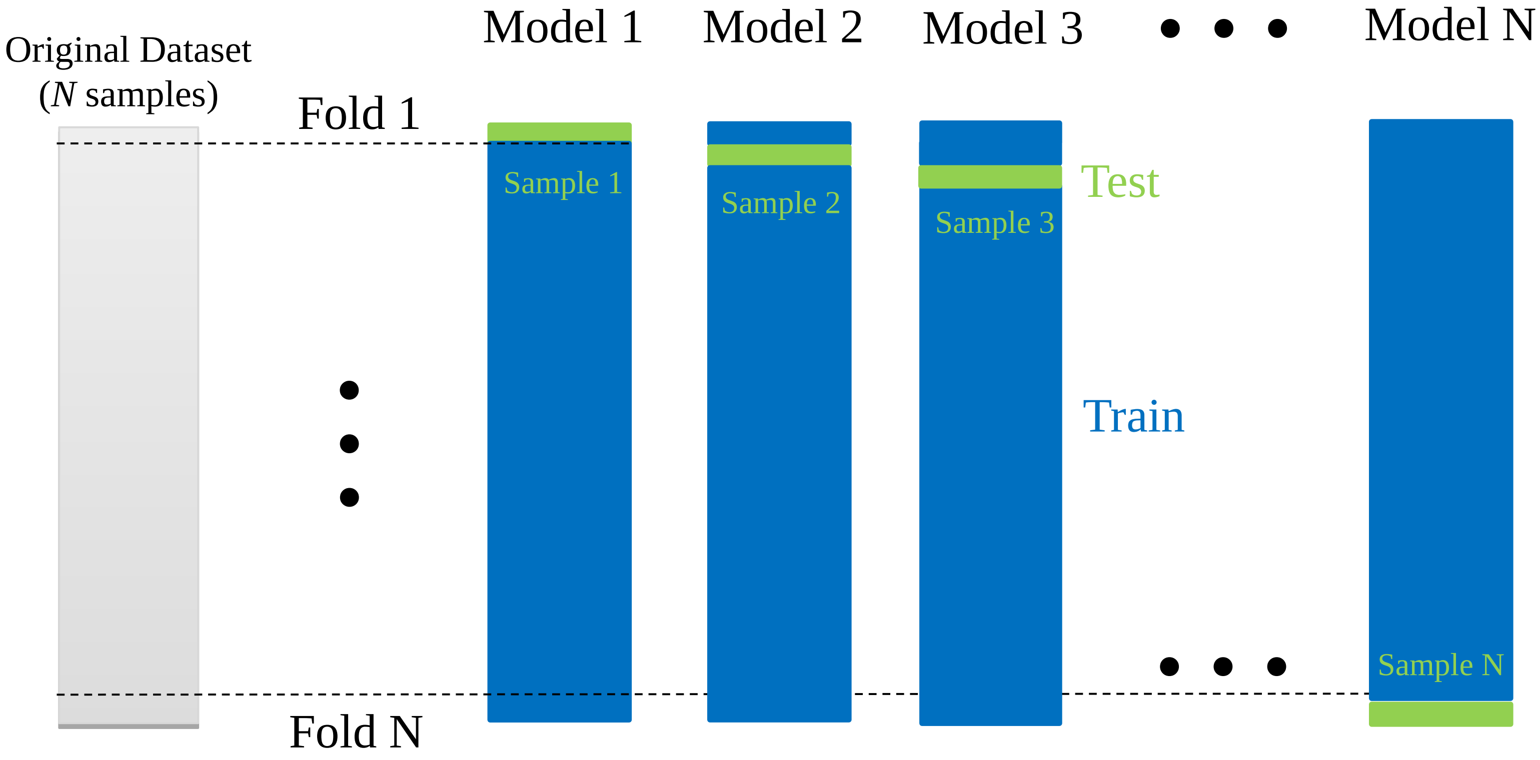
Stratified Cross-validation

fruit_label	fruit_name
1	Apple
1	Apple
1	Apple
1	Apple
1	Apple
2	Mandarin
...	...
3	Orange
...	...
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon



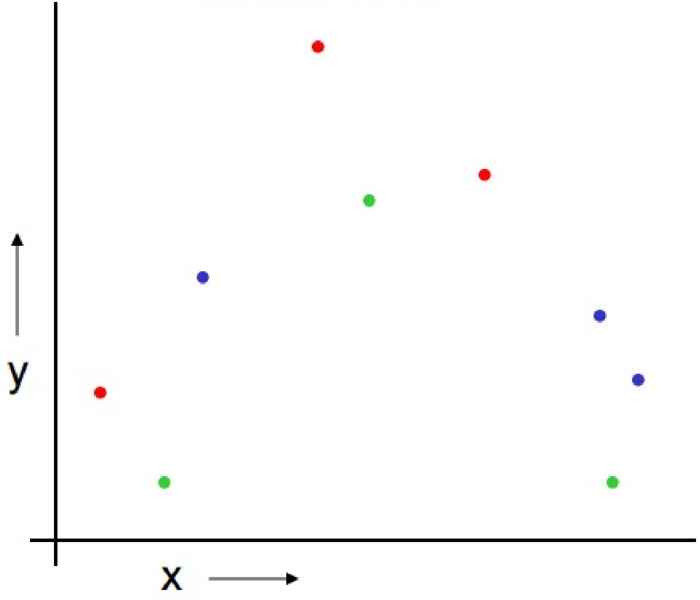
Stratified folds each contain a proportion of classes that matches the overall dataset. Now, all classes will be fairly represented in the test set.

Leave-one-out cross-validation (with *N* samples in dataset)



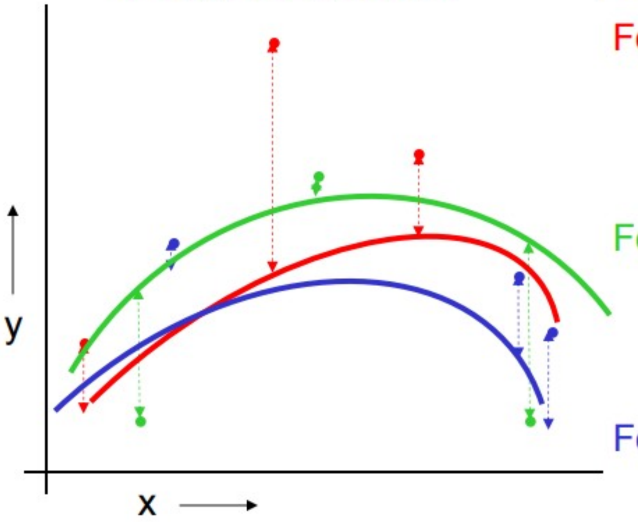
k-fold Cross Validation

Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Red Green and Blue)



