Supervised Machine Learning (Part 2)

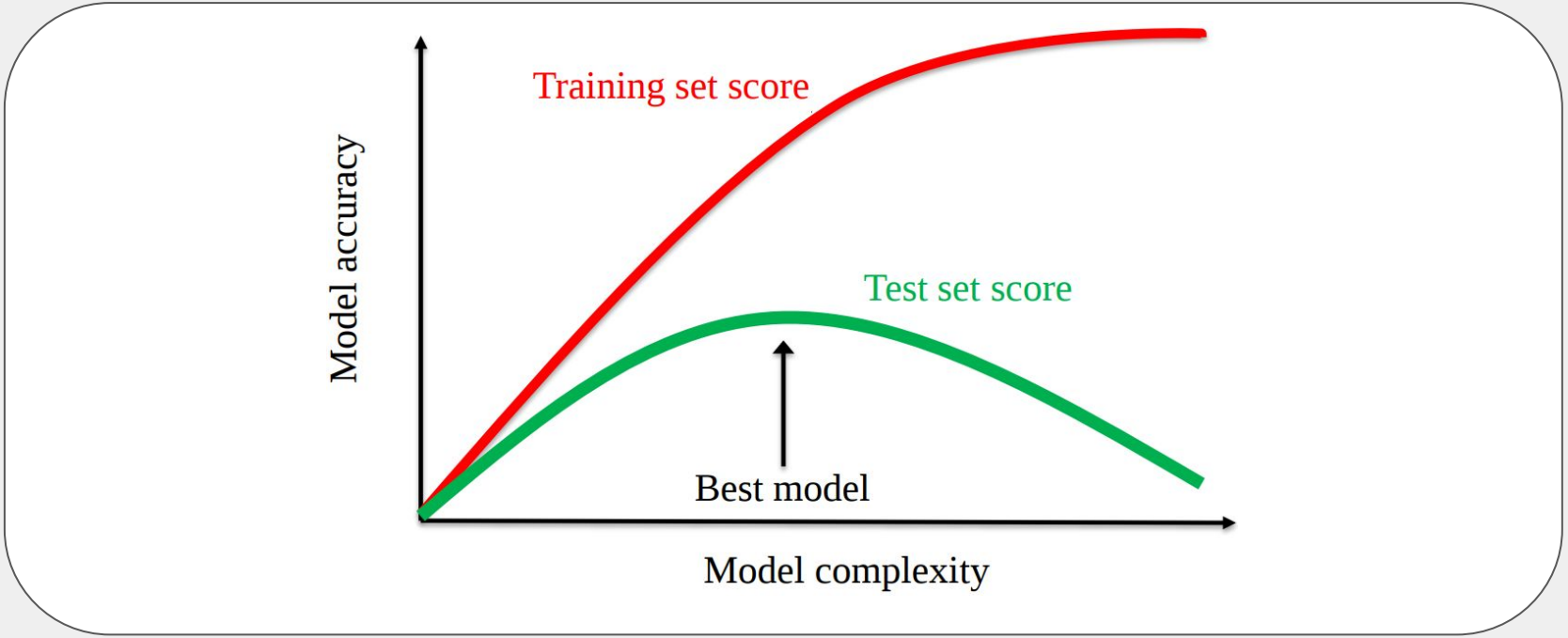
Taught by: Bedoor AlShebli

Based on Kevyn Collins-Thompson's Applied Machine Learning Course

Learning Objectives

1. Understand how a number of **different supervised learning algorithms** learn by estimating their parameters from data to make new predictions.
2. Understand the **strengths and weaknesses** of particular supervised learning methods.
3. Learn **how to apply** specific supervised machine learning algorithms in Python with scikit-learn.
4. Learn about **general principles** of supervised machine learning, like overfitting and how to avoid it.
5. What are the **key parameters that control each models complexity**.

The relationship between model complexity and training / test performance



Quick Review of Important Terms

- **Feature Representation:**
  - Converting properties of an object into a representation that will make it easier for a ML algorithm to use in order to predict future instances.

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192	8.4	7.3	0.55
1	1	apple	granny_smith	180	8.0	6.8	0.59
2	1	apple	granny_smith	176	7.4	7.2	0.60
3	2	mandarin	mandarin	86	6.2	4.7	0.80
4	2	mandarin	mandarin	84	6.0	4.6	0.79
5	2	mandarin	mandarin	80	5.8	4.3	0.77
6	2	mandarin	mandarin	80	5.9	4.3	0.81
7	2	mandarin	mandarin	76	5.8	4.0	0.81
8	1	apple	braeburn	178	7.1	7.8	0.92
9	1	apple	braeburn	172	7.4	7.0	0.89
10	1	apple	braeburn	166	6.9	7.3	0.93
11	1	apple	braeburn	172	7.1	7.6	0.92
12	1	apple	braeburn	154	7.0	7.1	0.88
13	1	apple	golden_delicious	164	7.3	7.7	0.70
14	1	apple	golden_delicious	152	7.6	7.3	0.69
15	1	apple	golden_delicious	156	7.7	7.1	0.69
16	1	apple	golden_delicious	156	7.6	7.5	0.67
17	1	apple	golden_delicious	168	7.5	7.6	0.73
18	1	apple	cripps_pink	162	7.5	7.1	0.83
19	1	apple	cripps_pink	162	7.4	7.2	0.85

Quick Review of Important Terms

- **Feature Representation:**
  - Converting properties of an object into a representation that will make it easier for a ML algorithm to use in order to predict future instances.
- **Data instances / Samples / Examples (X)**

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192	8.4	7.3	0.55
1	1	apple	granny_smith	180	8.0	6.8	0.59
2	1	apple	granny_smith	176	7.4	7.2	0.60
3	2	mandarin	mandarin	86	6.2	4.7	0.80
4	2	mandarin	mandarin	84	6.0	4.6	0.79
5	2	mandarin	mandarin	80	5.8	4.3	0.77
6	2	mandarin	mandarin	80	5.9	4.3	0.81
7	2	mandarin	mandarin	76	5.8	4.0	0.81
8	1	apple	braeburn	178	7.1	7.8	0.92
9	1	apple	braeburn	172	7.4	7.0	0.89
10	1	apple	braeburn	166	6.9	7.3	0.93
11	1	apple	braeburn	172	7.1	7.6	0.92
12	1	apple	braeburn	154	7.0	7.1	0.88
13	1	apple	golden_delicious	164	7.3	7.7	0.70
14	1	apple	golden_delicious	152	7.6	7.3	0.69
15	1	apple	golden_delicious	156	7.7	7.1	0.69
16	1	apple	golden_delicious	156	7.6	7.5	0.67
17	1	apple	golden_delicious	168	7.5	7.6	0.73
18	1	apple	cripps_pink	162	7.5	7.1	0.83
19	1	apple	cripps_pink	162	7.4	7.2	0.85

Quick Review of Important Terms

- **Feature Representation:**
  - Converting properties of an object into a representation that will make it easier for a ML algorithm to use in order to predict future instances.
- **Data instances / Samples / Examples (X)**
- **Target value (y)**
  - In classification, the label of an object is the target value
  - In regression, the continuous value you want to project from the input is your target value

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192	8.4	7.3	0.55
1	1	apple	granny_smith	180	8.0	6.8	0.59
2	1	apple	granny_smith	176	7.4	7.2	0.60
3	2	mandarin	mandarin	86	6.2	4.7	0.80
4	2	mandarin	mandarin	84	6.0	4.6	0.79
5	2	mandarin	mandarin	80	5.8	4.3	0.77
6	2	mandarin	mandarin	80	5.9	4.3	0.81
7	2	mandarin	mandarin	76	5.8	4.0	0.81
8	1	apple	braeburn	178	7.1	7.8	0.92
9	1	apple	braeburn	172	7.4	7.0	0.89
10	1	apple	braeburn	166	6.9	7.3	0.93
11	1	apple	braeburn	172	7.1	7.6	0.92
12	1	apple	braeburn	154	7.0	7.1	0.88
13	1	apple	golden_delicious	164	7.3	7.7	0.70
14	1	apple	golden_delicious	152	7.6	7.3	0.69
15	1	apple	golden_delicious	156	7.7	7.1	0.69
16	1	apple	golden_delicious	156	7.6	7.5	0.67
17	1	apple	golden_delicious	168	7.5	7.6	0.73
18	1	apple	cripps_pink	162	7.5	7.1	0.83
19	1	apple	cripps_pink	162	7.4	7.2	0.85

Quick Review of Important Terms

- **Training and Test Sets**

```
%matplotlib notebook
import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

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

X = fruits[['height', 'width', 'mass', 'color_score']]
y = fruits['fruit_label']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

knn = KNeighborsClassifier(n_neighbors = 5)
knn.fit(X_train, y_train)
print("Accuracy of K-NN classifier on test set: ", knn.score(X_test, y_test))

example_fruit = [[5.5, 2.2, 10, 0.70]]
print("Predicted fruit type for ", example_fruit, " is ", knn.predict(example_fruit))
```

Quick Review of Important Terms

- **Training and Test Sets**
- **Model / Estimator**
  - **“Model fitting”** produces a **“trained model”**
  - Training is the process of *estimating model parameters*

```
%matplotlib notebook
import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

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

X = fruits[['height', 'width', 'mass', 'color_score']]
y = fruits['fruit_label']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

knn = KNeighborsClassifier(n_neighbors = 5)
knn.fit(X_train, y_train)
print("Accuracy of K-NN classifier on test set: ", knn.score(X_test, y_test))

example_fruit = [[5.5, 2.2, 10, 0.70]]
print("Predicted fruit type for ", example_fruit, " is ", knn.predict(example_fruit))
```

Quick Review of Important Terms

- **Training and Test Sets**
- **Model / Estimator**
  - **“Model fitting”** produces a **“trained model”**
  - Training is the process of *estimating model parameters*
- **Evaluation Method**

```
%matplotlib notebook
import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

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

X = fruits[['height', 'width', 'mass', 'color_score']]
y = fruits['fruit_label']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

knn = KNeighborsClassifier(n_neighbors = 5)
knn.fit(X_train, y_train)
print("Accuracy of K-NN classifier on test set: ", knn.score(X_test, y_test))

example_fruit = [[5.5, 2.2, 10, 0.70]]
print("Predicted fruit type for ", example_fruit, " is ", knn.predict(example_fruit))
```

Two Kinds of Supervised Learning:

Classification and **Regression**

Classification

- Both classification and regression take a set of training instances and learn a mapping to a **target value**.
- For **classification,** the **target value is discrete**
  a. Binary: Target value is either 0 (negative class) or 1 (positive class)
     - Eg, detecting fraudulent credit card transaction
  b. Multi-class: Target value is one of a set of discrete values
     - Eg, labeling the type of fruit from its physical attributes
  c. Multi-label: Multiple target values (labels)
     - Eg, labelling the topics discussed on a webpage

Regression

- For **regression**, the **target value is continuous** (floating point / real-value)
    a. Eg, predicting the selling price of a house from its attributes
- Looking at the target value's type will guide you on what supervised learning technique to use.
- Many supervised learning methods have "flavors" for both regression and classification.

Overfitting (and Underfitting)

Generalization, Overfitting, and Underfitting

- **Generalization** 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.
  - Models that are accurate on the training set are expected to be accurate on the test set.

