Regression in Use

How to Apply Linear Regression, Logistic regression and KNN in python.

Ridge Regression

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

$$RSS_{RIDGE}(w, b) = \sum_{\{i=1\}}^{N} (y_i - (w \cdot x_i + b))^2 + \alpha \sum_{\{j=1\}}^{p} w_j^2$$

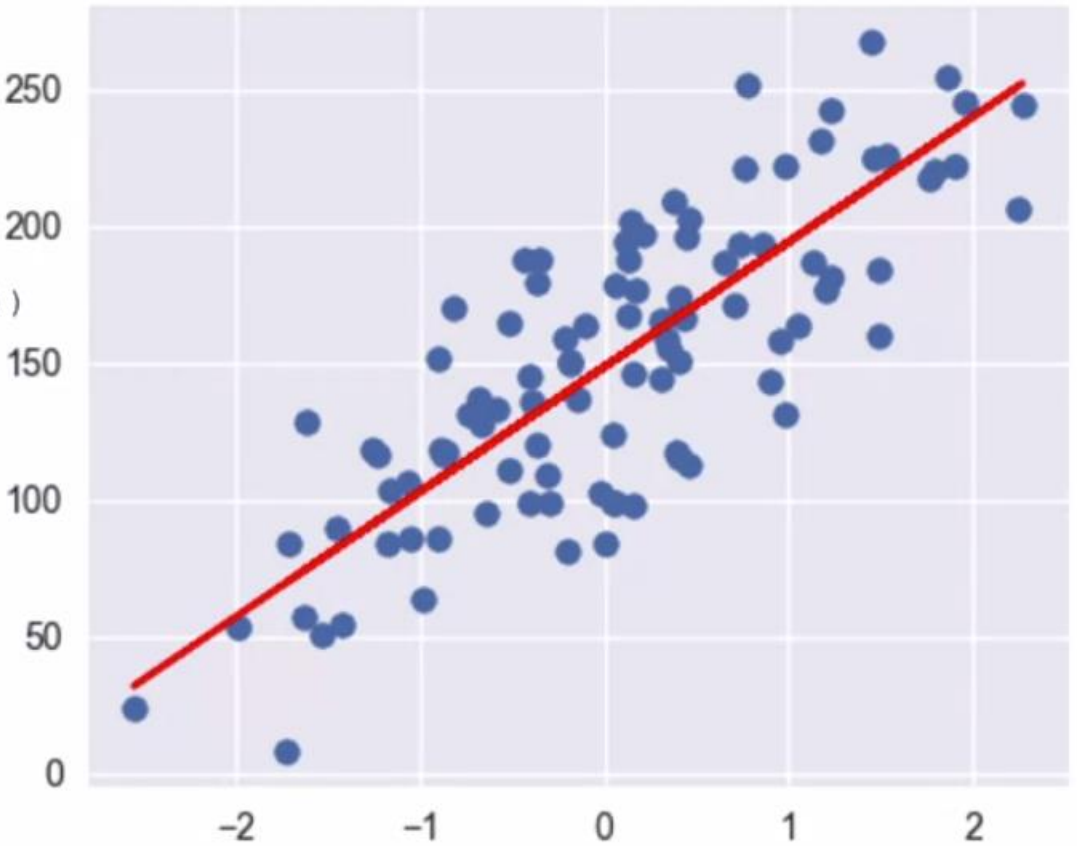
- Once the parameters are learned, the ridge regression prediction formula is the same as ordinary least-squares.
- The addition of a parameter penalty is called regularization. Regularization prevents overfitting by restricting the model, typically to reduce its complexity.
- Ridge regression uses L2 regularization: minimize sum of squares of *w* entries
- The influence of the regularization term is controlled by the α parameter.
- Higher alpha means more regularization and simpler models.

Ridge Regression and Normalization.

- Ridge regression is the linear regression with L2 regularization.
- Normalization works fine with ridge regression since it enables to apply fair penalty to each of the coefficients

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_

$$\hat{y} = \boldsymbol{w_0} x_0 + b$$

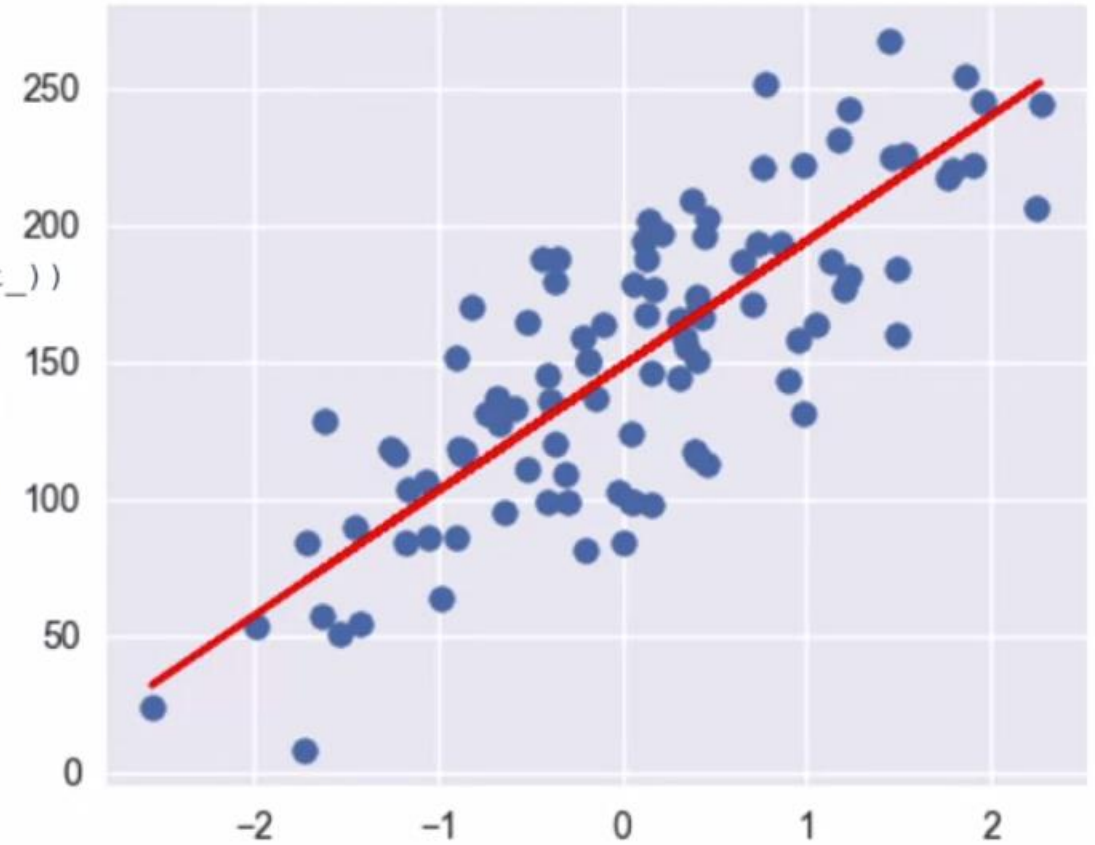
Least-Squares Linear Regression in Scikit-Learn

```
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_))
```

Underscore denotes a quantity derived from training data, as opposed to a user setting.