Graphic courtesy of Andrew W. Moore: https://www.autonlab.org/tutorials

k-fold Cross Validation



Quadratic Regression
$MSE_{3FOLD}=1.11$

Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Red Green and Blue)

For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

Then report the mean error

Graphic courtesy of Andrew W. Moore: https://www.autonlab.org/tutorials

K-fold Cross Validation

- **Code to run**

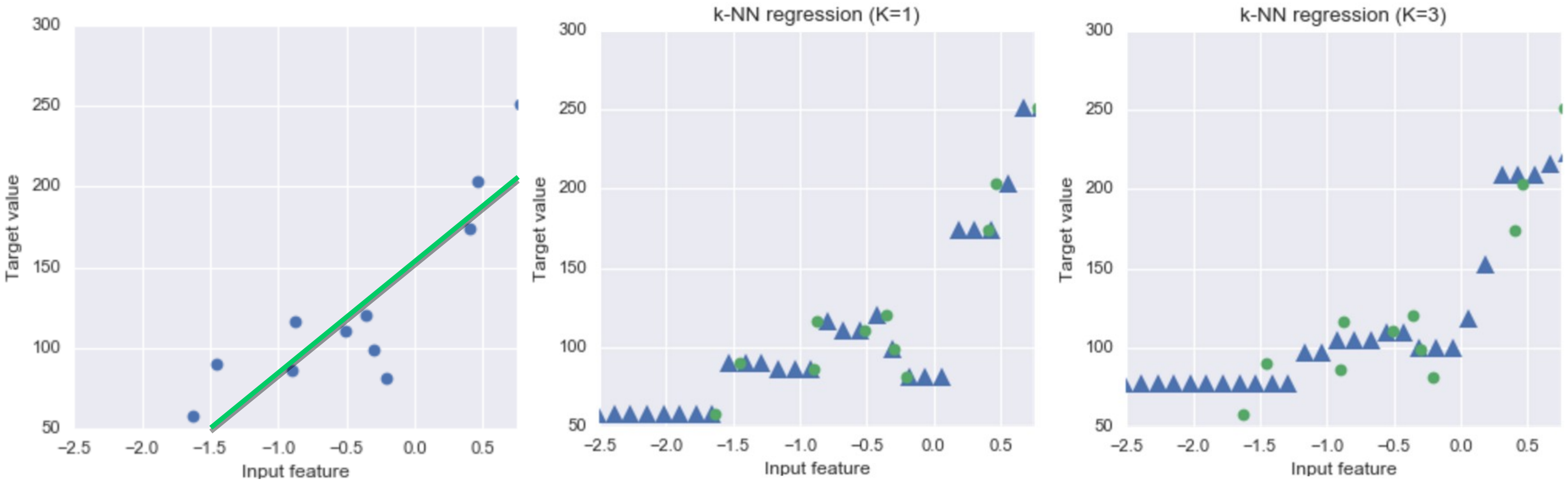
```
>>> scores = cross_validate(clf, iris.data, iris.target,
...                         scoring='precision_macro', cv=5,
...                         return_estimator=True)
>>> sorted(scores.keys())
['estimator', 'fit_time', 'score_time', 'test_score']
```

Think, Pair, Share

- **In 10 fold cross validation, how many different predictors are learned?**
  - *A) One, all the changes is the test set*
  - *B) Nine, you learn one for each fold not in the test set.*
  - *C) Ten, you a different predictor to test each of the ten folds*

Linear versus k-NN Regression

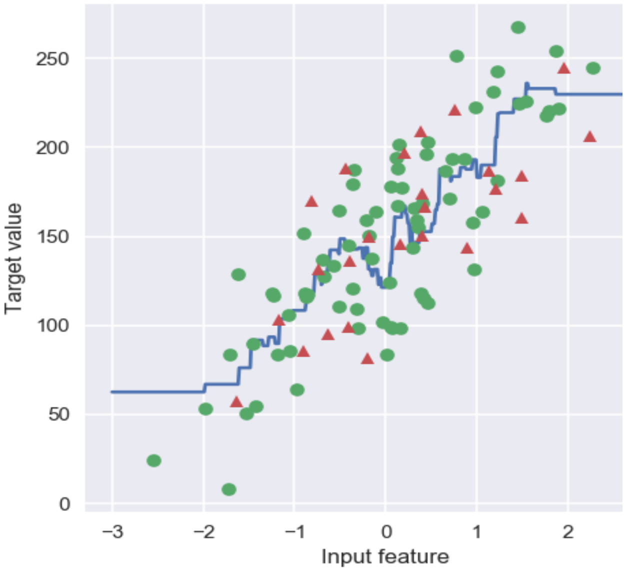
k-Nearest Neighbors Regression



K-NN Regression vs Least-Squares Linear Regression

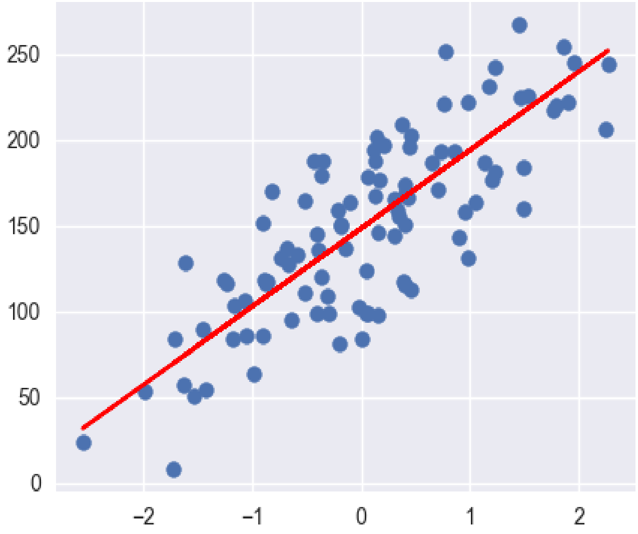
k-NN (k=7)

```
R-squared score (training): 0.720
R-squared score (test): 0.471
```



LS linear

```
R-squared score (training): 0.679
R-squared score (test): 0.492
```



Linear versus k-NN Regression

Linear Regression

- **Few parameters**
- **Can learn from few points**
- **Generalizes beyond points**

K-NN regression

- **Non-parametric**
- **Requires many point**
  - *especially in high dimensions*
- **Limited generalization**

Datasets

Scikit-Learn Data Sets

- **Two types of internal data sets:**
  - *Empirical*
  - *Synthetic*
- **Can also load external data from file.**

Loading from scikit-learn: Bunch objects

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
```

The resulting object from scikit-learn is a Bunch object, similar to a dictionary.