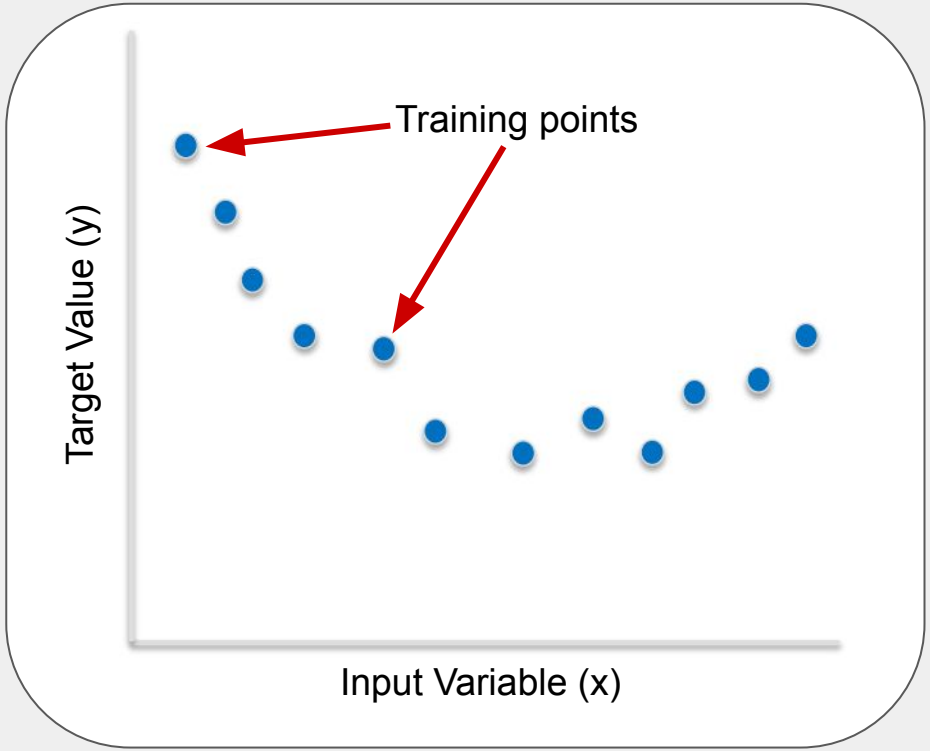
What happens if the trained model is tuned

“too specifically” to the training set?

Generalization, Overfitting, and Underfitting

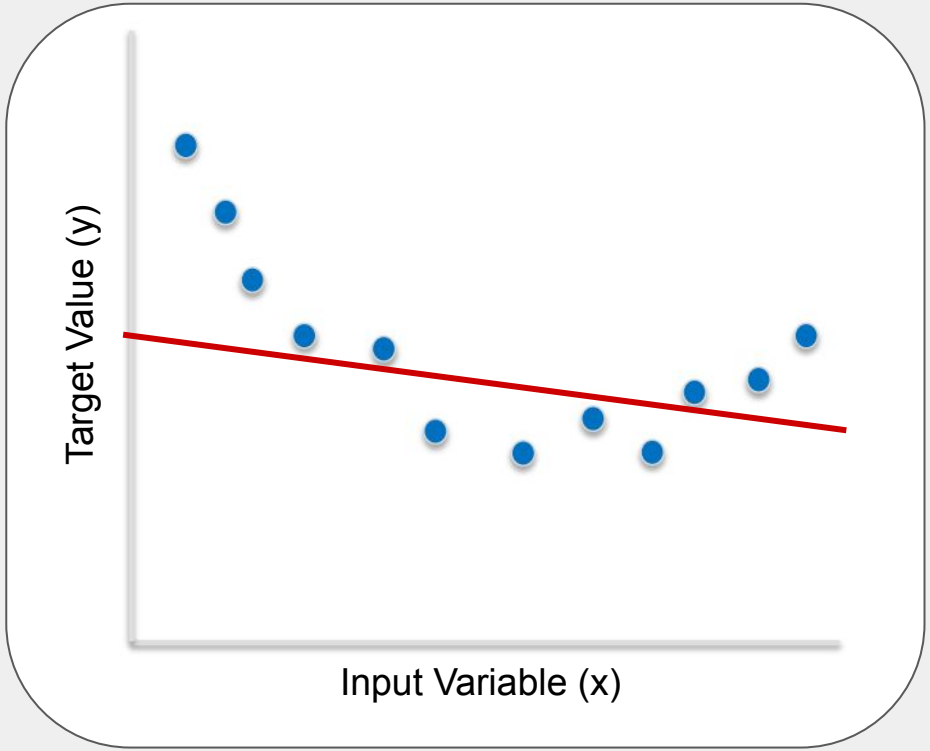
- **Generalization** refers to an algorithm's ability to give accurate predictions for new, previously unseen data.
- Assumptions:
  - Future unseen data (test set) will have the same properties as the current training sets.
  - Models that are accurate on the training set are expected to be accurate 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**, and don't even do well on the training data, **are said to underfit** and also not likely to generalize well.

Overfitting in Regression



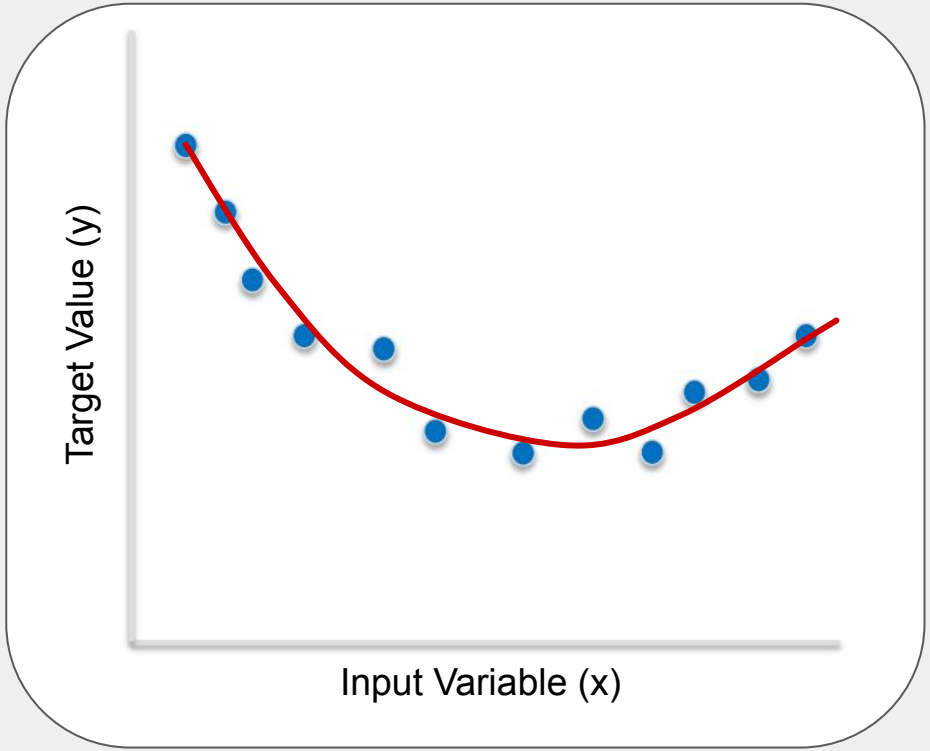
Overfitting in Regression

- Say we want to fit a linear model to predict the linear relationship between the input and target variables.
- In this case, the model has **“underfit”** the data, i.e. the model is too simple for the actual trends that are present.
- The model doesn’t even do well on the training points.
- As a result, such a model is not likely to generalize well to test data.
- **Underfitting = poor performance on the training data, poor generalization to other data.**



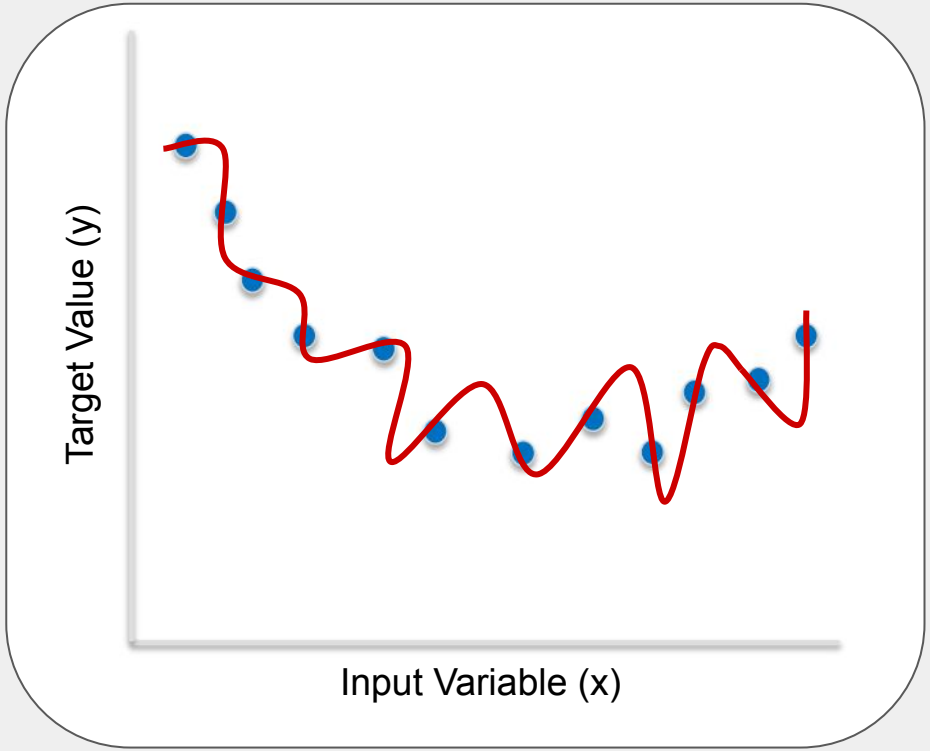
Overfitting in Regression

- A better model would be a quadratic / polynomial one.
- It would be a good fit, and captures the general trend of the point while ignoring the little variations that might exist due to noise.
- **Good fit = good performance on the training data, good generalization to other data.**



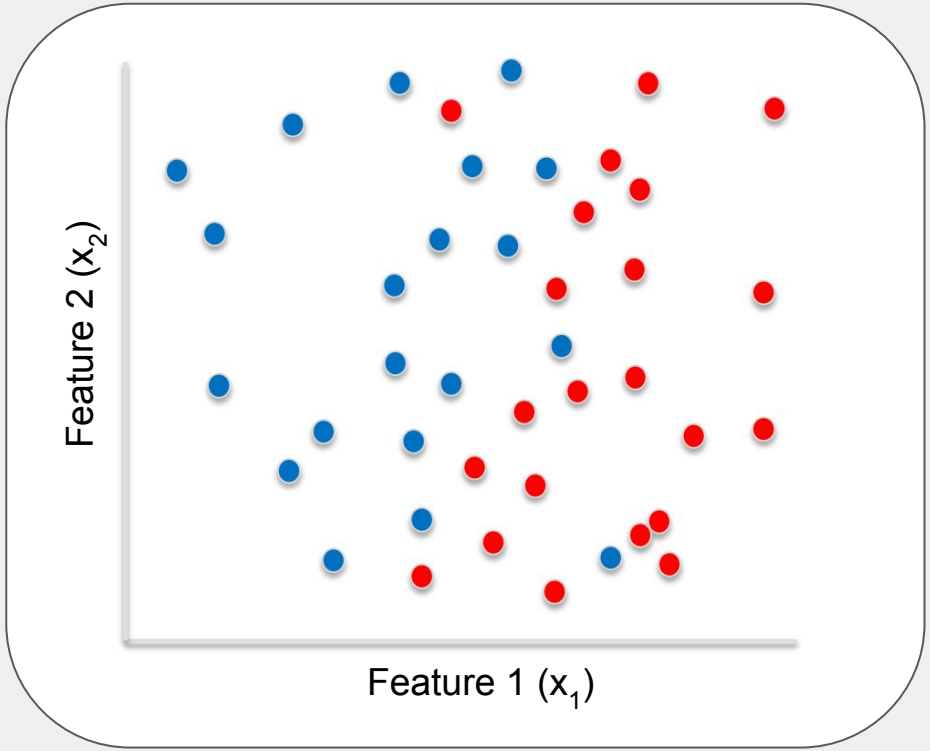
Overfitting in Regression

- We can fit the training data to a more complex model that is a function of several different parameters.
- Such a model is high in variance and focuses on capturing the local variations of the training data rather than capturing the global trend.
- In this case, the model has "**overfit**" the data, i.e. there isn't enough data to constrain the parameters enough to recognize a global trend.
- **Overfitting = good performance on the training data, poor generalization to other data.**



