Scikit-learn and Tour of Classifiers

Recap

Classification pipeline

- **Goal**: Train a classifier that **generalizes** well on **unseen** data
- Train (for training) / Test (unseen) split to get a better estimate of the generalization error via the error on the test set
  - **Randomize and stratify** to get equal class distributions in the test and training set
- Feature Scaling/Transformations
  - Always apply **identical** transformations to the test and training set
  - **Avoid information leakage!** Standardization: Compute mean/std **on the training set**
- **Accuracy:** 1.0 – misclassified_samples / all_samples

Overfitting / Underfitting

Overfitting

- Overfitting is a common problem in machine learning
- We want the model to **generalize** well to **unseen** data
- A model that is overfitted performs well on the training data but does not generalise well to unseen data (test data)

Overfitting – Underfitting

A model that suffers from **overfitting**

- Has high variance
- High variance may be caused by too many parameters that lead to a model that is too complex given the training data

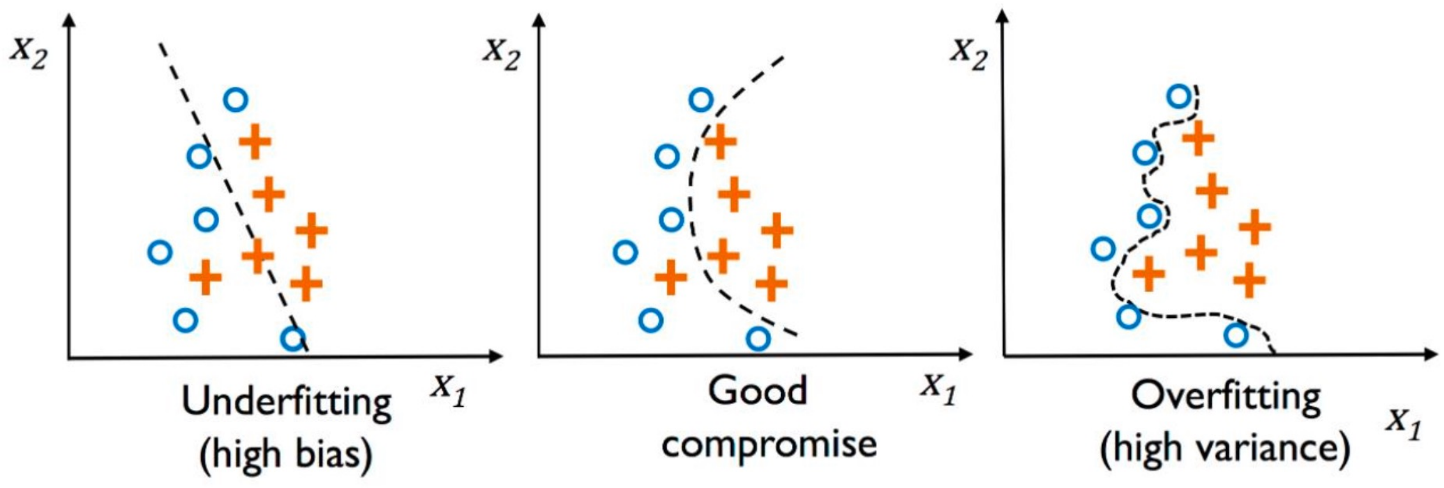
→ **poor performance** on unseen data because of high variance

A model that suffers from **underfitting**

- Has high bias
- High bias means that the model is not complex enough to capture the patterns in the training data well

→ **poor performance** on unseen data because of high bias

Overfitting – Underfitting (Examples)



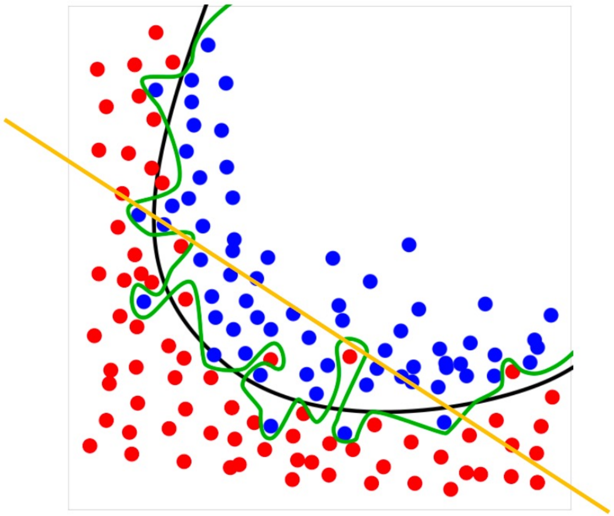
Linear decision boundary

Nonlinear decision boundary

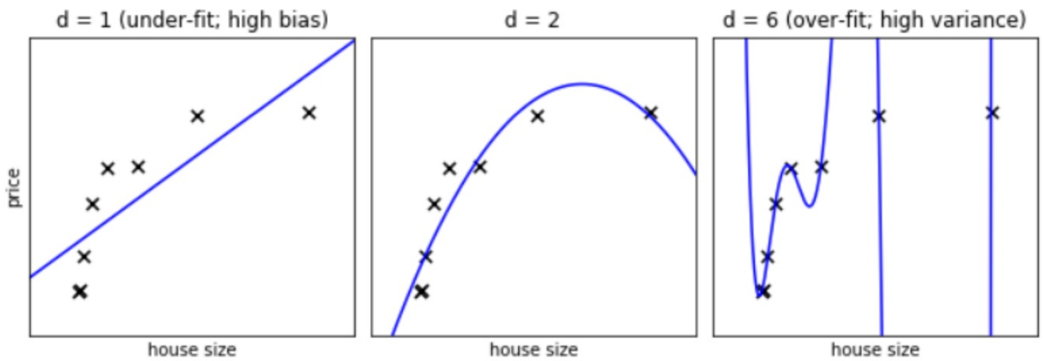
Nonlinear decision boundary

Figure from Python Machine Learning (Raschka & Mirjalili)

Overfitting – Underfitting (Examples)



Classification



Regression

Figure from Python Machine Learning (Raschka & Mirjalili)

Bias – Variance and Overfitting – Underfitting

Overfitting

Underfitting

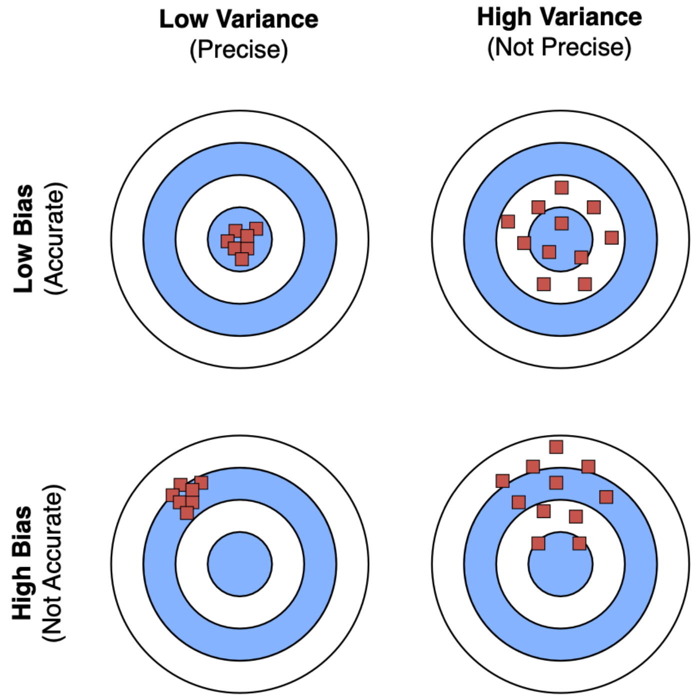


Figure by Sebastian Raschka Lecture Notes STAT451 FS20

Bias – Variance

Variance

- Measures the **consistency** (**variability**) of the model **prediction** for a particular sample
- If the model is retrained on different subsets of the training data and the prediction for a particular sample differs a lot each time → **high variance**
- The model is **sensitive** to the **randomness** in the **training data**