`linreg.coef_` `linreg.intercept_`

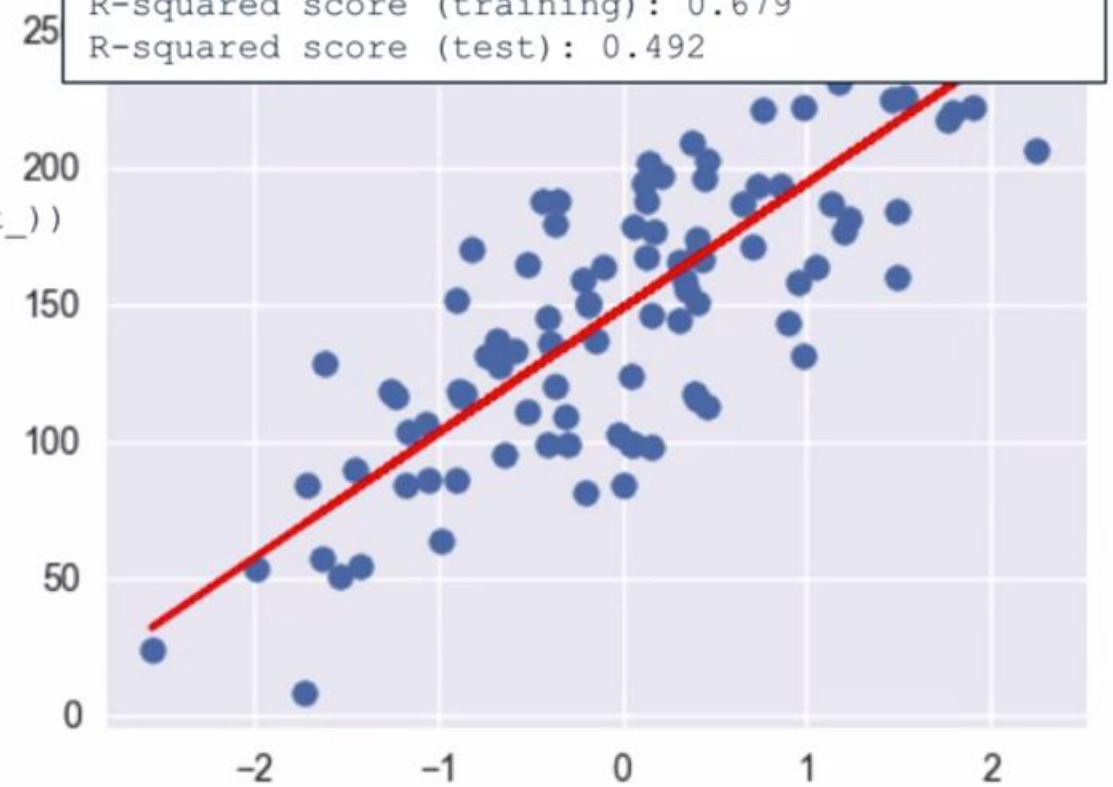
$$\hat{y} = \boldsymbol{w_0} x_0 + b$$



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_))
```

```
linear model coeff (w): [ 45.70870465]
linear model intercept (b): 148.44575345658873
R-squared score (training): 0.679
R-squared score (test): 0.492
```



linreg.coef_ linreg.intercept_

$$\hat{y} = \boldsymbol{w_0} x_0 + b$$

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, …*
- Can also depend on the data. More on feature engineering later in the course. For now, we do MinMax scaling of the features:
  - *For each feature x_i : compute the min value x_i^{MIN} and the max value x_i^{MAX} achieved across all instances in the training set.*
  - *For each feature: transform a given feature x_i value to a scaled version x_i' using the formula*

$$x_i' = (x_i - x_i^{MIN})/(x_i^{MAX} - x_i^{MIN})$$

Feature Normalization: The test set must use identical scaling to the training set

- Fit the scaler using the training set, then apply the same scaler to transform the test set.
- Do not scale the training and test sets using different scalers: this could lead to random skew in the data.
- Do not fit the scaler using any part of the test data: referencing the test data can lead to a form of *data leakage*. More on this issue later in the course.

Lasso regression is another form of 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

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

- 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.*

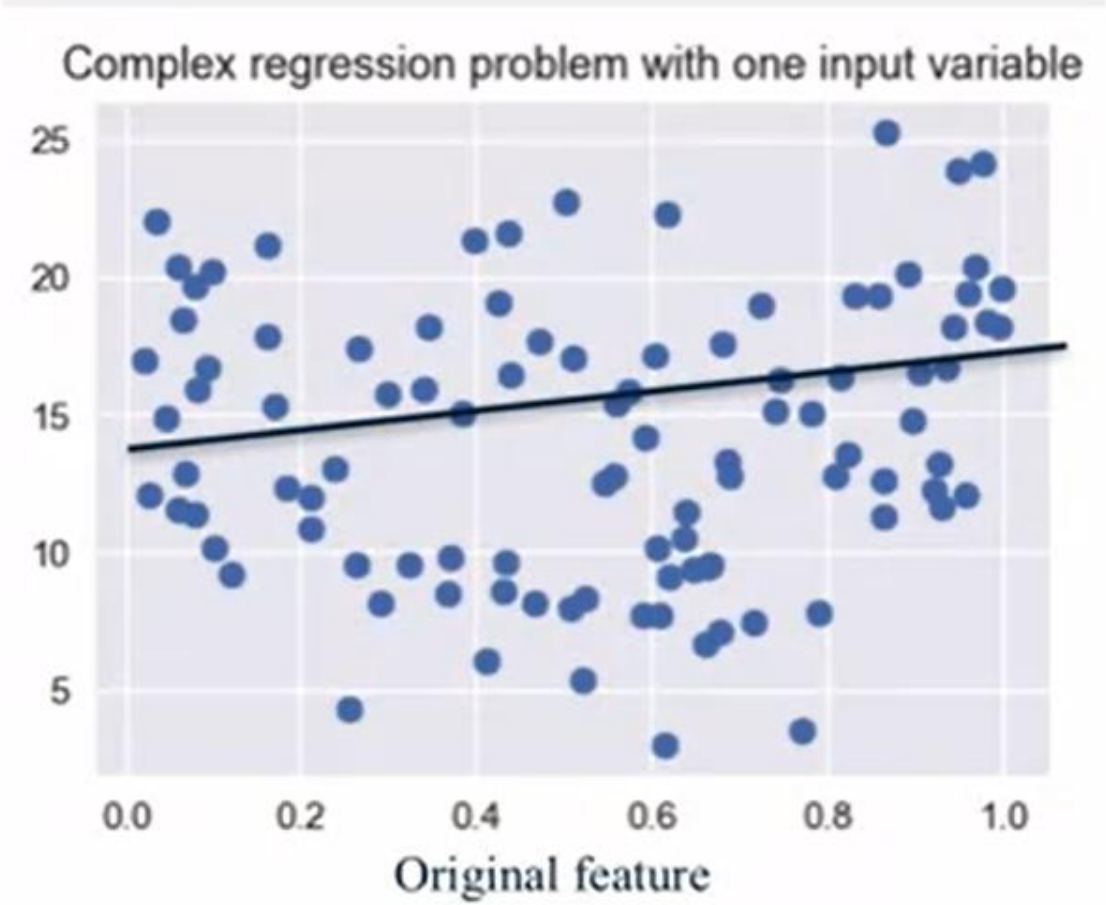
Polynomial Features with Linear Regression

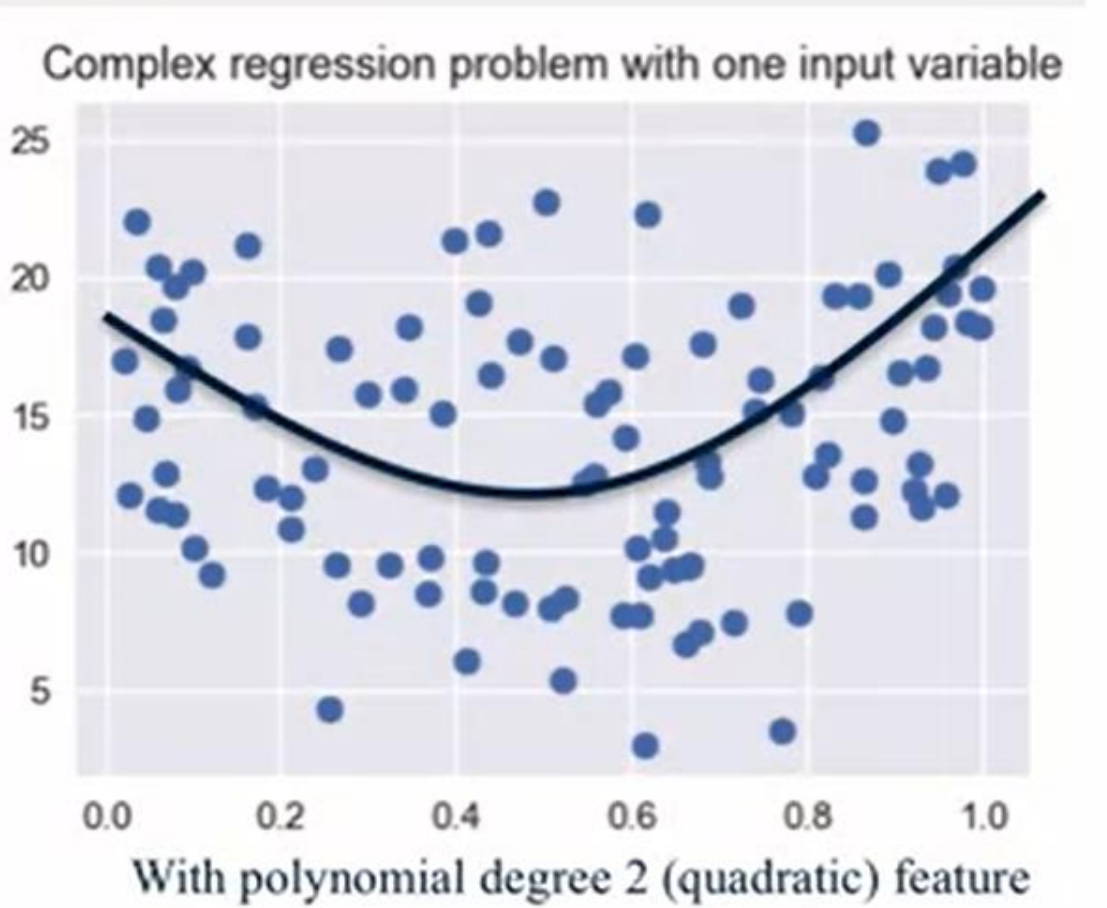