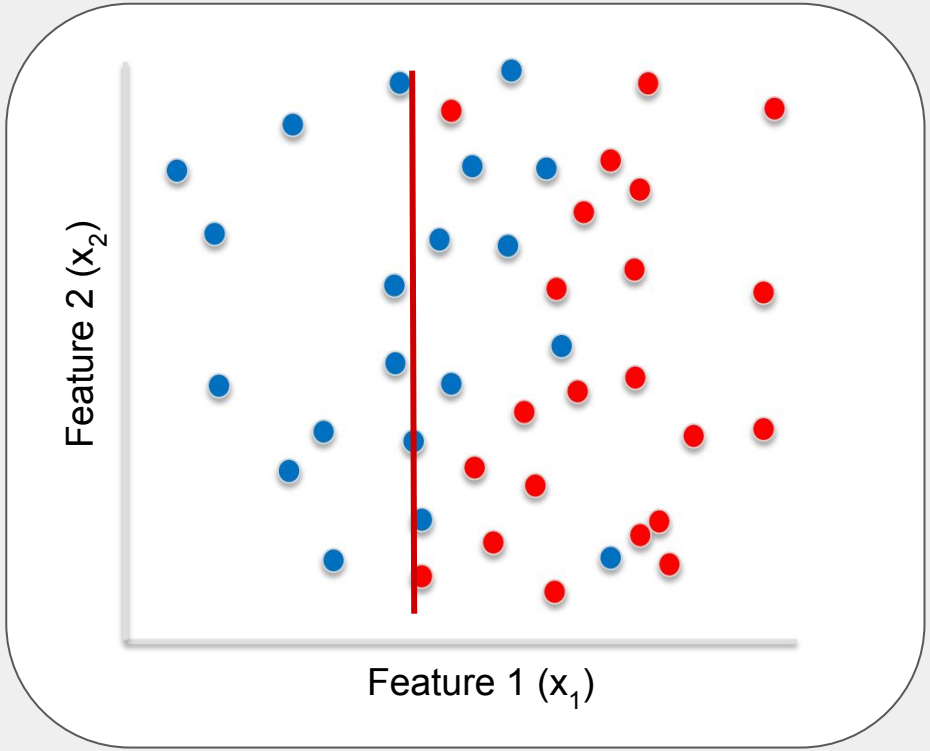
Overfitting in Classification

- There are 2 features associated with each data instance
- Identify the decision boundary of this binary classification problem.
    - Blue points are class 0
    - Red points are class 1



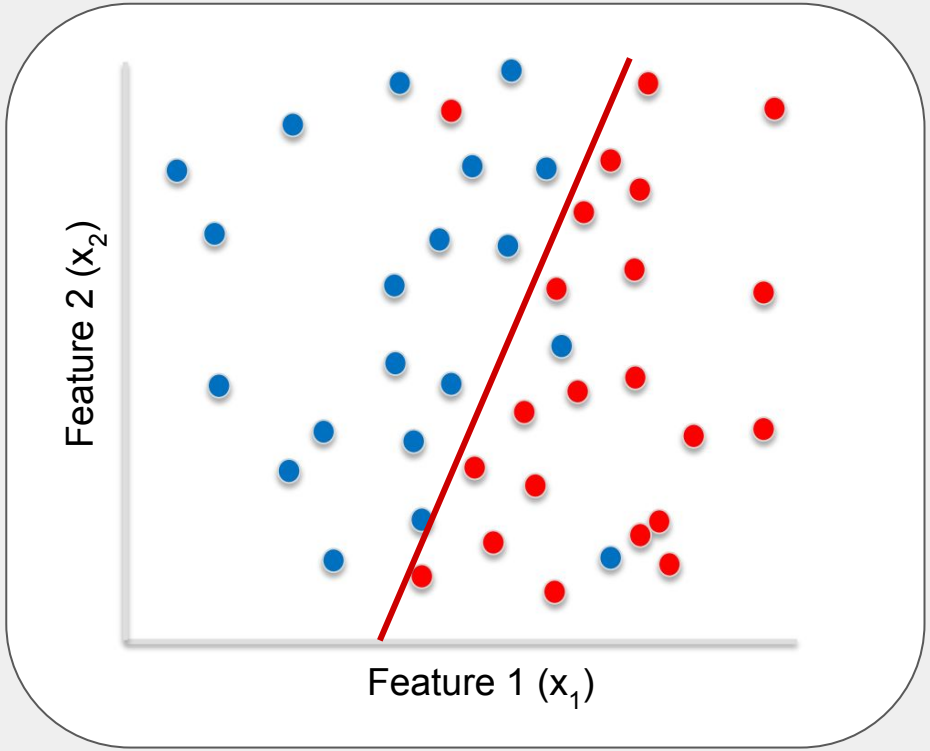
Overfitting in Classification

- We can use a **linear classifier** and draw a straight line through the space (as our decision boundary).
- If we design a simple classifier that only looks at Feature 1, the division between the classes in the training data will **not likely generalize** well to new data.
- As such, this classifier has **underfit** the data.



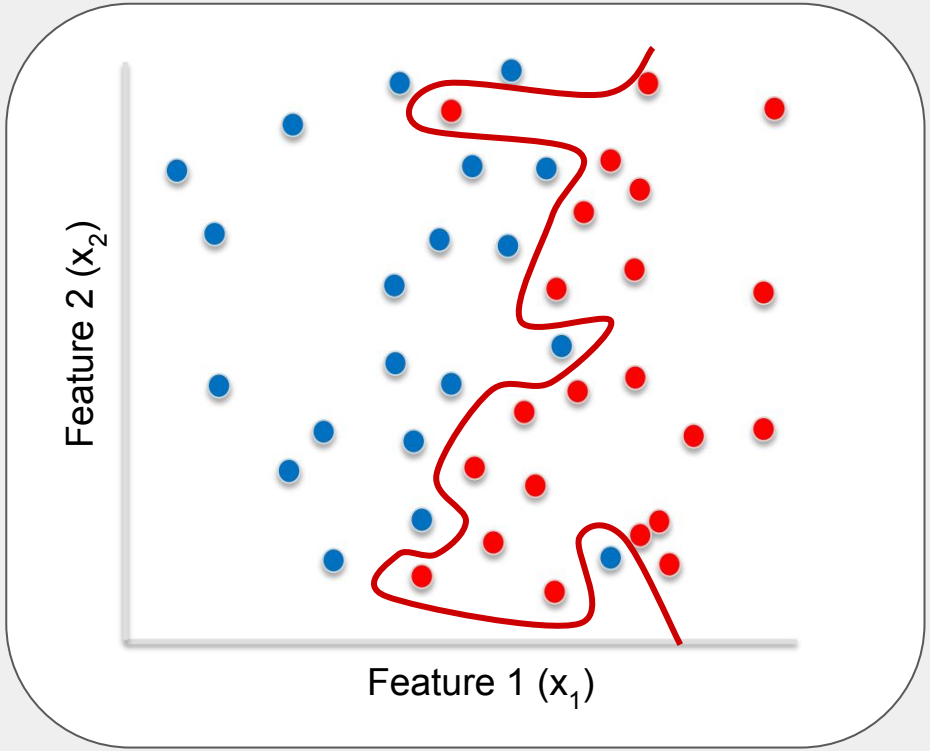
Overfitting in Classification

- **Good fit** model is one that finds a general difference between classes 0 and 1.
- While there may be occasionally points on the wrong side of the boundary line, it did a good job at finding a global separation between the two classes.



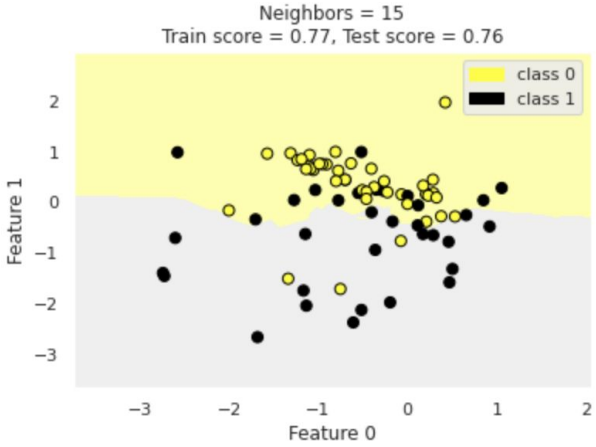
Overfitting in Classification

- An **overfit** model would be one that has lots of parameters.
- This results in a highly variable decision boundary
- It does not have enough data to see the global trend that would have resulted in better overall generalization.

