# ML Fundamentals

Instituto Balseiro

Instituto Balseiro
30/08/2022

# Solving conflicts, no editing

Relevant for conflicts with binaries:


```
$ git checkout --ours -- path/to/conflicted-file.txt
$ git checkout --theirs -- path/to/conflicted-file.txt
```

https://stackoverflow.com/questions/278081/resolving-a-git-conflict-with-binary-files

# Params/Hyperparams/Grid search

- **Parameters**: to be found when training, i.e., optimizing a Loss function so as to have low validation error, and (maybe) low test error.

- **Hyperparameters**: fixed quantities asociated with a particular model and modeling process. They do not change during training. When several, one looks for the best combination (lowest error).

- **Grid search**: one of many possible strategies to decide which hyperparameters assign to the final model. Semi-exhaustive search over range of hyperparameters.

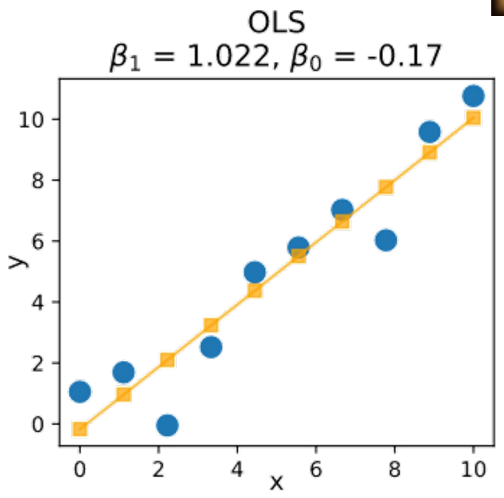

Lecture 5
Linear

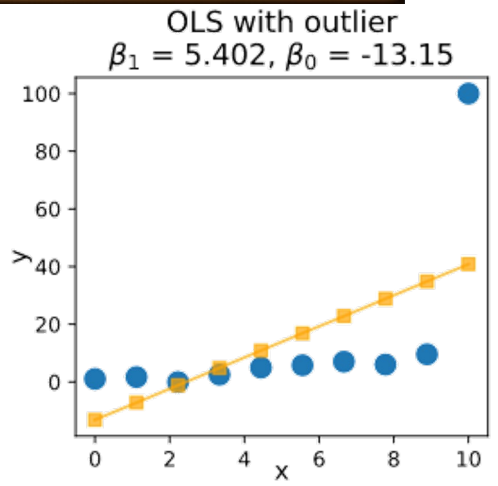
# Linear models

- Why linear models
- Linear regression
- Regularization
  - L1, L2, and related models
- Linear classification
  - Logistic regression
  - SVMs
- Non-parametric models
  - KNNs

# Why linear models?

- **PROS**
  - Interpretable
  - Fast
  - Basis of ANN/DNN
  - Expandable
  - Tractable
  - Very general

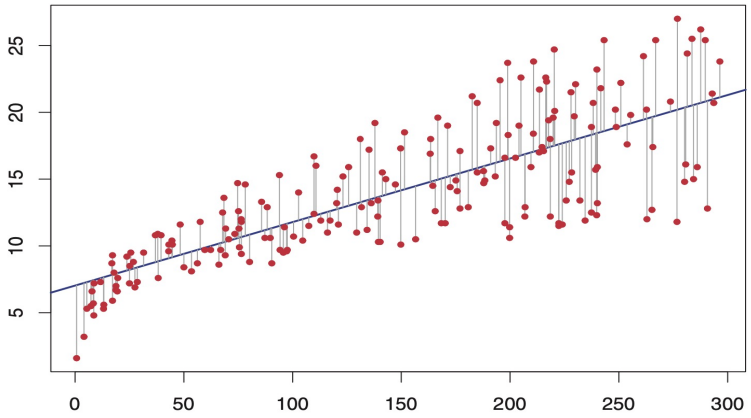


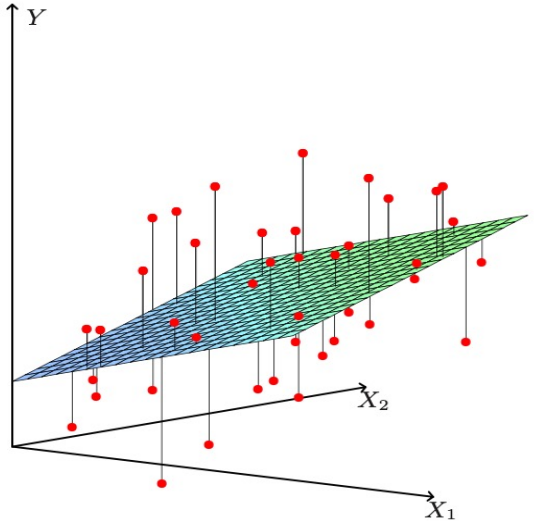




- **CONS**
  - Low complexity (large bias)
  - Sensitive to outliers
    - but regularization
  - Not general at all

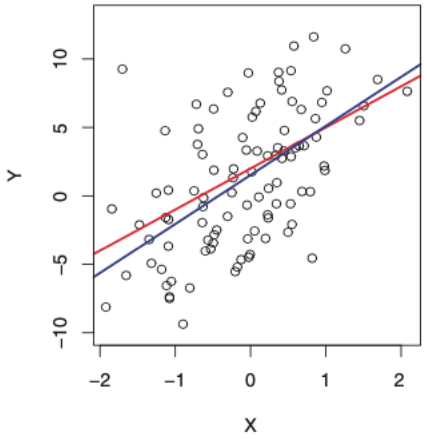
# Ordinary Least Squares (OLS)

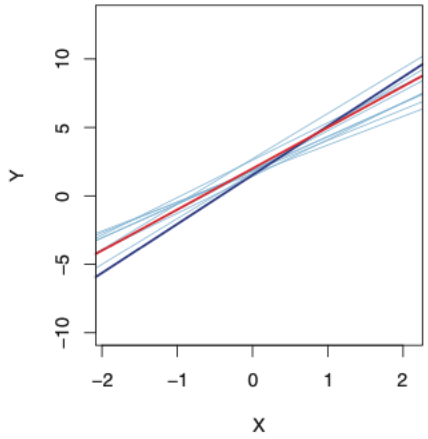
$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j. \qquad \mathrm{RSS}(\beta) = \sum_{i=1}^{N} (y_i - f(x_i))^2$$

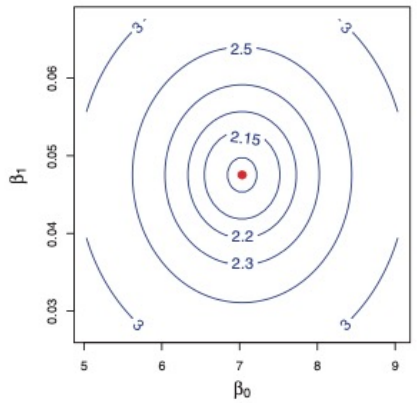


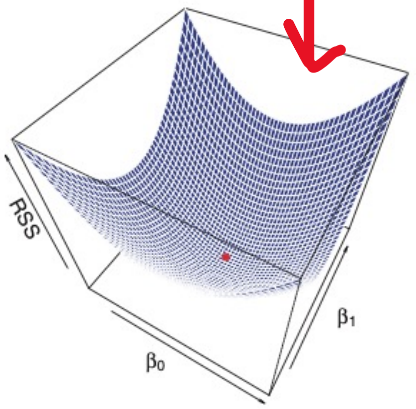




Dataset {xi}
dependant



# Performance metrics

- Mean Absolute Error (MAE)
- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE).
- $R^2$ statistic (determination coefficient)

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}|$$

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

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2}$$

$$R^2 = 1 - \frac{\sum(y_i - \hat{y})^2}{\sum(y_i - \bar{y})^2}$$

Where,

$\hat{y}$ – *predicted value of* $y$

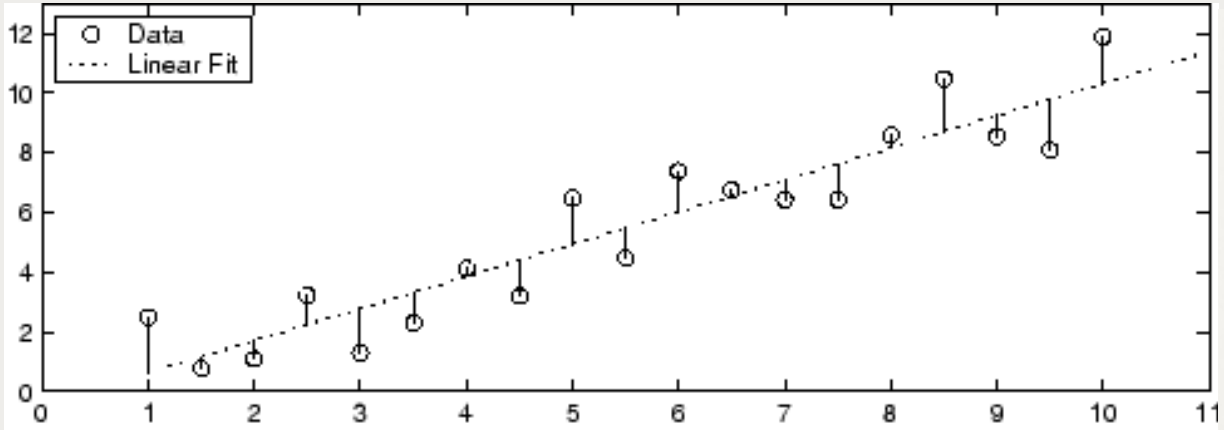
$\bar{y}$ – *mean value of* $y$

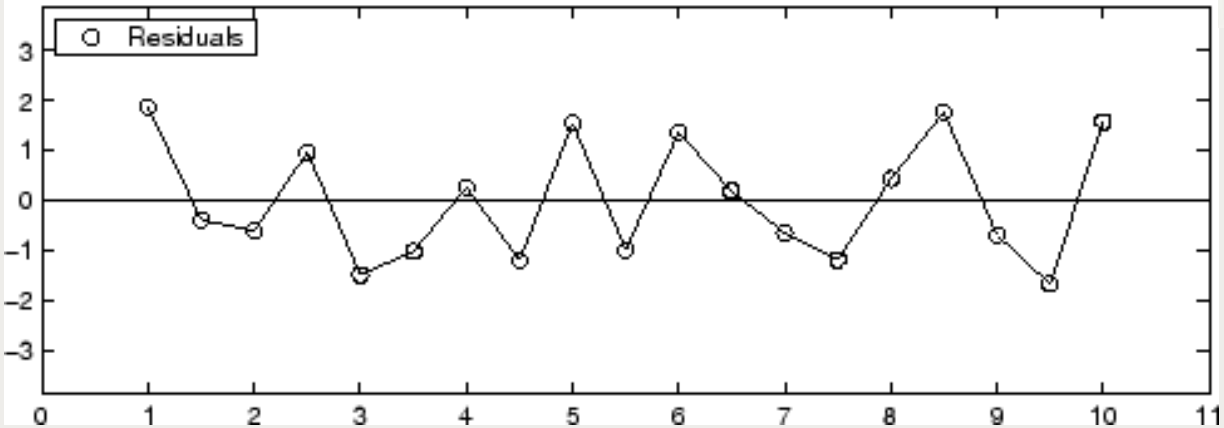
# Stats assumptions

Assumptions of linearity and multicollinearity

Assumptions of independence, homoscedasticity, and normality

$$\text{Response variable} = f(\text{Predicting variables}) + \text{Error term}$$




**Sandile Goje**
*Meeting of Two Cultures*
1993

Statistical Modeling: The Two Cultures Leo Breiman (2001)

statsmodels

https://www.statsmodels.org/

# Normal Equations

$$f(X) = \beta_0 + \sum_{p} X_j \beta_j.$$

$$\begin{aligned} \mathrm{RSS}(\beta) &= \sum_{i=1}^{N} (y_i - f(x_i))^2 \\ &= \sum_{i=1}^{N} \Big( y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \Big)^2. \end{aligned}$$

Loss function

$$\mathrm{RSS}(\beta) = (\mathbf{y} - \mathbf{X}\beta)^T (\mathbf{y} - \mathbf{X}\beta).$$

$$\frac{\partial \mathrm{RSS}}{\partial \beta} = -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta)$$

$$\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0$$

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}.$$

2013 - Book - Hastie - The Elements of Statistical Learning= Data Mining, Inference, and Prediction.pdf

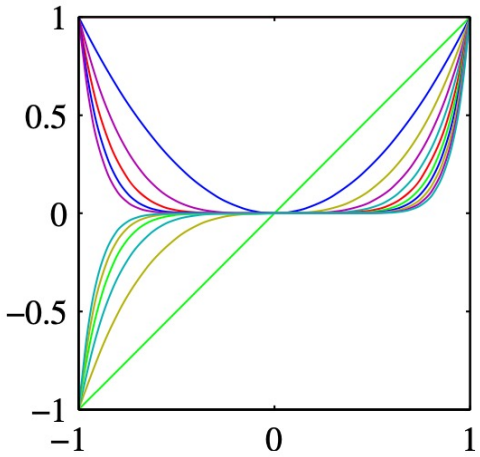
# Linear Basis Function Models

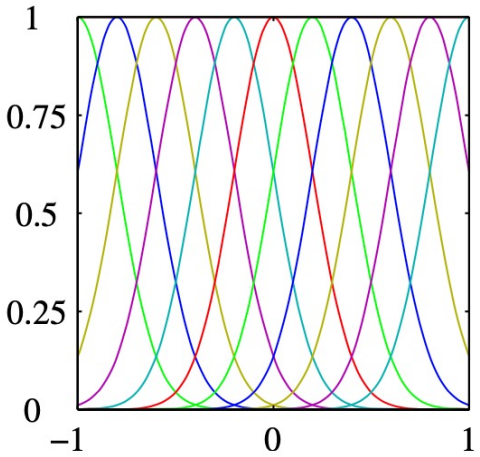
$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \ldots + w_D x_D$$