$$\boldsymbol{x} = (x_0, x_1) \Rightarrow \boldsymbol{x}' = (x_0, x_1, x_0^2, x_0 x_1, x_1^2)$$

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

- 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 still a linear model, and can use same least-squares estimation method for *w* and *b*.

Least-Squares Polynomial Regression



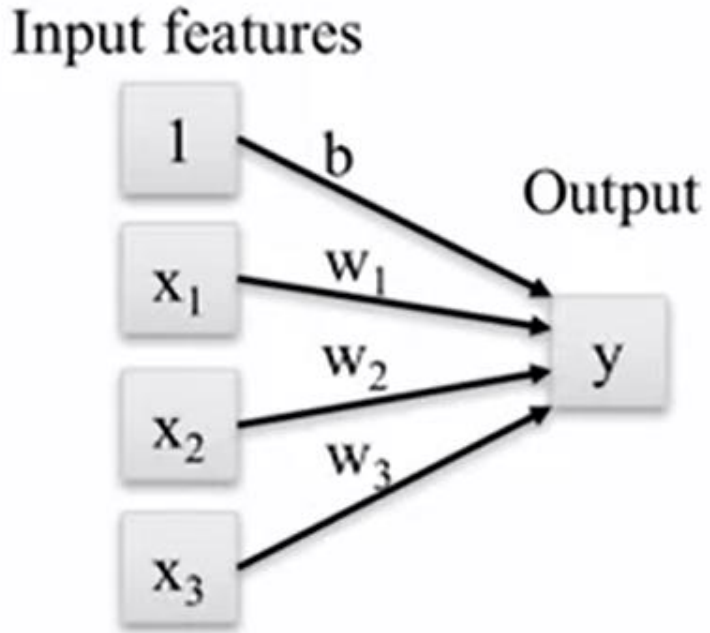


Polynomial Features with Linear Regression

- **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 degree, as this can lead to complex models that overfit**
  - *Thus, polynomial feature expansion is often combined with a regularized learning method like ridge regression.*

Logistic Regression

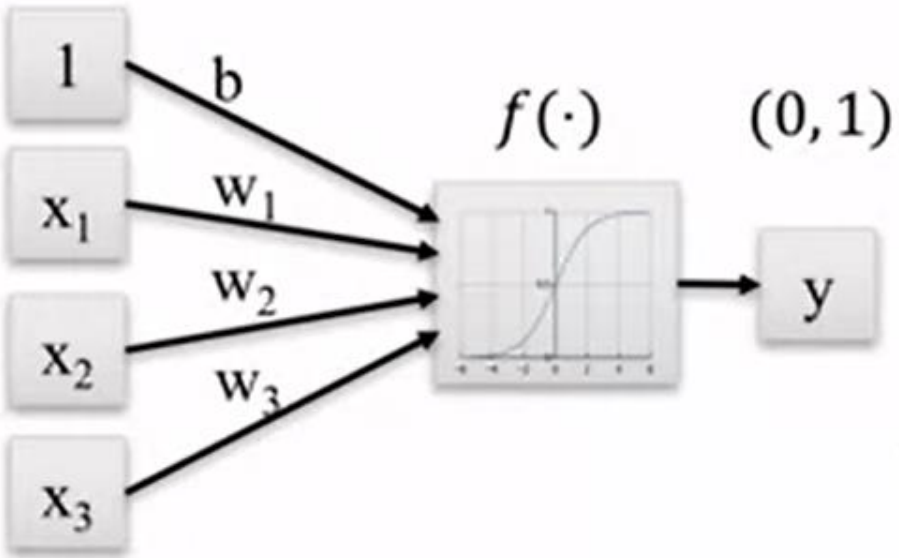
Linear Regression

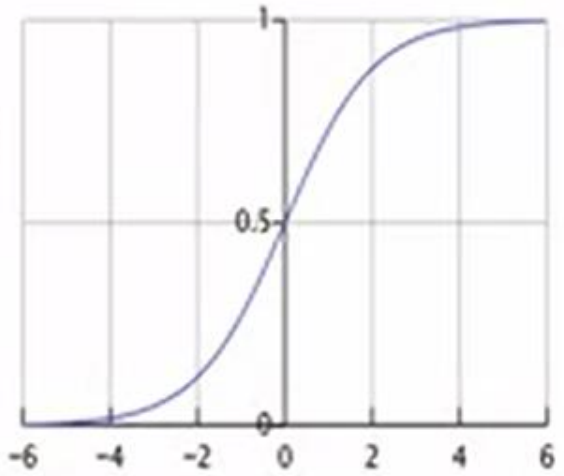


$$\hat{y} = \hat{b} + \hat{w}_1 \cdot x_1 + \cdots \hat{w}_n \cdot x_n$$

Linear models for classification: Logistic Regression

Input features



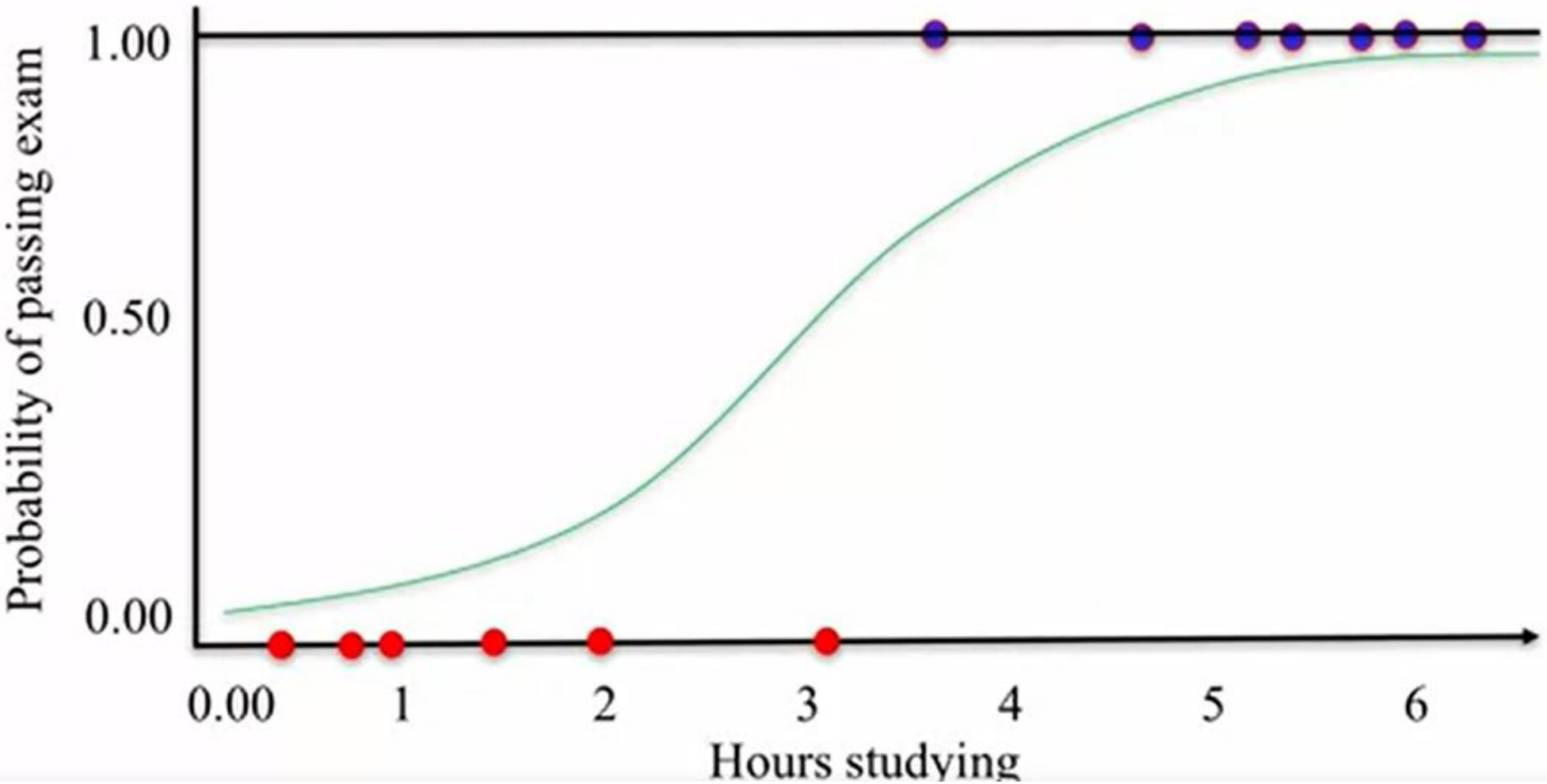


The logistic function transforms real-valued input to an output number y between 0 and 1, interpreted as the probability the input object belongs to the positive class, given its input features $(x_0, x_1, \ldots, x_n)$

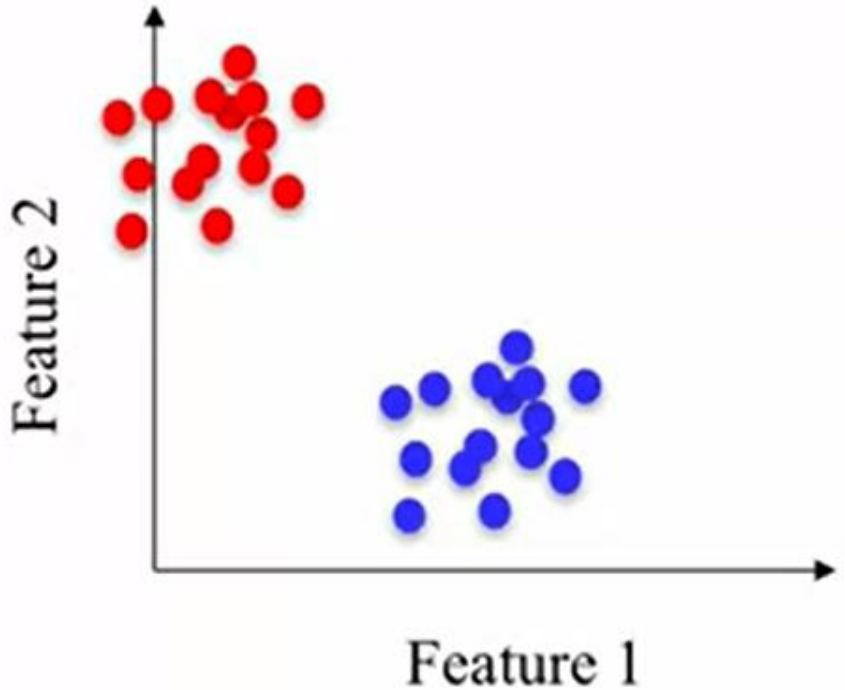
$$\hat{y} = \text{logistic}(\hat{b} + \hat{w}_1 \cdot x_1 + \cdots \hat{w}_n \cdot x_n)$$

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

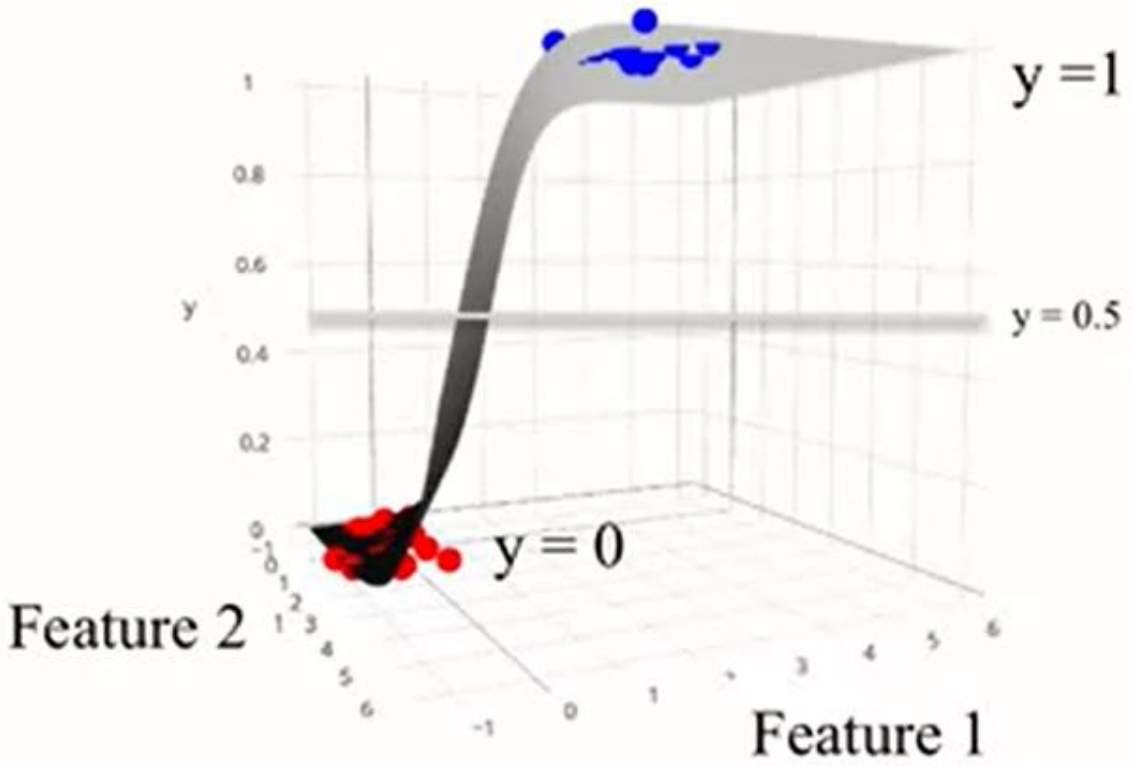
Linear models for classification: Logistic Regression



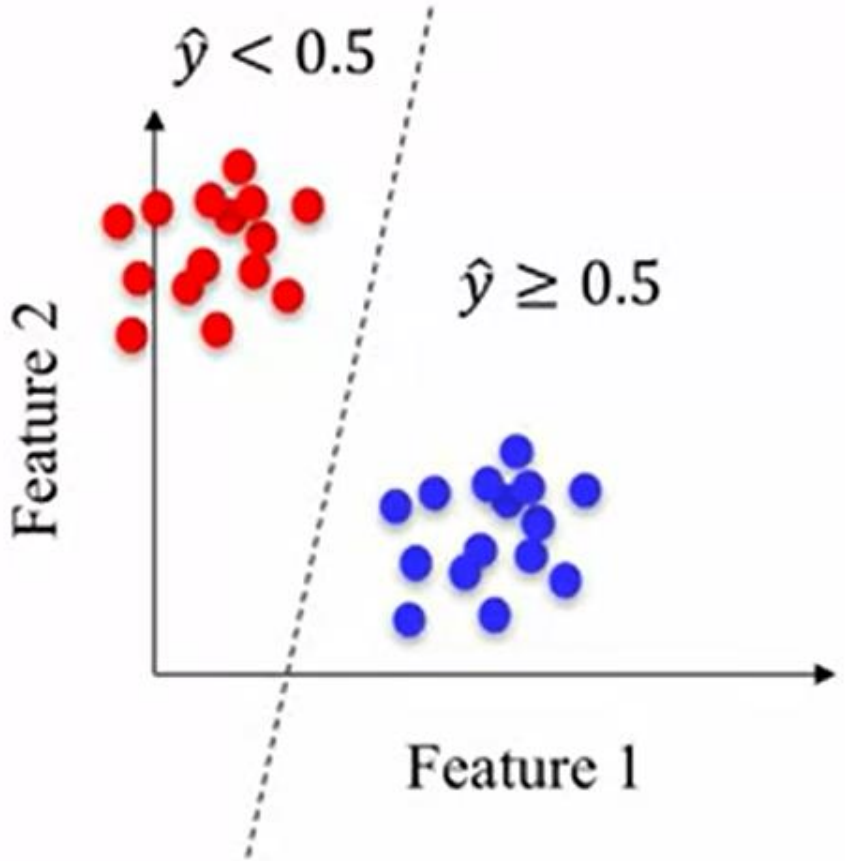
Logistic Regression for binary classification



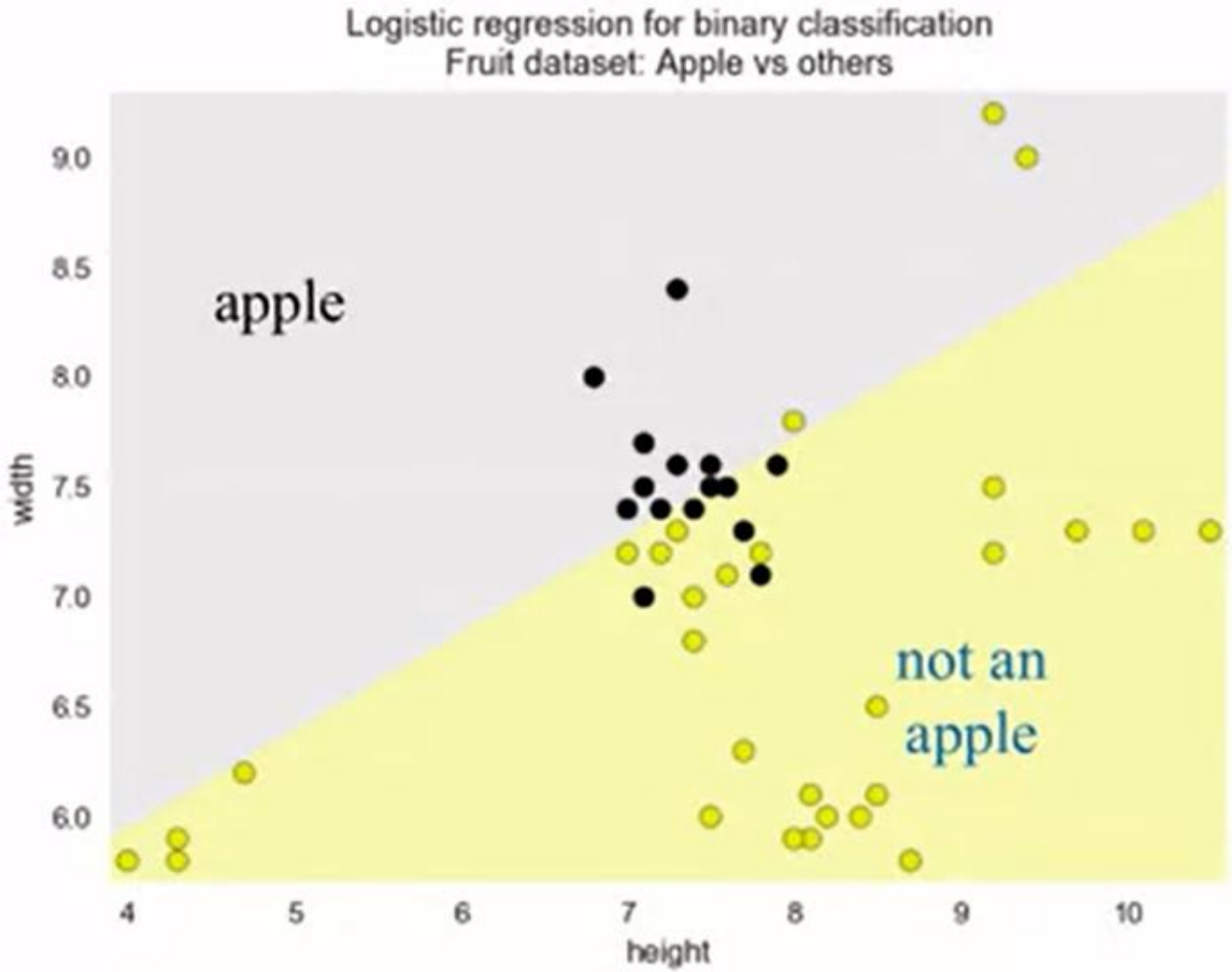
Logistic Regression for binary classification



Logistic Regression for binary classification

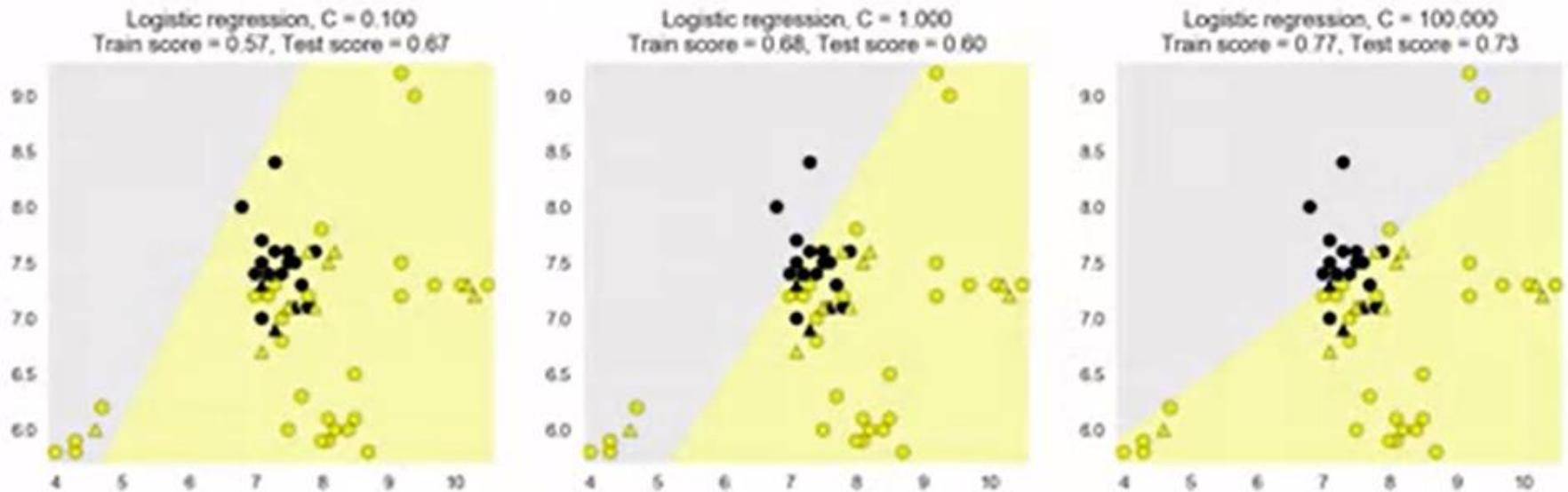