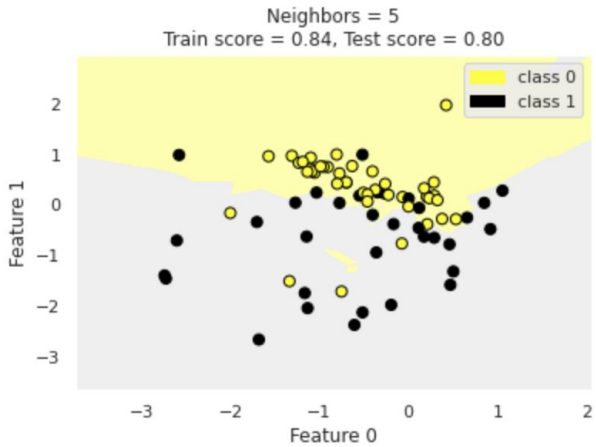


Overfitting with KNN Classifier

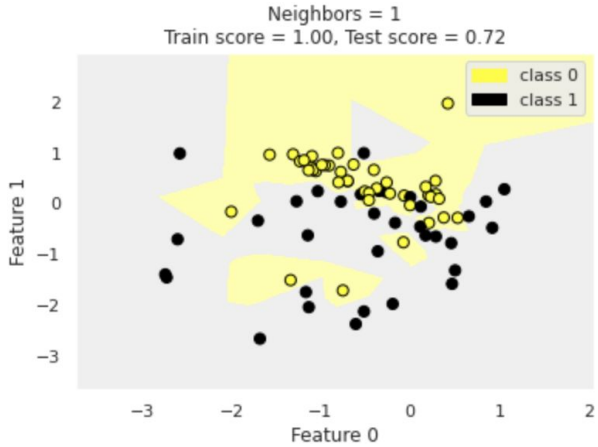
underfit ← → **overfit**



k=15



k=5

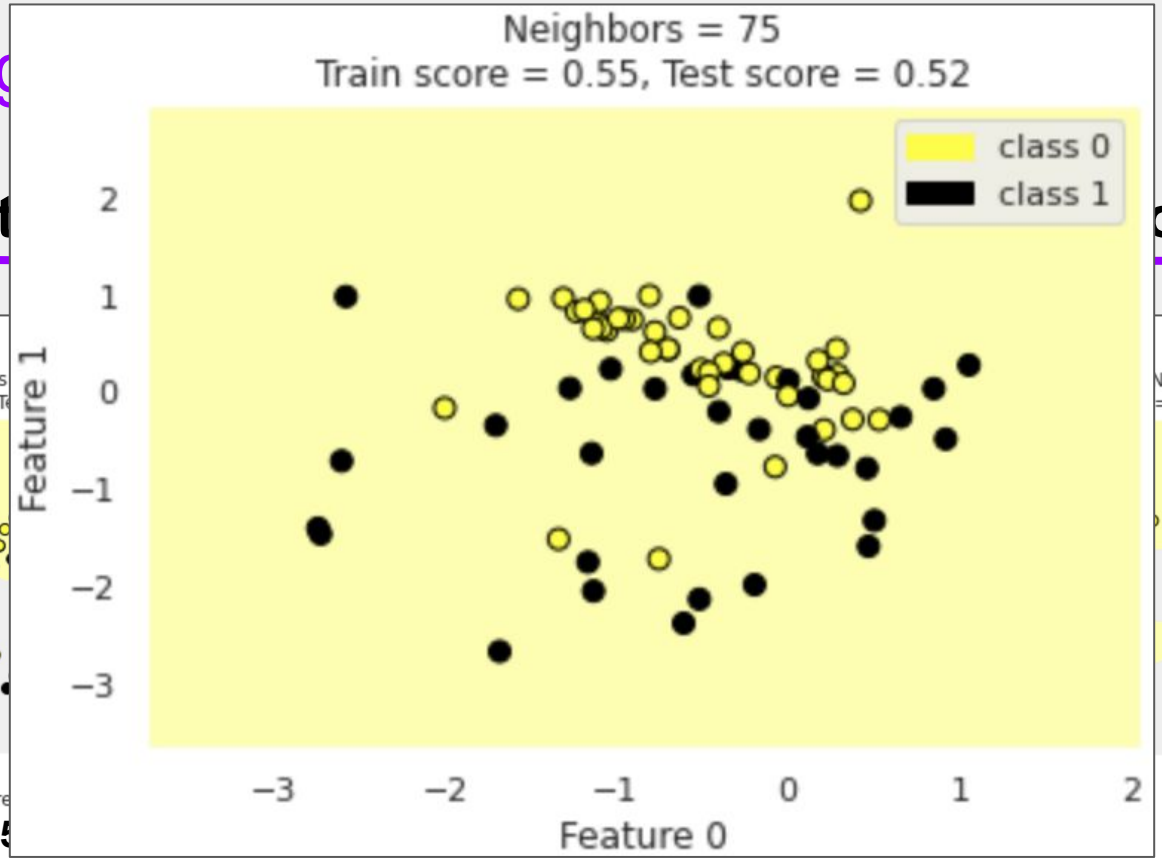


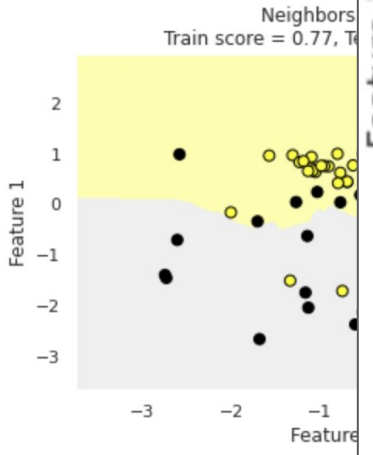
k=1

What happens when k = # training points?

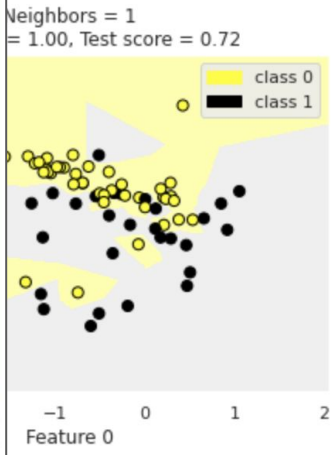
Overfitting

underfit ← → **overfit**





k=15



k=1

What happens when k = # training points?

Pop Quiz!

Which of the following is true about the k-nearest neighbors classification algorithm, assuming uniform weighting on the k neighbors? Select all that apply.

- A low value of “k” (close to 1) is more likely to overfit the training data and lead to worse accuracy on the test data, compared to higher values of “k”.
- A low value of “k” (close to 1) is less likely to overfit the training data and lead to better accuracy on the test data, compared to higher values of “k”.
- Setting “k” to the number of points in the training set will result in a classifier that *always* predicts the majority class.
- Setting “k” to the number of points in the training set will result in a classifier that *never* predicts the majority class.
- The k-nearest neighbors classification algorithm has to memorize all of the training examples to make a prediction.
- The performance of a k-nearest neighbors classifier is relatively insensitive to the choice of “k” on most datasets.

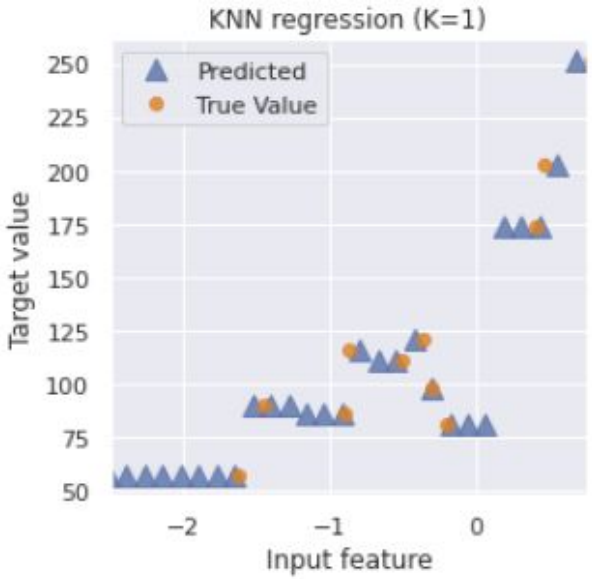
Pop Quiz!

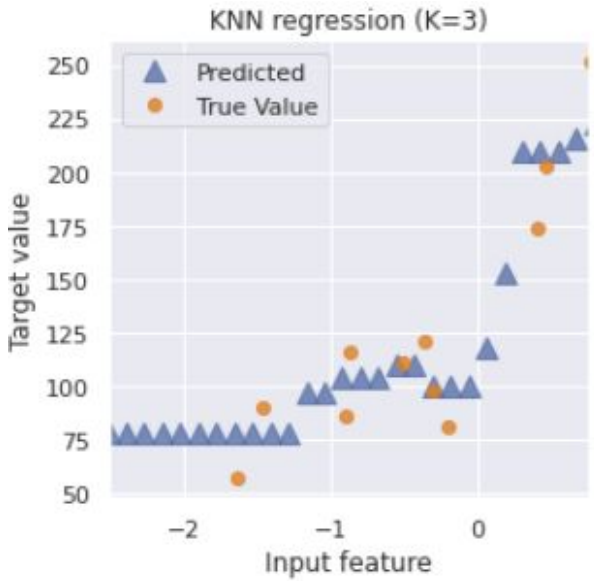
Which of the following is true about the k-nearest neighbors classification algorithm, assuming uniform weighting on the k neighbors? *Select all that apply.*

- A low value of “k” (close to 1) is more likely to overfit the training data and lead to worse accuracy on the test data, compared to higher values of “k”.
- A low value of “k” (close to 1) is less likely to overfit the training data and lead to better accuracy on the test data, compared to higher values of “k”.
- Setting “k” to the number of points in the training set will result in a classifier that *always* predicts the majority class.
- Setting “k” to the number of points in the training set will result in a classifier that *never* predicts the majority class.
- The k-nearest neighbors classification algorithm has to memorize all of the training examples to make a prediction.
- The performance of a k-nearest neighbors classifier is relatively insensitive to the choice of “k” on most datasets.