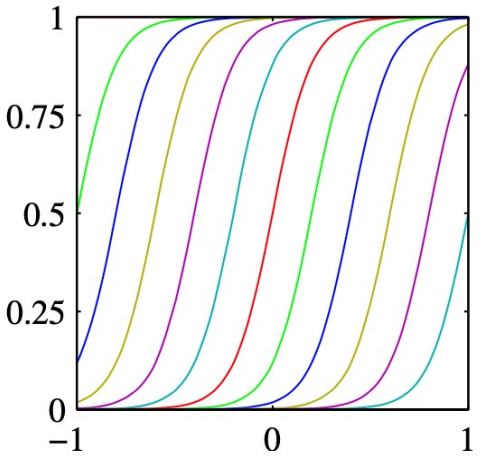
Linear ***in the parameters***

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x})$$

$$\phi_j(x) = \exp\left\{-\frac{(x-\mu_j)^2}{2s^2}\right\} \qquad \phi_j(x) = \sigma\left(\frac{x-\mu_j}{s}\right)$$



# Linear regression –Python implementation

*Ordinary least squares* (OLS)

In [26]:
```
from sklearn.linear_model import LinearRegression
X, y = mglearn.datasets.make_wave(n_samples=60)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

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

In [27]:
```
print("lr.coef_:", lr.coef_)
print("lr.intercept_:", lr.intercept_)
```

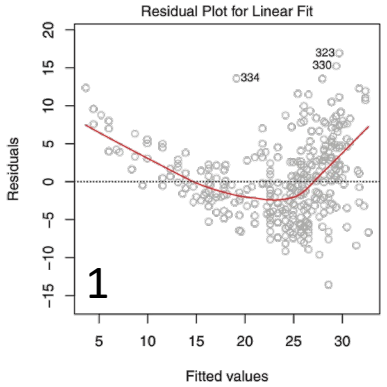
```
lr.coef_: [0.394]
lr.intercept_: -0.031804343026759746
```

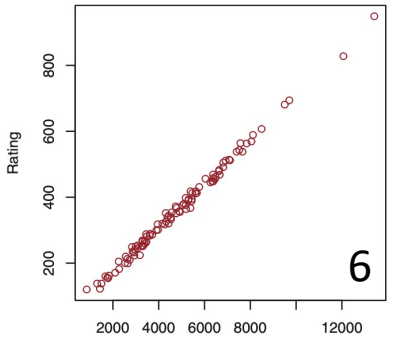
In [28]:
```
print("Training set score: {:.2f}".format(lr.score(X_train, y_train)))
print("Test set score: {:.2f}".format(lr.score(X_test, y_test)))
```

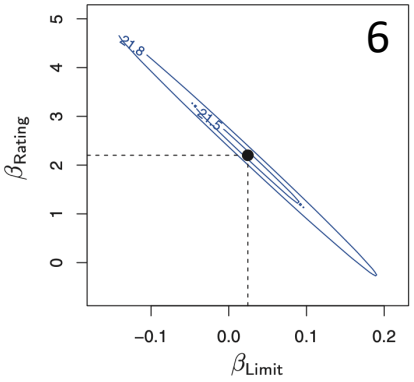
```
Training set score: 0.67
Test set score: 0.66
```

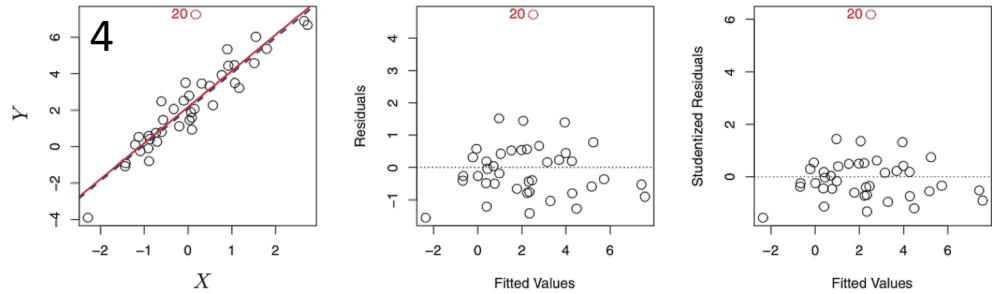
# Potential issues

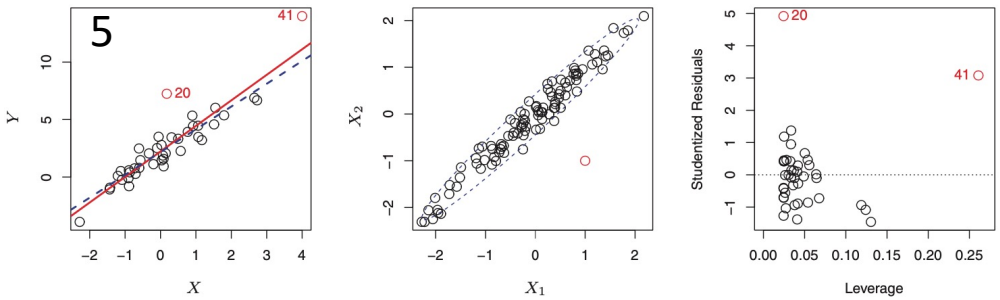
1. Non-linearity of the target-attribute relationships.
2. Correlation of error terms.
3. Non-constant variance of error terms.
4. Outliers.
5. High-leverage points.
6. Collinearity and more predictors than obervations

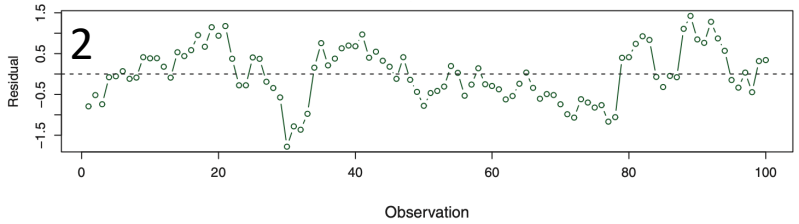


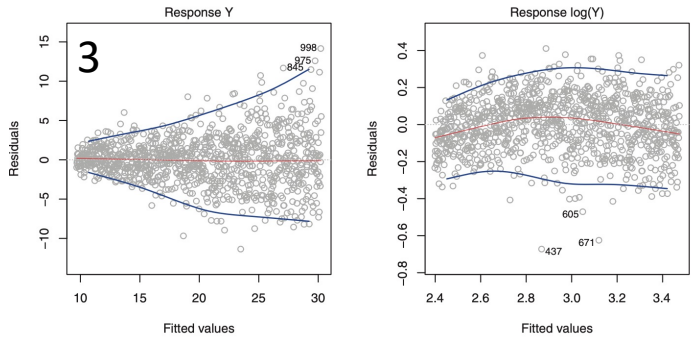



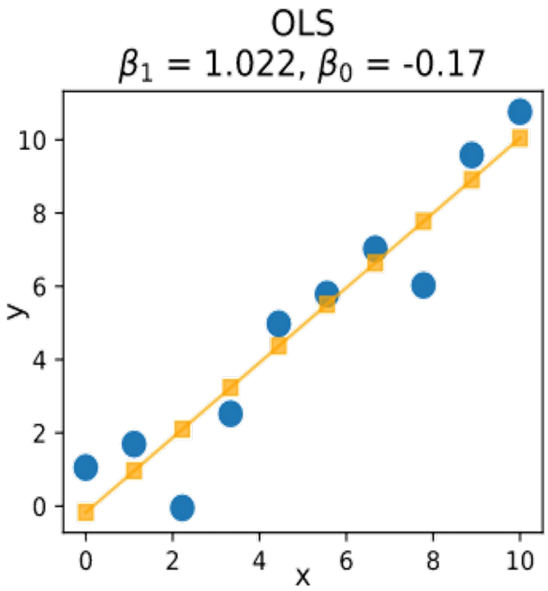



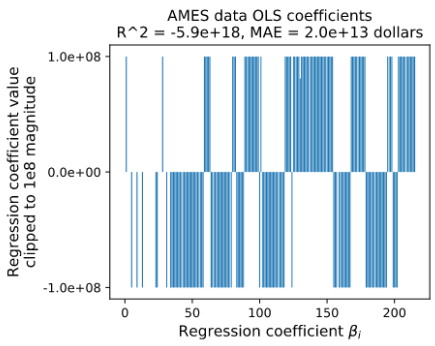



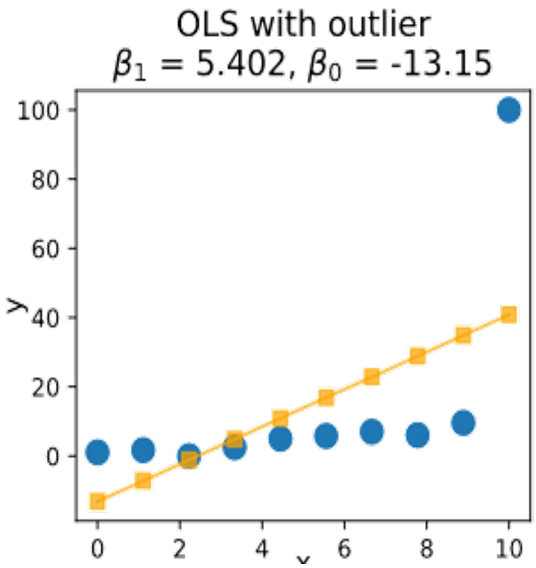



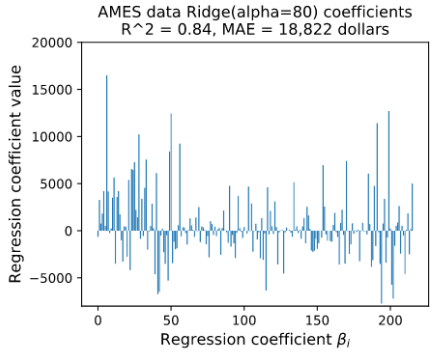



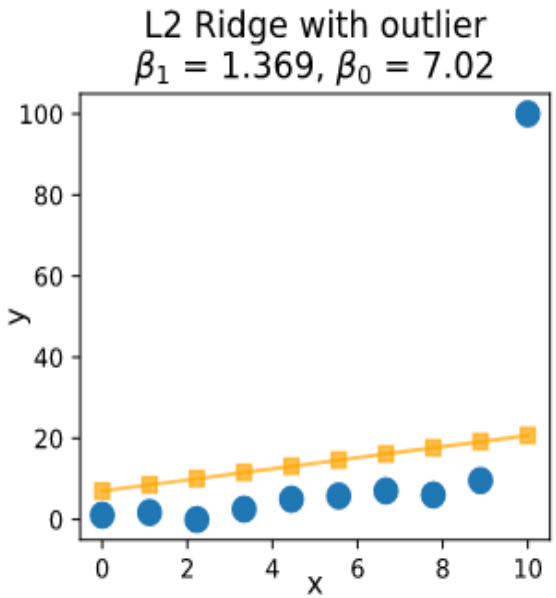
# Extreme coefficients are unlikely to yield models that generalize well.



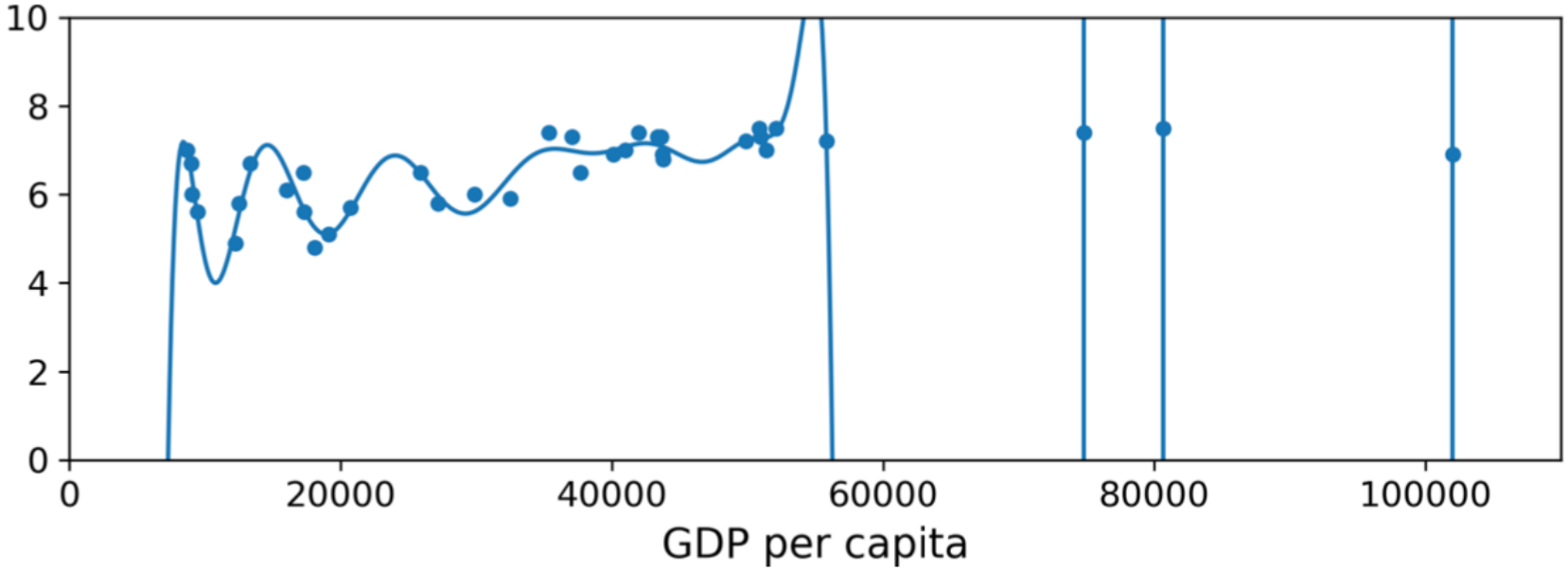



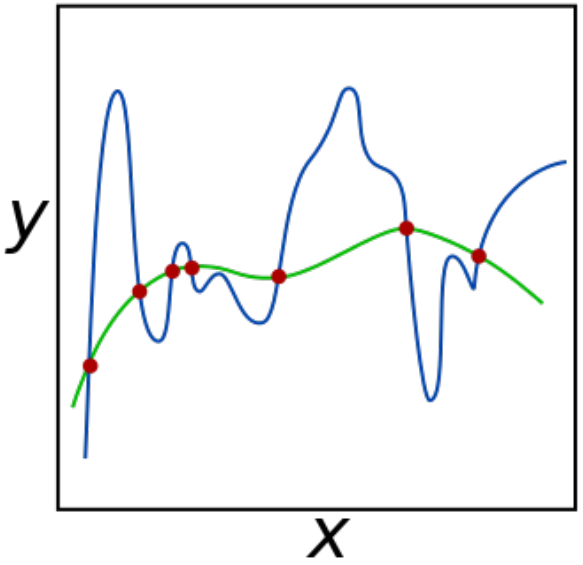




L2 regression

# Regularization



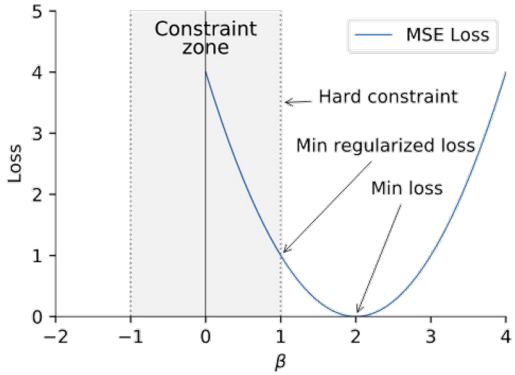

Strategies for restricting model parameters so as to minimize *overfitting risk*

*i.e. to improve the training/test error ratio*

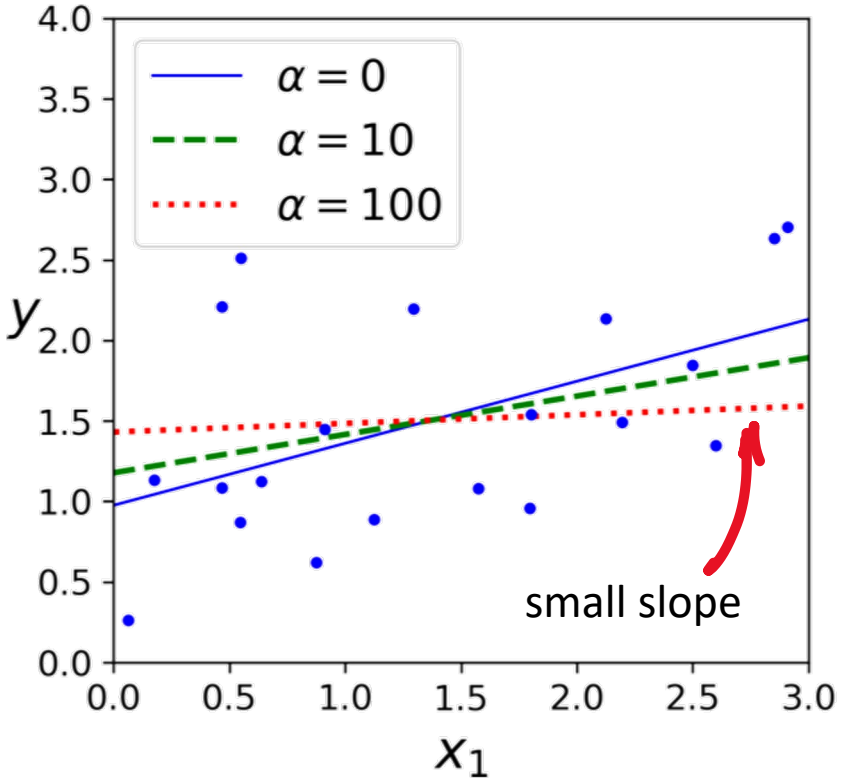
# Ridge or $L_2$ regression

*aka Tikhonov regularization*

*aka Weight decay in Deep Learning*




$$J(\boldsymbol{\theta}) = \text{MSE}(\boldsymbol{\theta}) + \alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$$




$$\hat{\beta}^{\text{ridge}} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{N} \left( y_i - \beta_0 - \sum_{j=1}^{p} x_{ij} \beta_j \right)^2,$$

$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 \leq t,$$