Simple logistic regression problem: two-class, two-feature version of the fruit dataset



Logistic Regression: Regularization

- L2 regularization is 'on' by default (like ridge regression)
- Parameter C controls amount of regularization (default 1.0) The inverse of alpha…
- As with regularized linear regression, it can be important to normalize all features so that they are on the same scale.



k-NN Algorithm

Classification

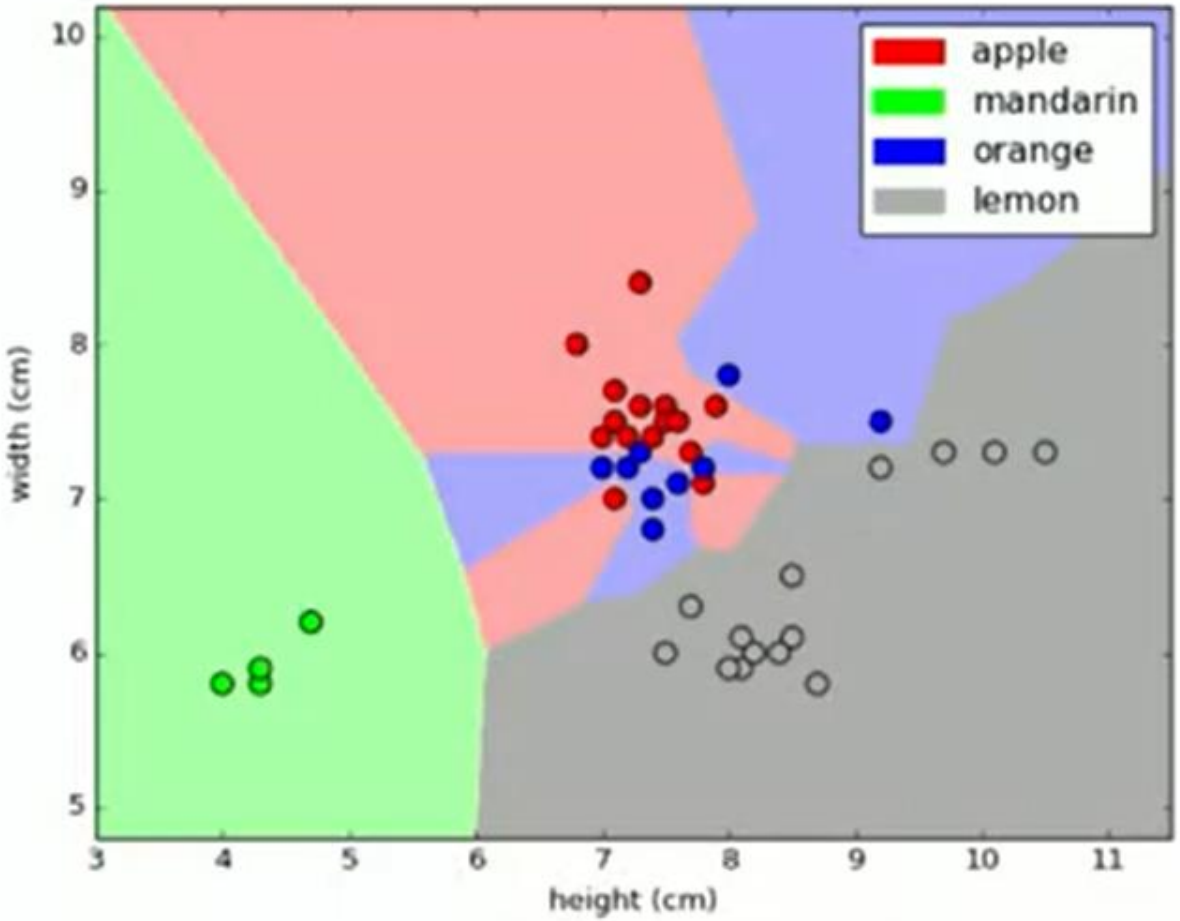
Regression

The k-Nearest Neighbor (k-NN) Classifier Algorithm

Given a training set X_train with labels y_train, and given a new instance x_test to be classified:

1. Find the most similar instances (let's call them X_NN) to x_test that are in X_train.
2. Get the labels y_NN for the instances in X_NN
3. Predict the label for x_test by combining the labels y_NN
   e.g. simple majority vote

A visual explanation of k-NN classifiers



Fruit dataset
Decision boundaries
with k = 1

A nearest neighbor algorithm needs four things specified

1. A distance metric
2. How many 'nearest' neighbors to look at?
3. Optional weighting function on the neighbor points
4. Method for aggregating the classes of neighbor points

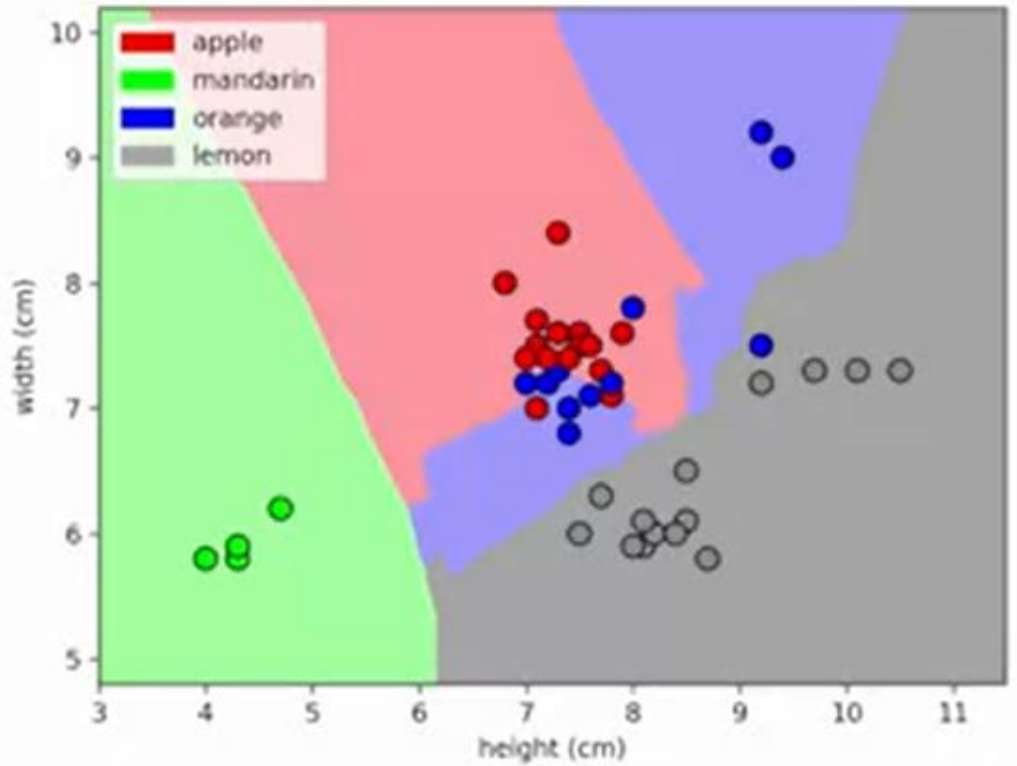
A nearest neighbor algorithm needs four things specified

1. A distance metric
   Typically Euclidean (Minkowski with p = 2)
2. How many 'nearest' neighbors to look at?
   e.g. five
3. Optional weighting function on the neighbor points
   Ignored
4. How to aggregate the classes of neighbor points
   Simple majority vote
   (Class with the most representatives among nearest neighbors)

Plot the decision boundaries of the k-NN classifier

In [10]:
```
from adspy_shared_utilities import plot_fruit_knn

