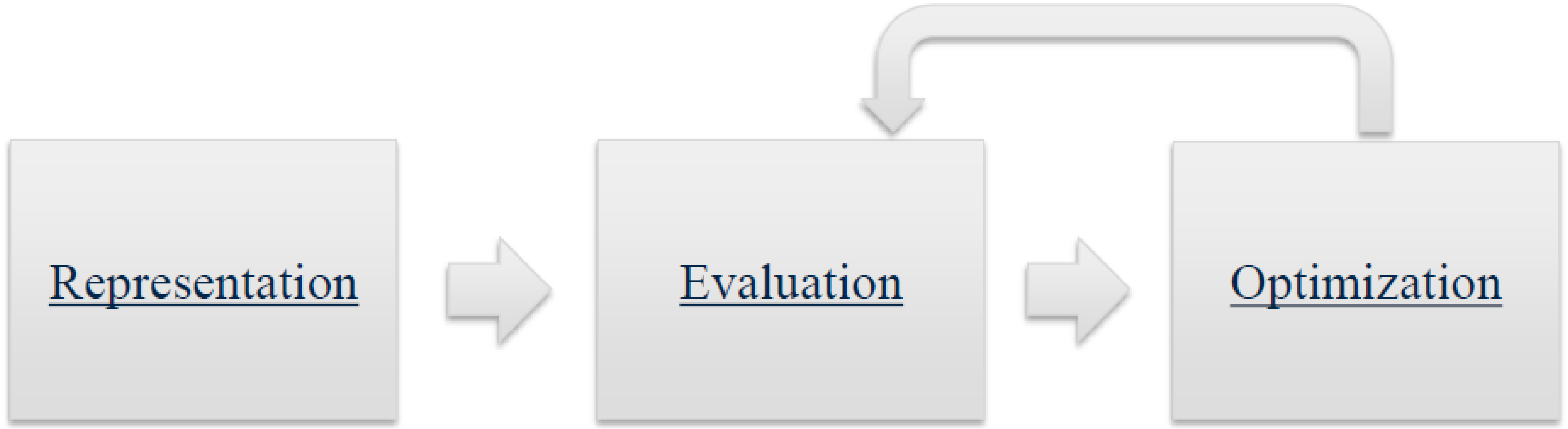
Introduction to Supervised Learning

Dr. Qiwei Gan

A Basic Machine Learning Workflow



Choose:
- A feature representation
- Type of classifier to use

e.g. image pixels, with
k-nearest neighbor classifier

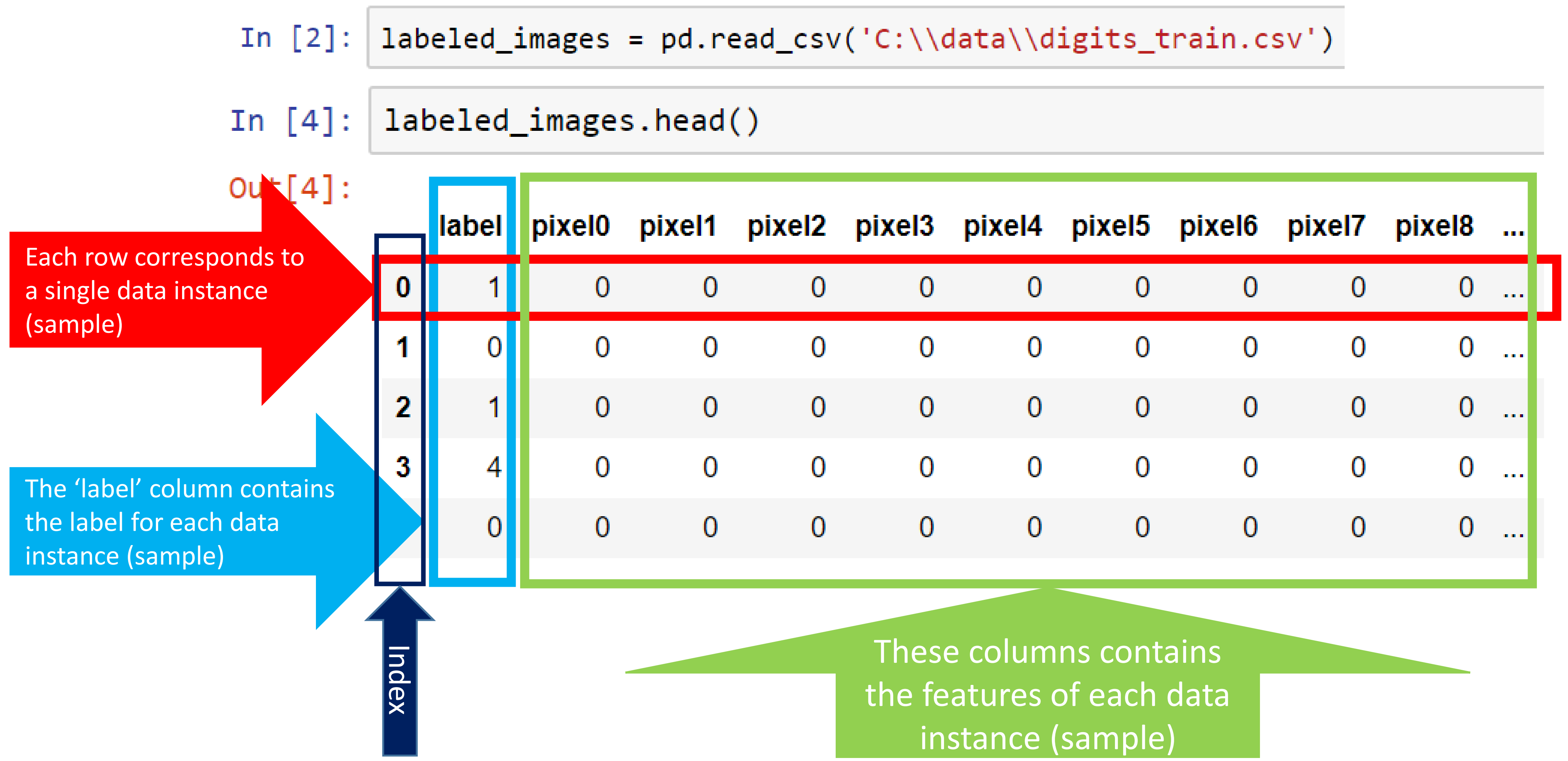
Choose:
- What criterion distinguishes good vs. bad classifiers?

e.g. % correct predictions on test set

Choose:
- How to search for the settings/parameters that give the best classifier for this evaluation criterion