Bias

- Measures **how far off** the predictions are from the **correct values** in general
- If the model is retrained multiple times on different **training** datasets, bias measures the **systematic error** that is **not due** to randomness

Bias – Variance

Given a true value $\boldsymbol{y}$ and an estimator $\widehat{\boldsymbol{y}}$

Bias

$$Bias(\widehat{\boldsymbol{y}}) := E[\widehat{\boldsymbol{y}}] - \boldsymbol{y}$$

Variance

$$Var(\widehat{\boldsymbol{y}}) := E[(E[\widehat{\boldsymbol{y}}] - \widehat{\boldsymbol{y}})^2] = E[E[\widehat{\boldsymbol{y}}]^2 + \widehat{\boldsymbol{y}}^2 - 2E[\widehat{\boldsymbol{y}}]\widehat{\boldsymbol{y}}]$$

$$= E[\widehat{\boldsymbol{y}}]^2 + E[\widehat{\boldsymbol{y}}^2] - 2E[\widehat{\boldsymbol{y}}]E[\widehat{\boldsymbol{y}}] = E[\widehat{\boldsymbol{y}}]^2 + E[\widehat{\boldsymbol{y}}^2] - 2E[\widehat{\boldsymbol{y}}]^2 = E[\widehat{\boldsymbol{y}}^2] - E[\widehat{\boldsymbol{y}}]^2$$

$E[\widehat{\boldsymbol{y}}]$ **:** The expected value (expectation) of the estimator $\hat{y}$ (here: expectation over the training sets)

Bias – Variance decomposition of the squared loss

Given a true value $\boldsymbol{y}$ and an estimator $\hat{\boldsymbol{y}}$; $Bias(\hat{\boldsymbol{y}}) = E[\hat{\boldsymbol{y}}] - \boldsymbol{y}$; $Var(\hat{\boldsymbol{y}}) = E[(E[\hat{\boldsymbol{y}}] - \hat{\boldsymbol{y}})^2]$

For $\boldsymbol{y}$ (true outcome) and $\hat{\boldsymbol{y}}$ (estimated outcome):

Using: $(a+b)^2 = a^2 + b^2 + 2ab$

Squared loss: $L = (\boldsymbol{y} - \hat{\boldsymbol{y}})^2 = (\boldsymbol{y} - E[\hat{\boldsymbol{y}}] + E[\hat{\boldsymbol{y}}] - \hat{\boldsymbol{y}})^2$

$$= (\boldsymbol{y} - E[\hat{\boldsymbol{y}}])^2 + (E[\hat{\boldsymbol{y}}] - \hat{\boldsymbol{y}})^2 + 2(\boldsymbol{y} - E[\hat{\boldsymbol{y}}])(E[\hat{\boldsymbol{y}}] - \hat{\boldsymbol{y}})$$

Take the expected value of both sides:

$$\boxed{E[L] = E[(\boldsymbol{y} - E[\hat{\boldsymbol{y}}])^2 + (E[\hat{\boldsymbol{y}}] - \hat{\boldsymbol{y}})^2] = Bias^2 + Variance}$$

Using: $E[2(\boldsymbol{y} - E[\hat{\boldsymbol{y}}])] = 2(\boldsymbol{y} - E[\hat{\boldsymbol{y}}])$ and $E[(E[\hat{\boldsymbol{y}}] - \hat{\boldsymbol{y}})] = (E[\hat{\boldsymbol{y}}] - E[\hat{\boldsymbol{y}}]) = 0$

Bias – Variance trade-off

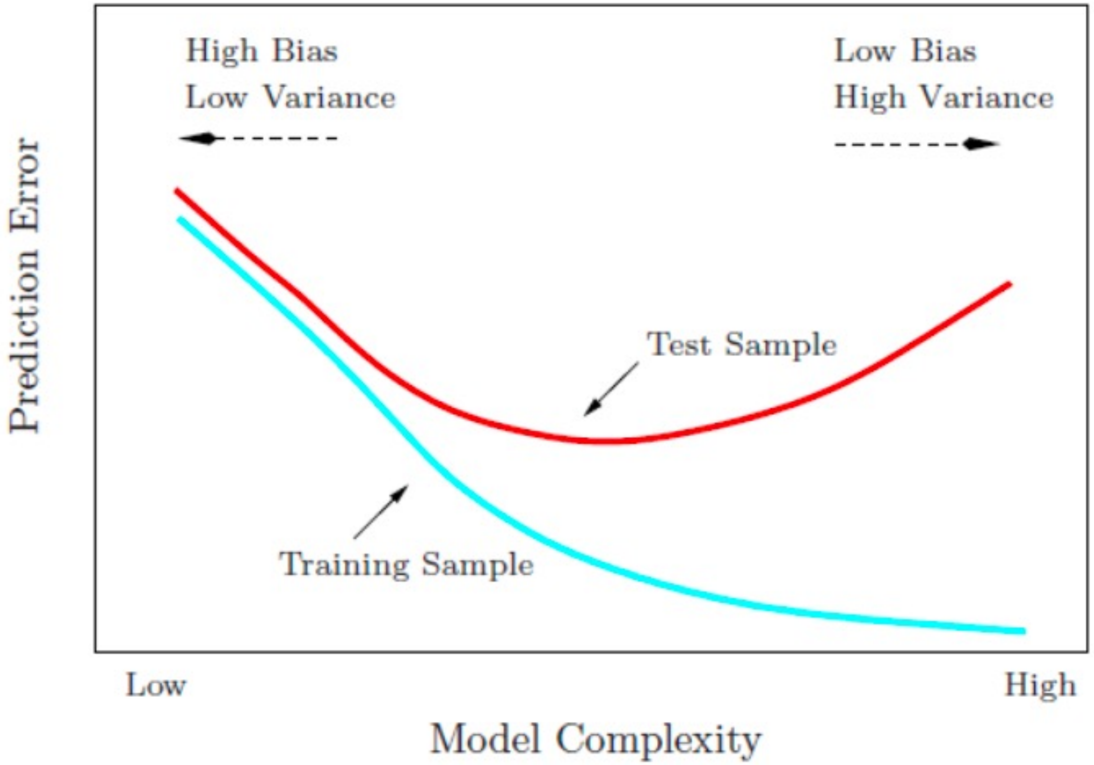


Figure from Elements of Statistical Learning (Hastie, Tibshirani, Friedman)