plot_fruit_knn(X_train, y_train, 5, 'uniform')
```

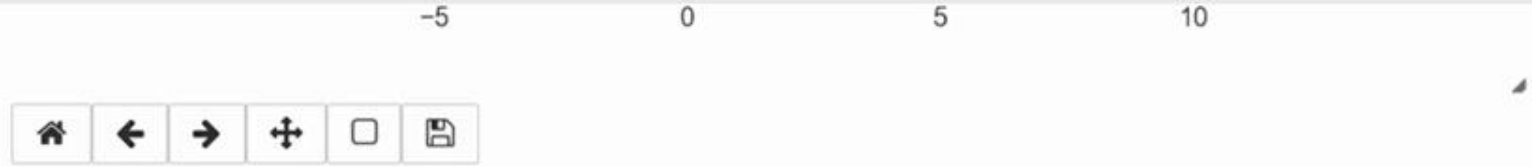
Figure 1



Bias – variance trade-off

- For larger values of K, the areas assigned to different classes are smoother and not as fragmented and more robust to noise in the individual points.
- But possibly with some mistakes, more mistakes in individual points.
- This is an example of what's known as the **bias variance tradeoff**.

- Consider the following example.



K-Nearest Neighbors

Classification

In [*]:
```
from adspy_shared_utilities import plot_two_class_knn