K-Nearest Neighbors Regression

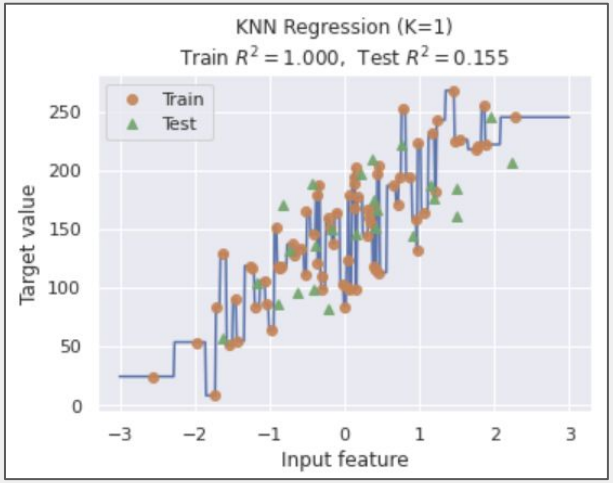
KNN Regression

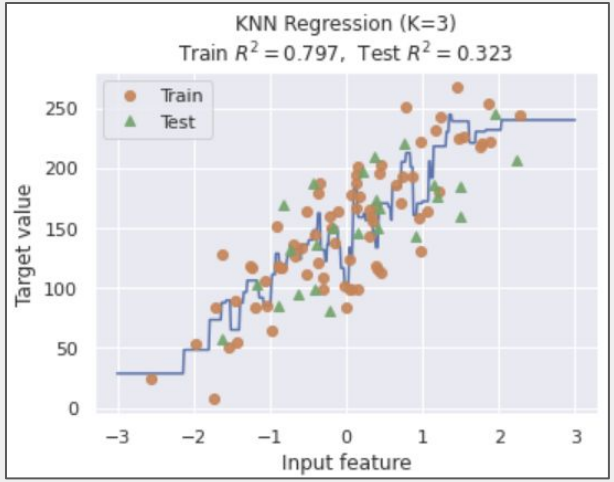


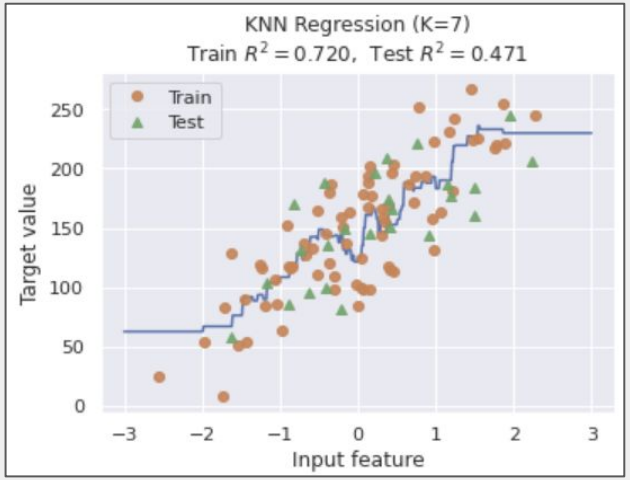


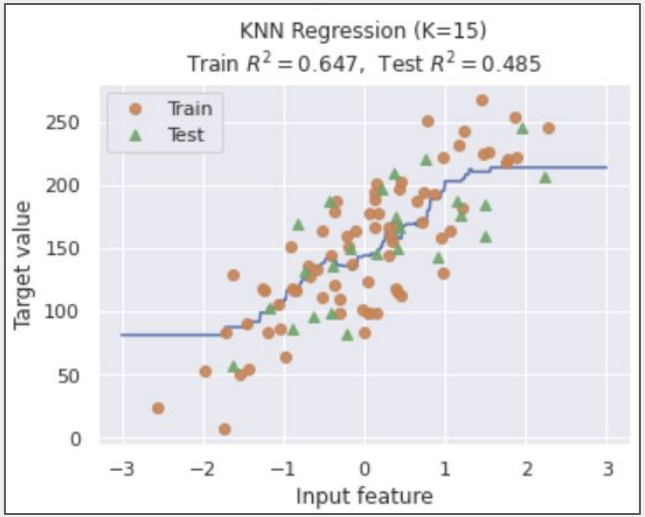
The R^2 (“r-squared”) Regression Score

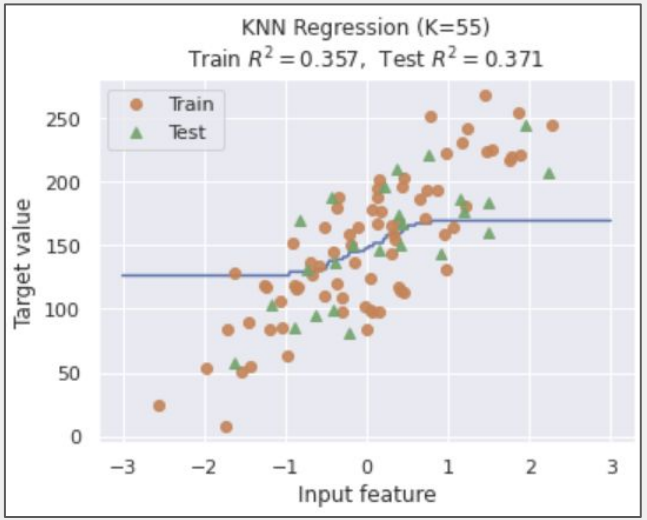
- As target values are continuous in a regression, different accuracy scores are needed than those used in a classifier.
- R^2 measures how well a regression model fits the given data.
  - The percent of variation in y explained by the x variables
- The score is between 0 to 1:
  - A value of 0 corresponds 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=3)
Train $R^2 = 0.797$, Test $R^2 = 0.323$
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=15)
Train $R^2 = 0.647$, Test $R^2 = 0.485$
Train
Test
Target value
Input feature

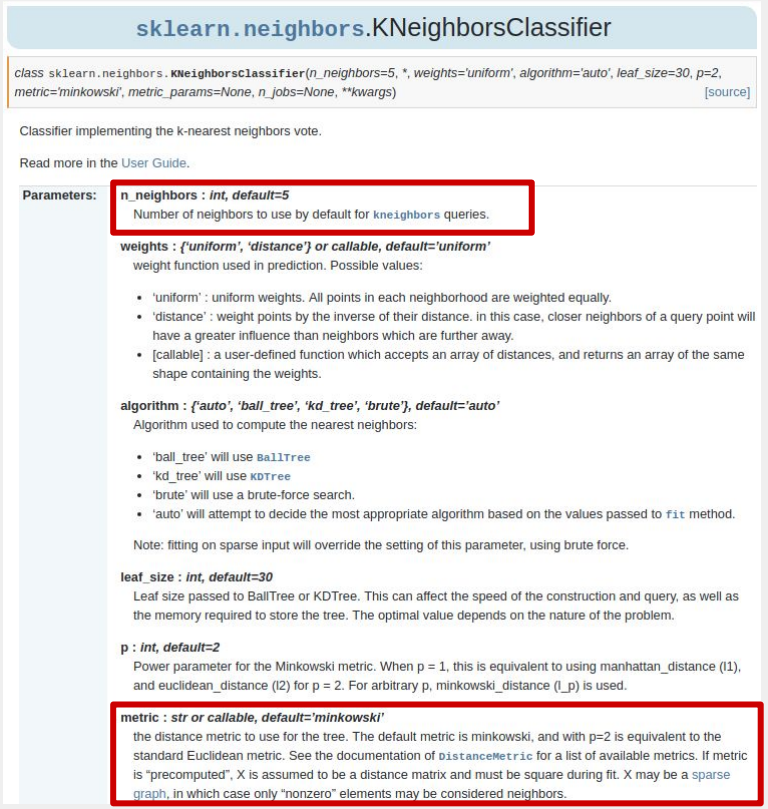

KNN Regression (K=55)
Train $R^2 = 0.357$, Test $R^2 = 0.371$
Train
Test
Target value
Input feature

Code it using Scikit-Learn

```
from sklearn.neighbors import KNeighborsRegressor

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

knnreg = KNeighborsRegressor(n_neighbors = 5).fit(X_train, y_train)

print(knnreg.predict(X_test))
print('R-squared test score: {:.3f}'.format(knnreg.score(X_test, y_test)))
```

Key Parameters in `KNeighborsClassifier` and `KNeighborsRegressor`

- **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)



sklearn.neighbors.KNeighborsClassifier

class sklearn.neighbors.**KNeighborsClassifier**(*n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs*) [source]

Classifier implementing the k-nearest neighbors vote.

Read more in the User Guide.

Parameters:

n_neighbors : *int, default=5*
Number of neighbors to use by default for `kneighbors` queries.

weights : *{'uniform', 'distance'} or callable, default='uniform'*
weight function used in prediction. Possible values:

- 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
- 'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
- [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

algorithm : *{'auto', 'ball_tree', 'kd_tree', 'brute'}, default='auto'*
Algorithm used to compute the nearest neighbors:

- 'ball_tree' will use `BallTree`
- 'kd_tree' will use `KDTree`
- 'brute' will use a brute-force search.
- 'auto' will attempt to decide the most appropriate algorithm based on the values passed to `fit` method.

Note: fitting on sparse input will override the setting of this parameter, using brute force.

leaf_size : *int, default=30*
Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.

p : *int, default=2*
Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.

metric : *str or callable, default='minkowski'*
the distance metric to use for the tree. The default metric is minkowski, and with p=2 is equivalent to the standard Euclidean metric. See the documentation of `DistanceMetric` for a list of available metrics. If metric is "precomputed", X is assumed to be a distance matrix and must be square during fit. X may be a sparse graph, in which case only "nonzero" elements may be considered neighbors.

Linear Regression: Least Squares

Linear Models

- A linear model is a **sum of weighted variables** that predicts a target output value given an input data instance.
  - *Example:* linear regression for predicting the market value of a house given two “informative” features: property yearly tax and the age of the house.
    - **Model:**

$$\widehat{Y_{PRICE}} = 212000 + 109X_{TAX} - 2000X_{AGE}$$

    - **House features:**

taxes per year (X_{TAX}) and age in years (X_{AGE})

    - **Predicted selling price:** A house with feature values (X_{TAX}, X_{AGE}) of (10000, 75) would have a predicted selling price of:

$$\widehat{Y_{PRICE}} = 212000 + 109 * 10000 - 2000 * 75 = 1152000$$

Linear Regression

- **Input instance:** feature vector: $x = (x_0, x_1, x_2, \ldots, x_n)$
- **Predicted output:** $\hat{y} = \hat{w}_0 x_0 + \hat{w}_1 x_1 + \ldots + + \hat{w}_n x_n + \hat{b}$
- **Parameters to estimate:** feature weights / model coefficients:

  $\hat{w} = (\hat{w}_0, \hat{w}_1, \ldots, \hat{w}_n)$

  constant bias term / intercept: $\hat{b}$

Linear Regression Model with one variable / feature

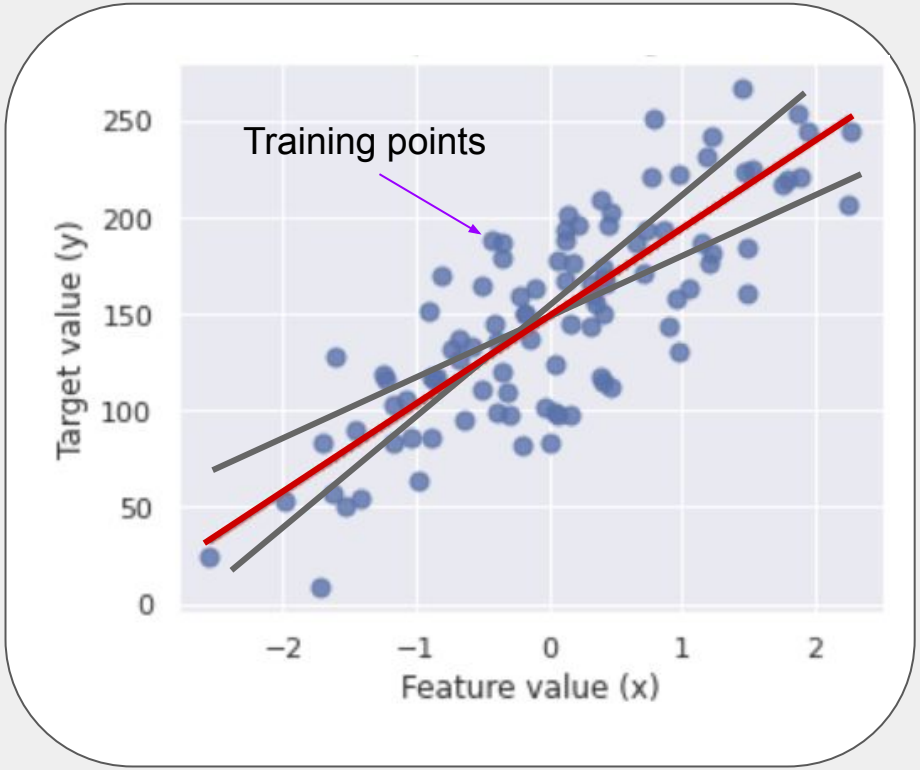
- **Input instance:**

$$x = (x_0)$$

- **Predicted output:**

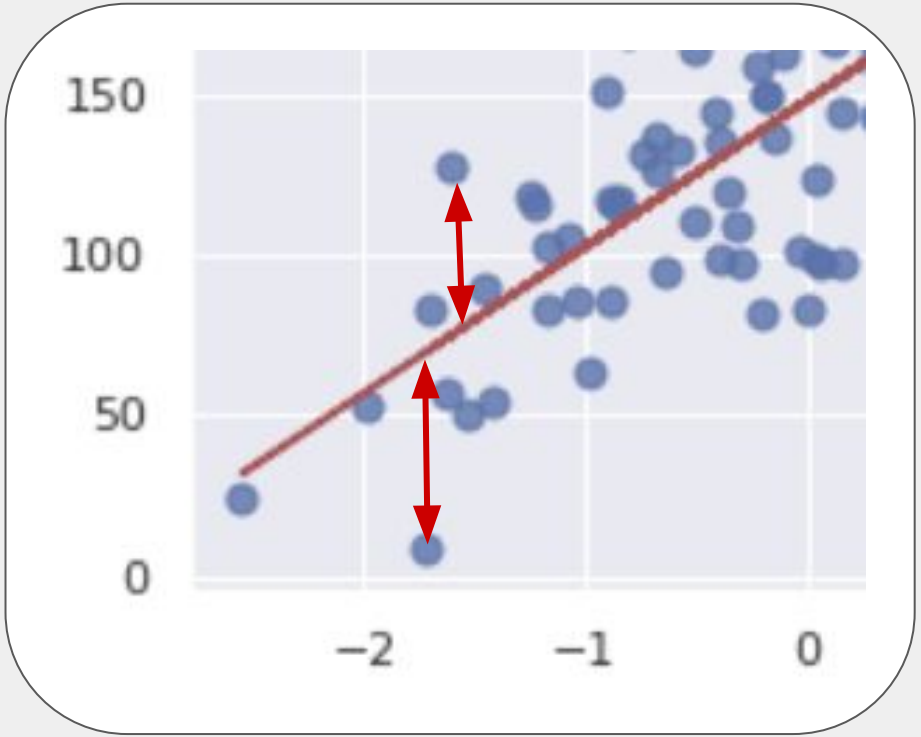
$$\hat{y} = \hat{w}_0 x_0 + \hat{b}$$

- **Parameters to estimate:**
  a. **Slope:** $\hat{w}_0$
  b. **intercept:** $\hat{b}$