Bias – Variance trade-off

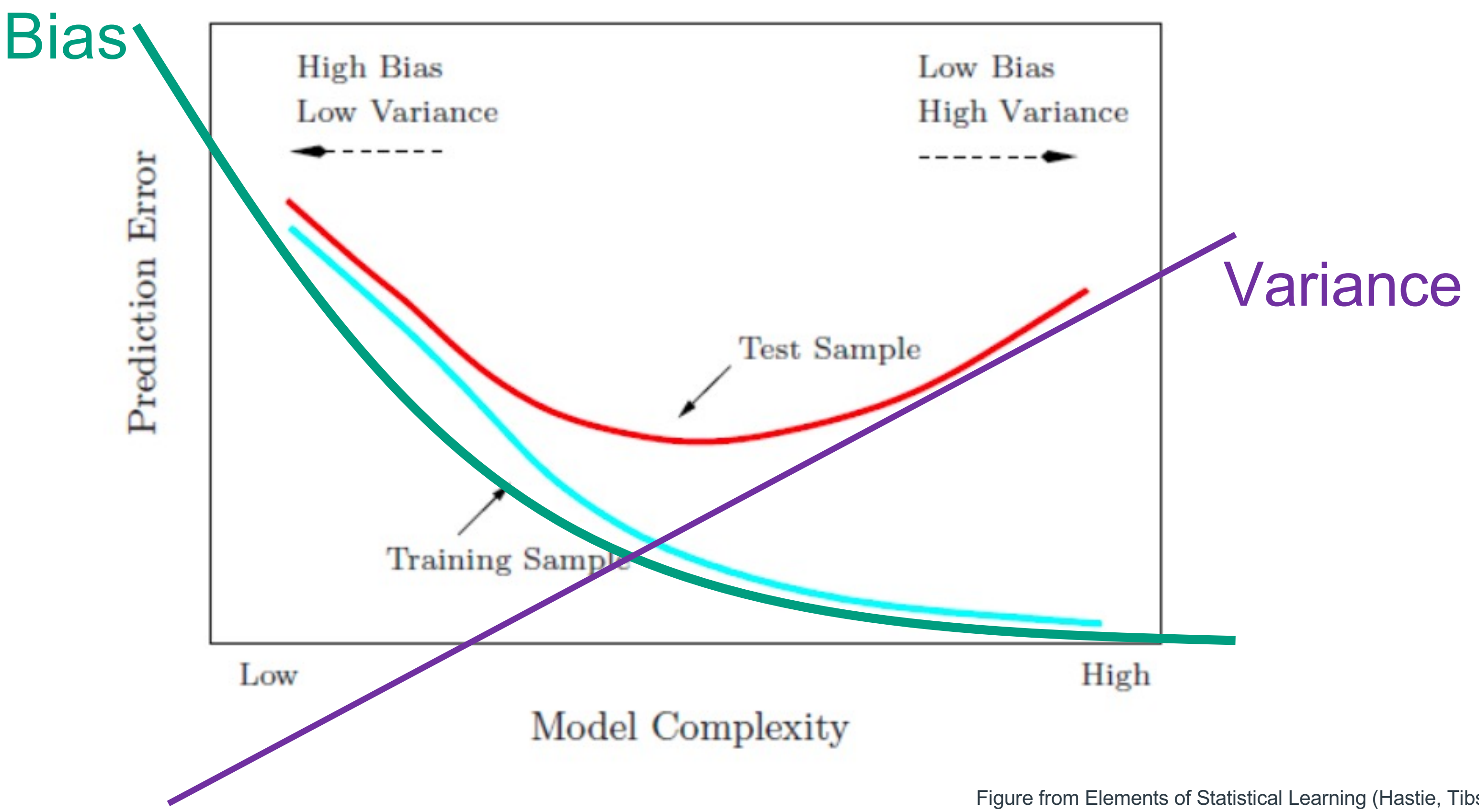


Figure from Elements of Statistical Learning (Hastie, Tibshirani, Friedman)

Bias – Variance trade-off

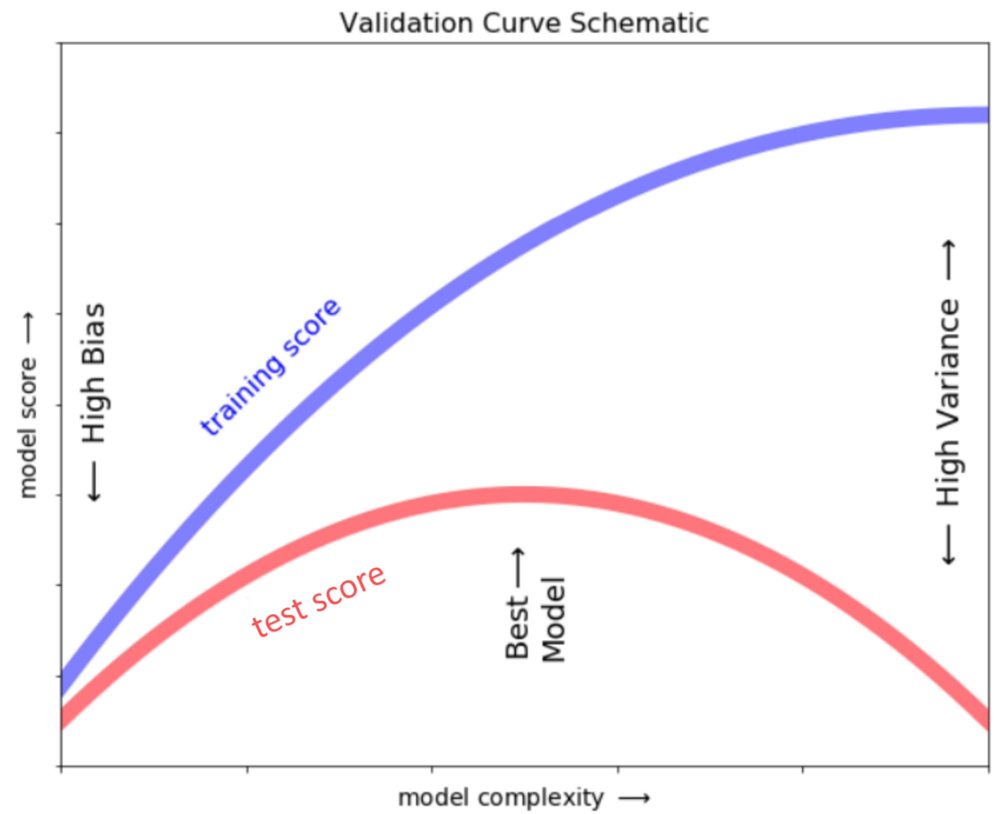


Figure from Python Machine Learning (Raschka & Mirjalili)

Tackling overfitting via regularization

Tackling overfitting

- Finding the appropriate **complexity** – **bias-variance trade-off** is important for good performance

  - Collect **more training data**
  - Introduce a **penalty for complexity** via **regularization**
  - Choose a **simpler model** with fewer parameters
  - **Reduce** the **dimensionality** of the data (feature selection, dimensionality reduction, Ch.06)

Tackling overfitting via regularization

- Good option: **Tune** the complexity of the model using **regularisation**
- Regularisation is very useful for
  - Handling **collinearity** (high **correlation** among **features**)
  - **Filtering out noise** from the data
  - **Preventing overfitting**
- To make regularisation work properly, features need to be **scaled** / **standardised**

- **Concept** of regularisation:
  - Introduce **additional information** (**bias**) to penalise **extreme parameter** (weight) **values**
  - The **most common** form of regularisation is so-called **L2 regularisation** (sometimes also called **L2 shrinkage** or **weight decay**) (For details: See Ch.04 in the course book)

ℓ_2- regularization (or L2 regularization / Ridge)

- ℓ_2-regularization: Add the Euclidean norm (two-norm) of the weights to the loss function

- **ℓ_2-regularized logistic regression:**

$$\boxed{L(\boldsymbol{\theta}) = -\sum_{i=1}^{n}\left[y^{(i)}\log(\sigma(z)) + (1-y^{(i)})\log(1-\sigma(z))\right] + \frac{\lambda}{2}||\mathbf{w}||_2^2}$$

Recall that $||\mathbf{w}||_2^2 = \sum_{i=1}^{m} w_i^2$ and $\boldsymbol{\theta} = \begin{bmatrix}\theta_0\\ \theta_1\\ \vdots\\ \theta_m\end{bmatrix} := \begin{bmatrix} b \\ \mathbf{w}\end{bmatrix}$

and $z := b + \mathbf{x}^{(i)}\mathbf{w} = \mathbf{x}^{(i)}\boldsymbol{\theta}$

ℓ_2- regularization

$$L(\boldsymbol{\theta}) = -\sum_{i=1}^{n}\left[y^{(i)}\log(\sigma(z)) + (1-y^{(i)})\log(1-\sigma(z))\right] \quad + \frac{\lambda}{2}||\mathbf{w}||_2^2$$

- Now **two** objectives: Minimize the “distance” to the data but **also** keep the **weights small**

- Other classifiers and regression models can be regularized in the same way!