X_train, X_test, y_train, y_test = train_test_split(X_C2, y_C2,
                                                    random_state=0)

plot_two_class_knn(X_train, y_train, 1, 'uniform', X_test, y_test)
plot_two_class_knn(X_train, y_train, 3, 'uniform', X_test, y_test)
plot_two_class_knn(X_train, y_train, 11, 'uniform', X_test, y_test)
```

In [ ]:
```

```

```
plot_two_class_knn(X_train, y_train, 11, 'uniform', X_test, y_test)
```

Figure 5

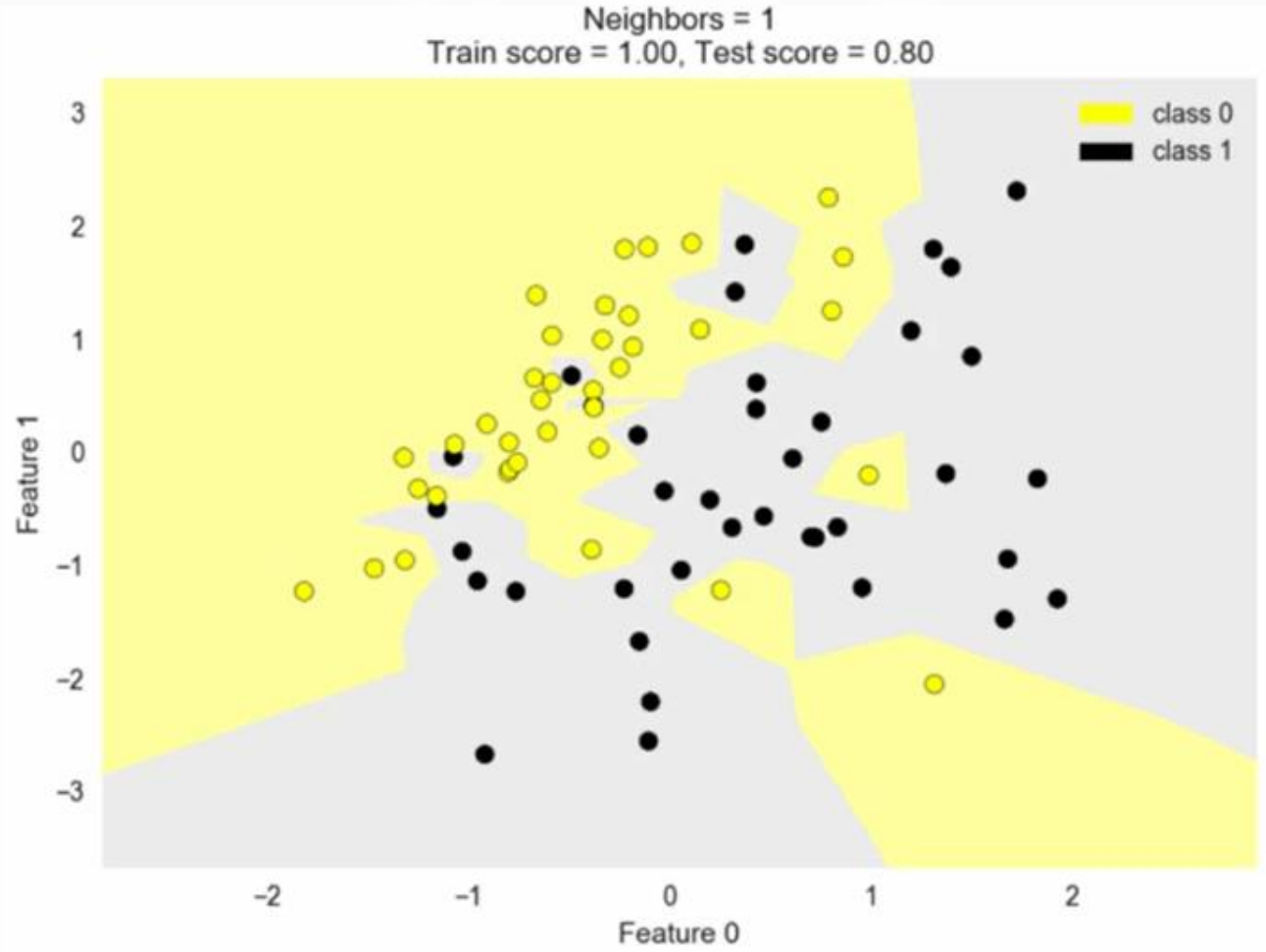


Figure 6