```
>>> from sklearn.linear_model import Ridge
>>> ridge_reg = Ridge(alpha=1, solver="cholesky")
>>> ridge_reg.fit(X, y)
>>> ridge_reg.predict([[1.5]])
array([[1.55071465]])
```

Or alternatively,

```
>>> sgd_reg = SGDRegressor(penalty="l2")
>>> sgd_reg.fit(X, y.ravel())
>>> sgd_reg.predict([[1.5]])
array([1.47012588])
```

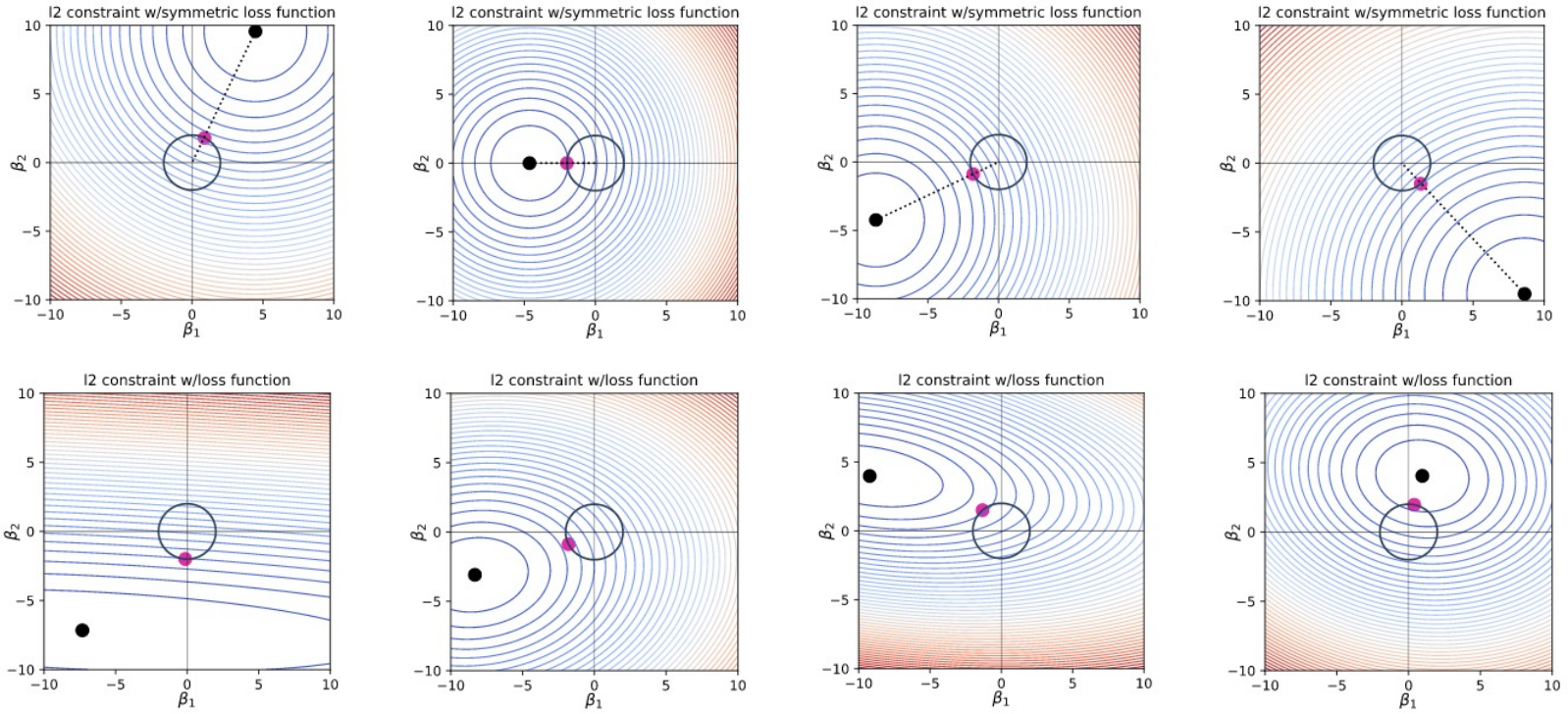
# $L_2$ Regularization

Soft constraint ('Lagrangian')

$$\hat{\beta}^{\text{ridge}} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{N} \left( y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2,$$

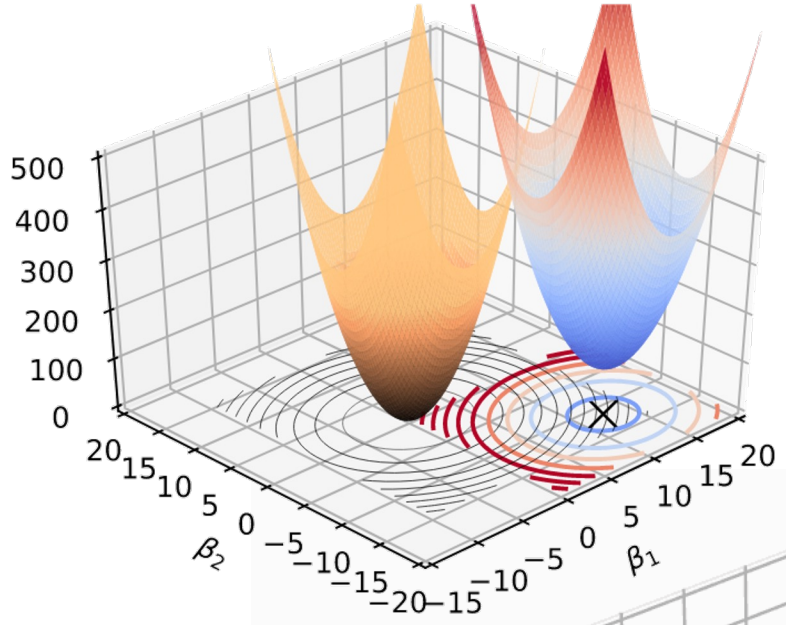
$$\hat{\beta}^{\text{ridge}} = \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^{N} \left( y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}.$$

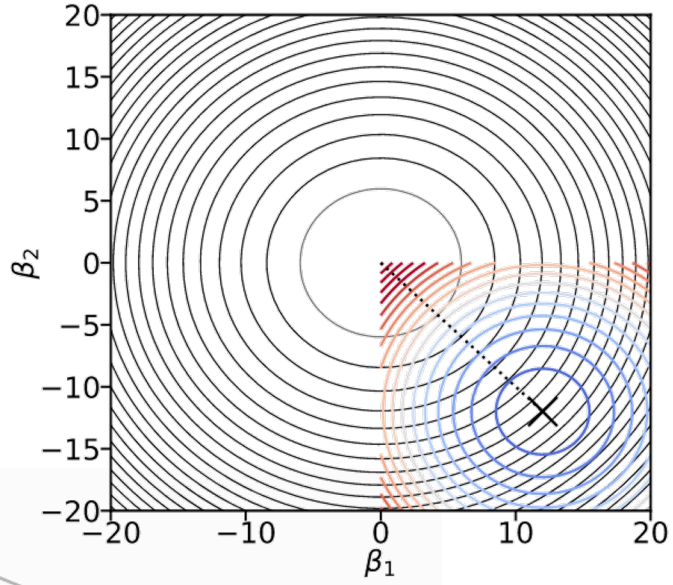
$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 \le t,$$ Hard constraint (Optimization)

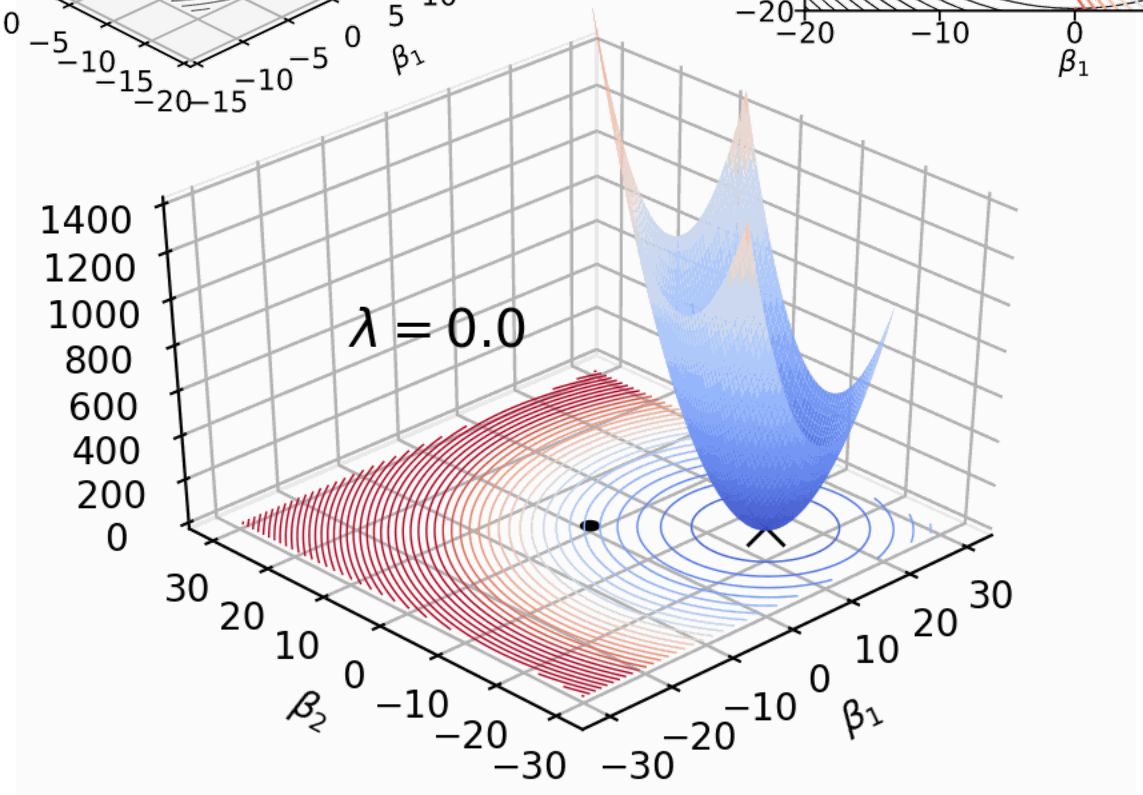



https://explained.ai/regularization/constraints.html

$$\hat{\beta}^{\text{ridge}} = \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^{N} \left( y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}.$$

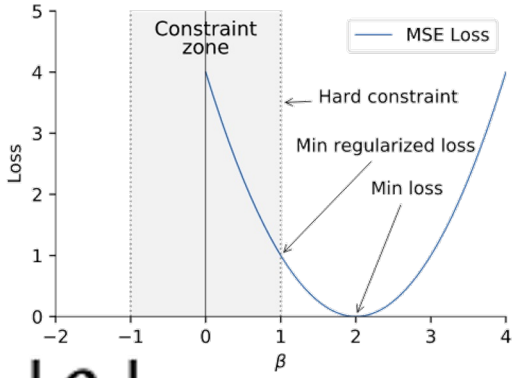


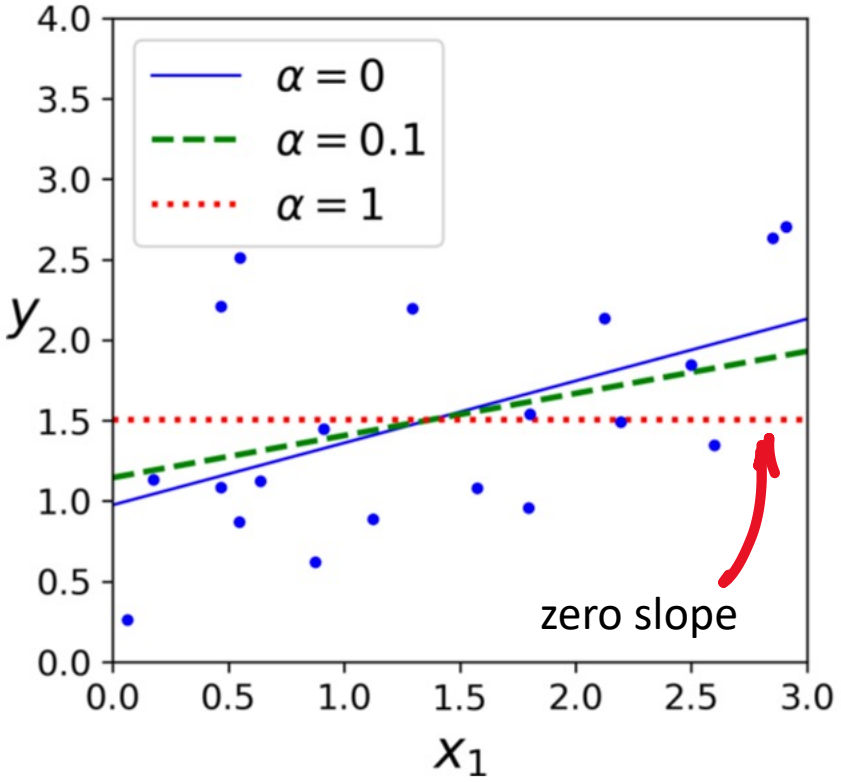



# LASSO or $L_1$ regression

*aka Least Absolute Shrinkage and Selection Operator Regression*




$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + \alpha \sum_{i=1}^{n} \left|\theta_i\right|$$




$$\hat{\beta}^{\text{lasso}} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2$$

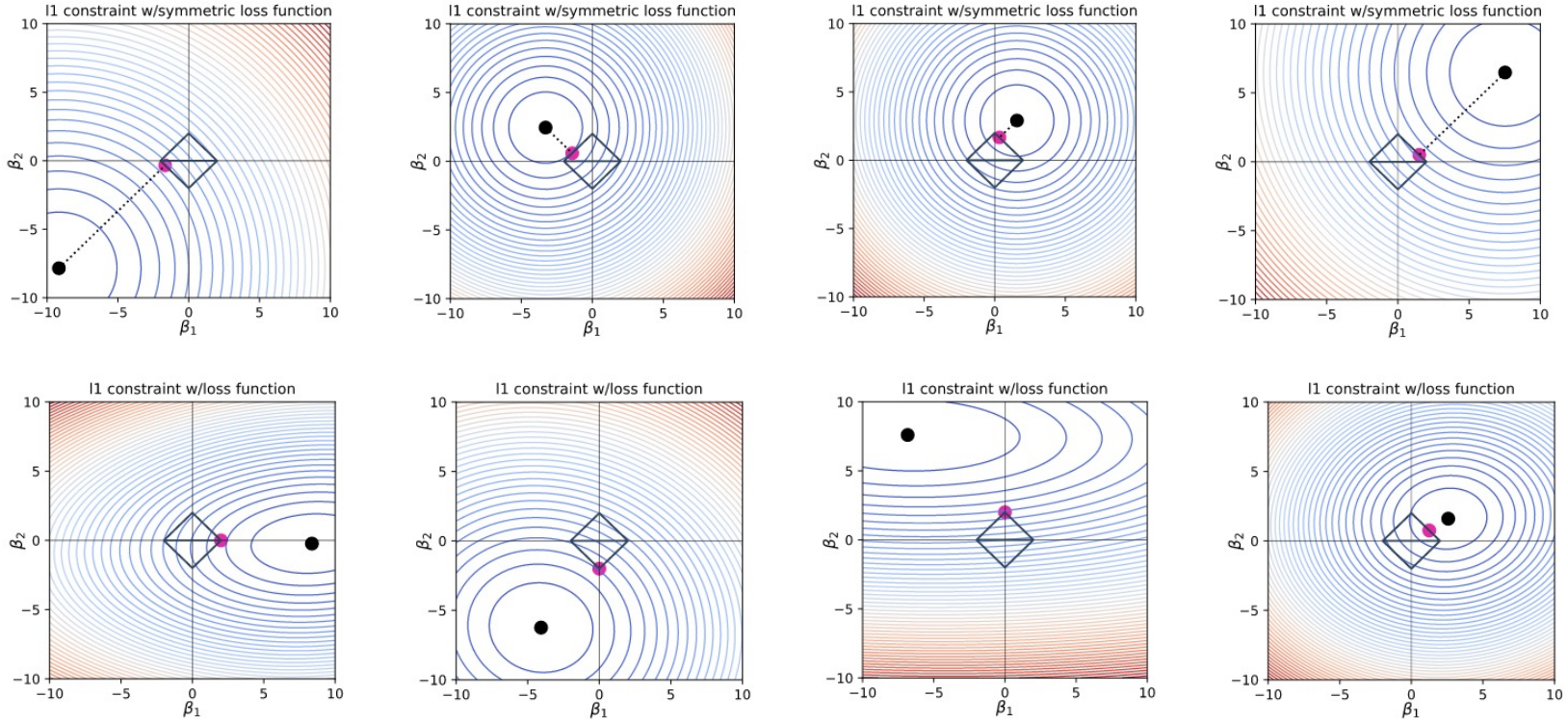
$$\text{subject to } \sum_{j=1}^{p} |\beta_j| \leq t.$$