Least Squares 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 `(RSS/#training points)`
- **No parameters to control model complexity.**



$$RSS(w,b) = \sum_{i=1}^{N} (y_i - (w.x_i + b))^2$$

Training set target value

Predicted target value using model

* RSS = Residual Sum of Squares

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 coeff (w): {}'.format(linreg.coef_))
print('linear model intercept (b): {:.3f}'.format(linreg.intercept_))
print('R-squared score (training): {:.3f}'.format(linreg.score(X_train, y_train)))
print('R-squared score (test): {:.3f}'.format(linreg.score(X_test, y_test)))
```

`linreg.coef_`

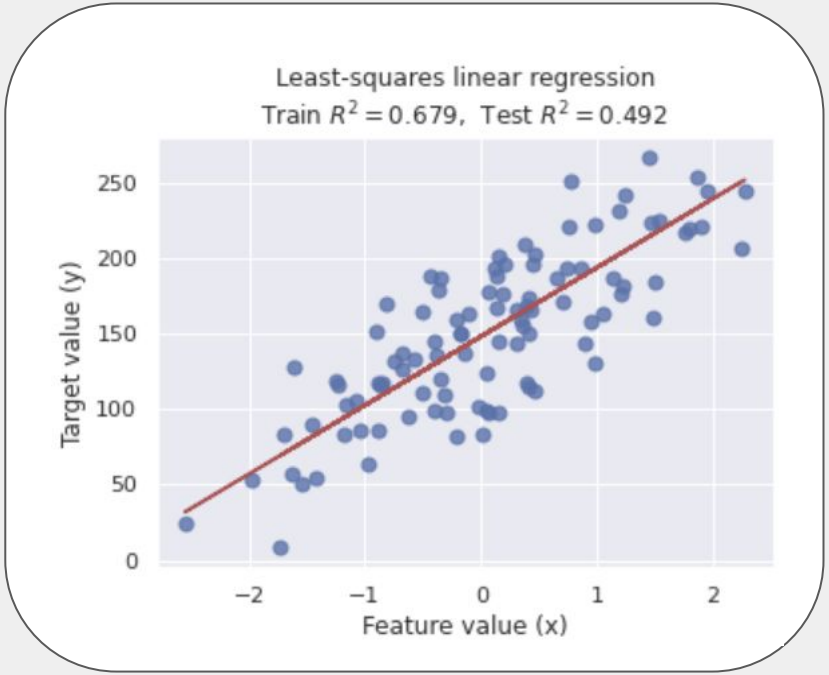
`linreg.intercept_`

$$\hat{y} = \hat{w}_0 x_0 + \hat{b}$$

```
linear model coeff (w): [45.71]
linear model intercept (b): 148.446
R-squared score (training): 0.679
R-squared score (test): 0.492
```

KNN Regression vs. Least-Squares Linear Regression

- Doesn't make assumptions about the structure of the data
- Gives potentially accurate but sometimes unstable predictions that are sensitive to small changes in the training data.



- Make strong assumptions about the structure of the data; i.e. the target value can be predicted using a weighted sum of input variables.
- Gives stable but potentially inaccurate predictions.

Quick Review on Linear Regression

1- Which of the following statement is true about outliers in Linear regression?

A) Linear regression is sensitive to outliers
B) Linear regression is not sensitive to outliers
C) Can’t say
D) None of these

Quick Review on Linear Regression

1- Which of the following statement is true about outliers in Linear regression?

A) Linear regression is sensitive to outliers
B) Linear regression is not sensitive to outliers
C) Can't say
D) None of these

The slope of the regression line will change due to outliers in most of the cases. So Linear Regression is sensitive to outliers.

2- What method do we use to find the best fit line for data in Linear Regression?

Quick Review on Linear Regression

1- Which of the following statement is true about outliers in Linear regression?

A) Linear regression is sensitive to outliers
B) Linear regression is not sensitive to outliers
C) Can't say
D) None of these

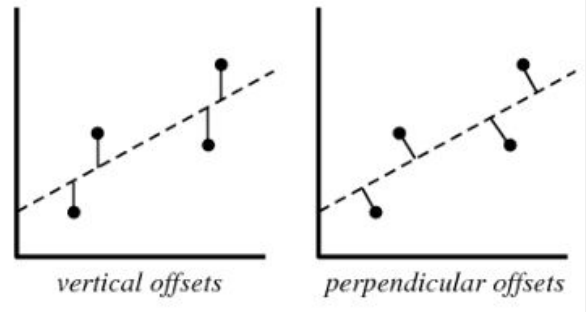
The slope of the regression line will change due to outliers in most of the cases. So Linear Regression is sensitive to outliers.

2- What method do we use to find the best fit line for data in Linear Regression?

Least square error

3- Which of the following offsets, do we use in linear regression's least square line fit? Suppose x-axis is a feature and y-axis is target value.

A) Vertical offset
B) Perpendicular offset
C) Both, depending on the situation
D) None of above



Quick Review on Linear Regression

1- Which of the following statement is true about outliers in Linear regression?

A) Linear regression is sensitive to outliers
B) Linear regression is not sensitive to outliers
C) Can't say
D) None of these

The slope of the regression line will change due to outliers in most of the cases. So Linear Regression is sensitive to outliers.

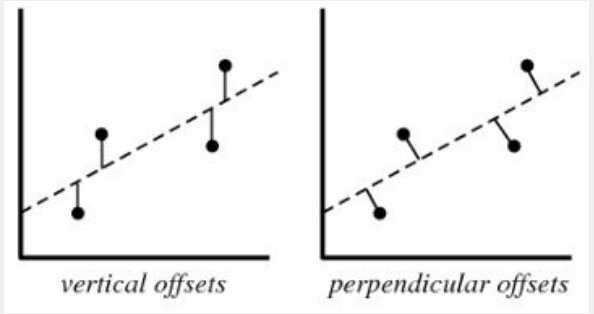
2- What method do we use to find the best fit line for data in Linear Regression?

Least square error

3- Which of the following offsets, do we use in linear regression's least square line fit? Suppose x-axis is a feature and y-axis is target value.

A) Vertical offset
B) Perpendicular offset
C) Both, depending on the situation
D) None of above

We always consider residuals as vertical offsets. We calculate the direct differences between actual value and the Y labels.



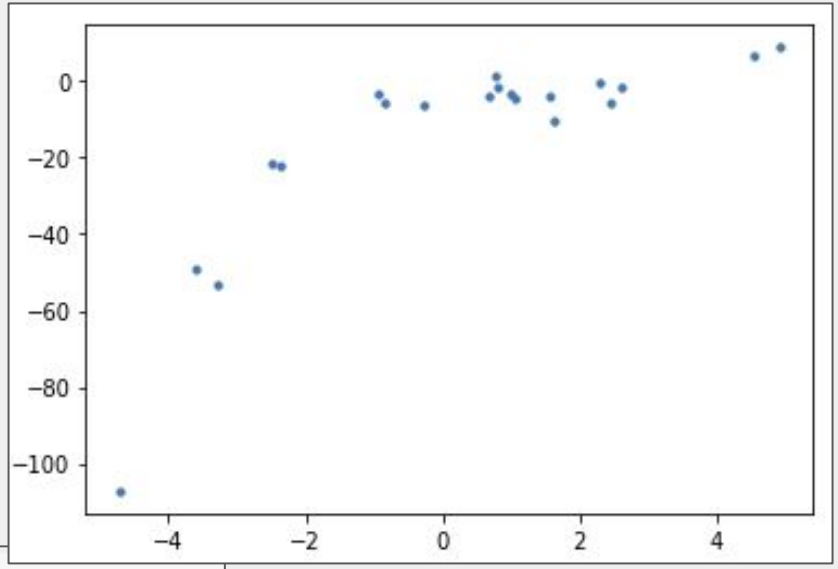
Polynomial Regression

Polynomial Regression

- Linear regression requires the relation between the dependent variable and the independent variable to be linear.
- *What if the distribution of the data was more complex as shown in this figure?*
- *Can linear models be used to fit non-linear data?*
- *How can we generate a curve that best captures the data as shown below?*

Why Polynomial Regression?

To understand the need for polynomial regression, let's generate some random training set first.

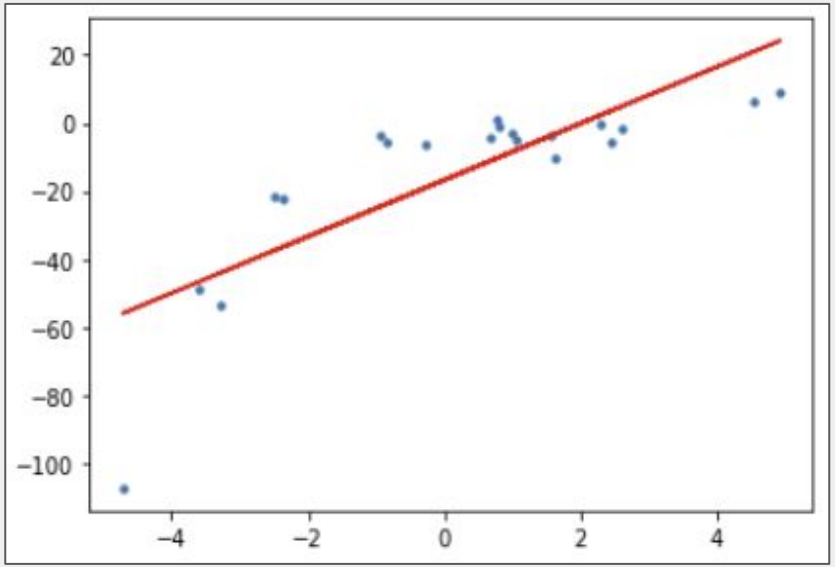



```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(0)
x = 2 - 3 * np.random.normal(0, 1, 20)
y = x - 2 * (x ** 2) + 0.5 * (x ** 3) + np.random.normal(-3, 3, 20)
plt.scatter(x,y, s=10)
plt.show()
```

Apply Linear Regression

Let's apply a linear regression model to this dataset.

We can see that the straight line is unable to capture the patterns in the data. This is an example of **under-fitting.**



Computing the RMSE (Root Mean Square Error) and R^2-score of the linear line gives:

```
Linear Regression RMSE = 15.908
Linear Regression R^2 = 0.639
```

How do we overcome under-fitting?

- Increase the complexity of the model!

How?

- Generate a higher order equation by adding powers of the original features as “new features”

The linear model:

$$\hat{y} = \hat{b} + \hat{w}_1 x_1$$

Can be transformed to:

$$\hat{y} = \hat{b} + \hat{w}_1 x_1 + \hat{w}_2 x_1^2$$

Notes:

- This is still considered a **linear model** as the weights / coefficients associated with the features are still linear!
- x^2 is considered a new feature.
- While linear, the curve we are fitting is **quadratic** in nature.

Let's look at how to code it...

To convert the original features into their higher order terms, we will use the `PolynomialFeatures` class provided by `scikit-learn`.