```
plot_two_class_knn(X_train, y_train, 11, 'uniform', X_test, y_test)
```

Figure 5

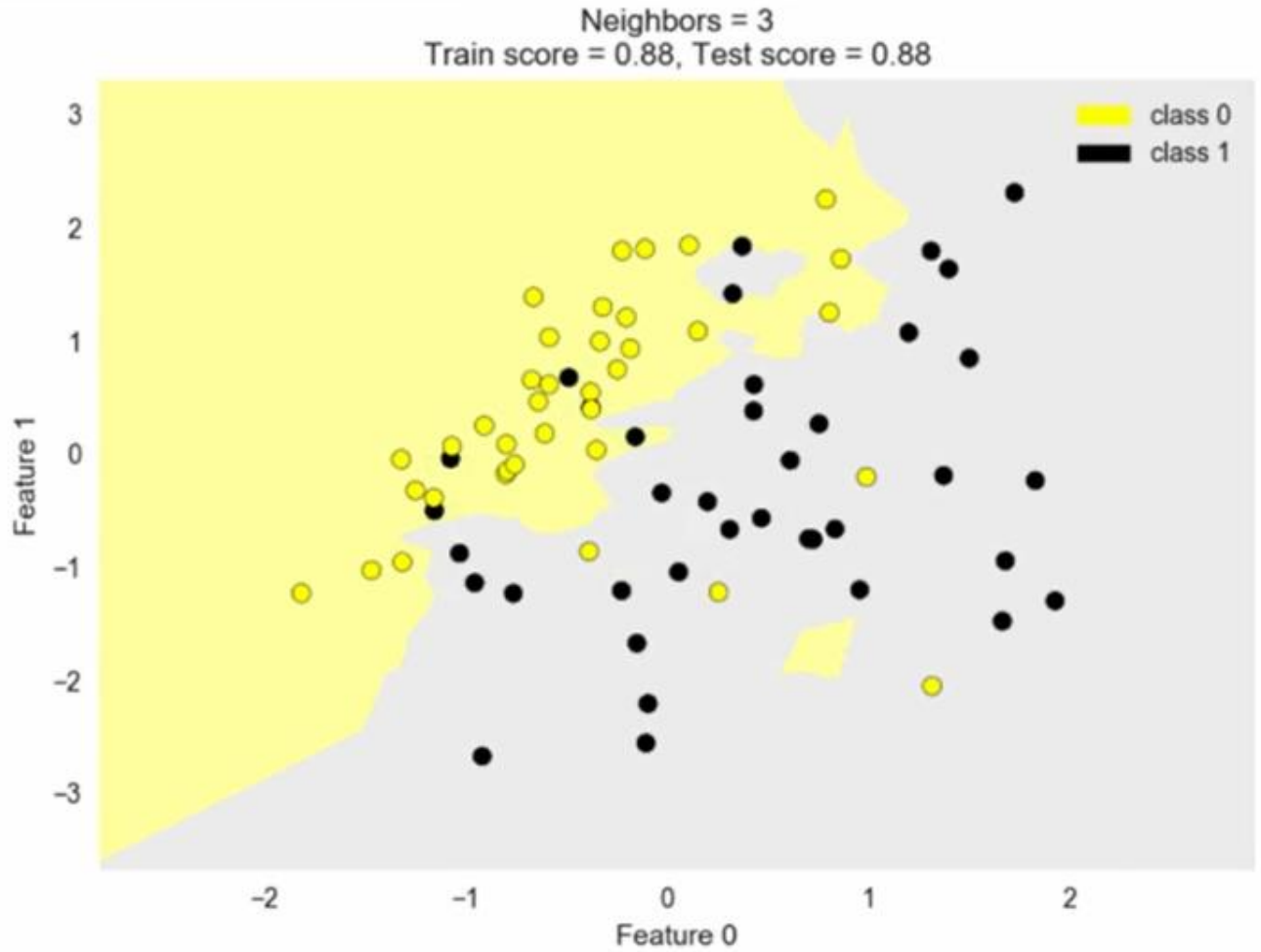


Figure 6