```
>>> from sklearn.linear_model import Lasso
>>> lasso_reg = Lasso(alpha=0.1)
>>> lasso_reg.fit(X, y)
>>> lasso_reg.predict([[1.5]])
array([1.53788174])
```

# $L_1$ Regularization

$$\hat{\beta}^{\text{lasso}} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{N} \Big( y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \Big)^2 \quad \text{subject to} \sum_{j=1}^{p} |\beta_j| \le t.$$

$$\hat{\beta}^{\text{lasso}} = \underset{\beta}{\operatorname{argmin}} \left\{ \frac{1}{2} \sum_{i=1}^{N} \big( y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \big)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}.$$



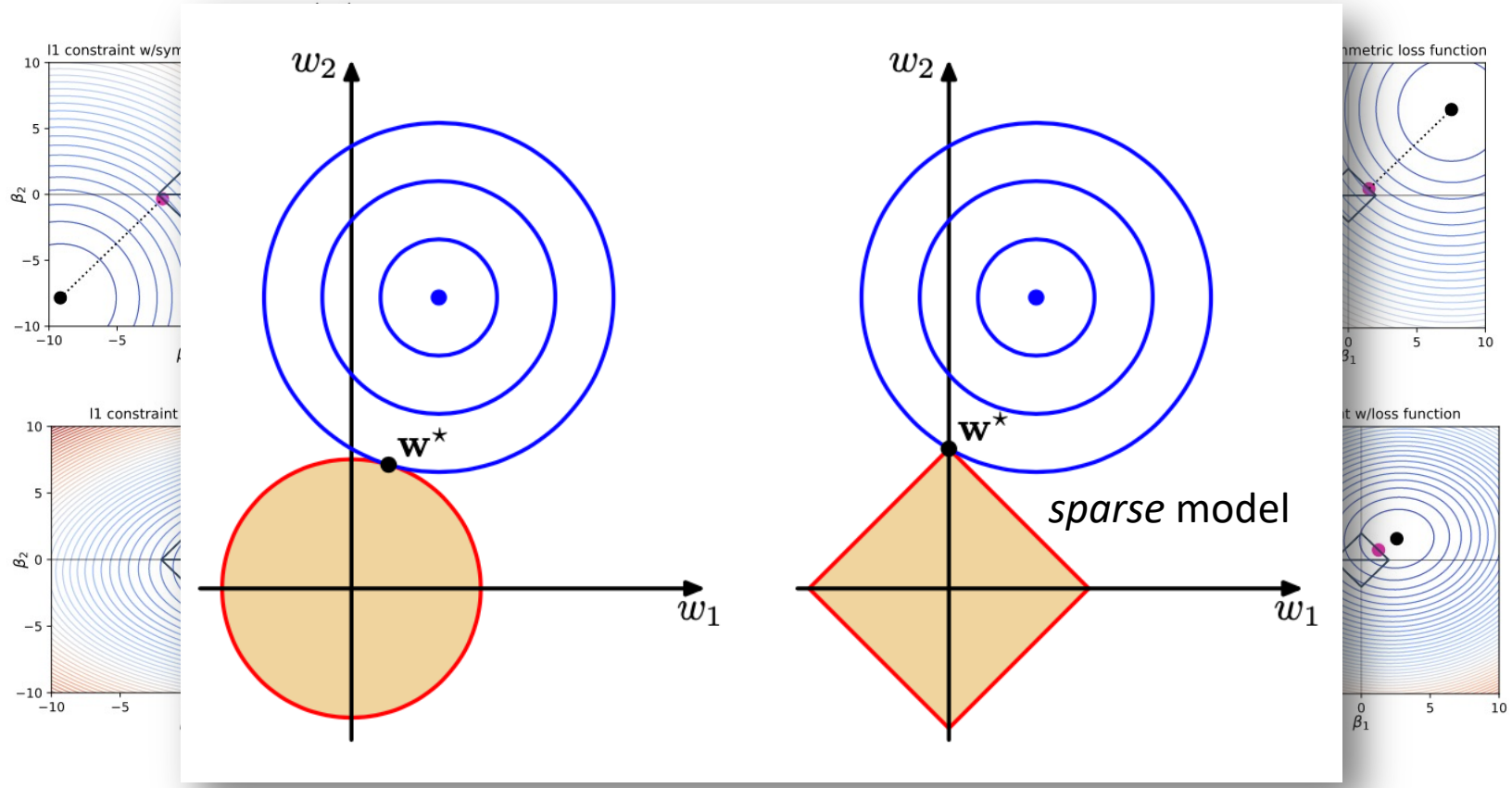

**L1 tends to give a lot more zero coefficients than L2, a form of *feature selection***

https://explained.ai/regularization/constraints.html

# $L_1$ Regularization

$$\hat{\beta}^{\text{lasso}} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{N} \Big( y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \Big)^2 \quad \text{subject to} \quad \sum_{j=1}^{p} |\beta_j| \le t.$$

$$\hat{\beta}^{\text{lasso}} = \underset{\beta}{\operatorname{argmin}} \left\{ \frac{1}{2} \sum_{i=1}^{N} \Big( y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \Big)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}.$$

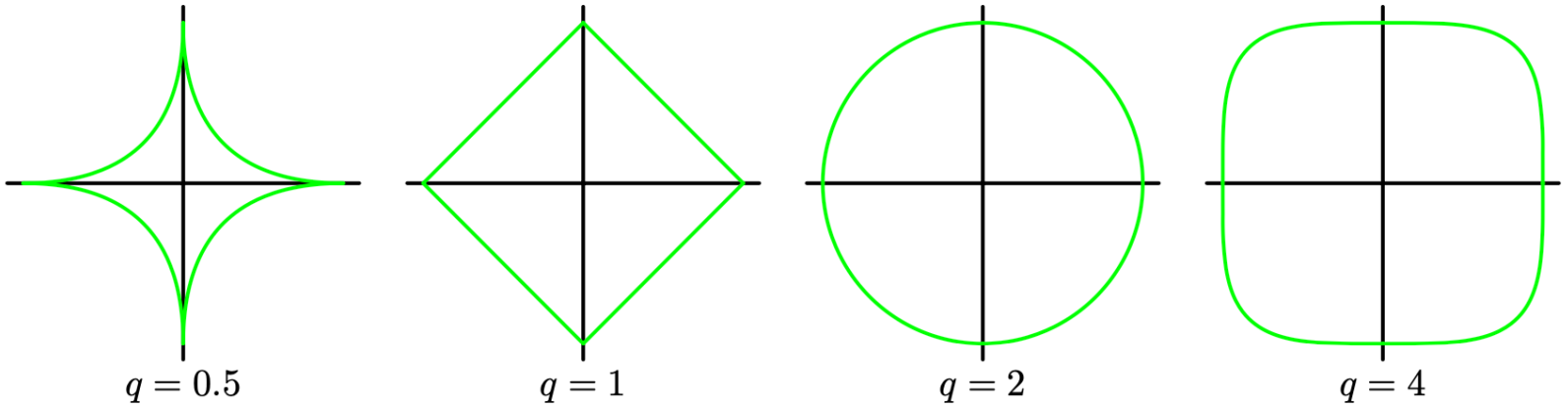



**L1 tends to give a lot more zero coefficients than L2, a form of *feature selection***

https://explained.ai/regularization/constraints.html

# Generalized regularization

$$\frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}^2 + \frac{\lambda}{2}\sum_{j=1}^{M}|w_j|^q$$



# Elastic net

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + r\alpha\Sigma_{i=1}^{n}\left|\theta_i\right| + \frac{1-r}{2}\alpha\Sigma_{i=1}^{n}\theta_i^2$$

r=1: lasso

r=0: ridge

```
>>> from sklearn.linear_model import ElasticNet
>>> elastic_net = ElasticNet(alpha=0.1, l1_ratio=0.5)
>>> elastic_net.fit(X, y)
>>> elastic_net.predict([[1.5]])
array([1.54333232])
```

From Géron's:

Least Squares < Lasso < Elastic Net < Ridge