```
dict_keys(['DESCR', 'data', 'target_names', 'target',
'feature_names'])
```

You can also request the dataset be split for you into X and y:

```
(X_cancer, y_cancer) = load_breast_cancer(return_X_y = True)
```

Some datasets have a key 'DESCR' that contains a more detailed description.

scikit-learn.datasets (Toy problems)

load_boston([return_X_y])	Load and return the boston house-prices dataset (regression).
load_iris([return_X_y])	Load and return the iris dataset (classification).
load_diabetes([return_X_y])	Load and return the diabetes dataset (regression).
load_digits([n_class, return_X_y])	Load and return the digits dataset (classification).
load_linnerud([return_X_y])	Load and return the linnerud dataset (multivariate regression). Physiological data.
load_wine([return_X_y])	Load and return the wine dataset (classification). Wine tasting.
load_breast_cancer([return_X_y])	Load and return the breast cancer wisconsin dataset (classification).

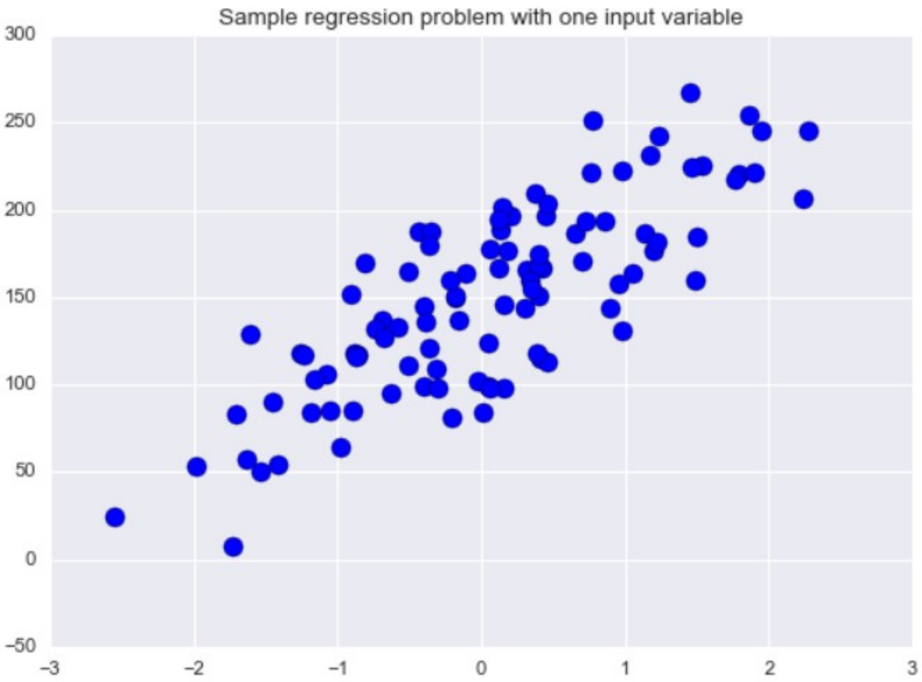
scikit-learn can easily create randomized synthetic datasets for you

- **Regression problems**
  - *make_regression(...)*

```
from sklearn.datasets import make_regression
plt.figure()
plt.title('Sample regression problem with one input variable')
X_R1, y_R1 = make_regression(n_samples = 100, n_features=1,
                            n_informative=1, bias = 150.0,
                            noise = 30, random_state=0)
plt.scatter(X_R1, y_R1, marker= 'o', s=50)
plt.show()
```

- make_friedman1(...) : more complex regression with lots of noise features

Simple Regression Dataset using make_regression



Synthetic classification problems with make_classification

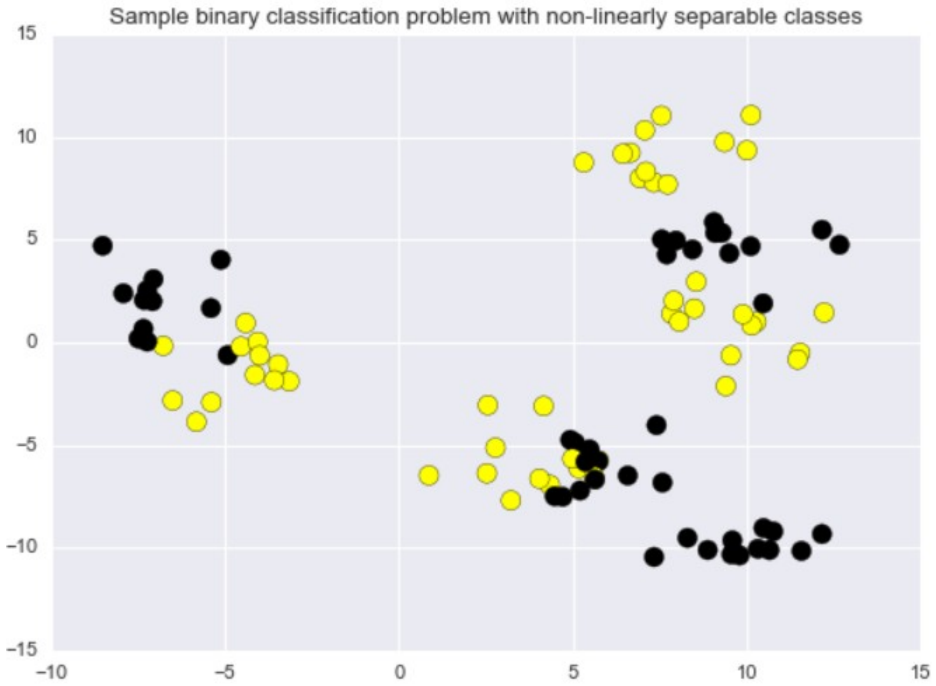
```
from sklearn.datasets import make_classification
plt.figure()
plt.title('Sample binary classification problem with two informative
features')
X_C2, y_C2 = make_classification(n_samples = 100, n_features=2,
                                n_redundant=0, n_informative=2,
                                n_clusters_per_class=1, flip_y = 0.1,
                                class_sep = 0.5, random_state=0)
plt.scatter(X_C2[:, 0], X_C2[:, 1], c=y_C2,
           marker= 'o', s=50, cmap=cmap_bold)
plt.show()
```

Simple Binary Classification Dataset

More complex classification problems with make_blobs