```
plot_two_class_knn(X_train, y_train, 11, 'uniform', X_test, y_test)
```

Figure 5

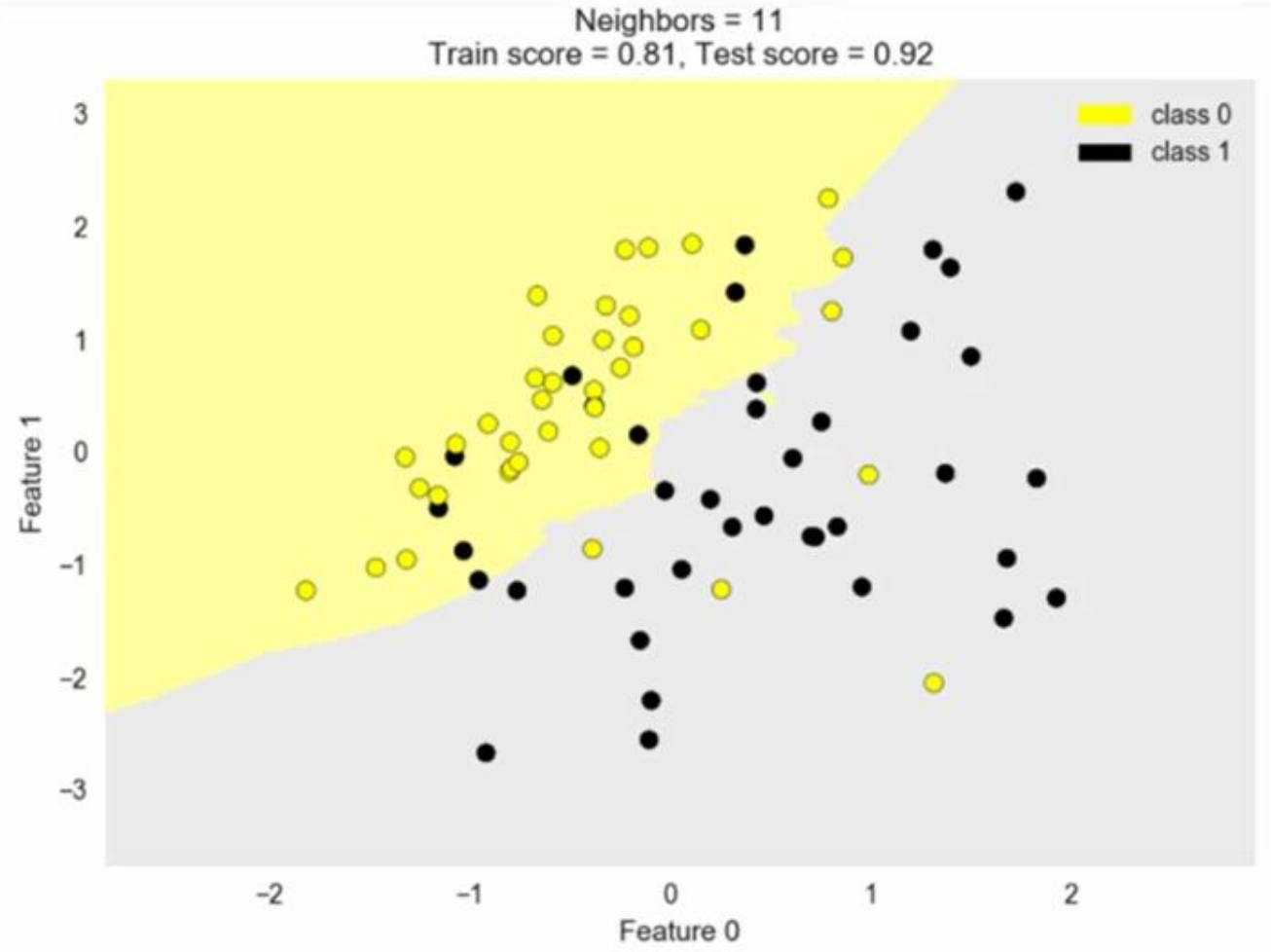
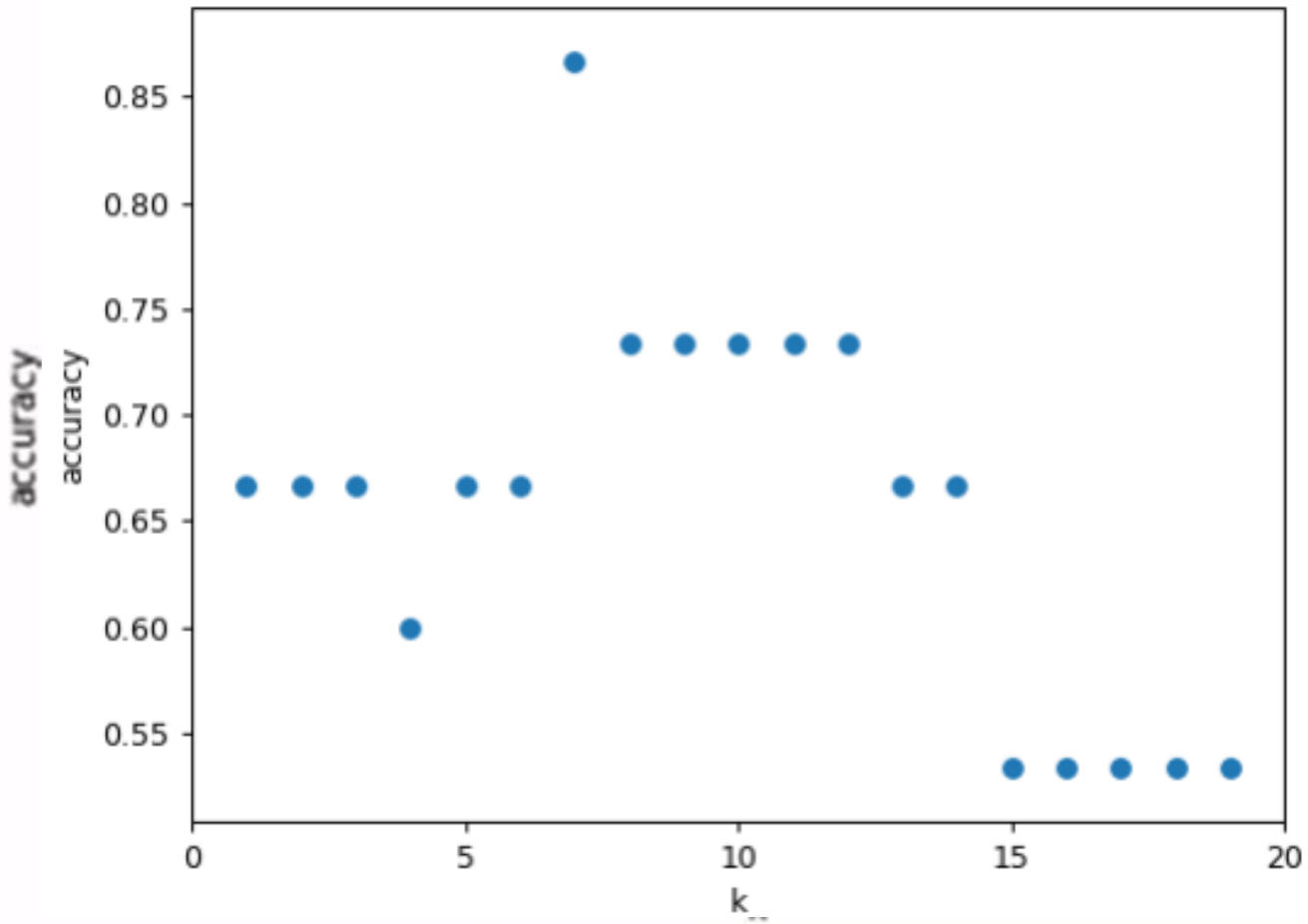


Figure 6

How sensitive is k-NN classifier accuracy to the choice of 'k' parameter?



Choose only colors as the feature set.

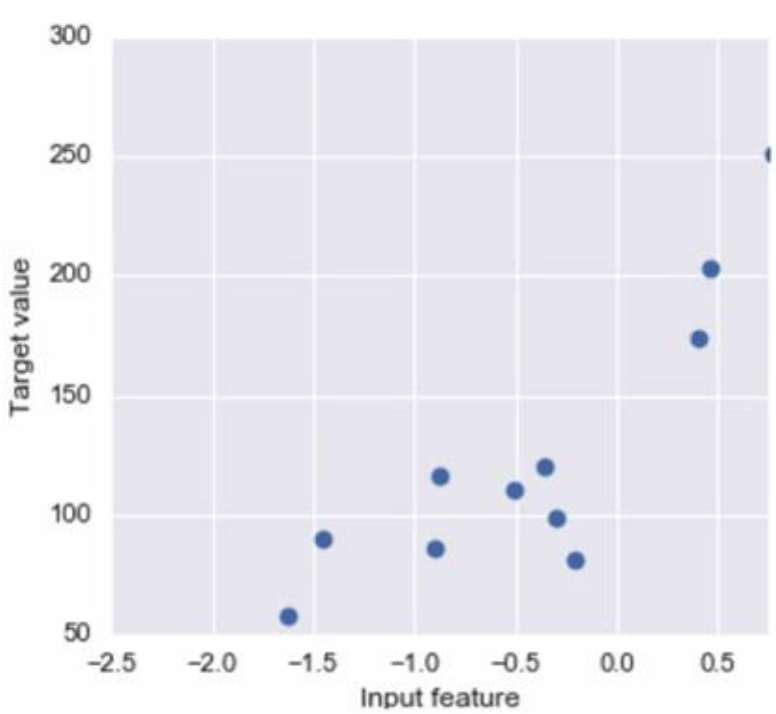
Fruit dataset with 75%/25% train-test split

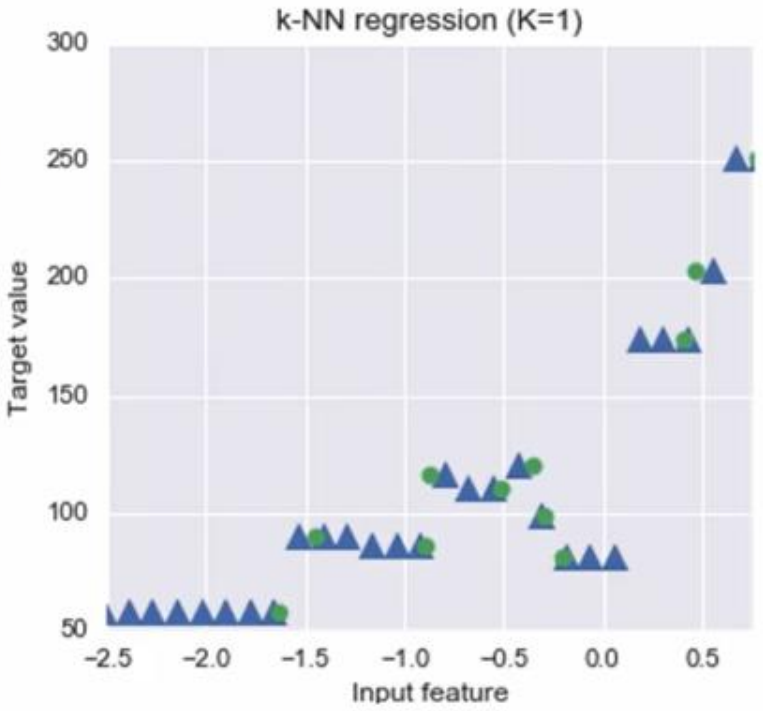
k-NN Algorithm

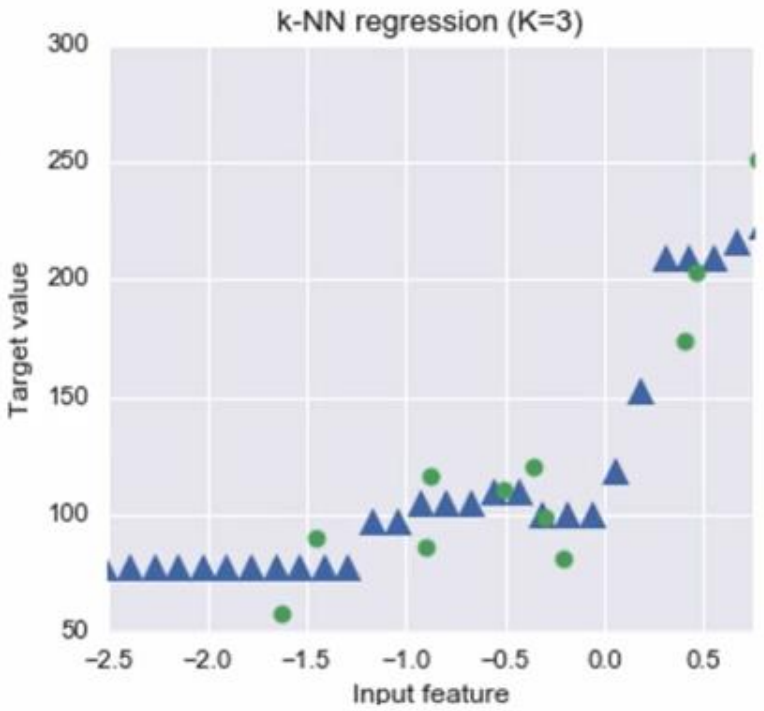
Classification

Regression

k-Nearest neighbors regression

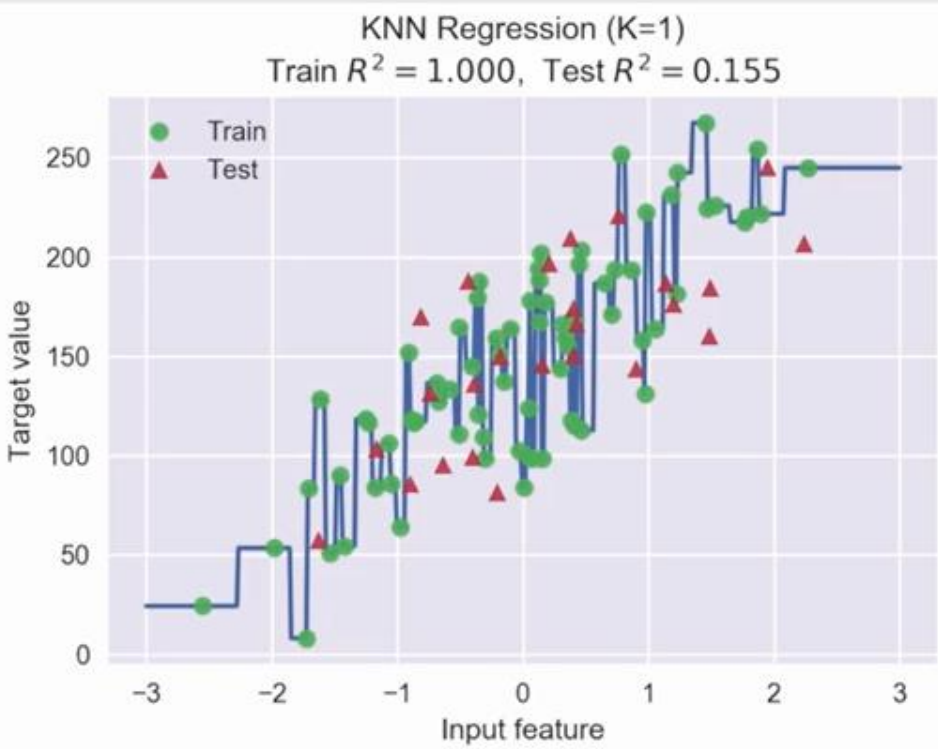


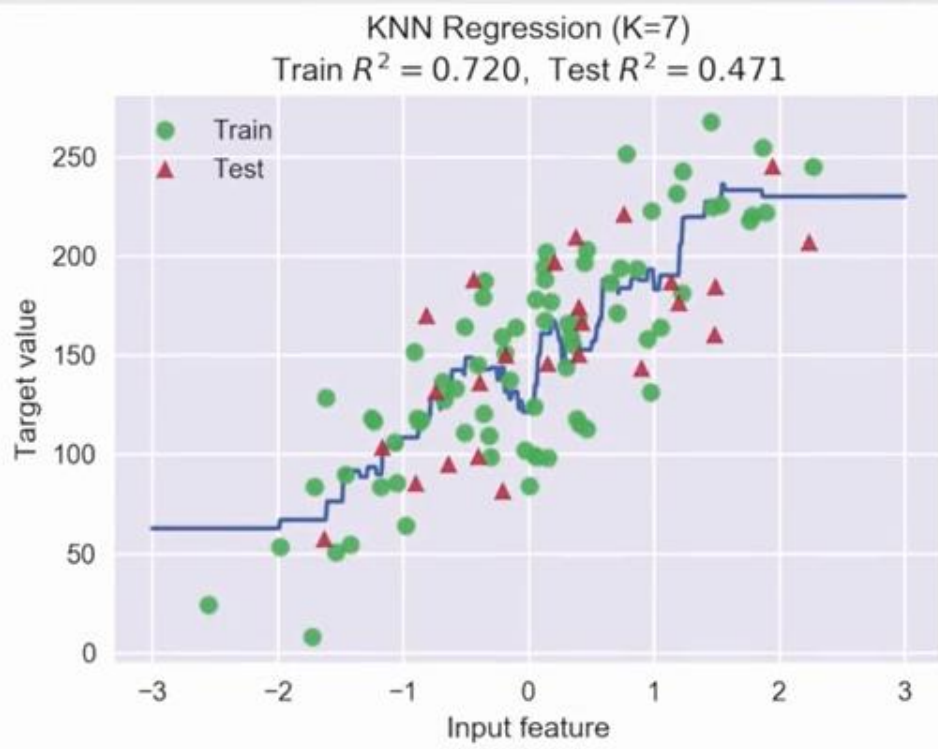


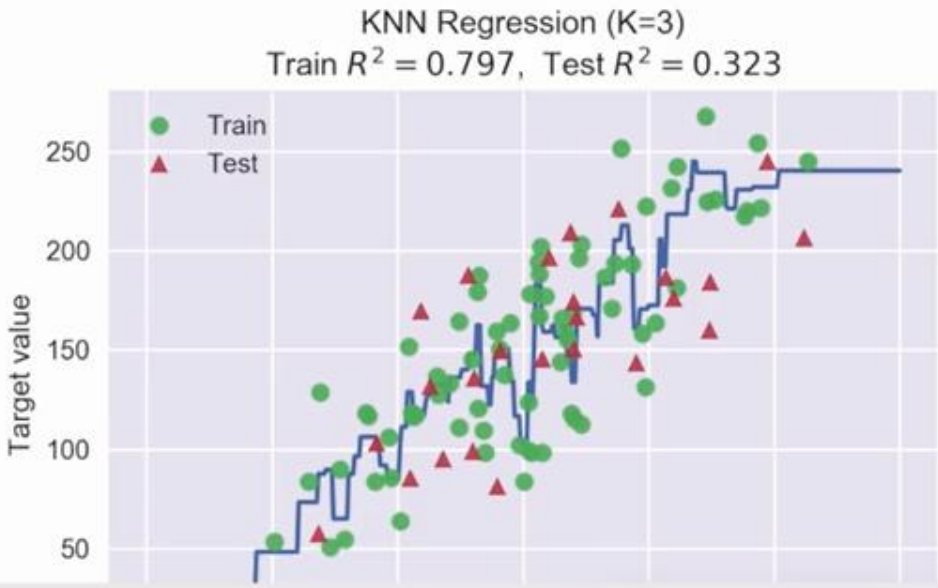


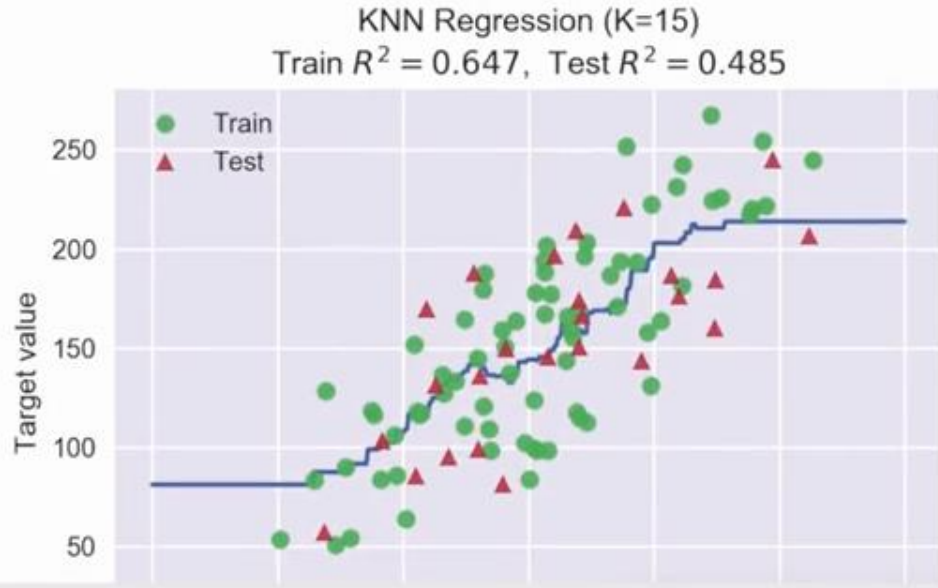
The R^2 ("r-squared") regression score

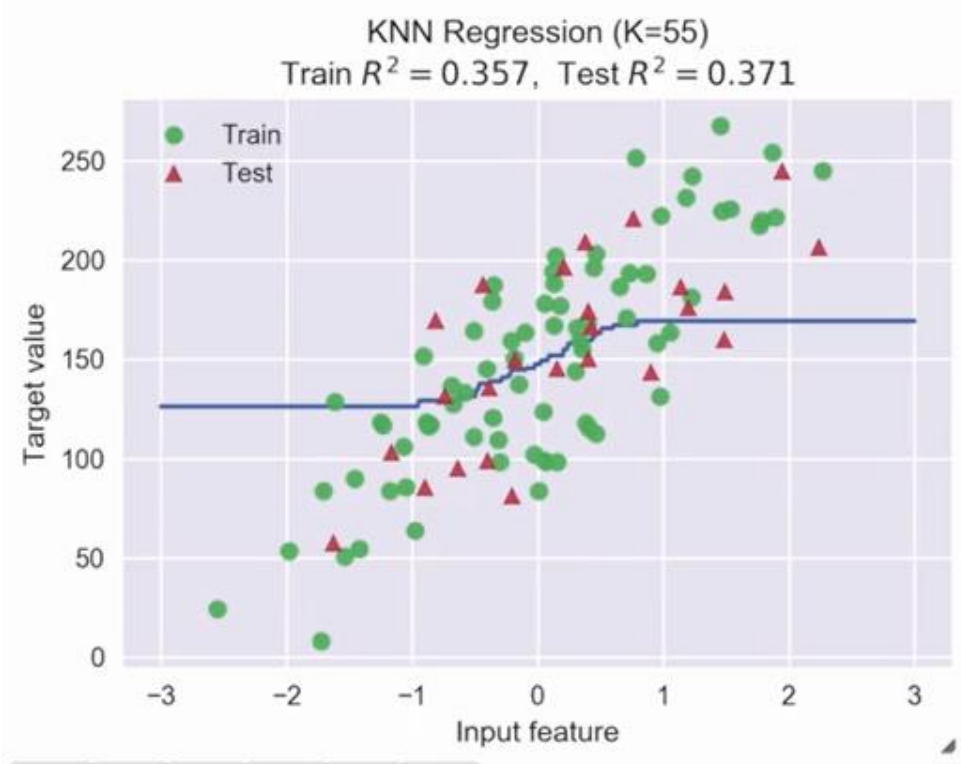
- **Measures how well a prediction model for regression fits the given data.**
- **The score is between 0 and 1:**
  - *A value of 0 corresponds to a constant model that predicts the mean value of all training target values.*
  - *A value of 1 corresponds to perfect prediction*
- **Also known as "coefficient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


KNN Regression (K=1)
Train $R^2 = 1.000$, Test $R^2 = 0.155$
Train
Test
Target value
Input feature


KNN Regression (K=7)
Train $R^2 = 0.720$, Test $R^2 = 0.471$
Train
Test
Target value
Input feature


KNN Regression (K=3)
Train $R^2 = 0.797$, Test $R^2 = 0.323$
Train
Test
Target value


KNN Regression (K=15)
Train $R^2 = 0.647$, Test $R^2 = 0.485$
Train
Test
Target value


KNN Regression (K=55)
Train $R^2 = 0.357$, Test $R^2 = 0.371$
Train
Test
Target value
Input feature
250
200
150
100
50
0
−3
−2
−1
0
1
2
3

KNeighborsClassifier and KNeighborsRegressor: important parameters

Model complexity

- *n_neighbors* : number of nearest neighbors (k) to consider
  - *Default = 5*

Model fitting

- *metric: distance function between data points*
  - *Default: Minkowski distance with power parameter p = 2 (Euclidean)*