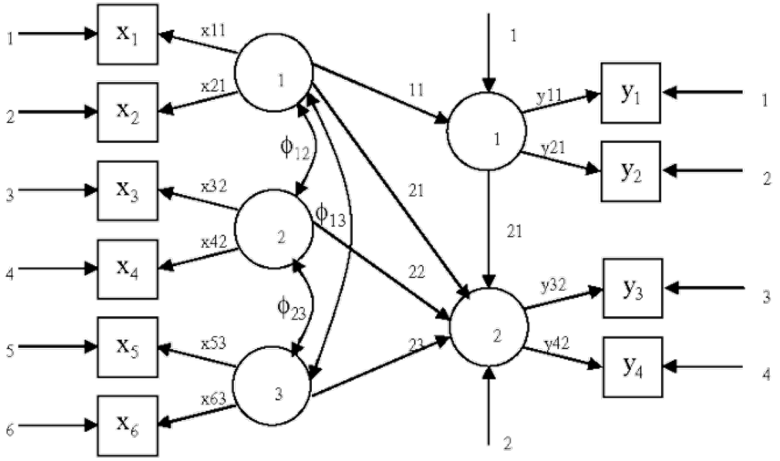
# More feature selection

Leave features out according to a given criteria:

- **Forward selection**
  - Start with one feature and add one by one, measuring some form of error and stopping with some rule
- **Backward selection**
  - Start with every feature and remove one by one, stopping with some rule
- **Mixed selection**
  - Add one by one, but remove according to some rule

# Other linear regression models

- Principal component regression (PCR)
  - Compute PCA reduced dimensions, throw some away
  - Regress with those dimensions and back
  - apt for multicollinearity
- Partial Least Squares (PLS)
  - adequate for more predictors than observations
  - also for multicollinearity



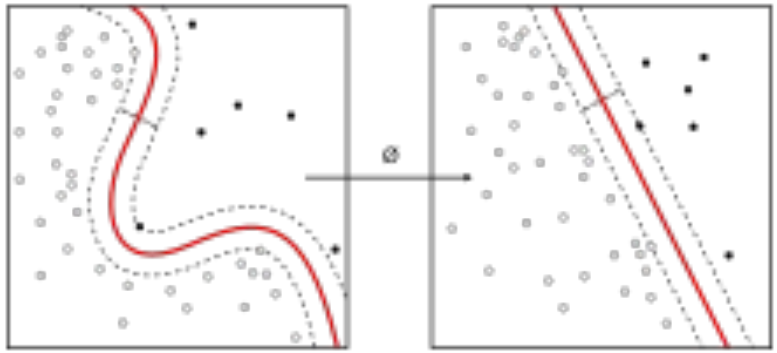

2004 - Haenlein - A Beginner's Guide to Partial Least Squares Analysis.pdf

# Linear classification

$$y(\mathbf{x}) = f\left(\mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0\right)$$

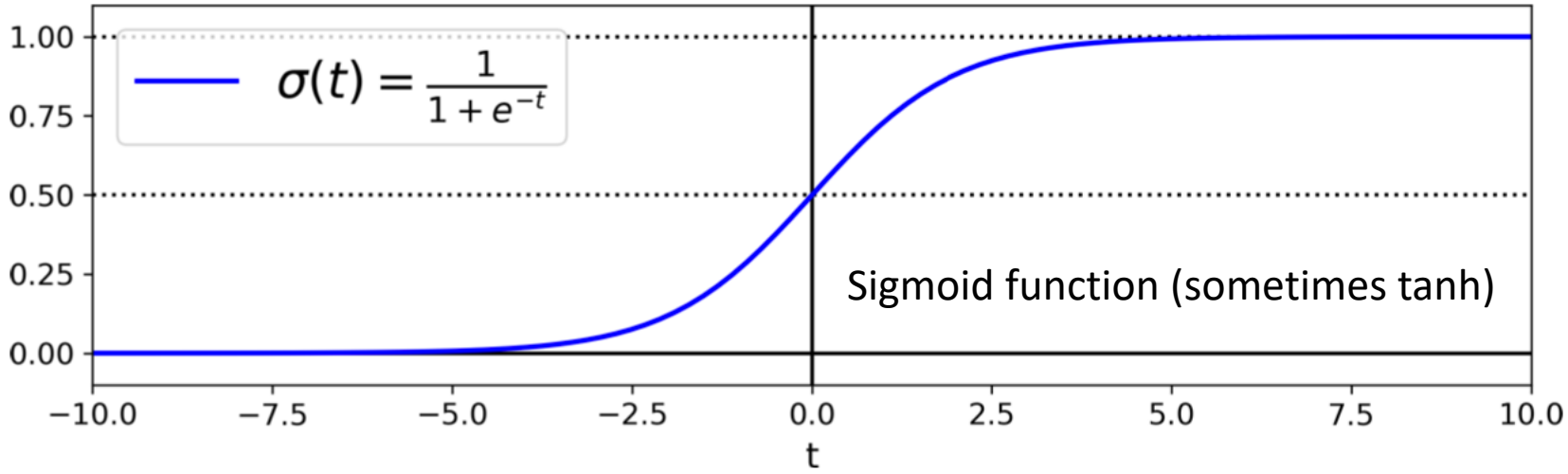
$$y(\mathbf{x}) = \text{constant}$$

$$\mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0 = \text{constant}$$



- Logistic regression (*!*)
  - Interpretable
  - Easy to implement
- Support vector machines (SVMs)
  - Can also do regression
  - with a little hot sauce: the kernel trick

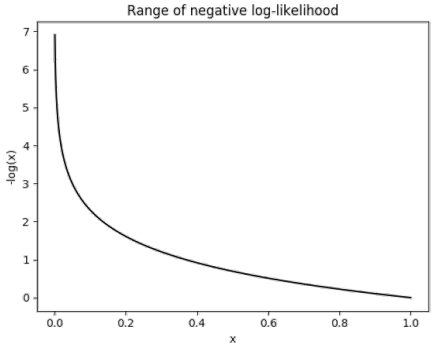
# Logistic regression




$$\hat{p} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma\left(\mathbf{x}^T\boldsymbol{\theta}\right) \longleftarrow \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n.$$

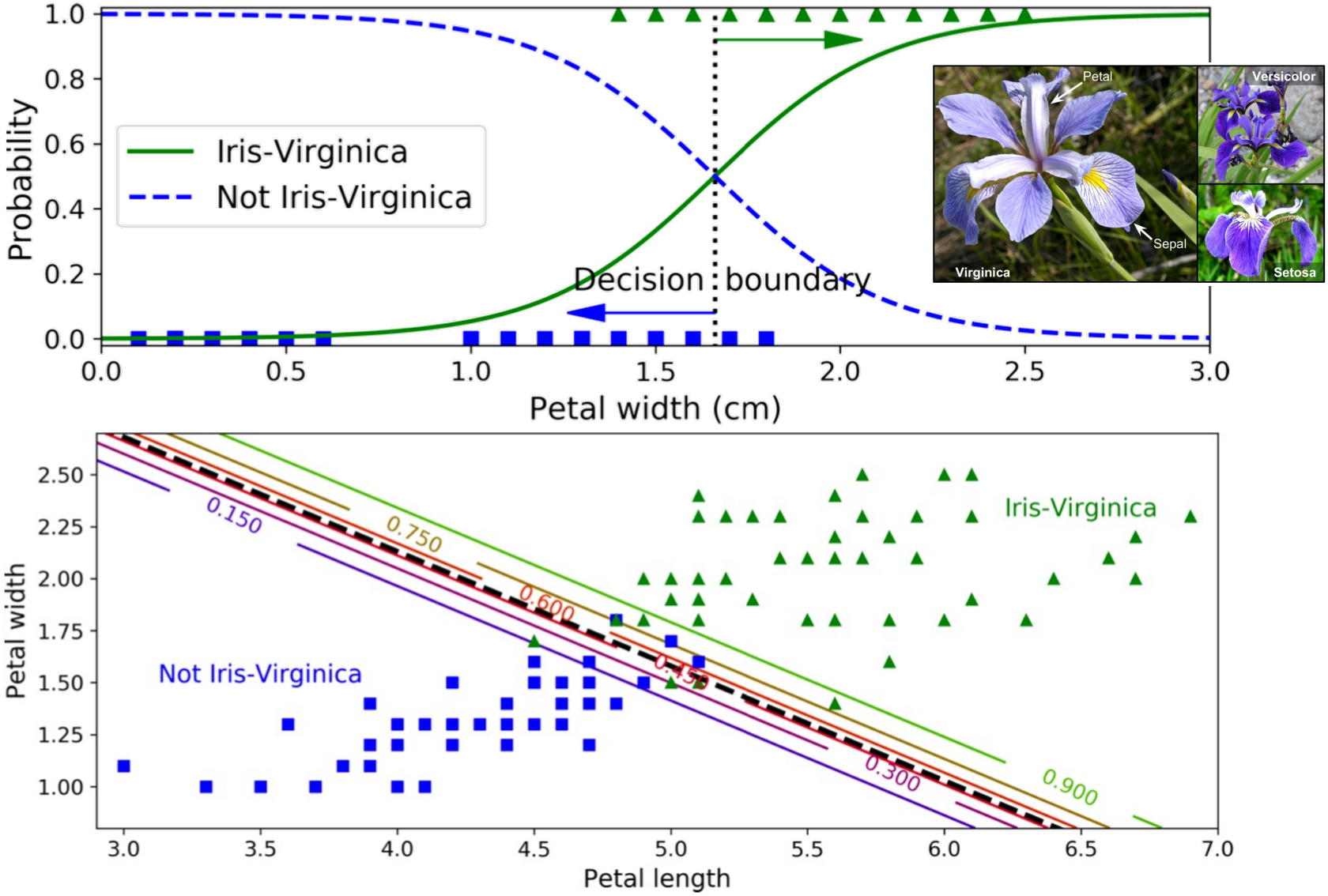
$$\hat{y} = \begin{cases} 0 \text{ if } \hat{p} < 0.5 \\ 1 \text{ if } \hat{p} \geq 0.5 \end{cases}$$

$$c(\boldsymbol{\theta}) = \begin{cases} -\log\left(\hat{p}\right) & \text{if } y = 1 \\ -\log\left(1-\hat{p}\right) & \text{if } y = 0 \end{cases}$$




Convex => SGD $$J(\boldsymbol{\theta}) = -\frac{1}{m}\Sigma_{i=1}^{m}\left[y^{(i)}log\left(\hat{p}^{(i)}\right) + \left(1-y^{(i)}\right)log\left(1-\hat{p}^{(i)}\right)\right]$$

# Logistic regression



# Logistic regression