```
from sklearn.preprocessing import PolynomialFeatures

#to generate the polynomial features
polynomial_features= PolynomialFeatures(degree=2)
x_poly = polynomial_features.fit_transform(x)
```

Output:

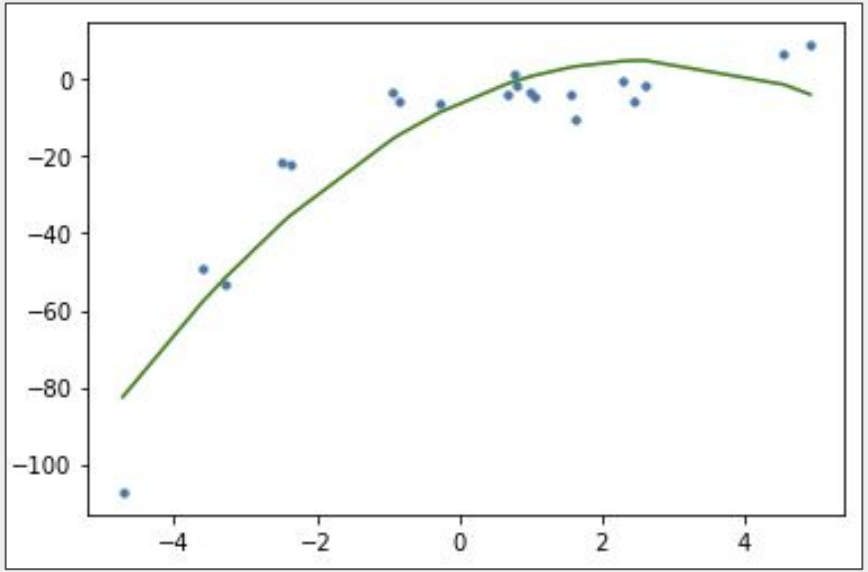
```
Let's take the first three rows of X (one feature):
[[-3.29215704]
 [ 0.79952837]
 [-0.93621395]]

If we apply polynomial transformation of degree 2, the feature vectors become:
[[ 1.         -3.29215704 10.83829796]
 [ 1.          0.79952837  0.63924562]
 [ 1.         -0.93621395  0.87649656]]
```

Let's look at how to code it...

Next, we train the model using `LinearRegression`.

```
model = LinearRegression()
model.fit(x_poly, y)
y_poly_pred = model.predict(x_poly)
```

Fitting a linear regression model on the transformed features gives the following plot, where the green line represents the model's `y_poly_pred` values.



It is quite clear from the plot that the quadratic curve is able to fit the data better than the linear line.

Computing the RMSE and R^2-score of the quadratic plot gives:

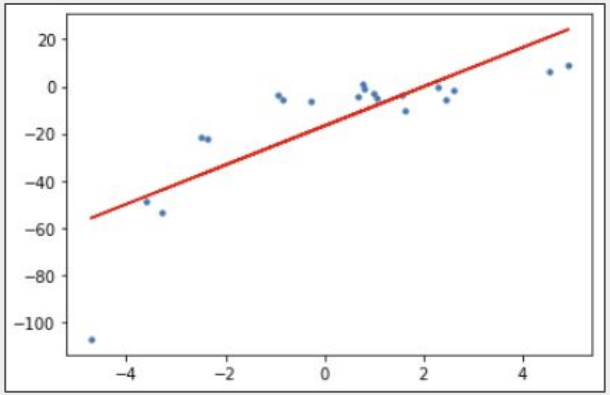
```
Polynomial Regression RMSE = 10.120
Polynomial Regression R^2 = 0.854
```

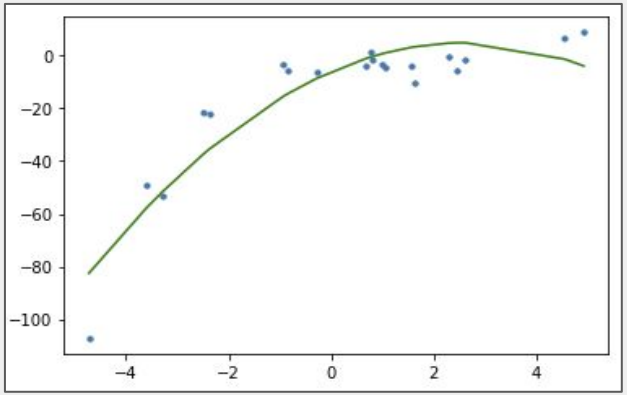
Let’s compare the polynomial to the linear regression...

We can see that RMSE has decreased and R^2-score has increased when compared to the linear line

```
Linear Regression RMSE = 15.908
Linear Regression R^2 = 0.639
```

```
Polynomial Regression RMSE = 10.120
Polynomial Regression R^2 = 0.854
```



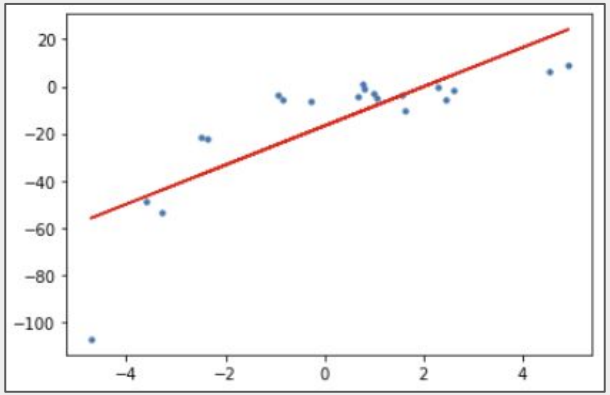


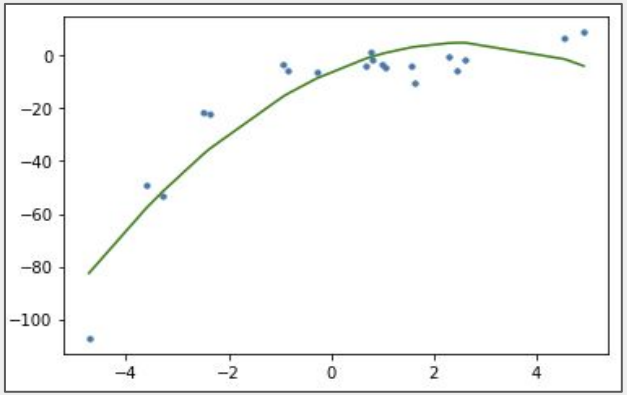
Let’s compare the polynomial to the linear regression...

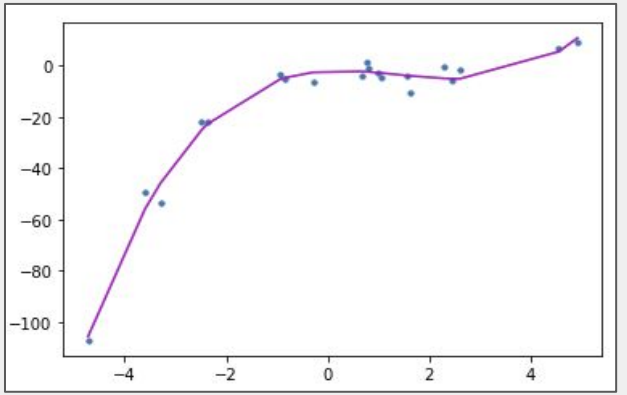
If we try to fit a cubic curve (set `degree=3`) to the dataset, we can see that it passes through more data points than the quadratic and the linear plots.

```
Linear Regression RMSE = 15.908
Linear Regression R^2 = 0.639
```

```
Polynomial Regression RMSE = 10.120
Polynomial Regression R^2 = 0.854
```



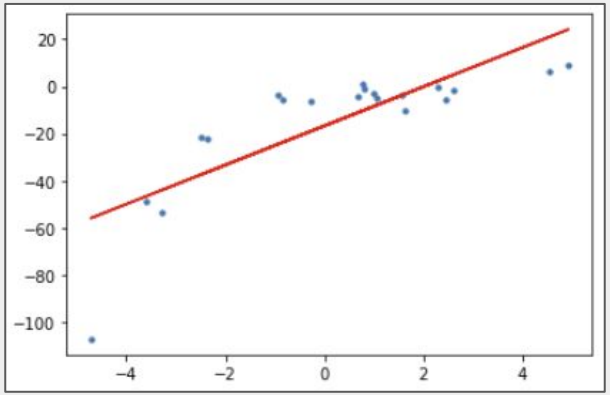




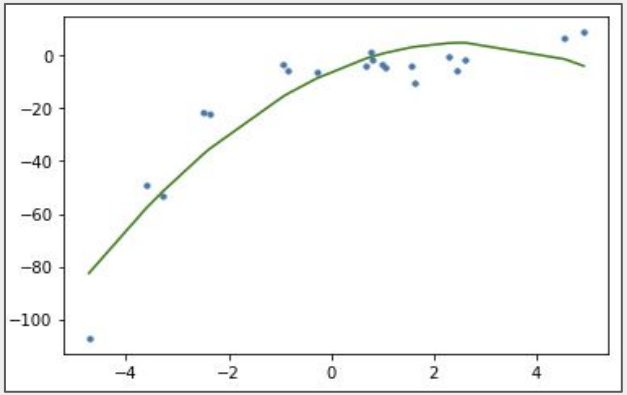
Let’s compare the polynomial to the linear regression...

… and we find that the cubic regression RMSE has decreased even more and R^2-score has increased when compared to the quadratic line.

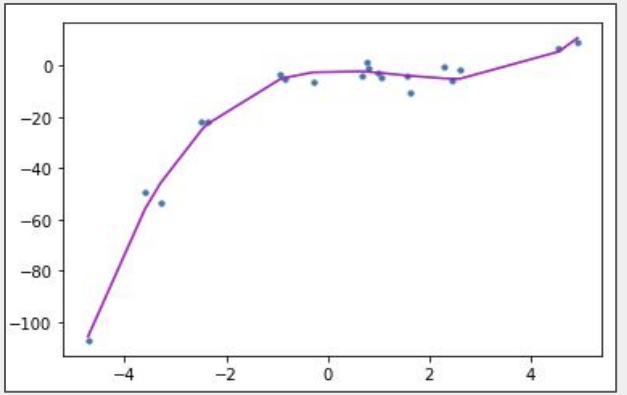
```
Linear Regression RMSE = 15.908
Linear Regression R^2 = 0.639
```




```
Polynomial Regression RMSE = 10.120
Polynomial Regression R^2 = 0.854
```



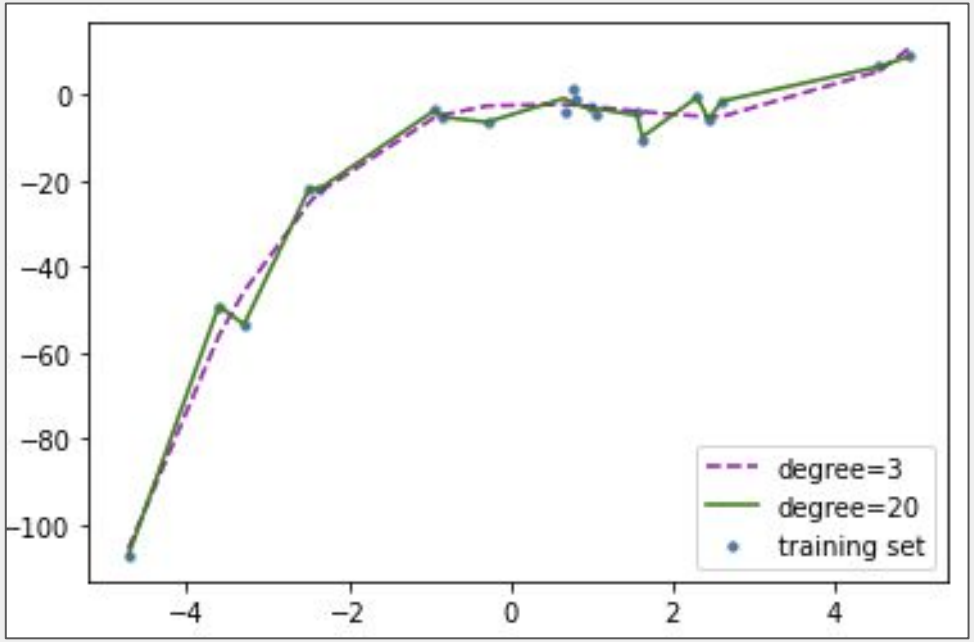

```
Polynomial Regression RMSE = 3.450
Polynomial Regression R^2 = 0.983
```



What if we increase the degree even more?

If we further increase the degree to 20, we can see that the curve passes through more data points.

For `degree=20`, the model is also capturing the noise in the data. This is an example of **over-fitting**. Even though this model passes through most of the data, it will fail to generalize on unseen data.