```
X_D2, y_D2 = make_blobs(n_samples = 100, n_features = 2, centers = 8,
                       cluster_std = 1.3, random_state = 4)
y_D2 = y_D2 % 2
plt.figure()
plt.title('Sample binary classification problem with non-linearly separable
classes')
plt.scatter(X_D2[:,0], X_D2[:,1], c=y_D2,
           marker= 'o', s=50, cmap=cmap_bold)
plt.show()
```

Complex Binary Classification Dataset



Real World Data Sets

6.3. Real world datasets

scikit-learn provides tools to load larger datasets, downloading them if necessary.

They can be loaded using the following functions:

`fetch_olivetti_faces` ([data_home, shuffle, …])	Load the Olivetti faces data-set from AT&T (classification).
`fetch_20newsgroups` ([data_home, subset, …])	Load the filenames and data from the 20 newsgroups dataset (classification).
`fetch_20newsgroups_vectorized` ([subset, …])	Load the 20 newsgroups dataset and vectorize it into token counts (classification).
`fetch_lfw_people` ([data_home, funneled, …])	Load the Labeled Faces in the Wild (LFW) people dataset (classification).
`fetch_lfw_pairs` ([subset, data_home, …])	Load the Labeled Faces in the Wild (LFW) pairs dataset (classification).
`fetch_covtype` ([data_home, …])	Load the covertype dataset (classification).
`fetch_rcv1` ([data_home, subset, …])	Load the RCV1 multilabel dataset (classification).
`fetch_kddcup99` ([subset, data_home, shuffle, …])	Load the kddcup99 dataset (classification).
`fetch_california_housing` ([data_home, …])	Load the California housing dataset (regression).

Communities and Crime Dataset

	population	householdsize	agePct12t21	agePct12t29	agePct16t24	agePct65up	numbUrban	pctUrban	medIncome	pctWWage	...	MedRentF
0	11980	3.10	12.47	21.44	10.93	11.33	11980	100.0	75122	89.24	...	23.8
1	23123	2.82	11.01	21.30	10.48	17.18	23123	100.0	47917	78.99	...	27.6
2	29344	2.43	11.36	25.88	11.01	10.28	29344	100.0	35669	82.00	...	24.1
3	16656	2.40	12.55	25.20	12.19	17.57	0	0.0	20580	68.15	...	28.7
5	140494	2.45	18.09	32.89	20.04	13.26	140494	100.0	21577	75.78	...	26.4
6	28700	2.60	11.17	27.41	12.76	14.42	28700	100.0	42805	79.47	...	24.4
7	59459	2.45	15.31	27.93	14.78	14.60	59449	100.0	23221	71.60	...	26.3
8	74111	2.46	16.64	35.16	20.33	8.58	74115	100.0	25326	83.69	...	25.2
9	103590	2.62	19.88	34.55	21.62	13.12	103590	100.0	17852	74.20	...	29.6
10	31601	2.54	15.73	28.57	15.16	14.26	31596	100.0	24763	73.92	...	23.8

Input features:
socio-economic data by location
from U.S. Census

Target variable:
Per capita violent crimes

Derived from the original UCI dataset at:

https://archive.ics.uci.edu/ml/datasets/Communities+and+Crime+Unnormalized

```
from adspy_shared_utilities import load_crime_dataset
crime = load_crime_dataset()
```

Loading from external data files

- Use pandas **read_table** to create a data frame from a local file

```
fruits = pd.read_table('fruit_data_with_colors.txt')
```

- Use Python urllib, plus NumPy **`loadtxt`** to read files on the Web