e.g. try a range of values for "k" parameter in k-nearest neighbor classifier

Input Data as a Table



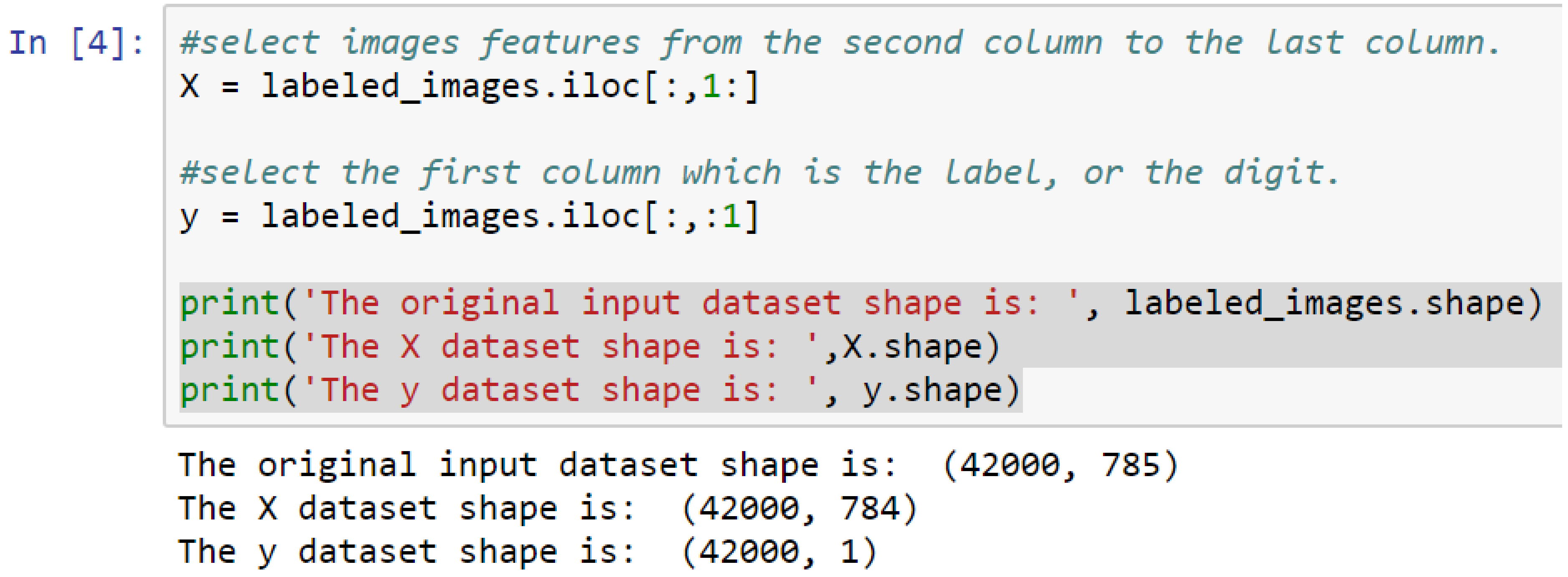
```
In [2]: labeled_images = pd.read_csv('C:\\data\\digits_train.csv')

In [4]: labeled_images.head()

Out[4]:
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...
0	1	0	0	0	0	0	0	0	0	0	...
1	0	0	0	0	0	0	0	0	0	0	...
2	1	0	0	0	0	0	0	0	0	0	...
3	4	0	0	0	0	0	0	0	0	0	...
	0	0	0	0	0	0	0	0	0	0	...

Each row corresponds to a single data instance (sample)
The 'label' column contains the label for each data instance (sample)
Index
These columns contains the features of each data instance (sample)

Conventions

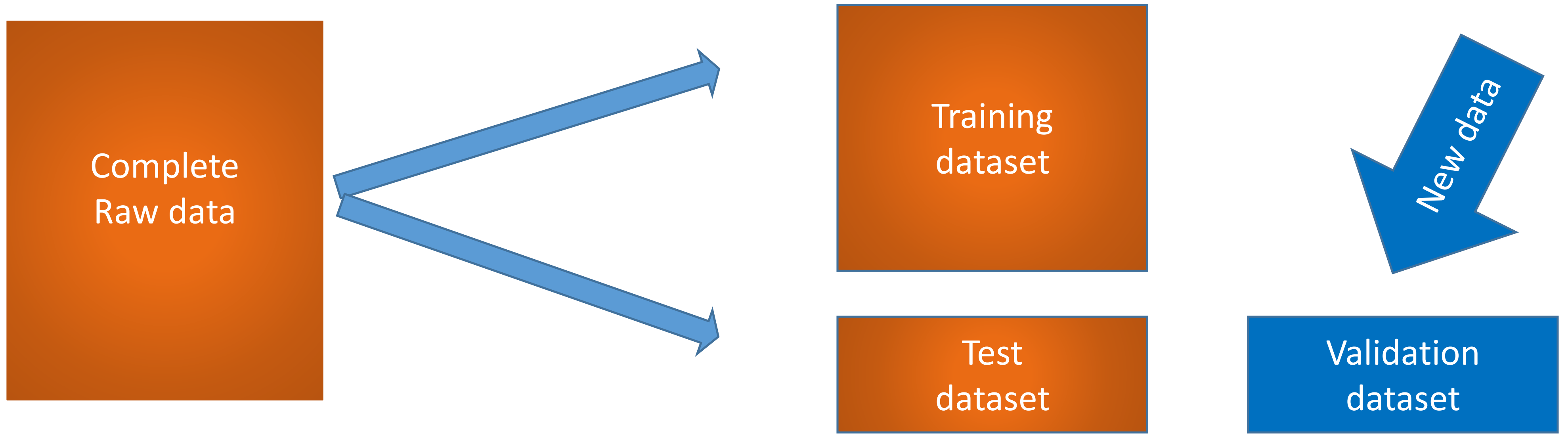
- Capital X: features matrix
- Lower y: label



```
In [4]: #select images features from the second column to the last column.
        X = labeled_images.iloc[:,1:]

        #select the first column which is the label, or the digit.
        y = labeled_images.iloc[:,:1]

        print('The original input dataset shape is: ', labeled_images.shape)
        print('The X dataset shape is: ',X.shape)
        print('The y dataset shape is: ', y.shape)
```

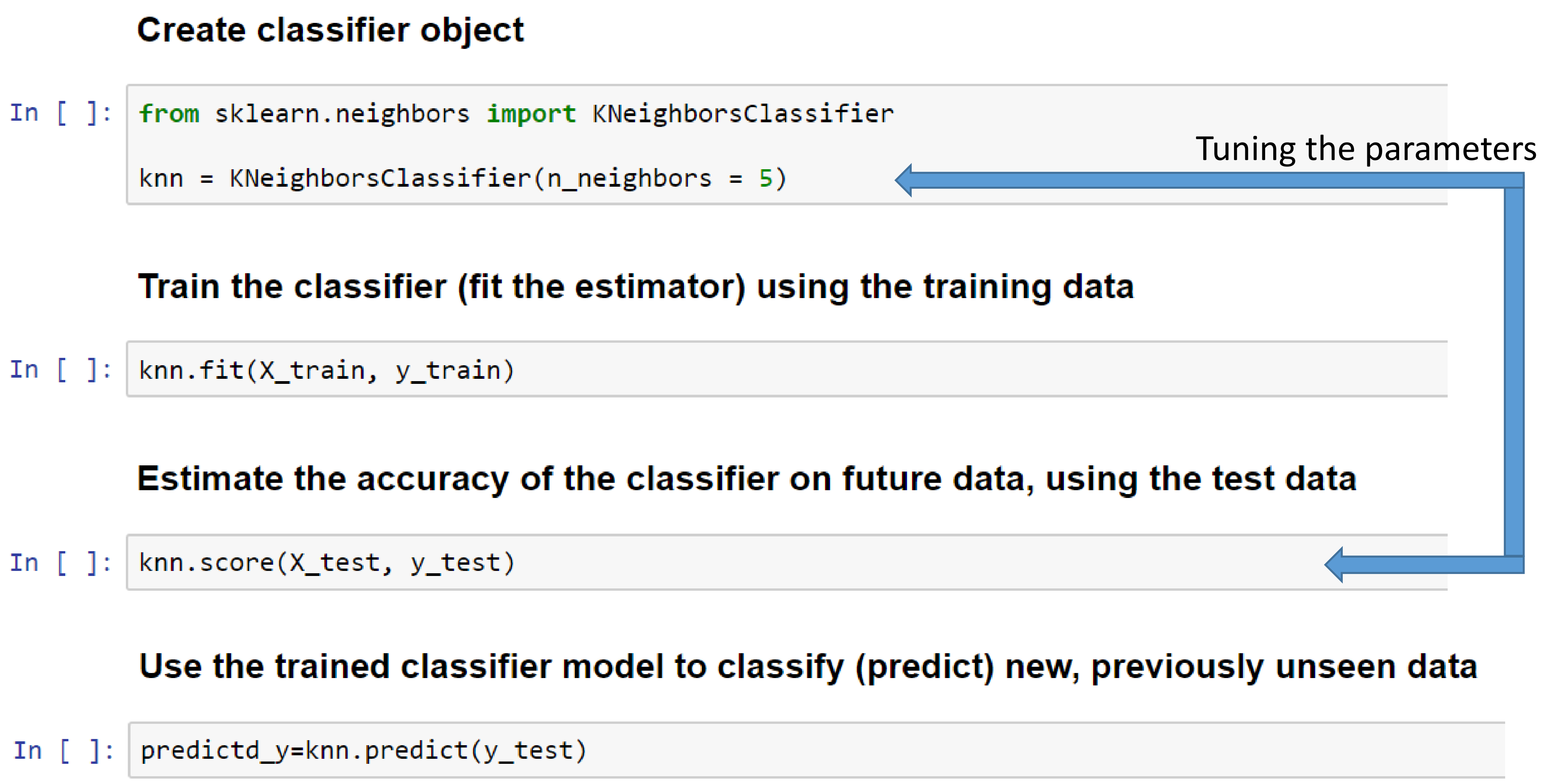
```
The original input dataset shape is:  (42000, 785)
The X dataset shape is:  (42000, 784)
The y dataset shape is:  (42000, 1)
```

Splitting Input Data into Train/Test



Data used to build the machine learning model, are called the *training data.*
The rest of the data will be used to assess how well the model works; these data are called *test data.*

Training Models



Create classifier object