ℓ_2- regularization

Regularisation parameter λ

- Provides **control** over **how well** the training data is fitted while keeping the weights small
- **Increasing** the value of $\lambda \rightarrow$ **increases** the regularisation **strength**
- The LogisticRegression class in scikit-learn implements a parameter C for regularisation
- C is the inverse of λ (C is the inverse regularisation parameter): $C := \frac{1}{\lambda}$
- Decreasing the value of C $\rightarrow$ **increases** regularisation strength

Train a logistic regression model

`03_logreg_iris.ipynb`

Code example:

- Use the **Iris** data set in scikit-learn
- Use ALL features
- Split the data into **training** and **test** set (test_size=0.3, random_state=1)
- Initialise the LogisticRegression class with LogisticRegression(C=100.0, random_state=1)
- Print out the number of **misclassified samples**
- Print out the **classification accuracy** for the **training** and **test data**

ℓ_2- regularized logistic regression

$$+\frac{\lambda}{2}||\mathbf{w}||_2^2$$

Code example:

- Use the **Wisconsin breast cancer** data set in scikit-learn
- Use ALL features
- Split the data 100 times into different training and test sets (test_size=0.3)
- Intialise the `LogisticRegression` class with `LogisticRegression(C=100.0, random_state=1)`
- Print out the average and standard deviation of the validation accuracy across the 100 splits

`03_logreg_cancer_variance.ipynb`

Train a logistic regression model on the cancer data set

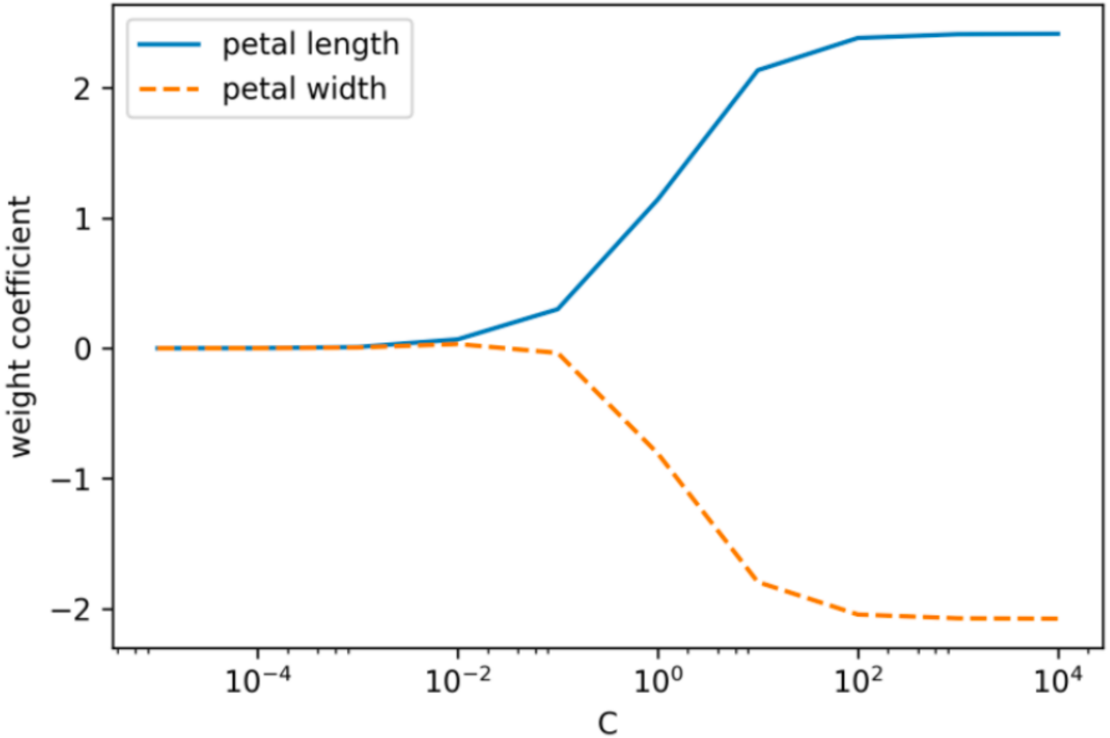
Exercise:

- Use the **Wisconsin breast cancer** data set in scikit-learn
- Use ALL features
- Split the data into **training** and **test** set (test_size=0.3, random_state=1)
- Initialise the LogisticRegression class with LogisticRegression(C=100.0, random_state=1)
- Print out the number of **misclassified samples**
- Print out the **classification accuracy** for **training** and **test data**
- **Does the accuracy change with C?**

ℓ_2- regularized logistic regression

$$+ \frac{\lambda}{2}||\mathbf{w}||_2^2$$

The effect of regularisation can be visualised by plotting the "ℓ_2**-regularization path**"



`03_logreg_iris_regularization.ipynb`

Figure from Python Machine Learning (Raschka & Mirjalili)