```
>>> from sklearn import datasets
>>> iris = datasets.load_iris()
>>> list(iris.keys())
['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename']
>>> X = iris["data"][:, 3:]  # petal width
>>> y = (iris["target"] == 2).astype(np.int)  # 1 if Iris-Virginica, else 0
```

```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression()
log_reg.fit(X, y)

X_new = np.linspace(0, 3, 1000).reshape(-1, 1)
y_proba = log_reg.predict_proba(X_new)
plt.plot(X_new, y_proba[:, 1], "g-", label="Iris-Virginica")
plt.plot(X_new, y_proba[:, 0], "b--", label="Not Iris-Virginica")
# + more Matplotlib code to make the image look pretty
```

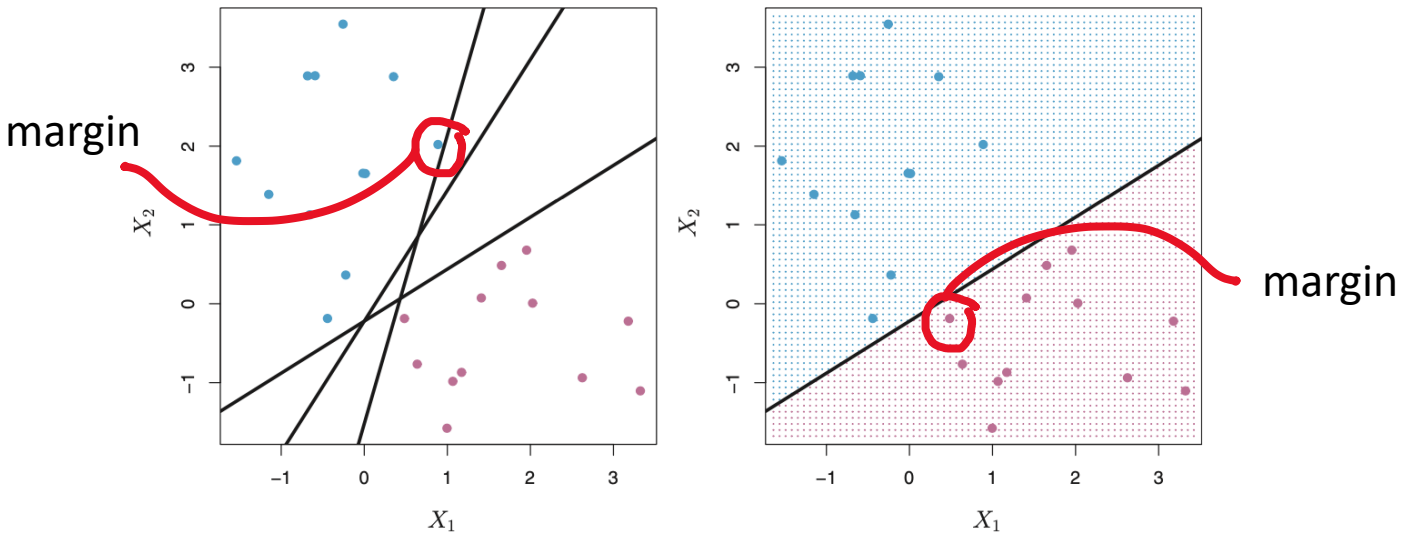
```
>>> log_reg.predict([[1.7], [1.5]])
array([1, 0])
```

# Support Vector Machines

- SVMs are a robust method from 90's (until GBM)
- *Based on the Maximal margin classifier*

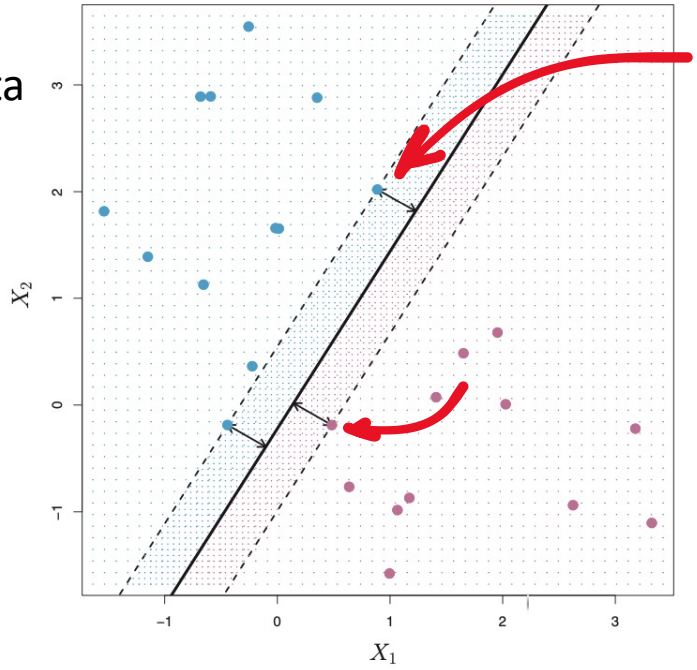
p-dimensional hyperplane:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p = 0$$



# Support vector classifier (SVC)

SVCs find the widest *strip* between data points (*large margin classification*)



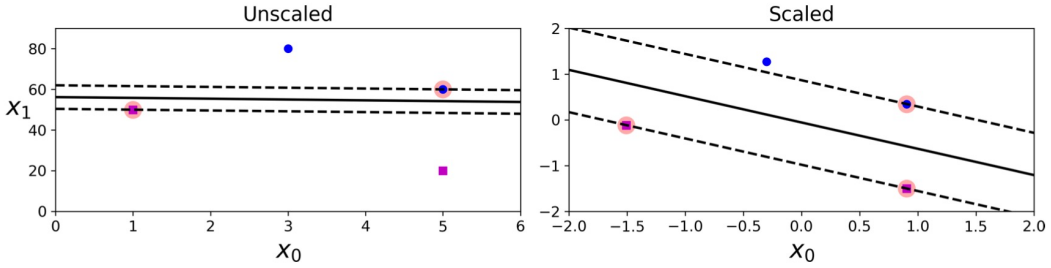

$$x_1, \ldots, x_n \in \mathbb{R}^p$$
$$y_1, \ldots, y_n \in \{-1, 1\}.$$

$$\underset{\beta_0, \beta_1, \ldots, \beta_p}{\text{maximize}}\, M$$

$$\text{subject to} \sum_{j=1}^{p} \beta_j^2 = 1,$$

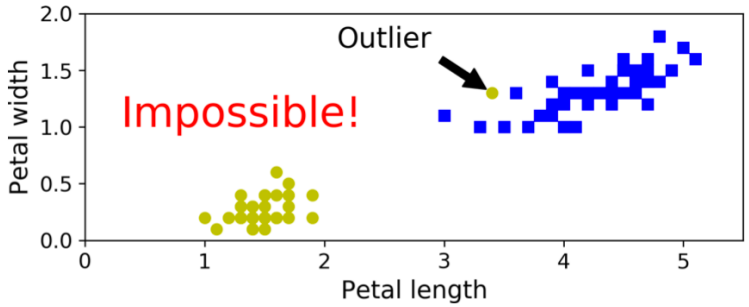
$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M \;\; \forall\, i = 1, \ldots, n.$$

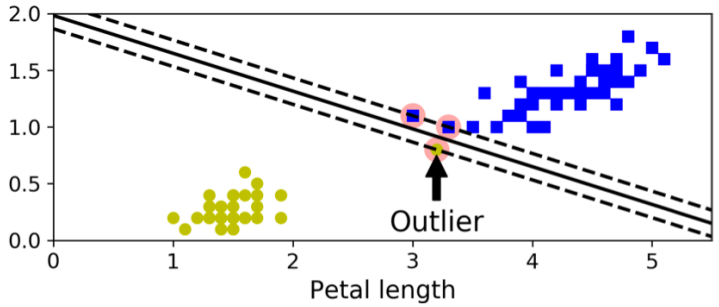
SVCs are **scale sensitive:**



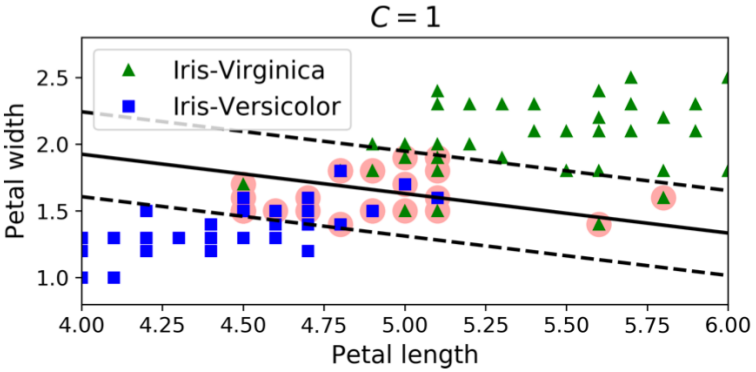
# Support vector classifier (SVC)

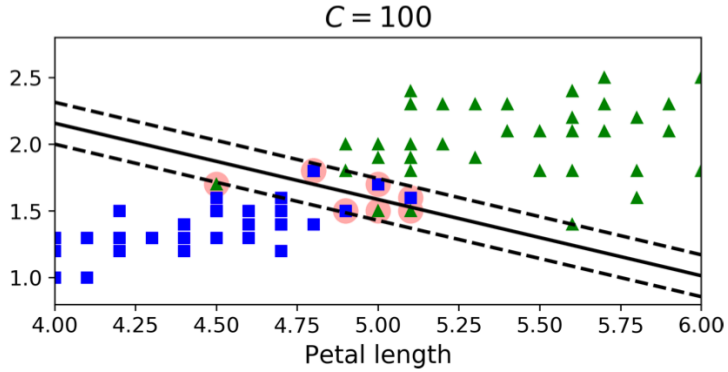
## Soft margin: training errors (FP, FN & the usual suspects)








$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n}{\text{maximize}} \quad M$$

$$\text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1 - \epsilon_i),$$

$$\epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C,$$

# Support vector classifier (SVC)