```
In [ ]: from sklearn.neighbors import KNeighborsClassifier

        knn = KNeighborsClassifier(n_neighbors = 5)
```

Tuning the parameters

Train the classifier (fit the estimator) using the training data

```
In [ ]: knn.fit(X_train, y_train)
```

Estimate the accuracy of the classifier on future data, using the test data

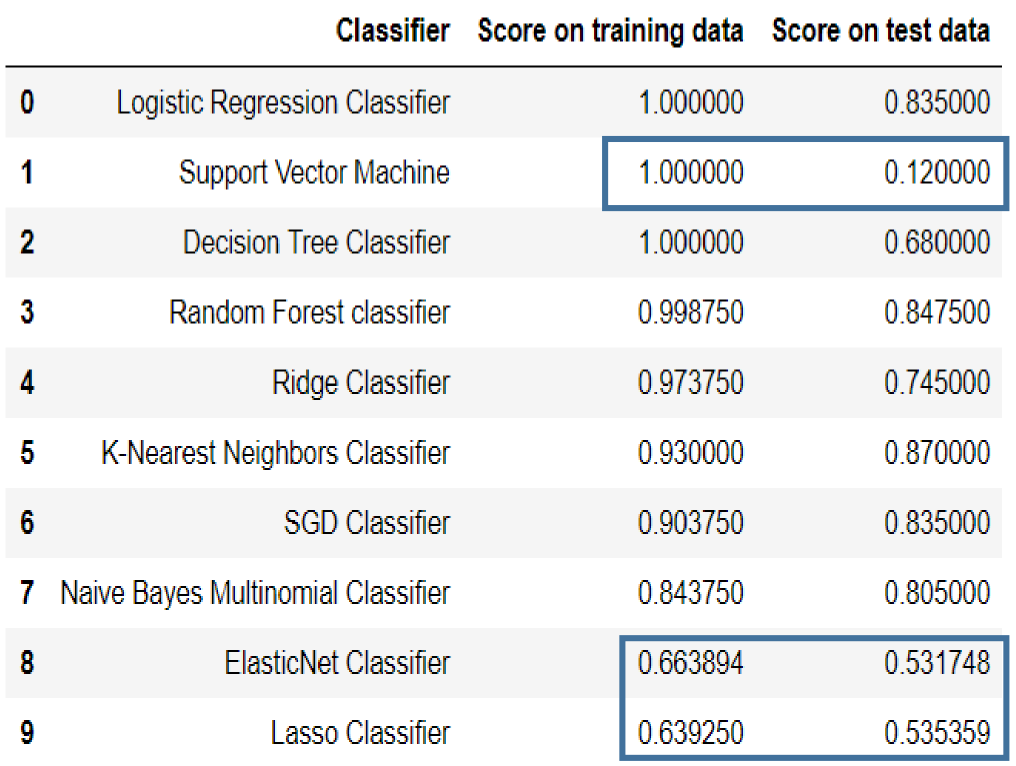
```
In [ ]: knn.score(X_test, y_test)
```

Use the trained classifier model to classify (predict) new, previously unseen data

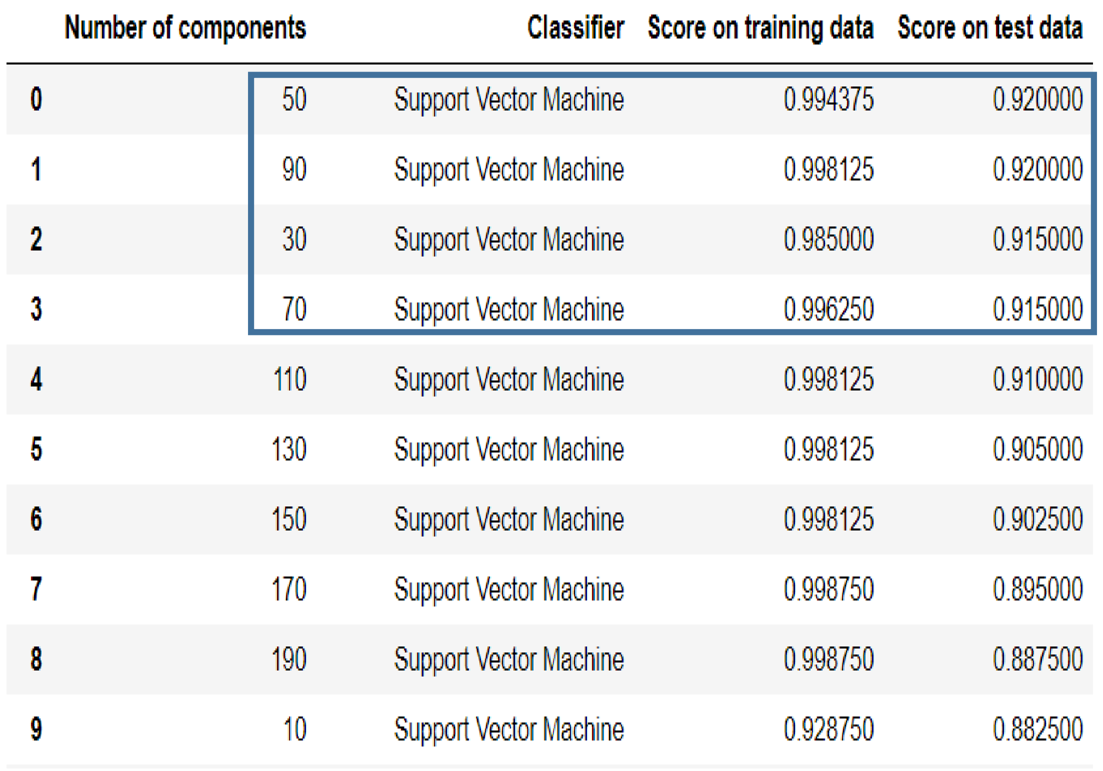
```
In [ ]: predictd_y=knn.predict(y_test)
```

Questions?

Overfitting and Underfitting



	Classifier	Score on training data	Score on test data
0	Logistic Regression Classifier	1.000000	0.835000
1	Support Vector Machine	1.000000	0.120000
2	Decision Tree Classifier	1.000000	0.680000
3	Random Forest classifier	0.998750	0.847500
4	Ridge Classifier	0.973750	0.745000
5	K-Nearest Neighbors Classifier	0.930000	0.870000
6	SGD Classifier	0.903750	0.835000
7	Naive Bayes Multinomial Classifier	0.843750	0.805000
8	ElasticNet Classifier	0.663894	0.531748
9	Lasso Classifier	0.639250	0.535359



	Number of components	Classifier	Score on training data	Score on test data
0	50	Support Vector Machine	0.994375	0.920000
1	90	Support Vector Machine	0.998125	0.920000
2	30	Support Vector Machine	0.985000	0.915000