ℓ_2- regularized logistic regression

$$+\frac{\lambda}{2}||\mathbf{w}||_2^2$$

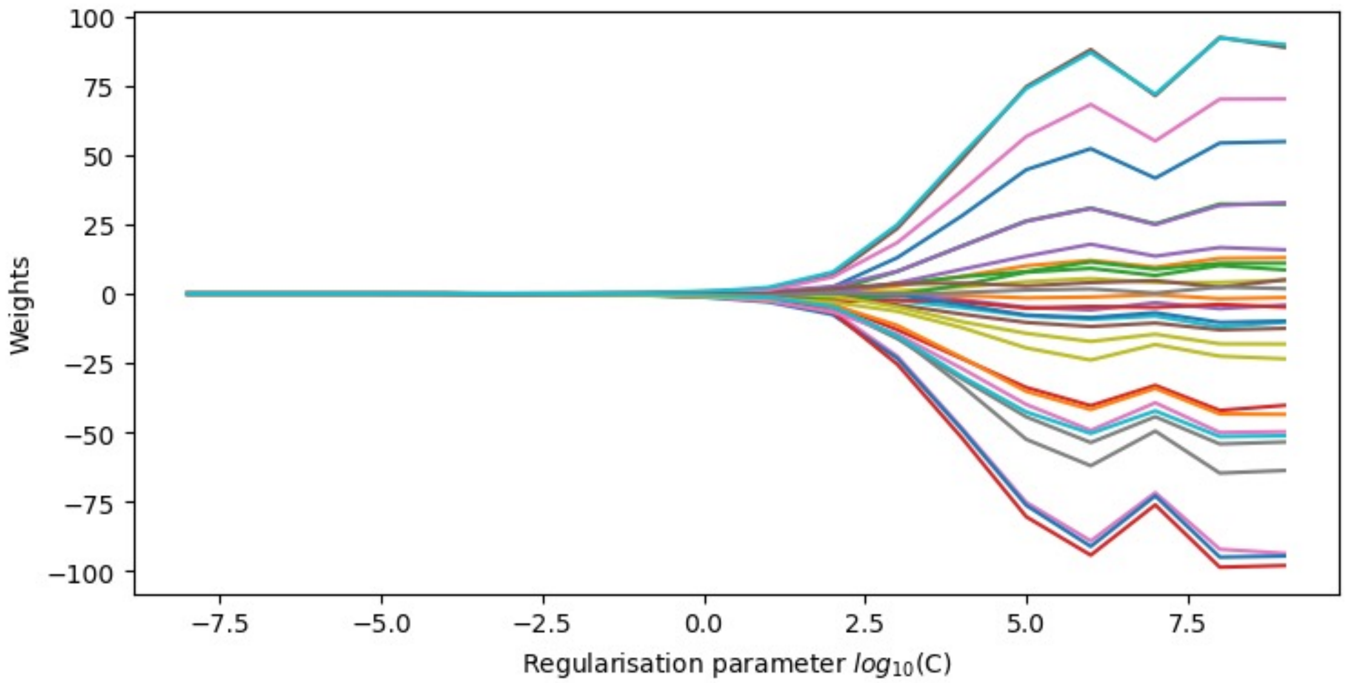
Exercise:

- Use the **Wisconsin breast cancer** data set in scikit-learn
- Use ALL features
- Split the data into a training set and a test set (test_size=0.3)
- Intialise the `LogisticRegression` class with `LogisticRegression(C=10.0**c, random_state=1)`
  - Small `c` varies from -8 to 10 (steps of 1)
- Plot the training and test accuracy across each C in one plot
- For which C does the model provide the best test accuracy? For which C values does the model overfit? For which C values does the model underfit?

`03_logreg_cancer_regularization.ipynb`

ℓ_2- regularized logistic regression

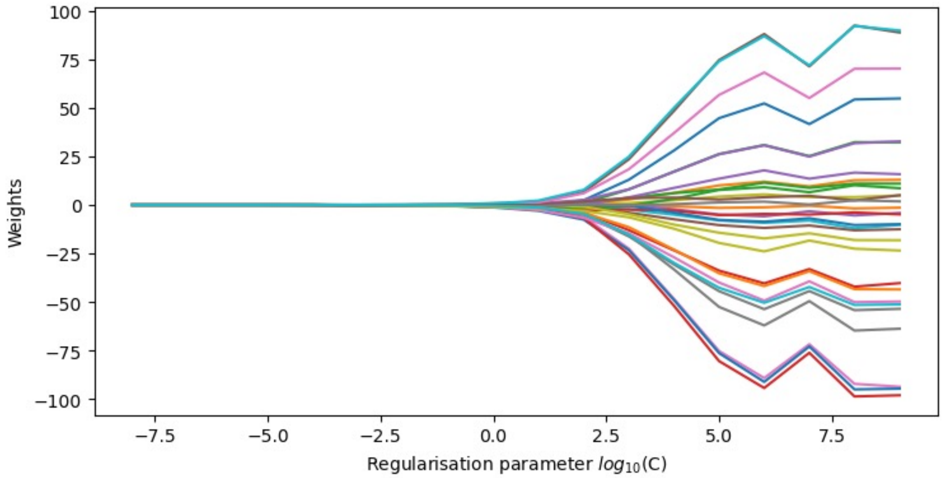
$$+\frac{\lambda}{2}||\mathbf{w}||_2^2$$

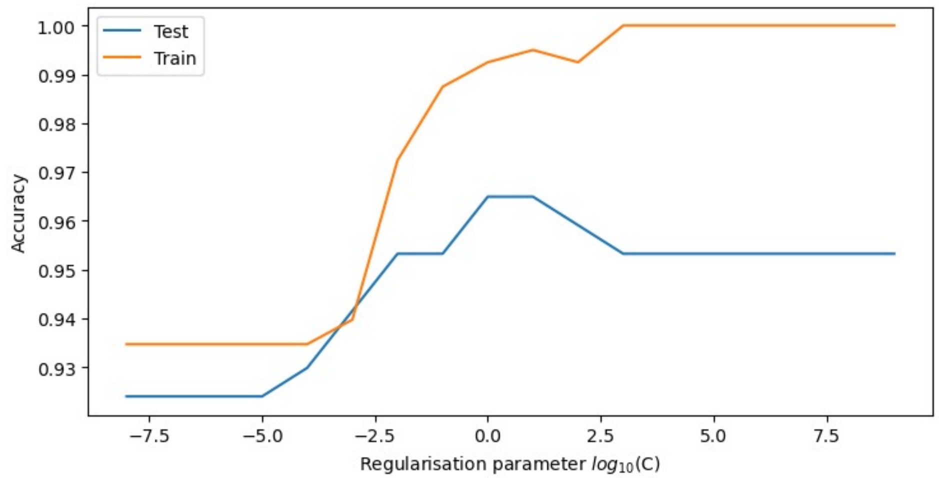


`03_logreg_cancer_regularization.ipynb`

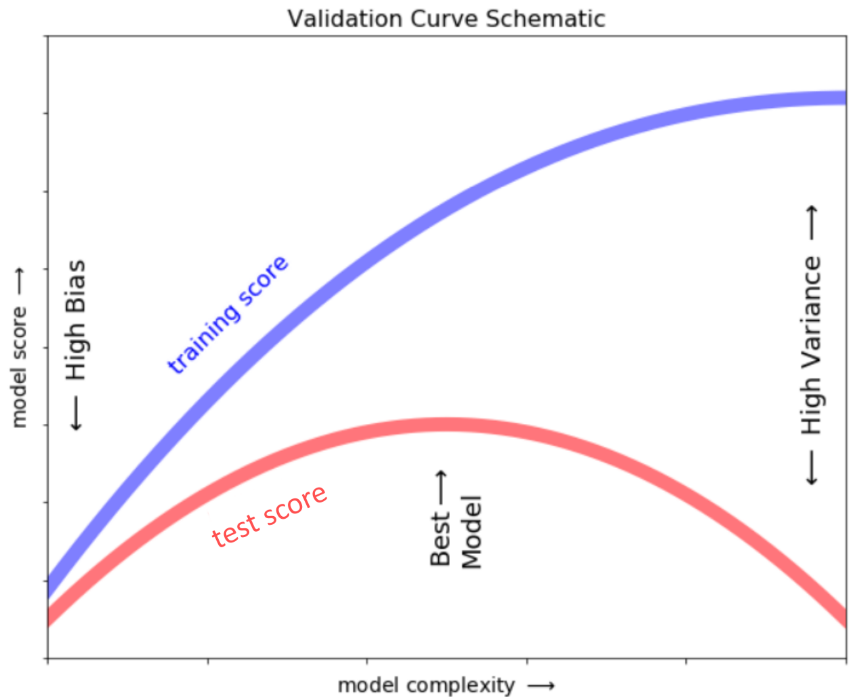
Figure from Python Machine Learning (Raschka & Mirjalili)

ℓ_2- regularized logistic regression (exercise)