```
# Load the Pima Indians diabetes dataset from CSV URL
import numpy as np
import urllib
# URL for the Pima Indians Diabetes dataset (UCI Machine Learning Repository)
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
# download the file
raw_data = urllib.urlopen(url)
# load the CSV file as a numpy matrix
dataset = np.loadtxt(raw_data, delimiter=",")
# separate the data from the target attributes
X = dataset[:,0:7]
y = dataset[:,8]
```

See UCI Machine Learning Repository for more "classic" datasets
https://archive.ics.uci.edu/ml/index.php

Feature Expansion

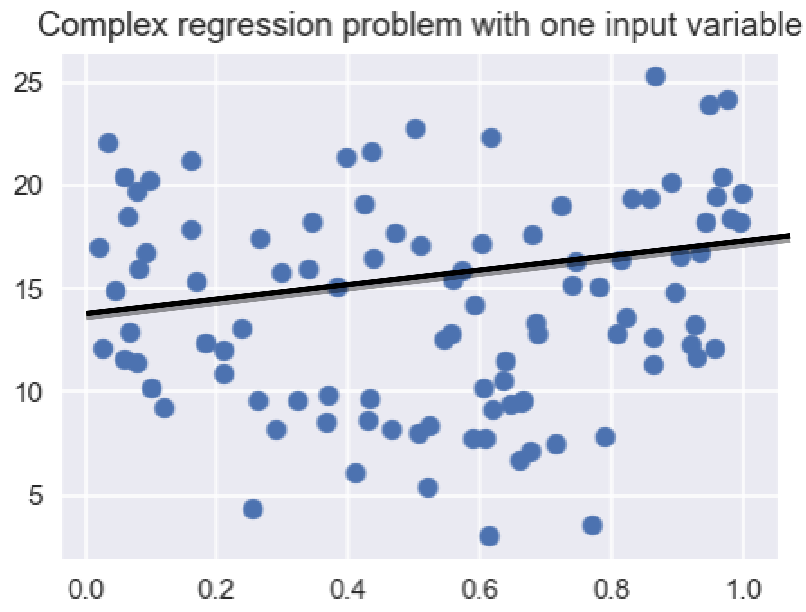
Polynomial Features with Linear Regression

$\boldsymbol{x} =$ $\Rightarrow$ $\boldsymbol{x}' =$

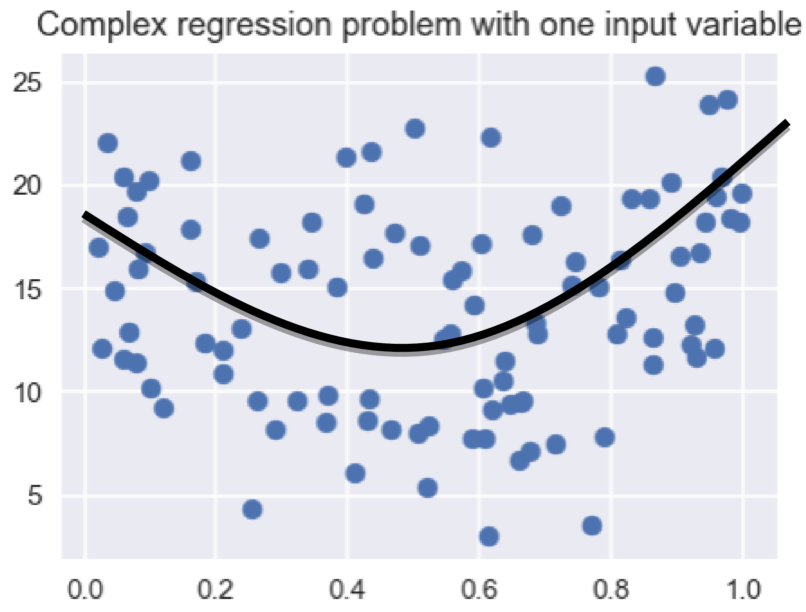
$=$

- Generate new features consisting of all polynomial combinations of the original two features .
- The *degree* of the polynomial specifies how many variables participate at a time in each new feature (above example: degree 2)
- This is still a weighted linear combination of features, so it's <u>still a linear model</u>, and can use same least-squares estimation method for *w* and *b*.

Least-Squares Polynomial Regression



Original feature



With polynomial degree 2 (quadratic) feature

Polynomial Features with Linear Regression

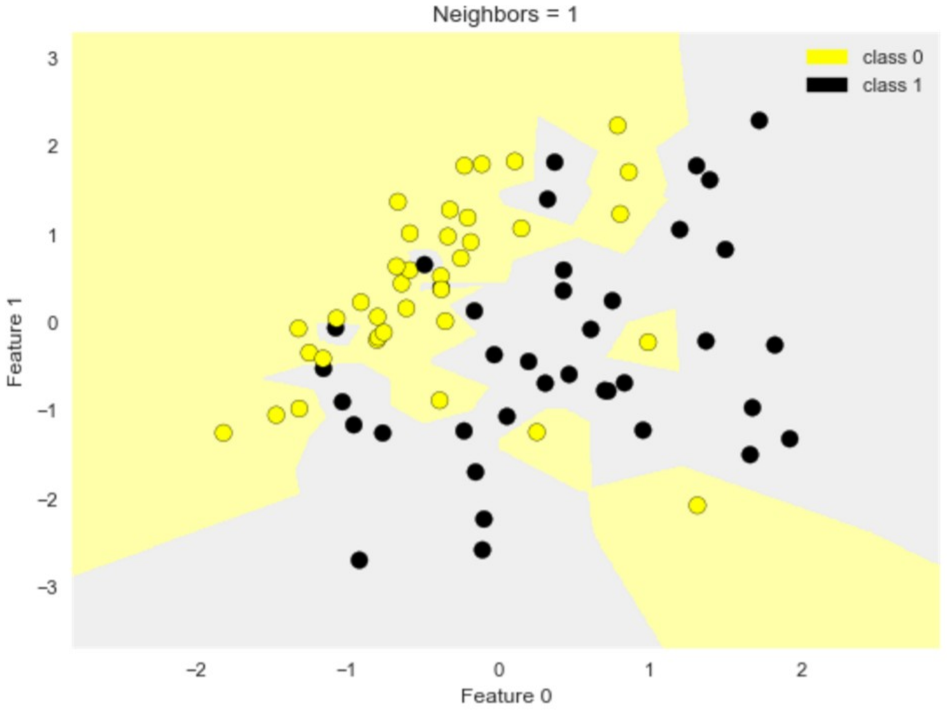
- **Why would we want to transform our data this way?**
  - *To capture interactions between the original features by adding them as features to the linear model.*
  - *To make a classification problem easier (we'll see this later).*
- **Beware of polynomial feature expansion with high as this can lead to complex models that overfit**
  - *Thus, polynomial feature expansion is often combined with a regularized learning method like ridge regression.*
- **Later: other transformations to create new features.**

Think, Pair, Share

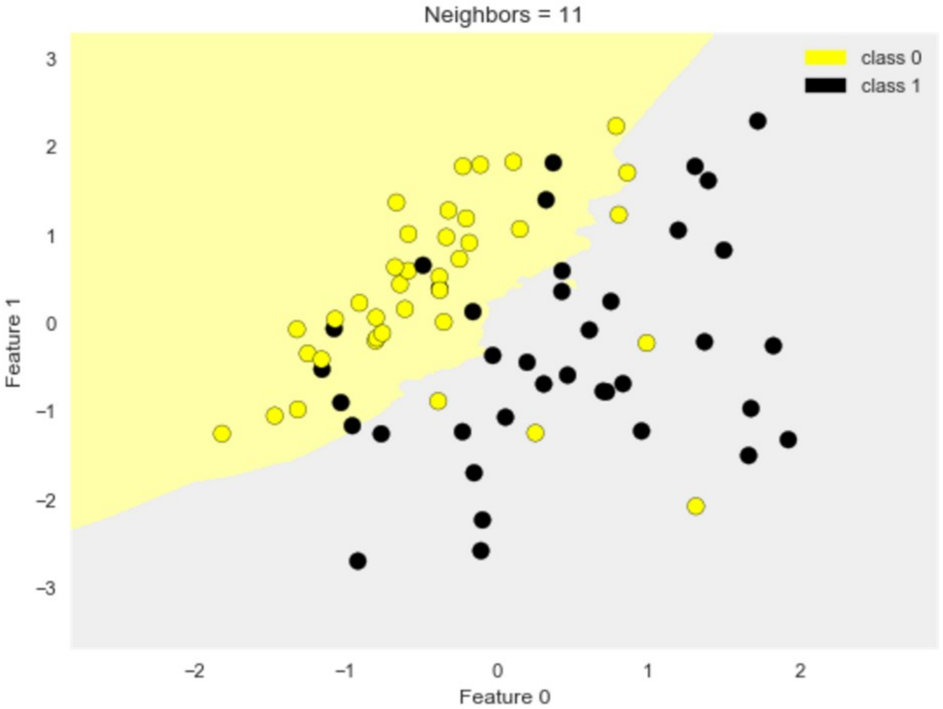
- **If you have 6 features and augment them to degree 2 polynomial features, how many features will you have?**
- **A) 6**
- **B) 28**
- **C) 36**
- **D) 6!**

Overfitting/Underfitting

Nearest Neighbors Classification (k=1)



Nearest Neighbors Classification (k=11)



Generalization, Overfitting, and Underfitting

- **Generalization ability refers to an algorithm's ability to give accurate predictions for new, previously unseen data.**
- **Assumptions:**
  - *Future unseen data (test set) will have the same properties as the current training sets.*
  - *Thus, models that are accurate on the training set are expected to be accurate on the test set.*
  - *But that may not happen if the trained model is tuned too specifically to the training set.*
- **Models that are too complex for the amount of training data available are said to overfit and are not likely to generalize well to new examples.**
- **Models that are too simple, that don't even do well on the training data, are said to underfit and also not likely to generalize well.**

2 Line Segment Title

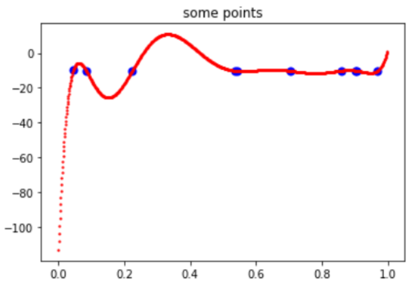
Overfitting in regression

In [123]:

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

num_points = 10

W = np.arange(0, 1, .001)
X = np.random.rand(num_points)
noise = np.random.rand(num_points)
Y = X*X*X*X - 1*X*X*X + 1* X*X - 1*X -10 + noise/10
reg = LinearRegression()
poly = PolynomialFeatures(9)
X_poly = poly.fit_transform(X.reshape(-1, 1))
W_poly = poly.transform(W.reshape(-1, 1))
reg.fit(X_poly, Y.reshape(-1, 1))
Z = reg.predict(W_poly)
plt.figure()
plt.title('some points')
plt.scatter(X, Y, color = 'b', marker= 'o', s=50)
plt.scatter(W, Z, c= 'r', marker= 'o', s=2)
```