```
import numpy as np
from sklearn import datasets
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC

iris = datasets.load_iris()
X = iris["data"][:, (2, 3)]  # petal length, petal width
y = (iris["target"] == 2).astype(np.float64)  # Iris-Virginica

svm_clf = Pipeline([
        ("scaler", StandardScaler()),
        ("linear_svc", LinearSVC(C=1, loss="hinge")),
    ])

svm_clf.fit(X, y)
```

```
>>> svm_clf.predict([[5.5, 1.7]])
array([1.])
```

# SVM = SVC + Kernel trick


$$\boldsymbol{w}^\top \boldsymbol{x} + b = b + \sum_{i=1}^{m} \alpha_i \boldsymbol{x}^\top \boldsymbol{x}^{(i)}$$

$$k(\boldsymbol{x}, \boldsymbol{x}^{(i)}) = \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{x}^{(i)})$$

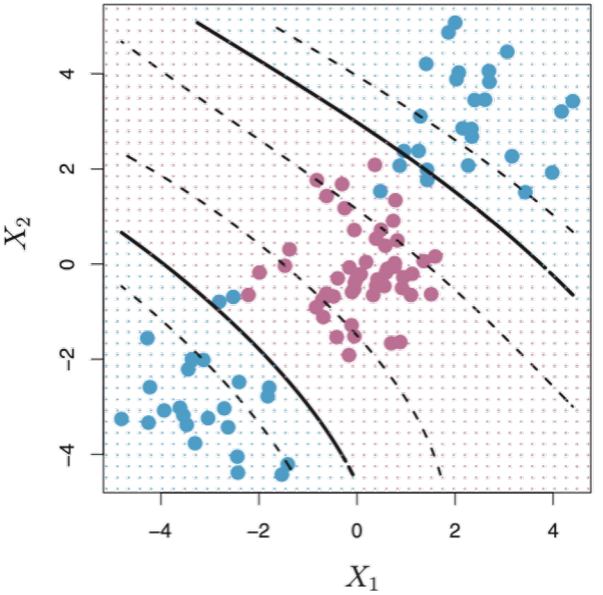
$$f(\boldsymbol{x}) = b + \sum_i \alpha_i k(\boldsymbol{x}, \boldsymbol{x}^{(i)}).$$

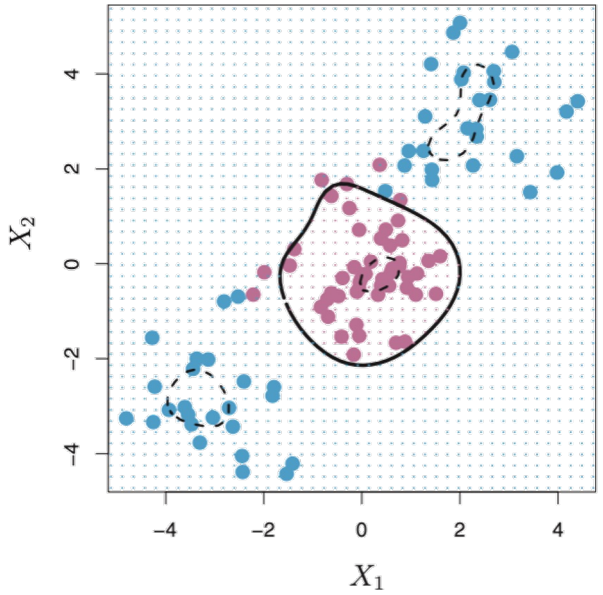
$$k(\mathbf{x}, \mathbf{x}') = \tanh\left(a\mathbf{x}^{\mathrm{T}}\mathbf{x}' + b\right)$$

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\|\mathbf{x} - \mathbf{x}'\|^2 / 2\sigma^2\right)$$

$$k(\mathbf{x}, \mathbf{z}) = \left(\mathbf{x}^{\mathrm{T}}\mathbf{z}\right)^2.$$

Also, multiclass SVMs, regression SVMs



# Many more linear classification models

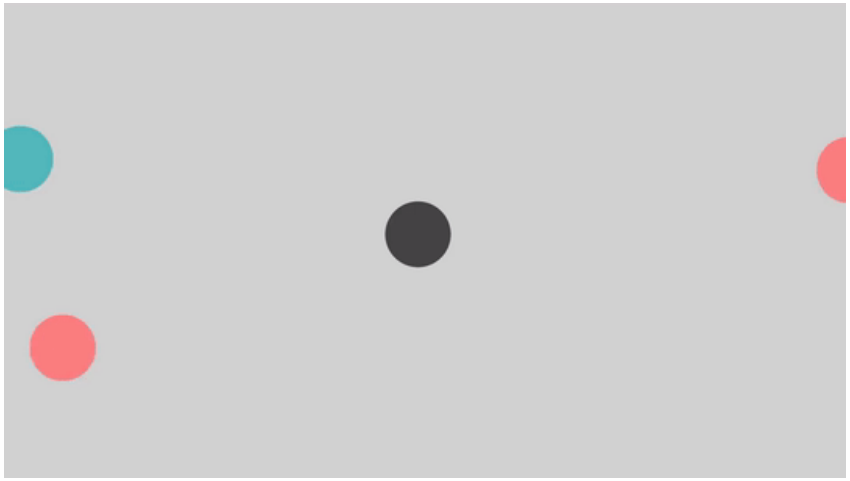
- Linear discriminant analysis (LDA)
- Mixture Discriminant Analysis
- Naive Bayes
- Perceptron
- etc.

# Non-parametric models

- Parametric models assume some relationship between attributes and target, and then train over that relationship
  - linear models, artificial neural networks

- Non-parametric models do not make explicit assumptions on the relationship, they derive it directly from data
  - knn, decision trees, SVMs, gradient boosting machines

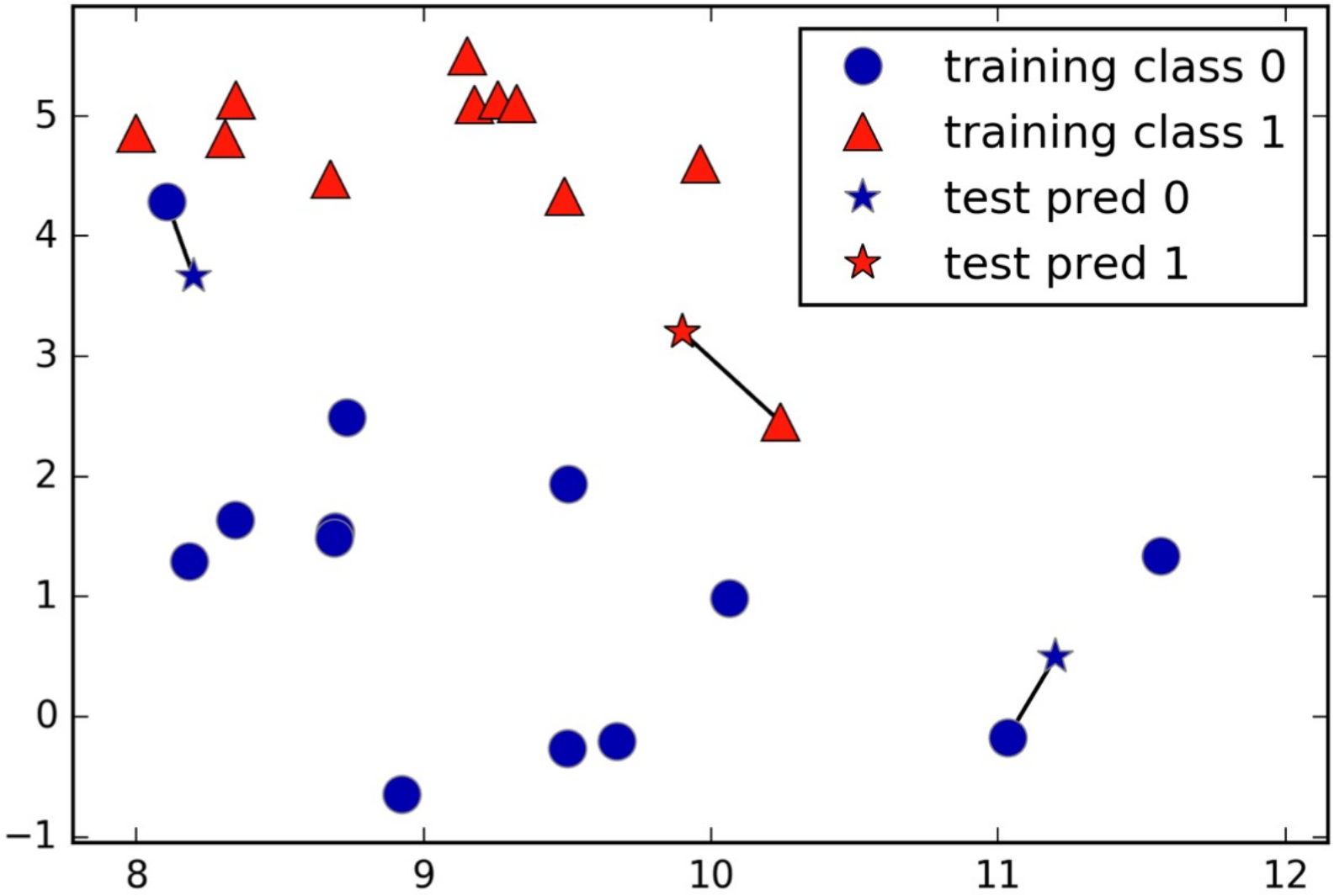
# KNNs



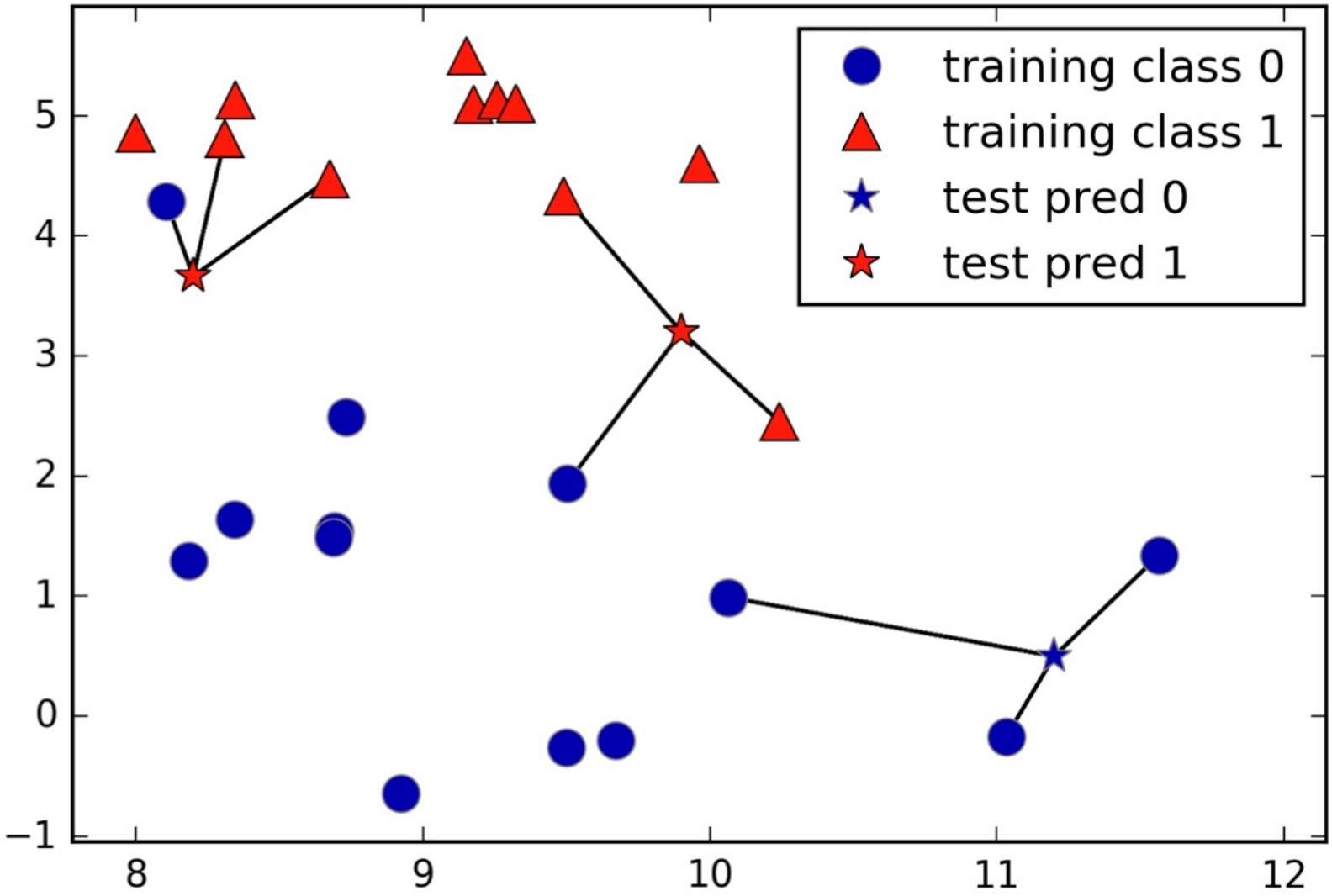
- Learn from k neighbours (supervised)
  - distance/metric
- Instance based learning
- Non-parametric
- May be expensive

...Where is the training? 🤔

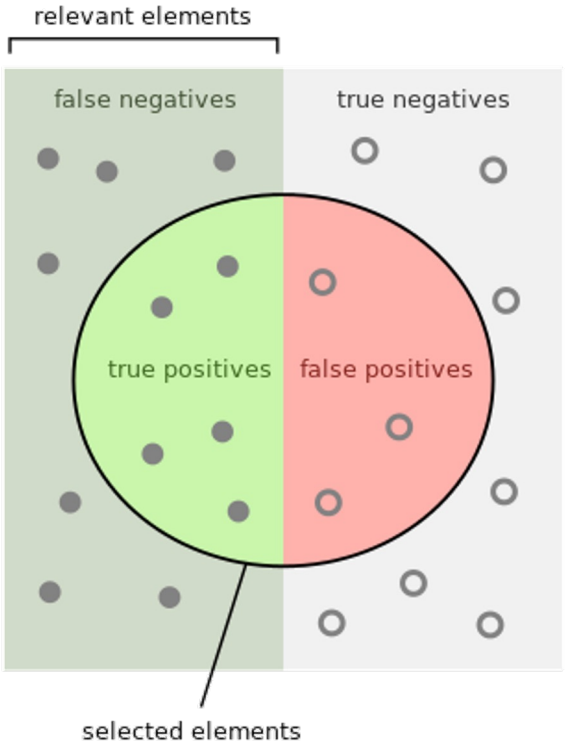
# 1-NN Classification



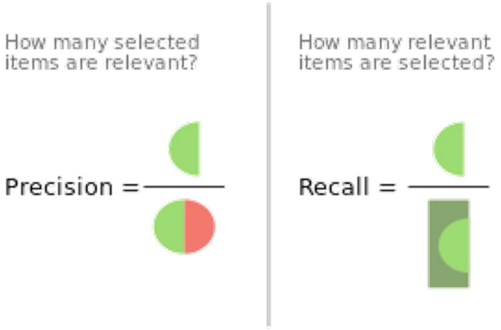
# 3-NN Classification



# The usual suspects


| | | **Predicted** class + | **Predicted** class - |
|---|---|---|---|
| **Actual** class | + | **TP**<br>True Positives | **FN**<br>False Negatives<br>Type II error |
| | - | **FP**<br>False Positives<br>Type I error | **TN**<br>True Negatives |




| Metric | Formula | Interpretation |
|---|---|---|
| Accuracy | $\frac{TP + TN}{TP + TN + FP + FN}$ | Overall performance of model |
| Precision | $\frac{TP}{TP + FP}$ | How accurate the positive predictions are |
| Recall<br>Sensitivity | $\frac{TP}{TP + FN}$ | Coverage of actual positive sample |
| Specificity | $\frac{TN}{TN + FP}$ | Coverage of actual negative sample |
| F1 score | $\frac{2TP}{2TP + FP + FN}$ | Hybrid metric useful for unbalanced classes |

# 3-NN

In [12]:
```
from sklearn.model_selection import train_test_split
X, y = mglearn.datasets.make_forge()