```
Polynomial Regression (degree = 20) RMSE = 1.113
Polynomial Regression (degree = 20) R^2 = 0.998
```

Remember, when the training set accuracy measure is close to 1, it indicates overfitting

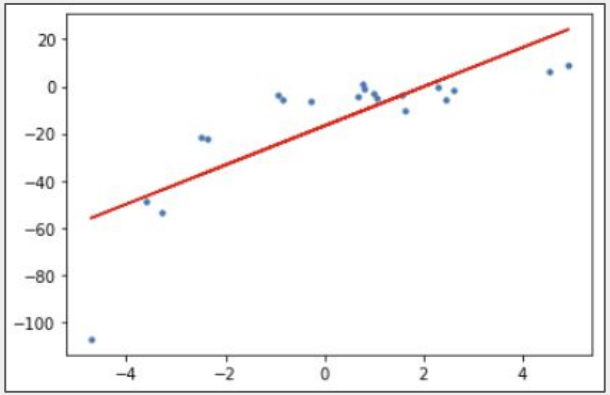
How do you choose the optimal model?

To answer this question we need to understand the **Bias-Variance Tradeoff**.

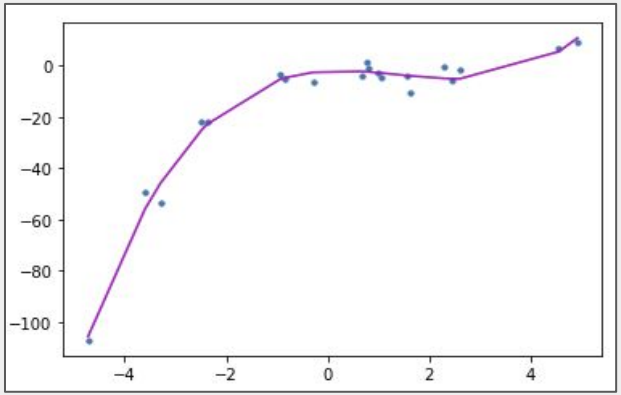
Bias happens due to the model's simplistic assumptions in fitting the data. A high bias means that the model is unable to capture the patterns in the data and this results in **under-fitting**.

Variance happens due to the complex model trying to fit the data. High variance means the model passes through most of the data points and it results in **over-fitting** the data.

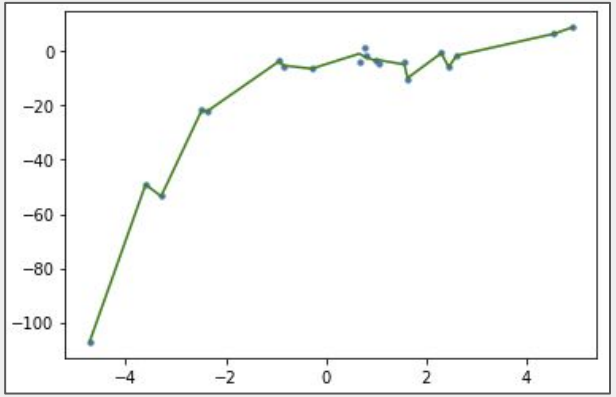
Underfit
= High Bias, Low Variance
= Overly "Simple" Model



Good fit
= Low Bias, Low Variance
= "Optimal" Model

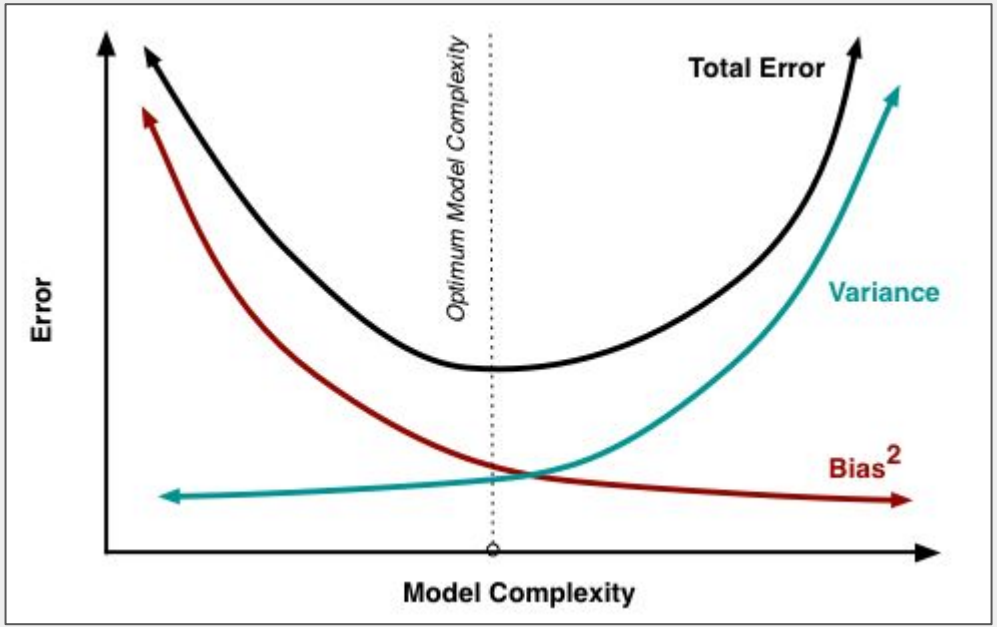


Overfit
= Low Bias, High Variance
= Overly "Complex" Model



Bias-Variance Tradeoff

- From the plot we can observe that as the model complexity increases, the bias decreases and the variance increases and vice-versa.
- Ideally, a machine learning model should have **low variance and low bias**. But practically it's impossible to have both.
- Therefore, to achieve a good model that performs well both on the train and unseen data, a **trade-off** is made.



How do we prevent overfitting?

1. **Add more training data** so that the ML algorithm doesn't learn the noise in the set, and can be more generalized.
    - Note that adding more data is "usually" a good thing, except when the additional training data is noisy by nature or doesn't match whatever you are trying to predict
2. **Remove features**
    - You can always improve a model's generalizability by removing irrelevant input features.
    - If using polynomial or logistic regression try reducing the degree of polynomial
    - An interesting way to do so is to tell a story about how each each feature fits into the model. If any feature doesn't make sense, or is hard to justify, then remove it.
    - Here are several feature selection heuristics that you can look at as a good starting point.

How do we prevent overfitting?

3. Apply regularization

- Regularization is a the concept of adding a “parameter penalty” in order to discourage learning a more complex or flexible model. By reducing a model’s complexity, you will avoid the risk of overfitting.

4. Cross Validation

- Cross Validation is a powerful preventative measure against overfitting.
- The idea is clever: use your initial training data to generate multiple mini train-test splits, and use these splits to train your model.
- It allows you to tune your parameters with only the original training set, and allows you to keep your test set as a truly unseen dataset for selecting your final optimal model.

How do we prevent underfitting?

1. **Add more features**
    - If using polynomial or logistic regression, try adding polynomial features.
2. **Reduce regularization**
    - If regularization is used, reduce the effect of the parameter penalty.

Let’s look at the code of a more complex polynomial regression…

Quick Review

Suppose that you have a dataset D1 and you design a linear regression model of degree 3 polynomial and you found that the training and testing error is “0” or in another terms it perfectly fits the data.

1- What will happen when you fit degree 4 polynomial in linear regression?
A) There is a high chance that a degree 4 polynomial will overfit the data
B) There is a high chance that a degree 4 polynomial will underfit the data
C) Can’t say
D) None of these

Quick Review

Suppose that you have a dataset D1 and you design a linear regression model of degree 3 polynomial and you found that the training and testing error is “0” or in another terms it perfectly fits the data.

1- What will happen when you fit degree 4 polynomial in linear regression?

A) There is a high chance that a degree 4 polynomial will overfit the data
B) There is a high chance that a degree 4 polynomial will underfit the data
C) Can’t say
D) None of these

If a degree 3 polynomial fits the data perfectly, it’s highly likely that a more complex model (degree 4 polynomial) might overfit the data.

2- What will happen when you fit degree 2 polynomial in linear regression?

A) There is a high chance that a degree 2 polynomial will overfit the data
B) There is a high chance that a degree 2 polynomial will underfit the data
C) Can’t say
D) None of these

Quick Review

Suppose that you have a dataset D1 and you design a linear regression model of degree 3 polynomial and you found that the training and testing error is “0” or in another terms it perfectly fits the data.

1- What will happen when you fit degree 4 polynomial in linear regression?
A) There is a high chance that a degree 4 polynomial will overfit the data
B) There is a high chance that a degree 4 polynomial will underfit the data
C) Can’t say
D) None of these

If a degree 3 polynomial fits the data perfectly, it’s highly likely that a more complex model (degree 4 polynomial) might overfit the data.

2- What will happen when you fit degree 2 polynomial in linear regression?
A) There is a high chance that a degree 2 polynomial will overfit the data
B) There is a high chance that a degree 2 polynomial will underfit the data
C) Can’t say
D) None of these

If a degree 3 polynomial fits the data perfectly, it’s highly likely that a simpler model (degree 2 polynomial) might under fit the data.

3- In terms of bias and variance. Which of the following is true when you fit degree 2 polynomial?
A) Bias will be high, variance will be high
B) Bias will be low, variance will be high
C) Bias will be high, variance will be low
D) Bias will be low, variance will be low

Quick Review

Suppose that you have a dataset D1 and you design a linear regression model of degree 3 polynomial and you found that the training and testing error is “0” or in another terms it perfectly fits the data.

1- What will happen when you fit degree 4 polynomial in linear regression?
A) There is a high chance that a degree 4 polynomial will overfit the data
B) There is a high chance that a degree 4 polynomial will underfit the data
C) Can’t say
D) None of these

If a degree 3 polynomial fits the data perfectly, it’s highly likely that a more complex model (degree 4 polynomial) might overfit the data.

2- What will happen when you fit degree 2 polynomial in linear regression?
A) There is a high chance that a degree 2 polynomial will overfit the data
B) There is a high chance that a degree 2 polynomial will underfit the data
C) Can’t say
D) None of these

If a degree 3 polynomial fits the data perfectly, it’s highly likely that a simpler model (degree 2 polynomial) might under fit the data.

3- In terms of bias and variance. Which of the following is true when you fit degree 2 polynomial?
A) Bias will be high, variance will be high
B) Bias will be low, variance will be high
C) Bias will be high, variance will be low
D) Bias will be low, variance will be low

Since a degree 2 polynomial will be less complex as compared to degree 3, the bias will be high and variance will be low.