Out[123]:

```
<matplotlib.collections.PathCollection at 0x1a4dc332b3
```

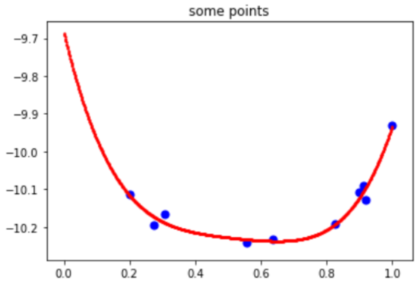


In [128]:

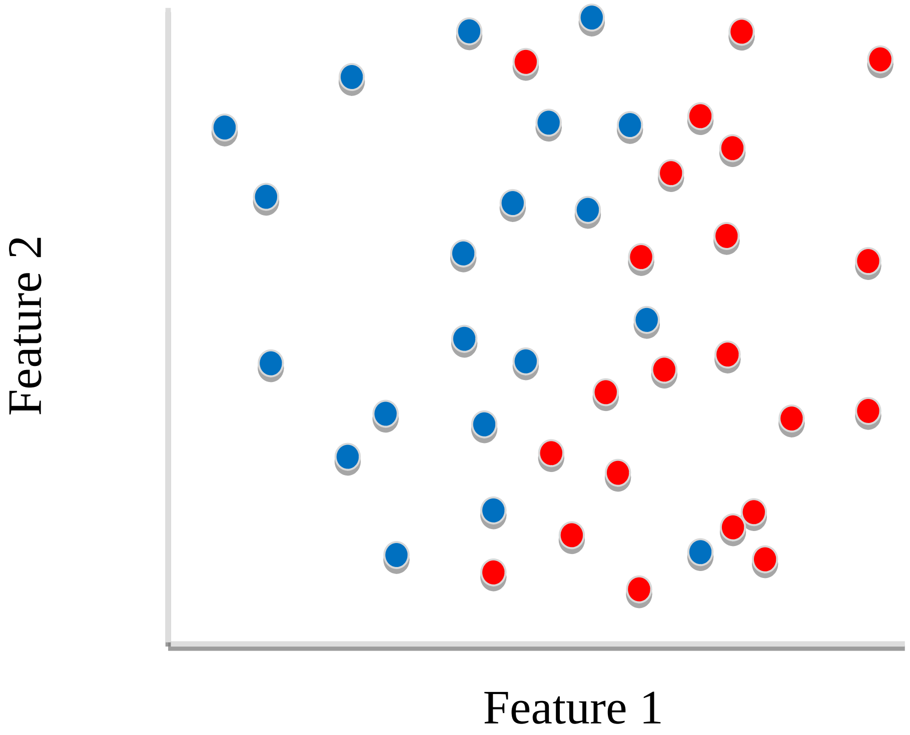
```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

num_points = 10


W = np.arange(0, 1, .001)
X = np.random.rand(num_points)
noise = np.random.rand(num_points)
Y = X*X*X*X - 1*X*X*X + 1* X*X - 1*X -10 + noise/10
reg = LinearRegression()
poly = PolynomialFeatures(4)
X_poly = poly.fit_transform(X.reshape(-1, 1))
W_poly = poly.transform(W.reshape(-1, 1))
reg.fit(X_poly, Y.reshape(-1, 1))
Z = reg.predict(W_poly)
plt.figure()
plt.title('some points')
plt.scatter(X, Y, color = 'b', marker= 'o', s=50)
plt.scatter(W, Z, c= 'r', marker= 'o', s=2)
```

Out[128]:

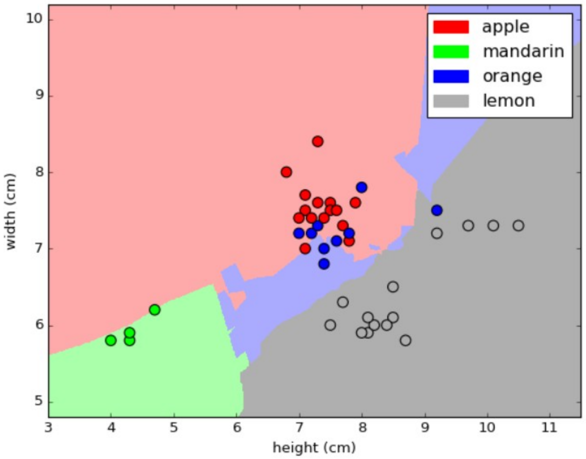
```
<matplotlib.collections.PathCollection at 0x1a4dc57fd68>
```