3	70	Support Vector Machine	0.996250	0.915000
4	110	Support Vector Machine	0.998125	0.910000
5	130	Support Vector Machine	0.998125	0.905000
6	150	Support Vector Machine	0.998125	0.902500
7	170	Support Vector Machine	0.998750	0.895000
8	190	Support Vector Machine	0.998750	0.887500
9	10	Support Vector Machine	0.928750	0.882500

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.

Trade-off between Complexity and Generalization

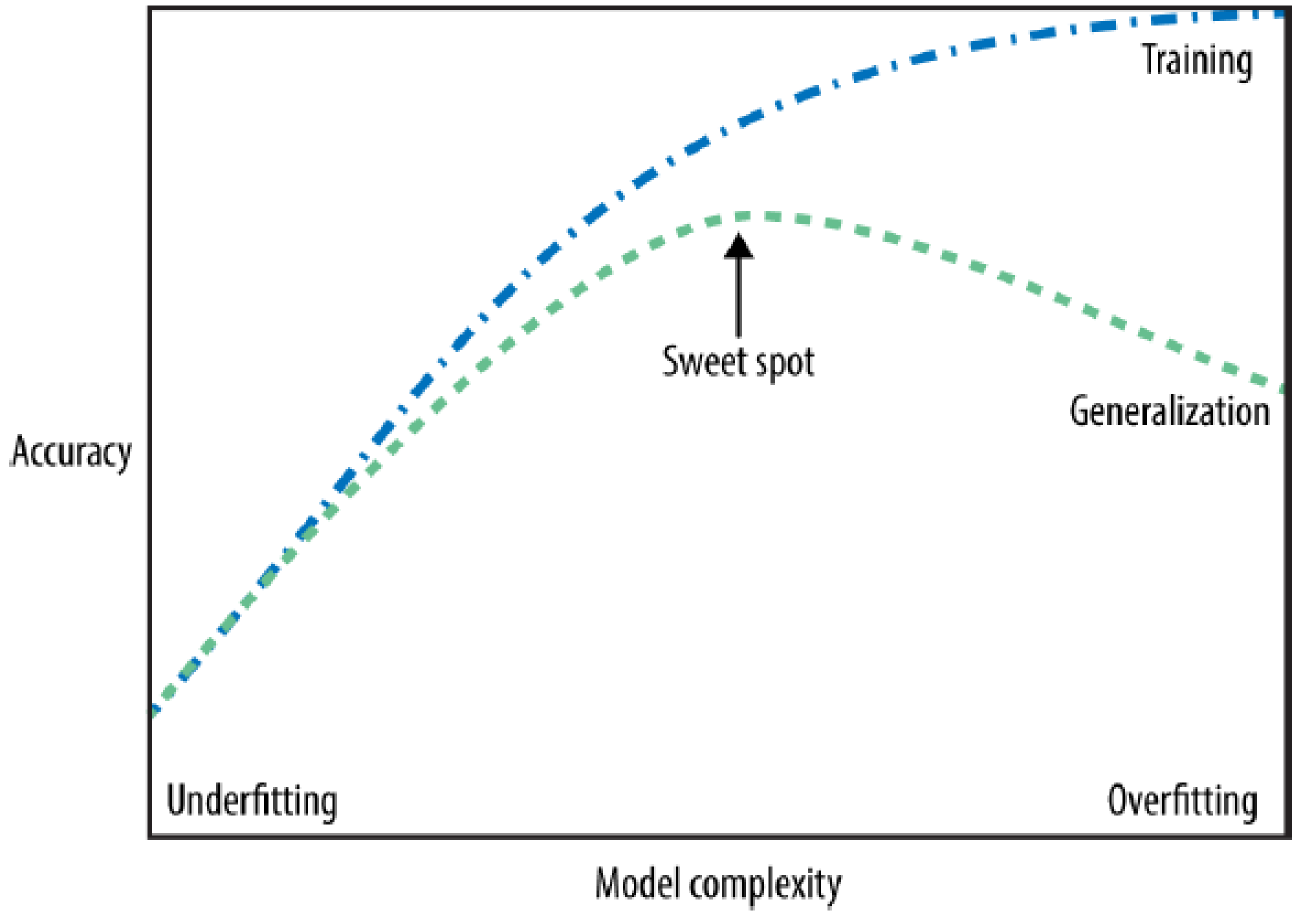
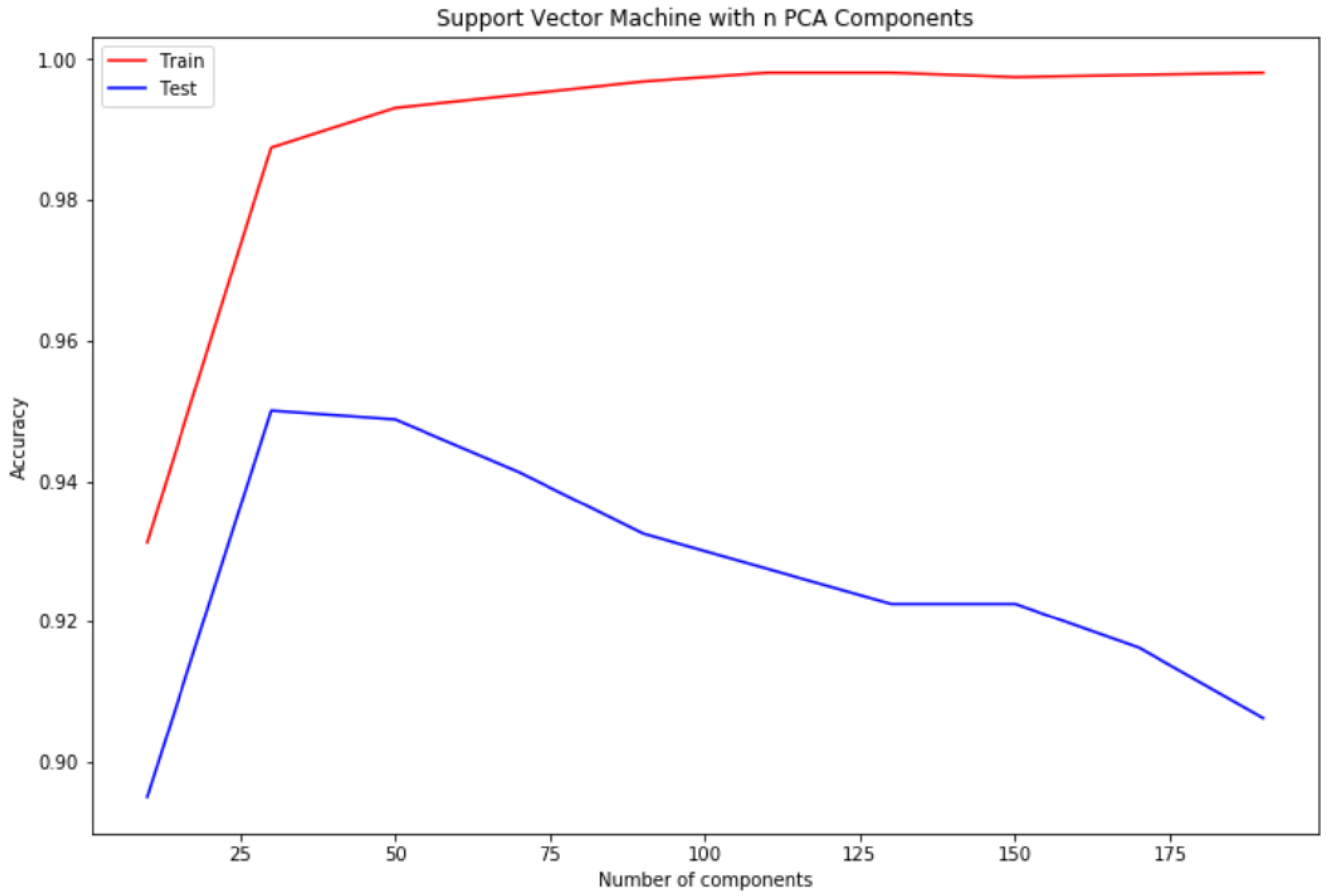


Figure 2-1. Trade-off of model complexity against training and test accuracy

Complexity vs. Accuracy



Supervised Learning Algorithms Overview

- **For each supervised learning method we'll explore:**
  - *How the method works conceptually at a high level.*
  - *What kind of feature preprocessing is typically needed.*
  - *Key parameters that control model complexity, to avoid under-and over-fitting.*
  - *Positives and negatives of the learning method.*
- This week:
  - *K-nearest neighbors*
  - *Linear model fit using least-squares*

K-nearest Neighbors

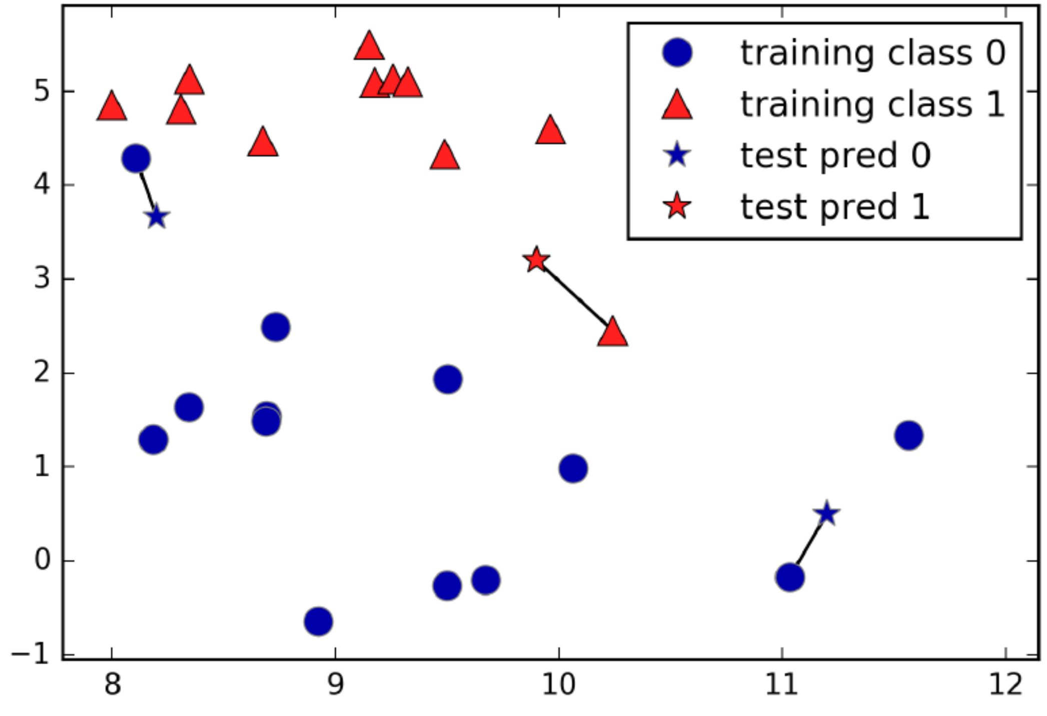
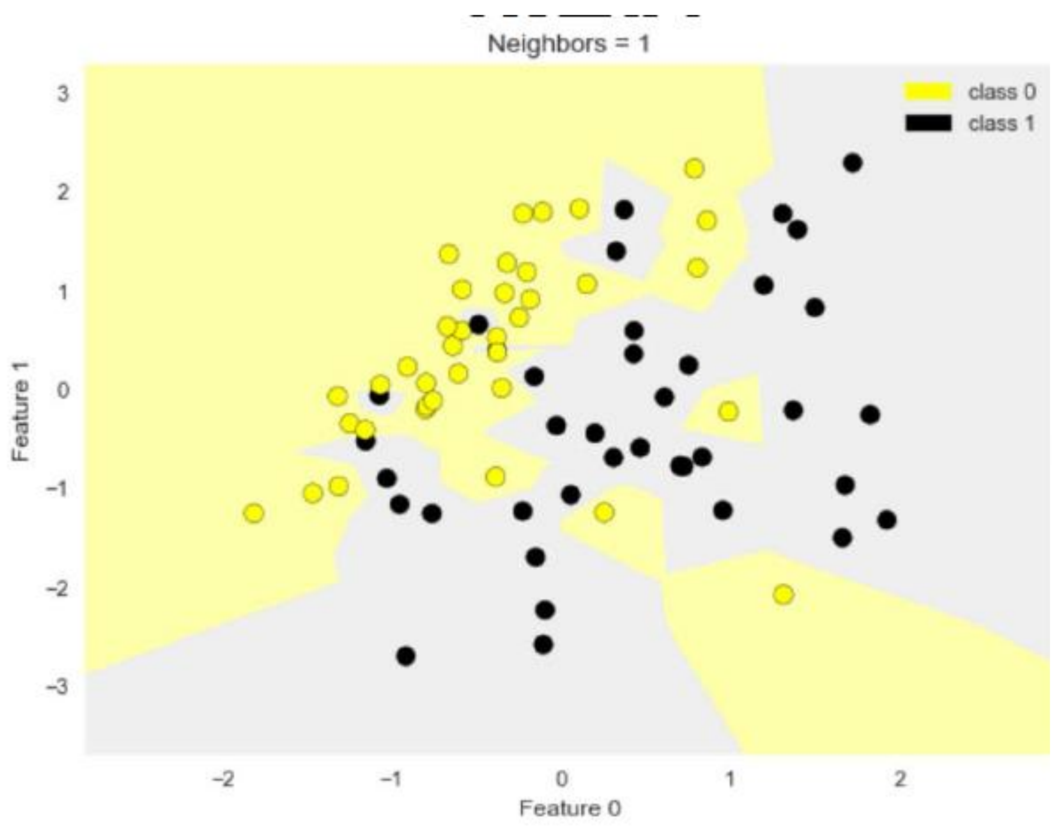
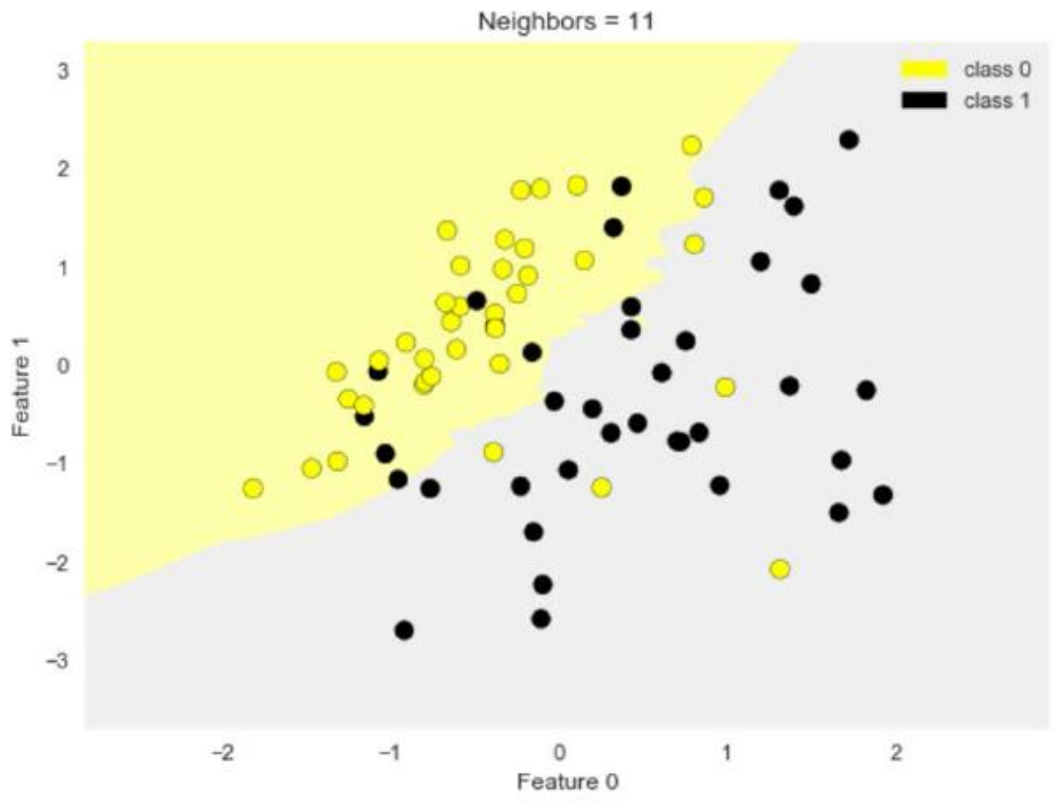


Figure 2-4. Predictions made by the one-nearest-neighbor model on the forge dataset