Bias – Variance trade-off



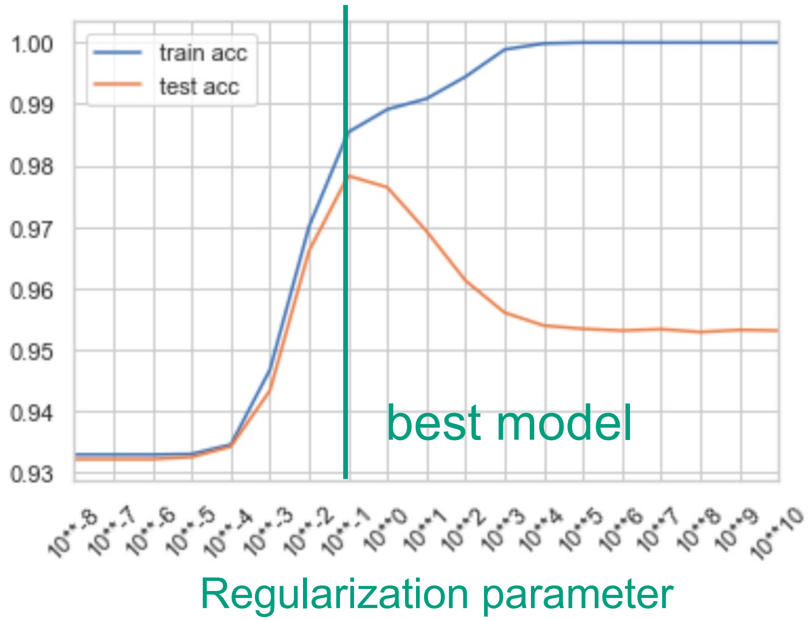


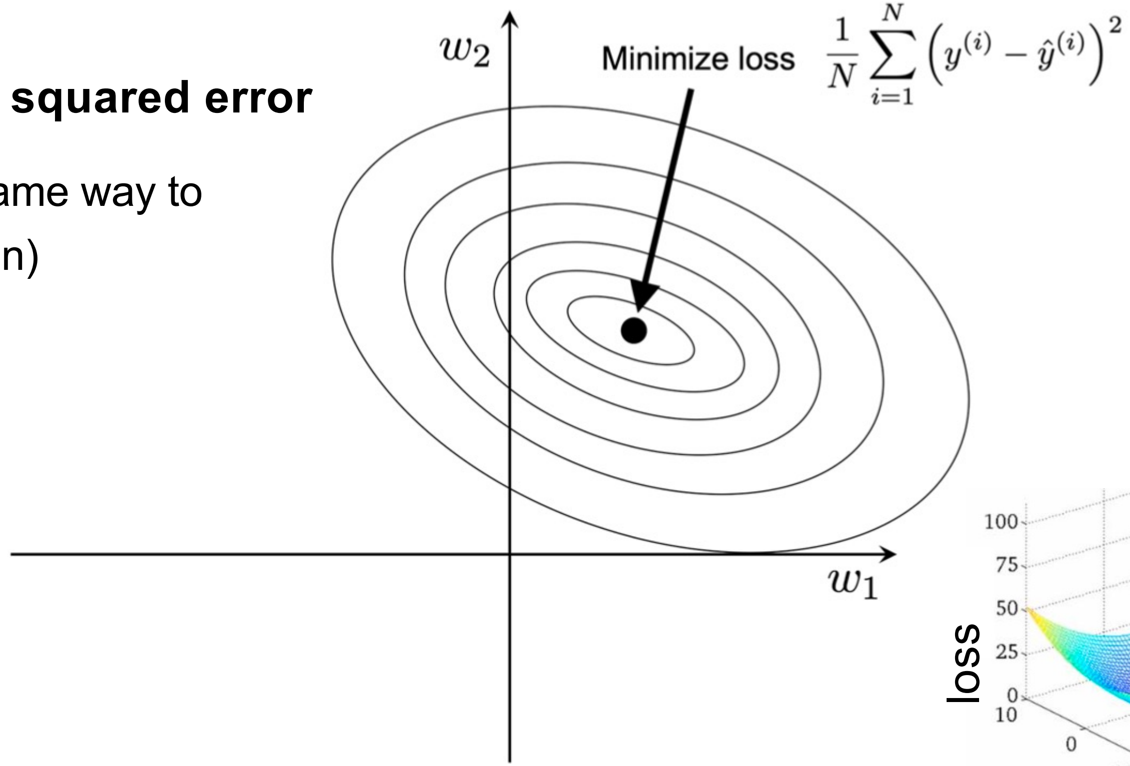
Figure from Python Machine Learning (Raschka & Mirjalili)

Geometric interpretation of regularization

- Example: **Mean squared error**

  (Applies in the same way to logistic regression)

- Two features



3D visualization

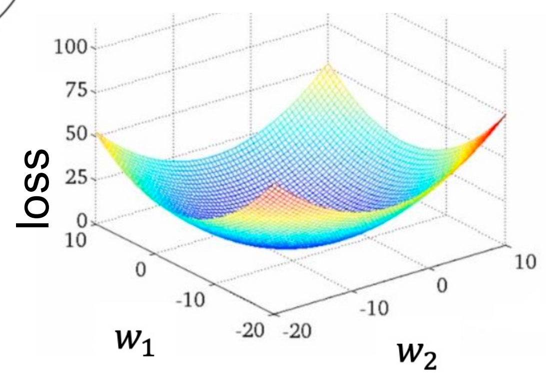


Figure from Python Machine Learning (Raschka & Mirjalili)

Geometric interpretation: ℓ_2- regularization

- **"Encourage small weights"** or **"Penalize large weights"**
- Increasing λ gives more weight to the penalty → More emphasis on smaller weights than fitting data