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

In [13]:
```
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=3)
```

In [14]:
```
clf.fit(X_train, y_train)
```

Out[14]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=3, p=2,
           weights='uniform')
```

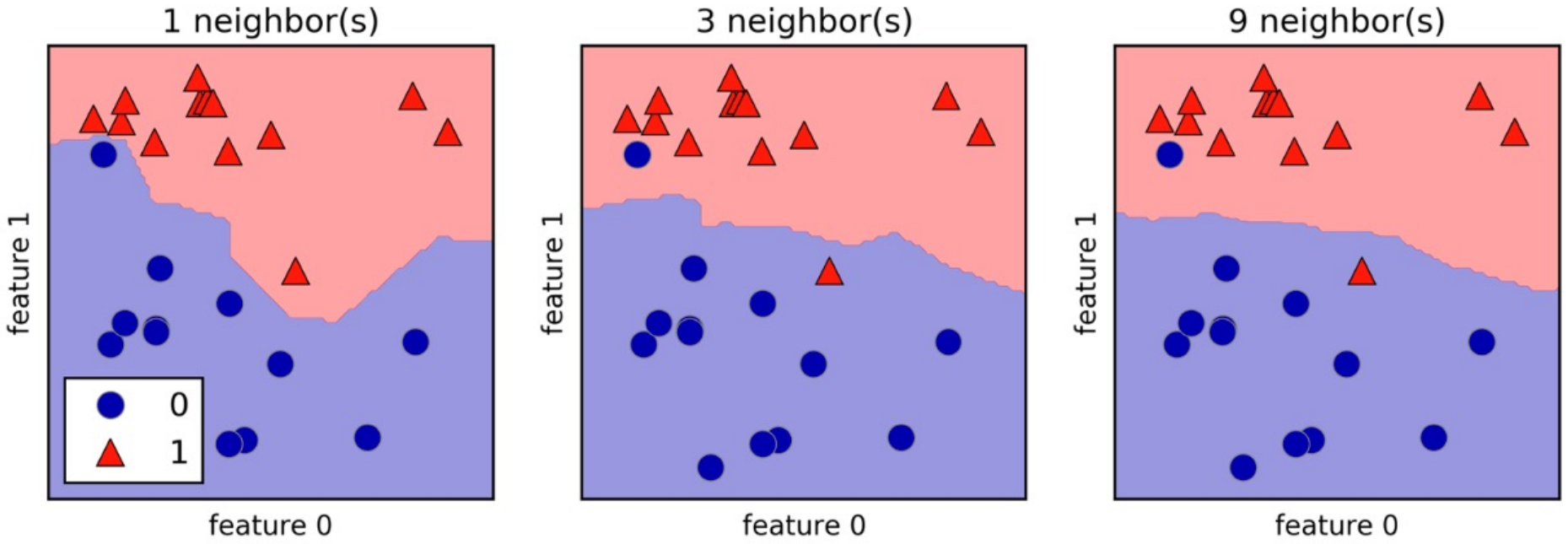
In [15]:
```
print("Test set predictions:", clf.predict(X_test))
```

```
Test set predictions: [1 0 1 0 1 0 0]
```

In [16]:
```
print("Test set accuracy: {:.2f}".format(clf.score(X_test, y_test)))
```

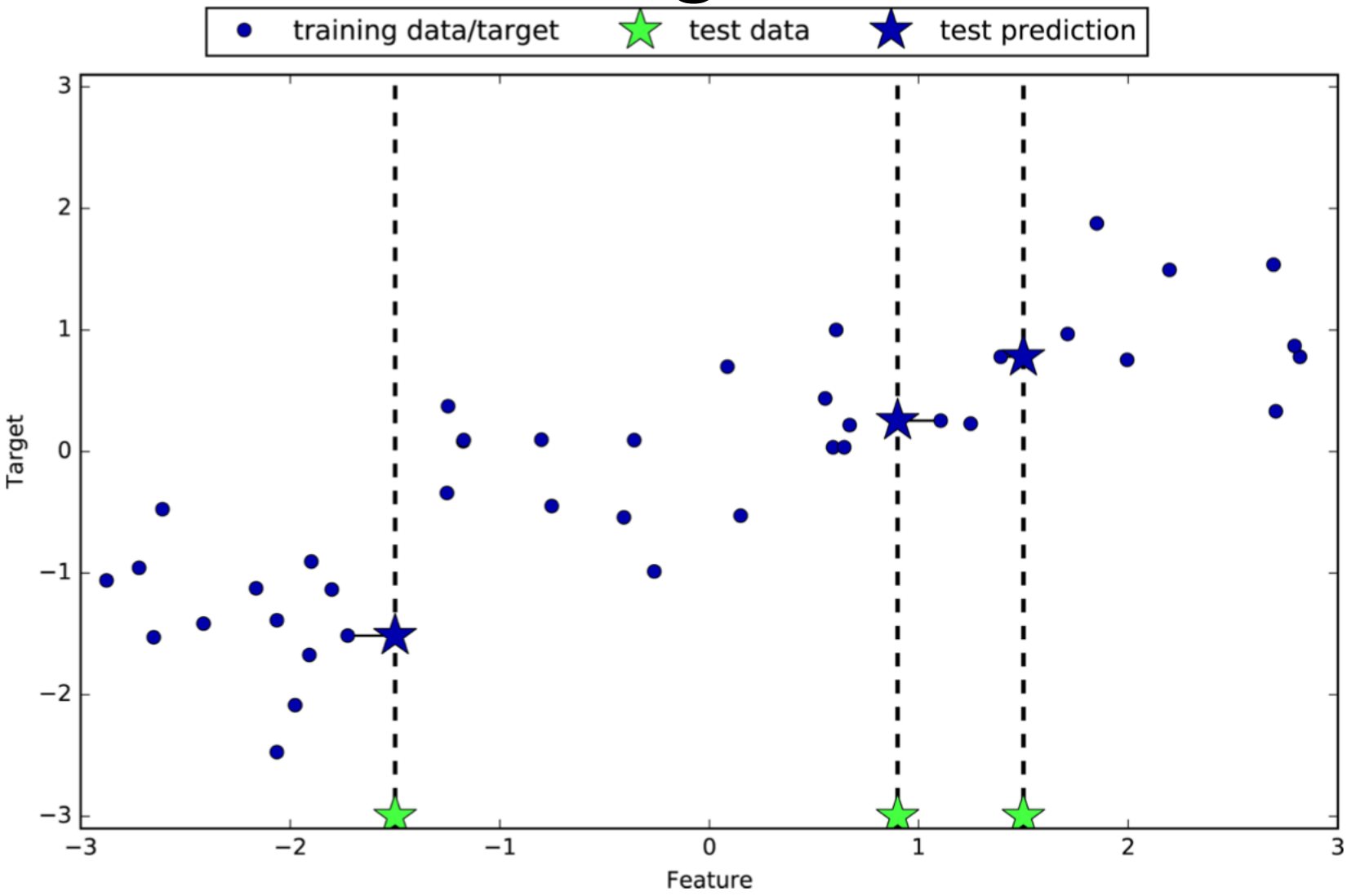
```
Test set accuracy: 0.857
```

# Decision boundaries

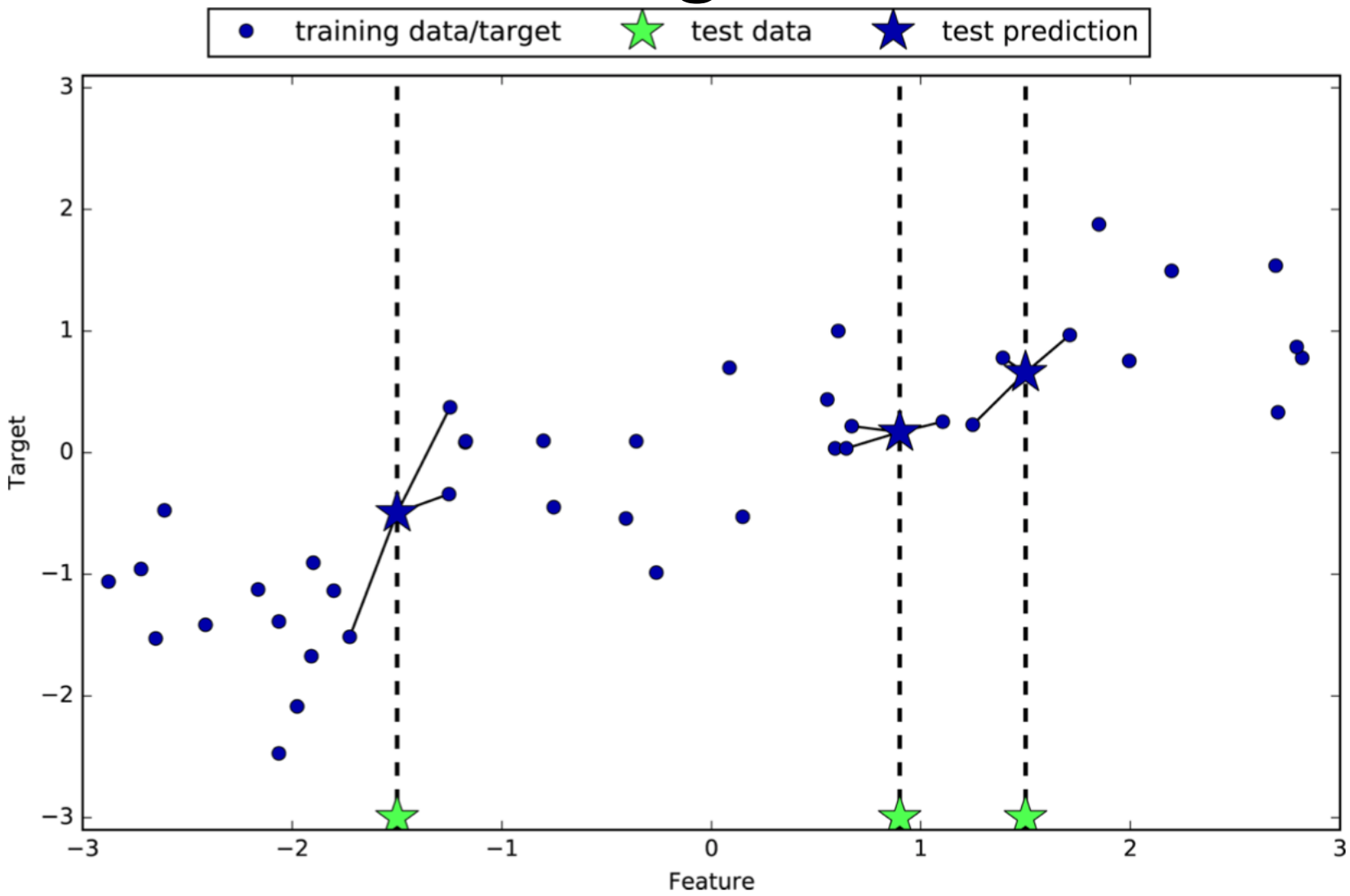



Which one has more bias and which one more variance?

# 1-NN regression



# 3-NN regression



# 3-NN Regression

In [21]:
```
from sklearn.neighbors import KNeighborsRegressor

X, y = mglearn.datasets.make_wave(n_samples=40)

# split the wave dataset into a training and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# instantiate the model and set the number of neighbors to consider to 3
reg = KNeighborsRegressor(n_neighbors=3)
# fit the model using the training data and training targets
reg.fit(X_train, y_train)
```

Out[21]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=None, n_neighbors=3, p=2,
          weights='uniform')
```

In [22]:
```
print("Test set predictions:\n", reg.predict(X_test))
```

```
Test set predictions:
 [-0.054  0.357  1.137 -1.894 -1.139 -1.631  0.357  0.912 -0.447 -1.139]
```

In [23]:
```
print("Test set R^2: {:.2f}".format(reg.score(X_test, y_test)))
```

```
Test set R^2: 0.83
```

# Linear models

- Why linear models
- Linear regression
- Regularization
  - L1, L2, and related models
- Linear classification
  - Logistic regression
  - SVMs
- Non-parametric models
  - KNNs

Now, practice!

Friday, trees