- To predict, find the closest data points in the training dataset—its "nearest neighbors."
- Import parameters:
  - Model complexity
    - n_neighbors
  - Model fitting
    - Metric: distance between data points
      - Default: Euclidean

Less Neighbors, More Complex





Pros and Cons

Pros

- Easy to understand
- Easy to interpret
- Fast training

Cons

- With large training dataset, prediction can be very slow
- Does not perform well on many features
- Does not perform well on sparse data (data with majority 0s)

Applications of KNN in Real World

- Useful in comparing similarities.
  - Concept search
  - Recommender systems

- Use as a benchmark for other more advanced algorithms
- Find out outliers

Linear Models

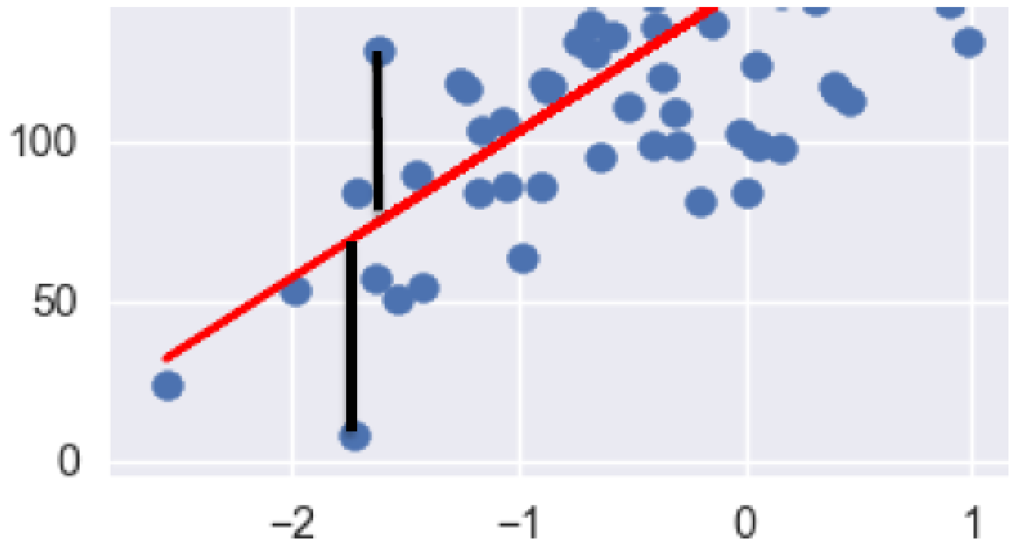
- $y = b + wX$
- $y\ is\ the\ label, or\ target\ to\ be\ predicted, or\ dependent\ variable$
- $b\ is\ the\ intercept$
- $w\ is\ a\ weight\ vector, or\ coefficient\ vector$
- $X\ is\ the\ feature\ matrix, or\ predictors\ matrix, or\ independent$
- $X = (x_0, x_1, x_2, x_3, \dots x_{n-1}, x_n), w = (w_0, w_1, w_2, w_3, \dots w_{n-1}, w_n)$
- $\hat{y} = \hat{b} + \hat{w}_0 x_0 + \hat{w}_1 x_1 + \dots + \hat{w}_{n-1} x_{n-1} + \hat{w}_n x_n$
- $\hat{b}, \hat{w}$ are the parameters to be estimated

Linear Models

- Regressions:
  - Linear regression (ordinary least squares)
  - Ridge regression
  - Lasso regression
  - ElasticNet regression
  - SGD regression (large numbers of samples and features)
- Classifications:
  - LinearSVC
  - Logistic regression
  - ElasticNet classifier
  - SGD classifier

Linear Regression (Ordinary Least Squares)

- Finds *w* and *b* that minimizes the sum of squared differences(RSS) over the training data between predicted target and actual target values.
- a.k.a. mean squared error of the linear model
- No parameters to control model complexity



$$RSS(\mathbf{w}, b) = \sum_{\{i=1\}}^{N} (\mathbf{y}_i - (\mathbf{w} \cdot \mathbf{x}_i + b))^2$$

Loss function

Linear Regression (Ordinary Least Squares)

- **Parameters are estimated from training data.**
- **There are many different ways to estimate *w* and *b*:**
  - *Different methods correspond to different "fit" criteria and goals and ways to control model complexity.*
- **The learning algorithm finds the parameters that optimize an <u>objective function</u>, typically to minimize some kind of <u>loss function</u> of the predicted target values *vs.* actual target values.**

Fit linear regression model

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

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

Check the R square of the regression model on training data