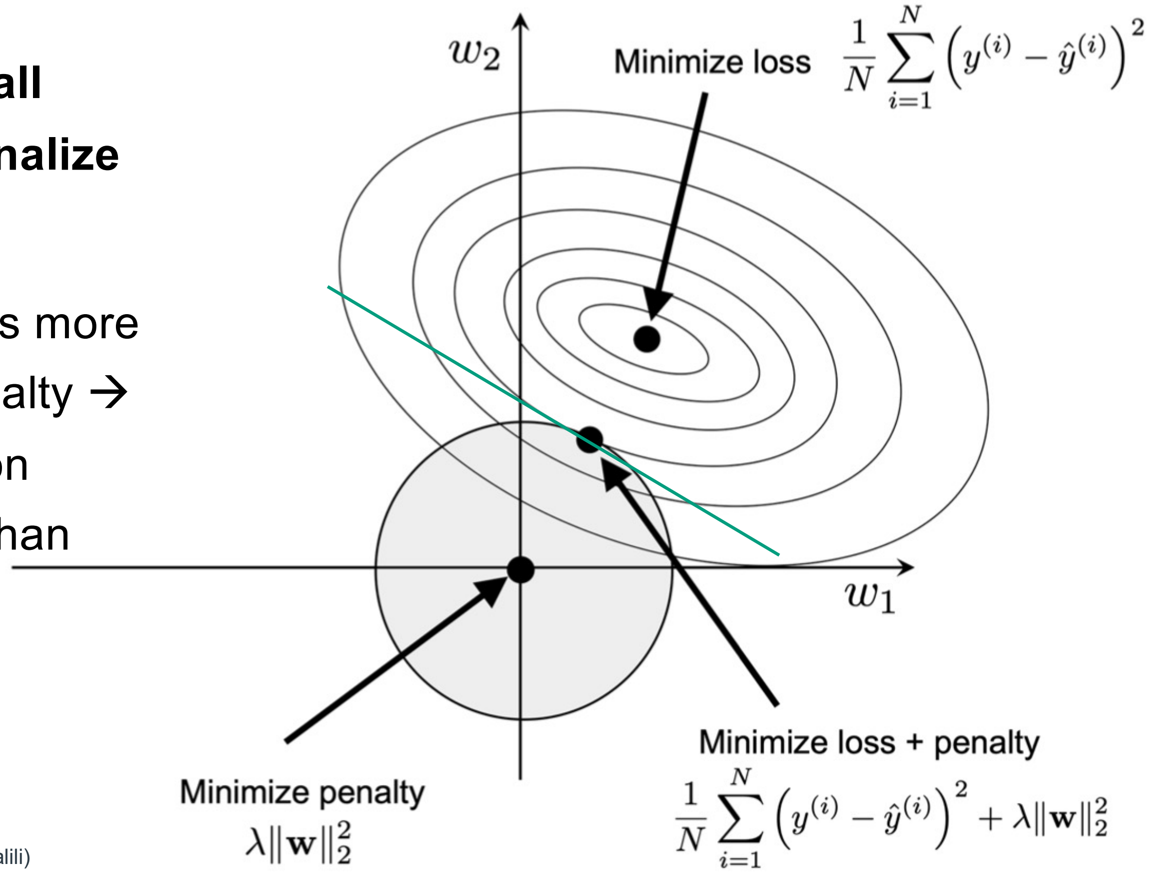


Figure from Python Machine Learning (Raschka & Mirjalili)

Tackling overfitting via regularization

Sparsity-promoting regularization

ℓ_1- regularization (or L1 regularization / LASSO)

LASSO – Least absolute shrinkage and selection operator

$$L(\boldsymbol{\theta}) = -\sum_{i=1}^{n}\left[y^{(i)}\log(\sigma(z)) + (1-y^{(i)})\log(1-\sigma(z))\right] + \lambda||\mathbf{w}||_1$$

Where $||\mathbf{w}||_1 = \sum_{i=1}^{m}|w_i|$ and $\boldsymbol{\theta} = \begin{bmatrix}\theta_0\\ \theta_1\\ \vdots\\ \theta_m\end{bmatrix} := \begin{bmatrix} b\\ \mathbf{w}\end{bmatrix}$

and $z := b + \mathbf{x}^{(i)}\mathbf{w} = \mathbf{x}^{(i)}\boldsymbol{\theta}$

Geometric interpretation: ℓ_1- regularization

- Example: Mean squared error (the same concept applies to the logistic loss function)

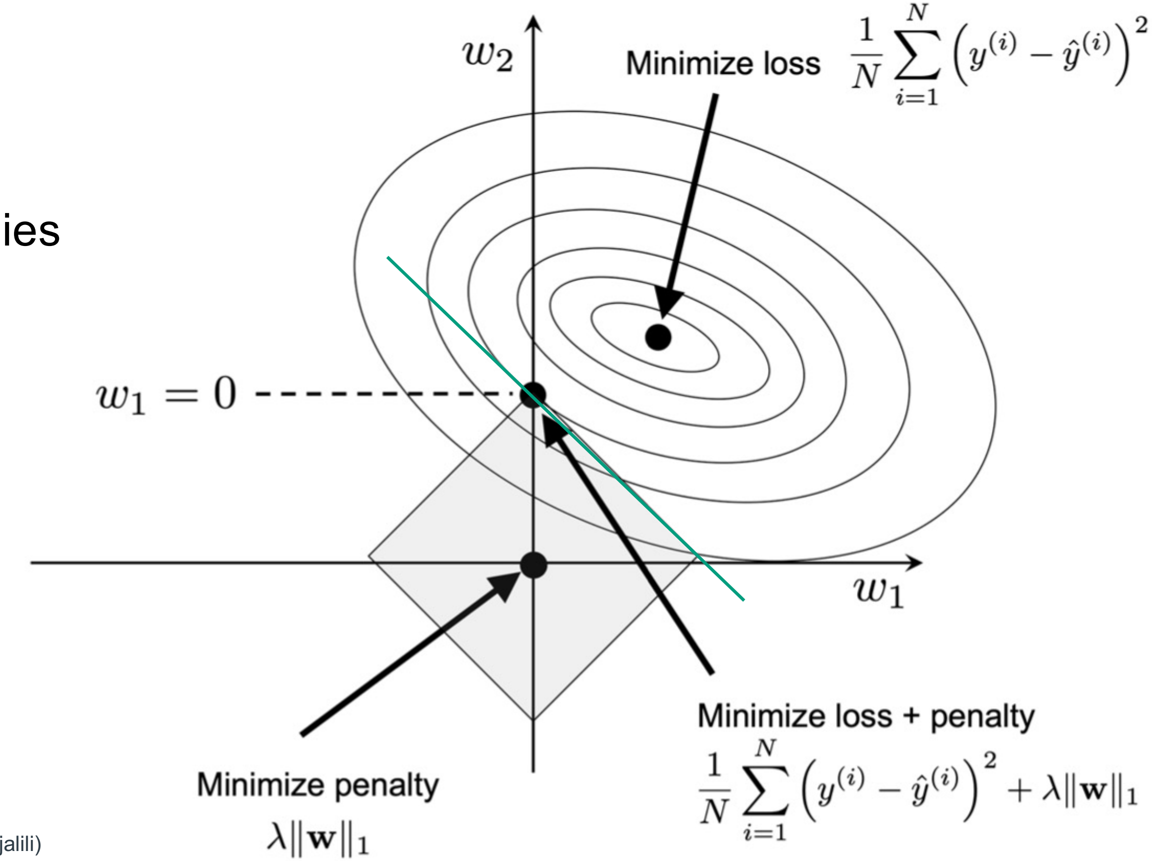


Figure from Python Machine Learning (Raschka & Mirjalili)

Geometric interpretation: ℓ_1- regularization

Due to the shape of the ℓ_1-loss this leads to a selection of weights such that many weights are zero (here: “automatic feature selection”)