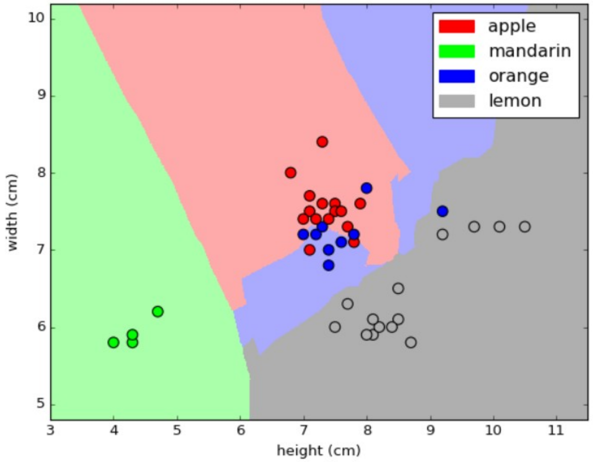
Overfitting in classification



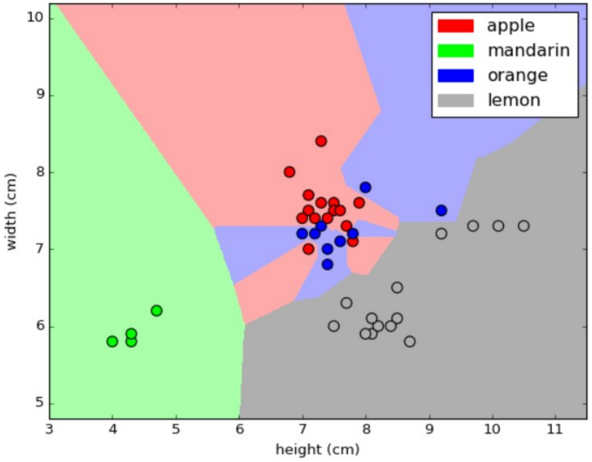
Overfitting with k-NN classifiers



K=10

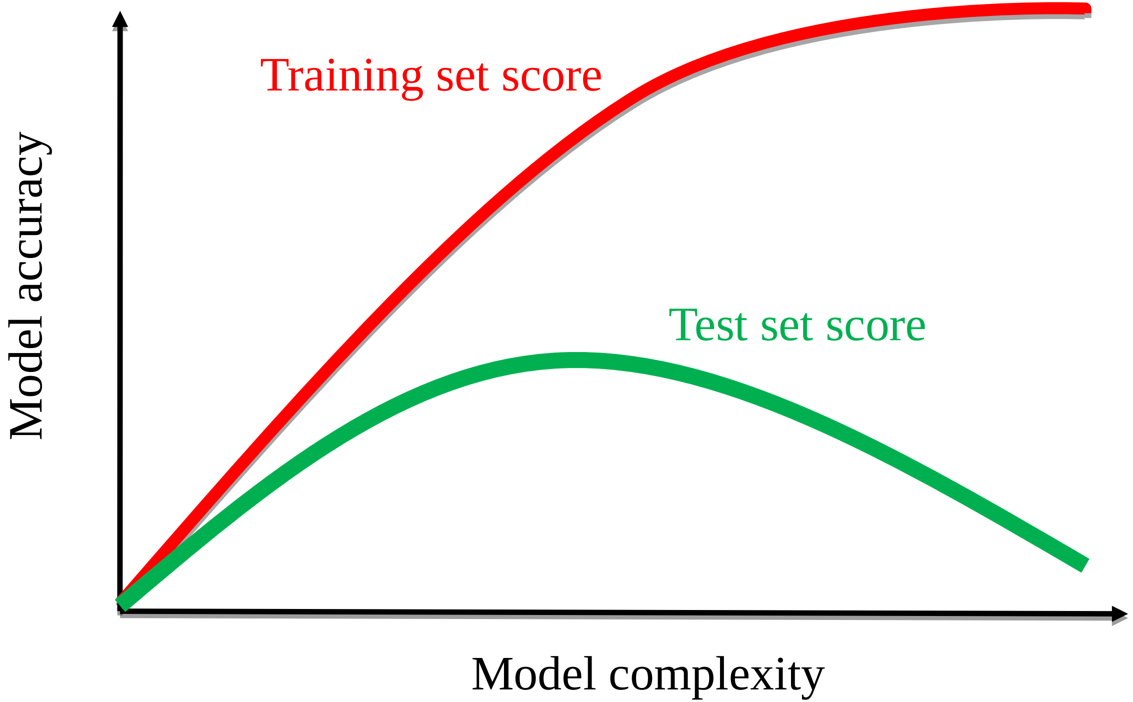


K=5

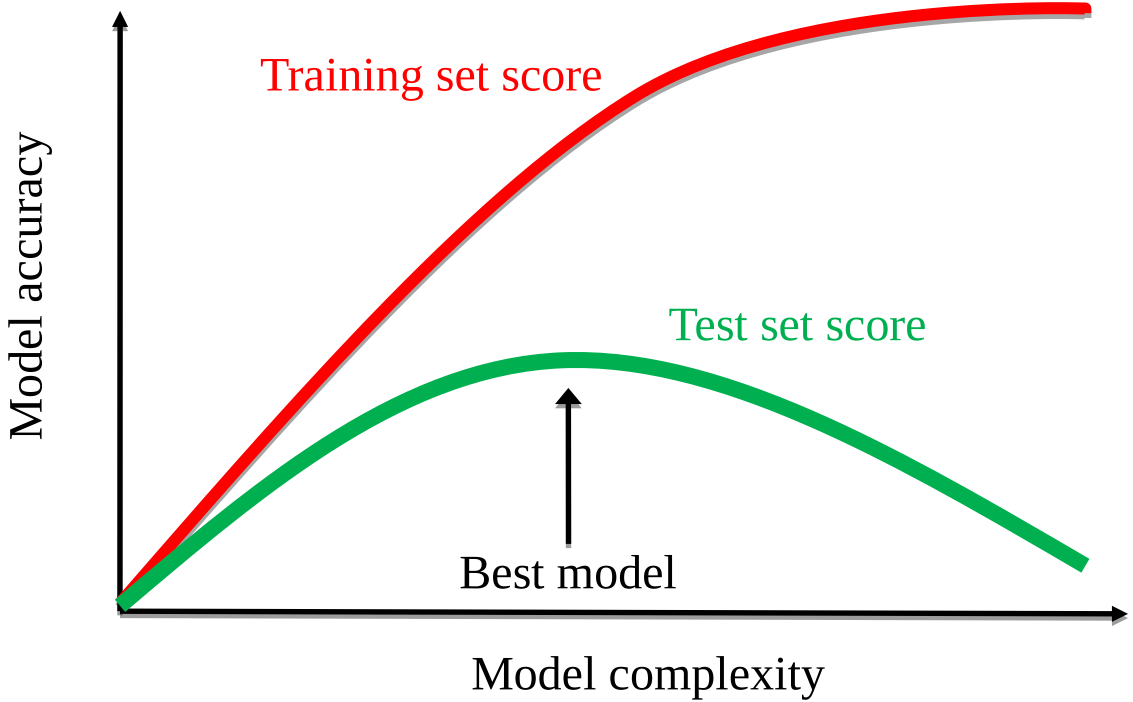


K=1

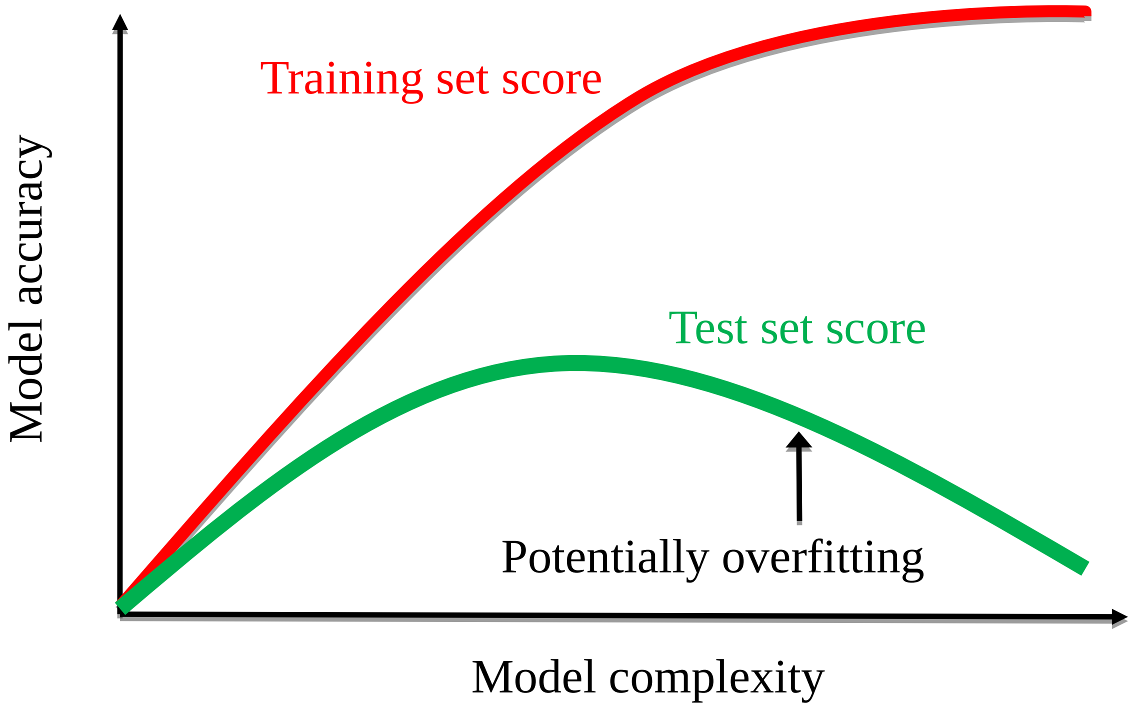
Where is the best model? (x-axis, model complexity)



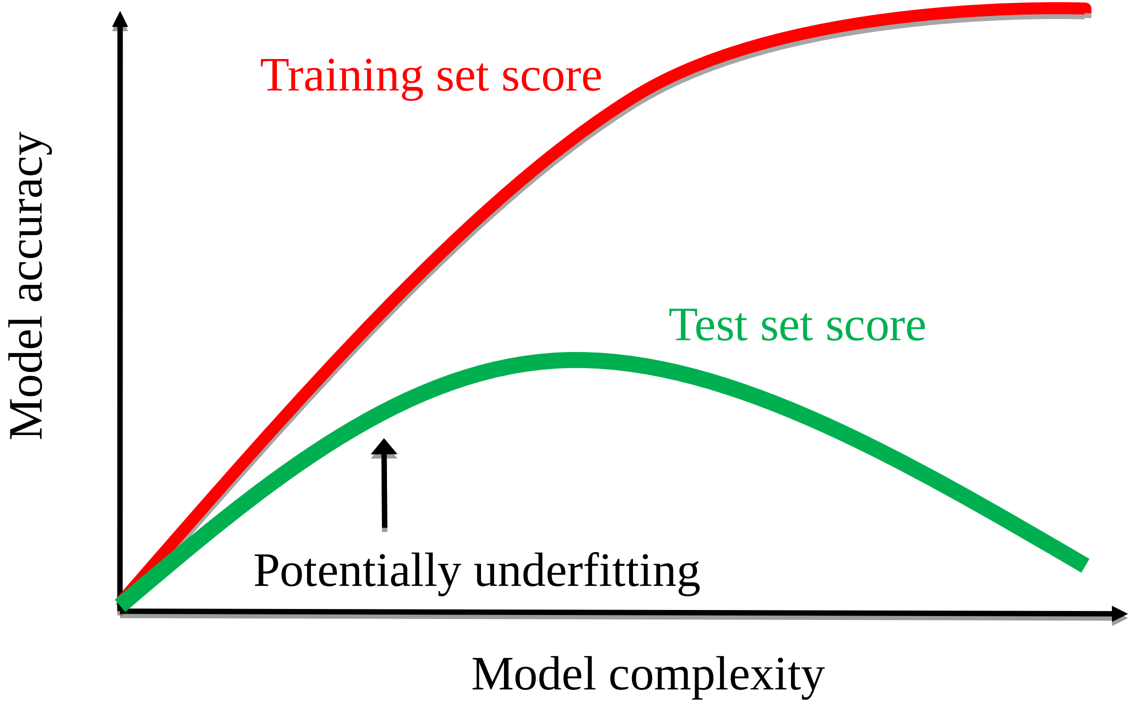
The relationship between model complexity and training/test performance



Detecting overfitting and underfitting



Detecting overfitting and underfitting



Dangerous overfitting situations

- **Your estimator has lots of parameters to be estimated, but you have a small dataset.**
  - *Linear regression: have at least 2 samples per "feature" (independent variable)*
  - *Logistic regression: have at least 5-10 samples per "feature"*

Ways to reduce (but not eliminate!) likelihood of overfitting

- Use cross-validation to get better estimates of training and test set error.
- Use regularization to restrict the complexity of the model (coming up).
- Check for data leakage of label info into training set (coming up).

Stats vs ML

- **(If time)**