```
clf.score(X_train,y_train)
```

```
0.7729718726571157 5
```

Check the R square of the regression model on test data

```
clf.score(X_test,y_test)
```

```
0.5892011519186440 8
```

$$y = b + wX$$

$$y = 38.14 - 0.1184X_0 + 0.0448X_1 + 0.0059X_2 + \cdots - 0.4936X_{12}$$

Display the intercept and coefficients

```
print('The intercept is {0} \nand coefficient vector is \n{1}'.format(clf.intercept_,clf.coef_))
```

```
The intercept is [ 38.13869271]
and coefficient vector is
[[ -1.18410318e-01   4.47550643e-02   5.85674689e-03   2.34230117e+00
   -1.61634024e+01   3.70135143e+00  -3.04553661e-03  -1.38664542e+00
    2.43784171e-01  -1.09856157e-02  -1.04699133e+00   8.22014729e-03
   -4.93642452e-01]]
```

Ridge Regression

Ridge regression learns *w*, *b* using the same least-squares criterion but adds a penalty for large variations in *w* parameters

$$RSS_{RIDGE}(\boldsymbol{w}, b) = \sum_{\{i=1\}}^{N} (\boldsymbol{y}_i - (\boldsymbol{w} \cdot \boldsymbol{x}_i + b))^2 + \boxed{\boxed{\alpha} \sum_{\{j=1\}}^{p} w_j^2}$$

Once the parameters are learned, the ridge regression <u>prediction</u> formula is the <u>same</u> as ordinary least-squares.

The addition of a parameter penalty is called <u>regularization</u>. Regularization prevents overfitting by restricting the model, typically to reduce its complexity.

Ridge regression uses <u>L2 regularization</u>: minimize sum of squares of *w* entries

The influence of the regularization term is controlled by $\boxed{\text{the } \alpha \text{ parameter.}}$

Higher alpha means more regularization and simpler models.

Lasso Regression

Lasso regression is another regularized linear regression that uses an L1 regularization penalty for training (instead of ridge's L2 penalty)

L1 penalty: Minimize the sum of the absolute values of the coefficients

$$RSS_{LASSO}(\mathbf{w}, b) = \sum_{\{i=1\}}^{N} (\mathbf{y}_i - (\mathbf{w} \cdot \mathbf{x}_i + b))^2 \boxed{+ \alpha \sum_{\{j=1\}}^{p} |w_j|}$$

This has the effect of setting parameter weights in *w* to zero for the least influential variables. This is called a sparse solution: a kind of feature selection

The parameter α controls amount of L1 regularization (default = 1.0).

The prediction formula is the same as ordinary least-squares.

When to use ridge vs lasso regression:

- *Many small/medium sized effects: use ridge.*
- *Only a few variables with medium/large effect: use lasso.*

ElasticNet Regression

- Linear regression with combined L1 and L2 priors as regularizer.

$$\min_{w} \frac{1}{2n_{samples}}||Xw - y||_2^2 + \alpha\rho||w||_1 + \frac{\alpha(1-\rho)}{2}||w||_2^2$$

- This combination allows for learning a sparse model where few of the weights are non-zero like Lasso, while still maintaining the regularization properties of Ridge.
- Parameters:
  - Alpha: High α means more regularization.
  - l1_ratio: with 0 <= l1_ratio <= 1. For l1_ratio = 0 the penalty is an L2 penalty. For l1_ratio = 1 it is an L1 penalty. For 0 < l1_ratio < 1, the penalty is a combination of L1 and L2.

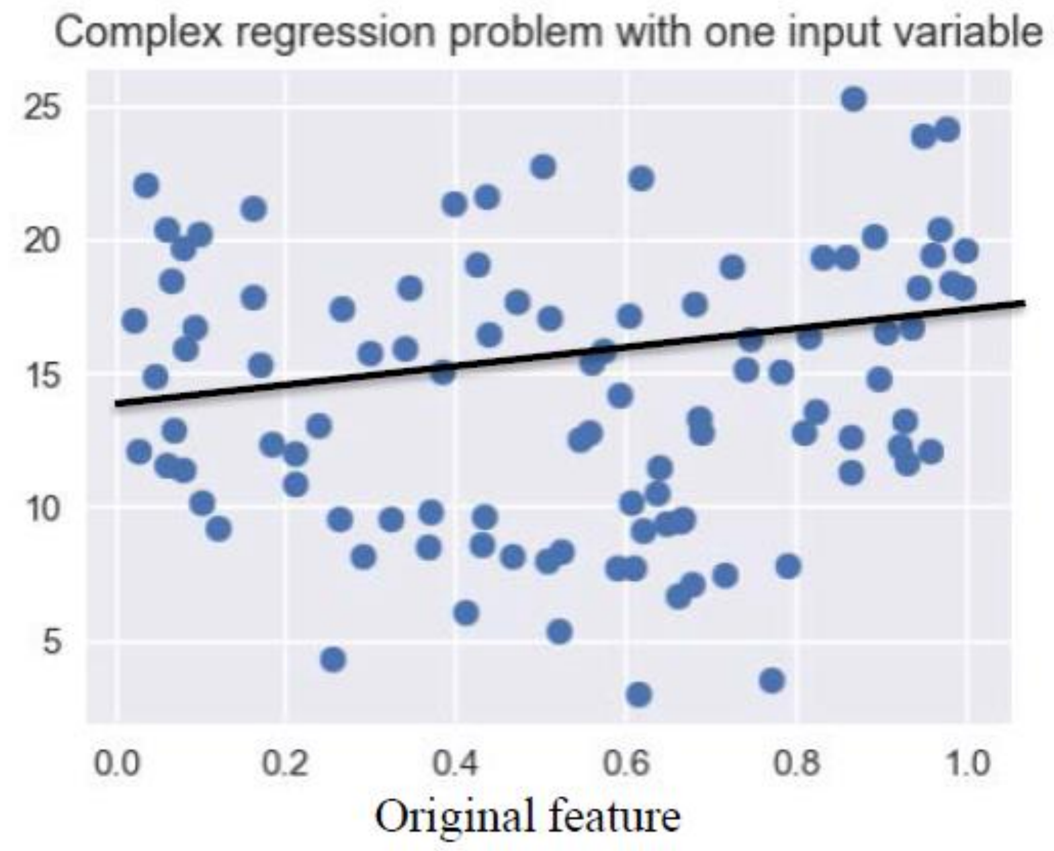
Polynomial Features with Linear Regression

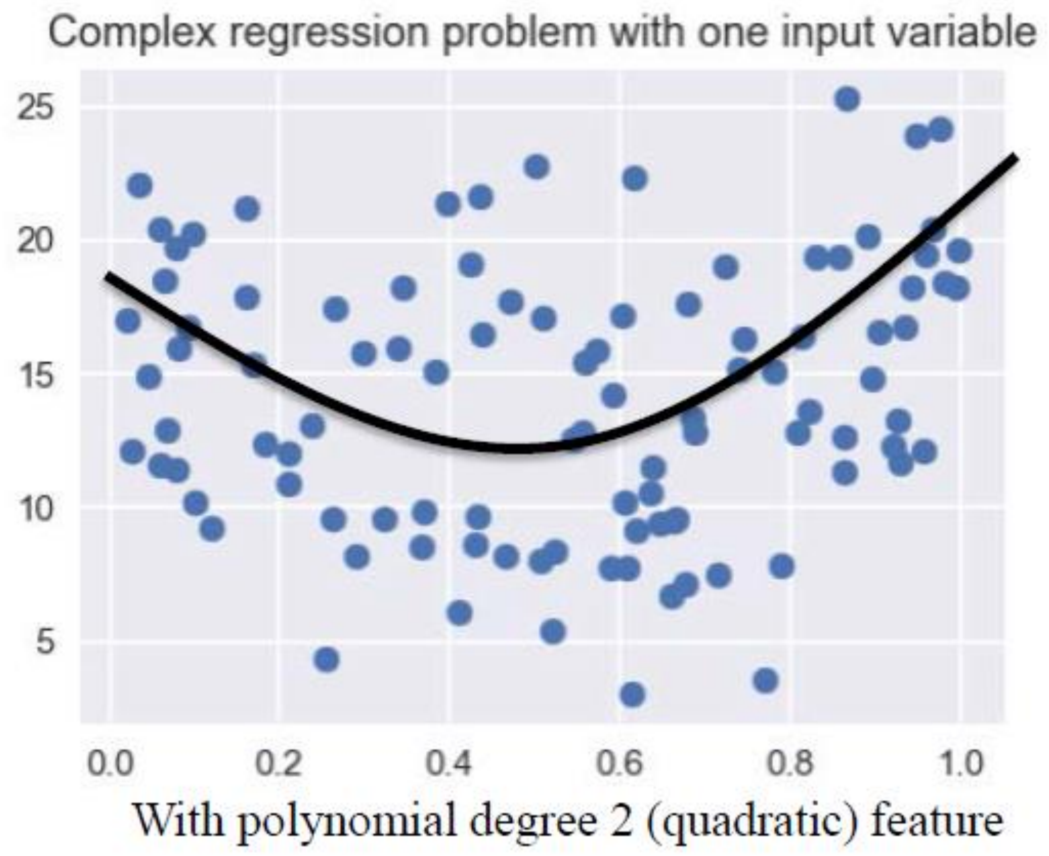