We say ℓ_1 regularization **promotes “sparsity”**

It leads to models with less parameters / lower complexity

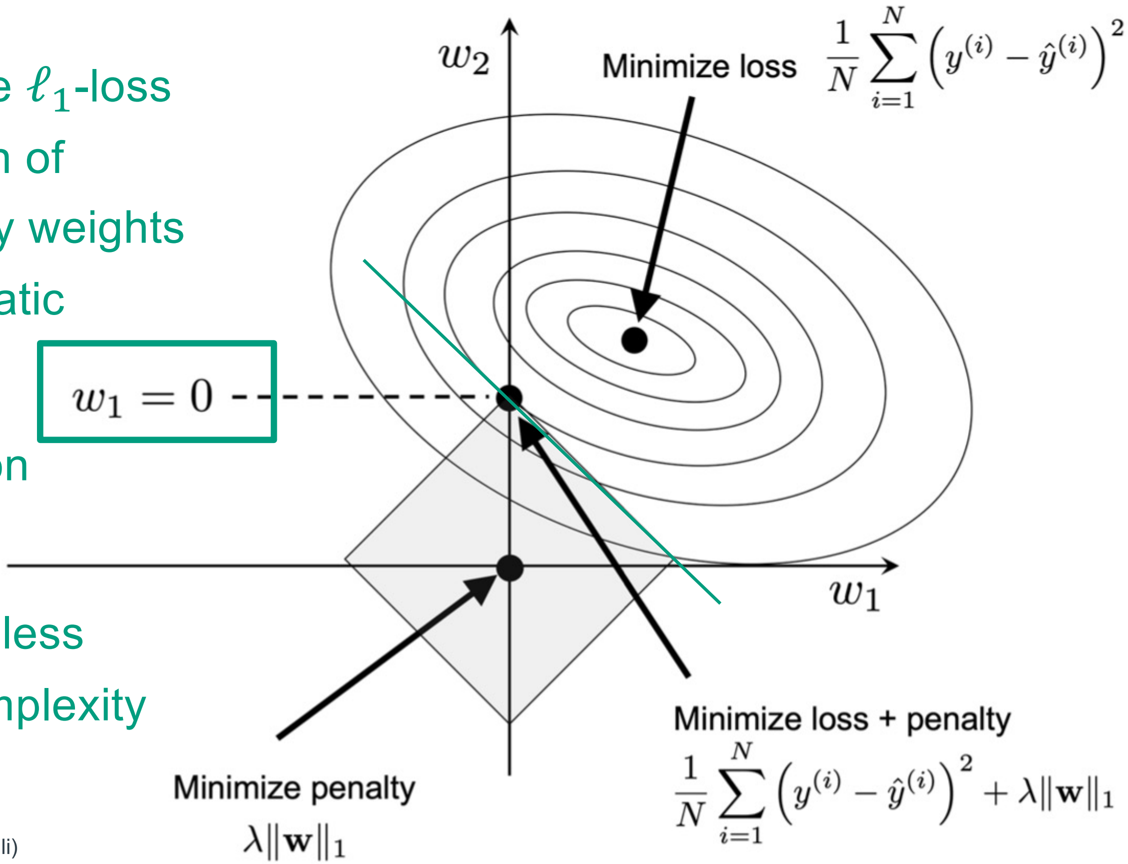
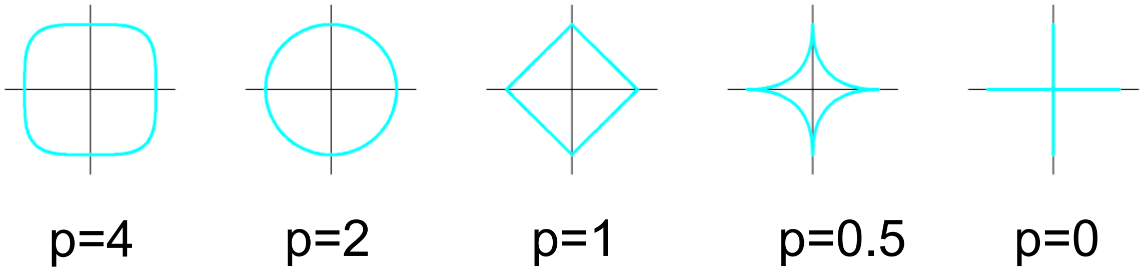


Figure from Python Machine Learning (Raschka & Mirjalili)

Geometric interpretation: ℓ_p- regularization

- Other theoretical options:



$$+\frac{\lambda}{p}||\mathbf{w}||_p^p$$

$$||\mathbf{w}||_p^p = \sum_{i=1}^{m} |w_i|^p$$

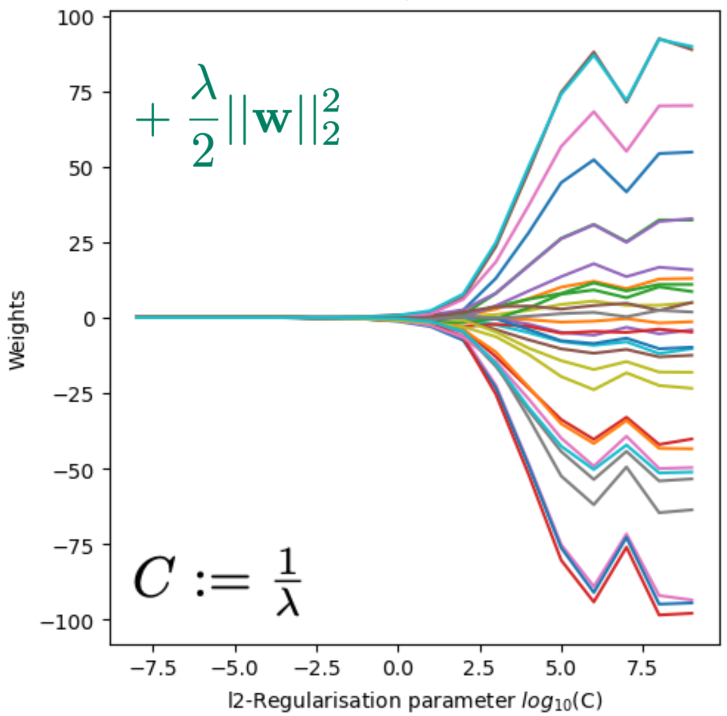
- Due to computational restrictions: **In practice, the options used are ℓ_1(Lasso), ℓ_2(Ridge) and ℓ_1+ ℓ_2 (Elastic net)**

Sparsity-promoting regularization

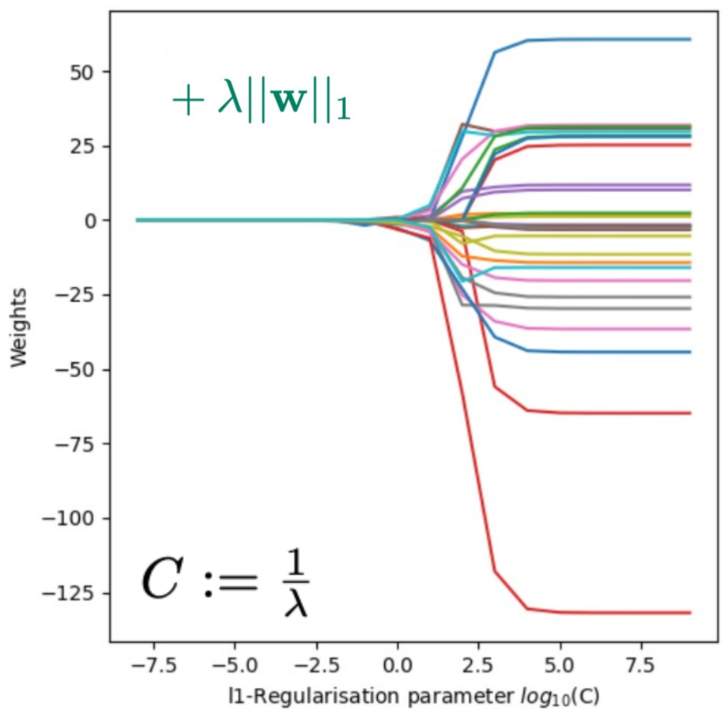
- L1 regularisation usually yields **sparse** feature vectors → most feature weights will be zero
- Sparsity can be useful in practice if we have a **high-dimensional** dataset with **many** features that are **irrelevant**
- L1 regularisation is especially useful in cases where more **irrelevant dimensions** than **samples** are present
- L1 regularisation can be understood as a technique for **feature selection**

Comparison of L1/L2 regularization

L2 / Ridge



L1 / LASSO



Some weights stay at 0

`03_logreg_cancer_regularization_L1_L2.ipynb`

Scikit-learn classifiers with a regularization option

- Perceptron
- Logistic regression
- Linear SVC

Options:

- L1 – a.k.a LASSO (Least Absolute shrinkage and Selection Operator)
- L2 – a.k.a. Ridge penalisation
- L1 + L2 – a.k.a. Elastic Net

NMBU