$$\boldsymbol{x} = (x_0, x_1) \Rightarrow \boldsymbol{x}' = (x_0, x_1, x_0^2, x_0 x_1, x_1^2)$$

$$\hat{y} = \widehat{w}_0 x_0 + \widehat{w}_1 x_1 + \widehat{w}_{00} x_0^2 + \widehat{w}_{01} x_0 x_1 + \widehat{w}_{11} x_1^2 + b$$

- **Generate new features consisting of all polynomial combinations of the original two features (x_0, x_1).**
- **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*.**

Polynomial Features with Linear Regression





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).*

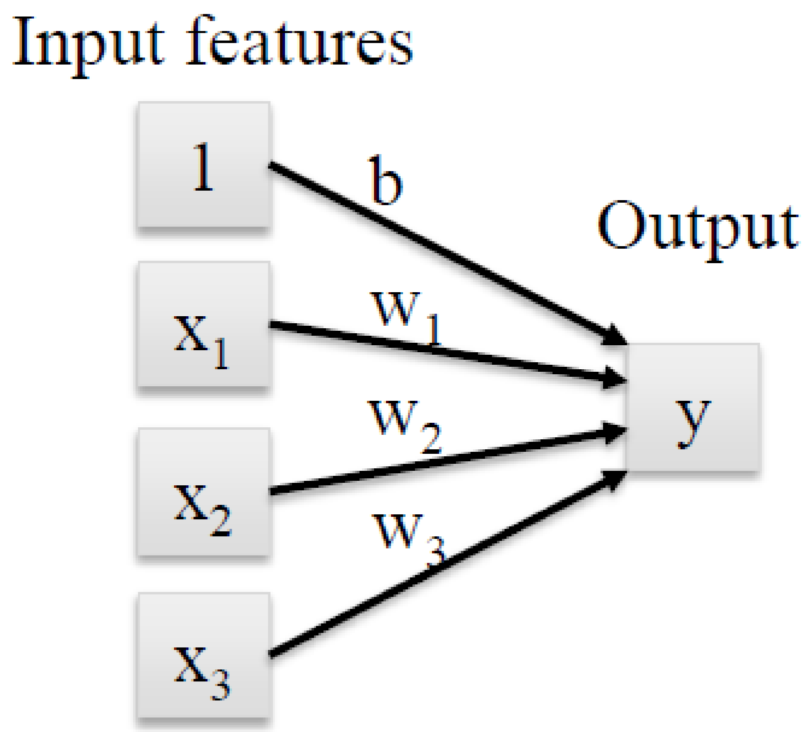
More generally, we can apply other non-linear transformations to create new features

- *(Technically, these are called non-linear basis functions)*

Beware of polynomial feature expansion with high as this can lead to complex models that overfit

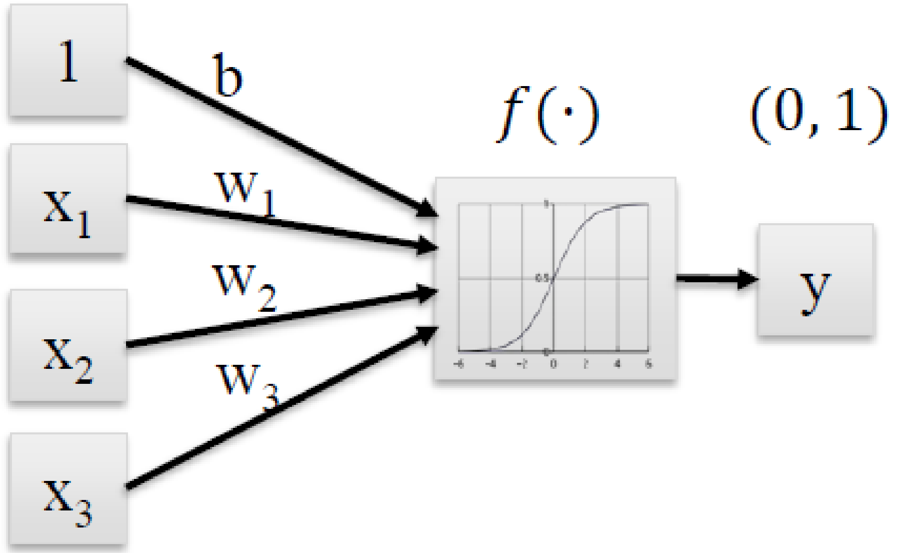
- *Thus, polynomial feature expansion is often combined with a regularized learning method like ridge regression.*

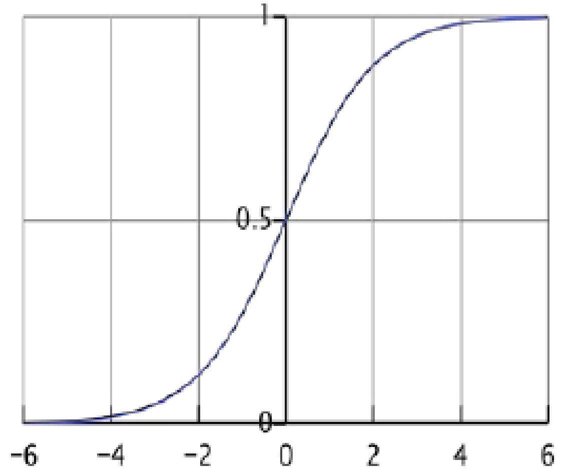
Linear Regression



$$\hat{y} = \hat{b} + \hat{w}_1 \cdot x_1 + \cdots \hat{w}_n \cdot x_n$$

Logistic Regression





The logistic function transforms real-valued input to an output number y between 0 and 1, interpreted as the probability the input object belongs to the positive class, given its input features $(x_0, x_1, \ldots, x_n)$

$$\hat{y} = \text{logistic}(\hat{b} + \hat{w}_1 \cdot x_1 + \cdots \hat{w}_n \cdot x_n)$$

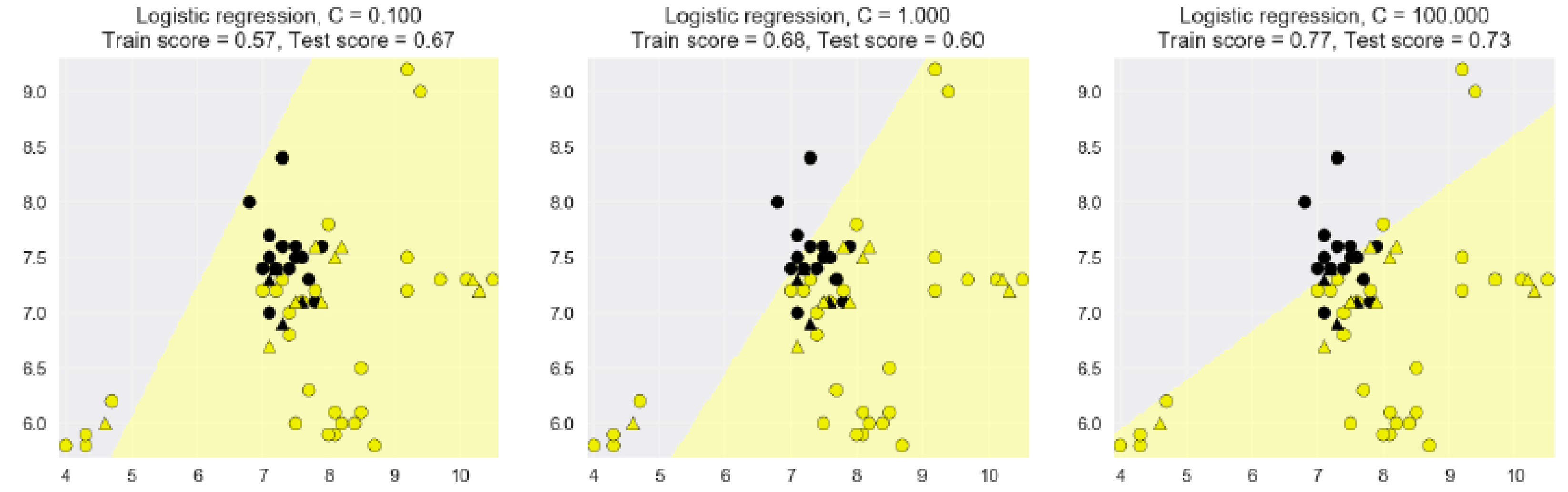
$$= \frac{1}{1 + \exp\left[-(\hat{b} + \hat{w}_1 \cdot x_1 + \cdots \hat{w}_n \cdot x_n)\right]}$$

Regularization of Logistic Regression

L2 regularization is 'on' by default (like ridge regression)

Parameter C controls amount of regularization (default 1.0)

As with regularized linear regression, it can be important to normalize all features so that they are on the same scale.



Linear Support Vector Machines Classifier

The strength of regularization is determined by C

Larger values of C: less regularization

- *Fit the training data as well as possible*
- *Each individual data point is important to classify correctly*

Smaller values of C: more regularization

- *More tolerant of errors on individual data points*

Important Parameters

Model complexity

- **alpha: weight given to the L1 or L2 regularization term in regression models**
  - *default = 1.0*
- **C: regularization weight for LinearSVC and LogisticRegression classification models**
  - *default = 1.0*

The Need for Feature Normalization

- **Important for some machine learning methods that all features are on the same scale (e.g. faster convergence in learning, more uniform or 'fair' influence for all weights)**
  - *e.g. regularized regression, k-NN, support vector machines, neural networks*

Linear Models

Pros:

- Simple and easy to train.
- Fast prediction.
- Scales well to very large datasets.
- Works well with sparse data.
- Reasons for prediction are relatively easy to interpret.

Cons:

- For lower-dimensional data, other models may have superior generalization performance.
- For classification, data may not be linearly separable (more on this in SVMs with non-linear kernels)

Questions?