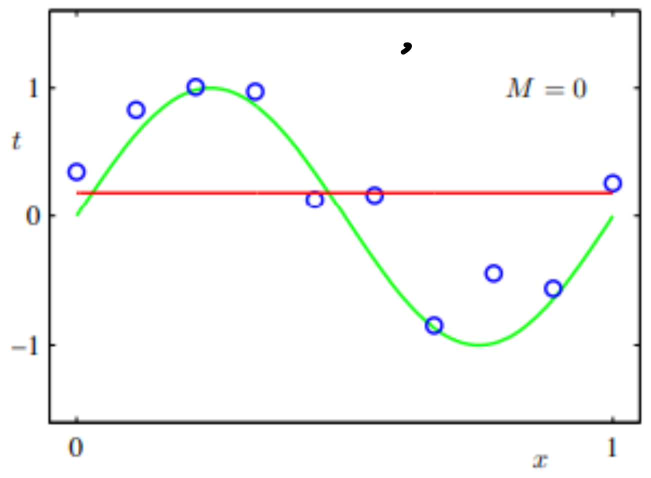
# Topic 4
# BVT + Logistic Regression

# Bias & Variance TradeOff

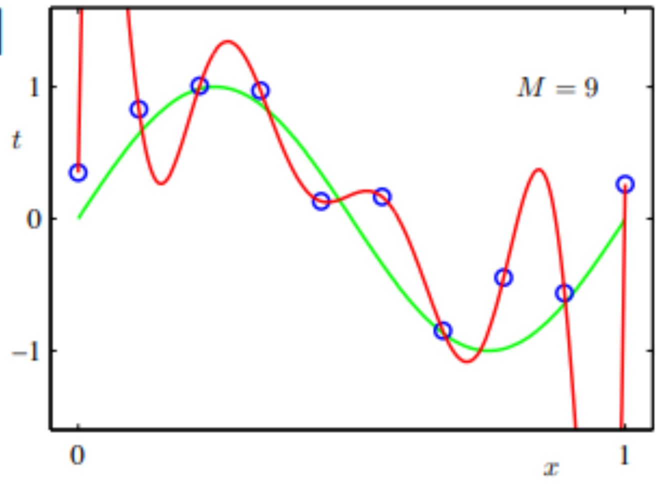
The bias-variance tradeoff is a fundamental concept in machine learning that helps in understanding and managing the sources of error in predictive modeling. It refers to the balance between two types of errors that affect the performance of a model: bias and variance.

- Model too simple: does not fit the data well
  - A *biased* solution

underfit



- Model too complex: small changes to the data, solution changes a lot
  - A *high-variance* solution



## The Bias/Variance Trade-Off

An important theoretical result of statistics and machine learning is the fact that a model's generalization error can be expressed as the sum of three very different errors:

*Bias*

This part of the generalization error is due to wrong assumptions, such as assuming that the data is linear when it is actually quadratic. A high-bias model is most likely to underfit the training data.[6]

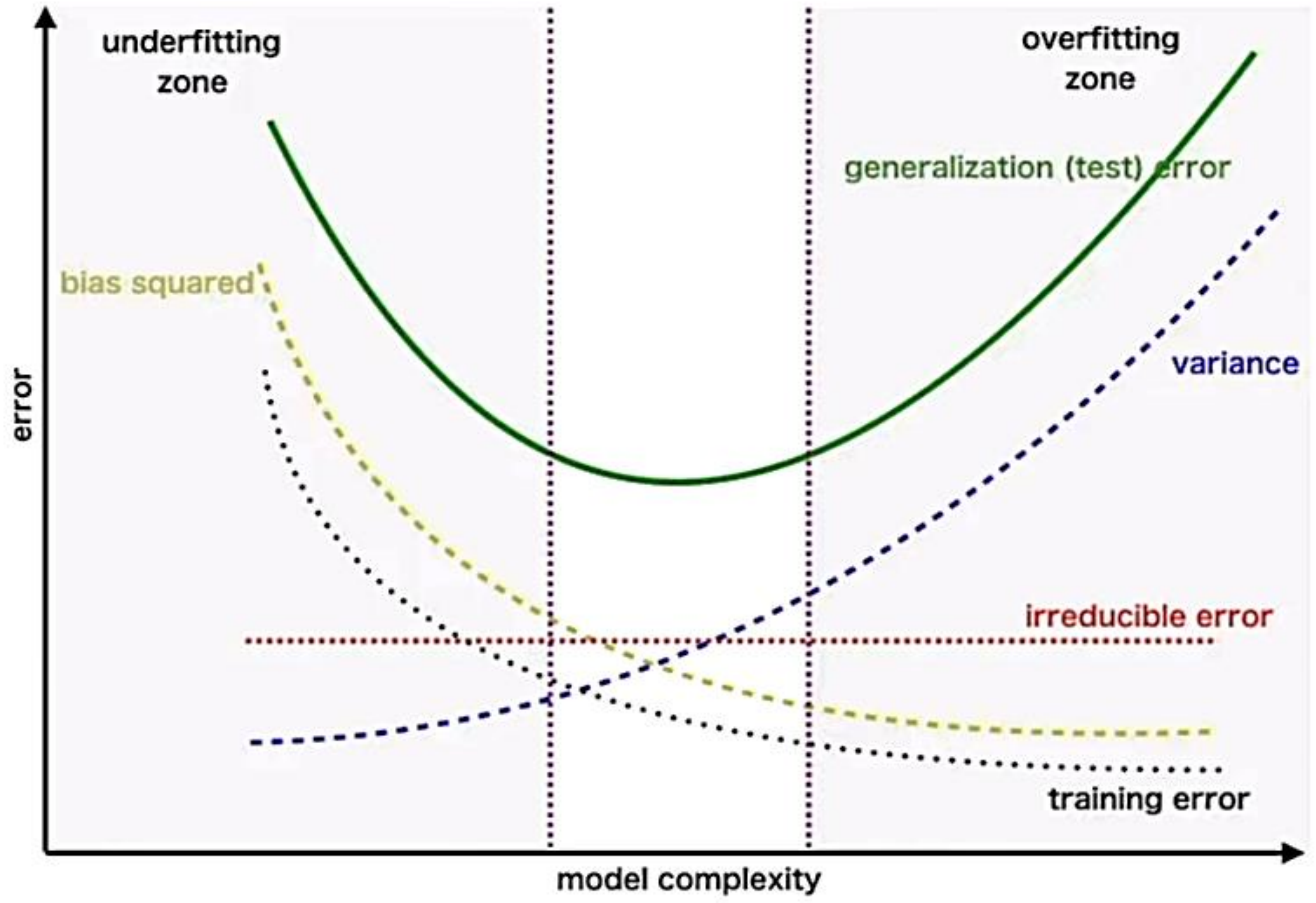
*Variance*

This part is due to the model's excessive sensitivity to small variations in the training data. A model with many degrees of freedom (such as a high-degree polynomial model) is likely to have high variance and thus overfit the training data.

*Irreducible error*

This part is due to the noisiness of the data itself. The only way to reduce this part of the error is to clean up the data (e.g., fix the data sources, such as broken sensors, or detect and remove outliers).

Increasing a model's complexity will typically increase its variance and reduce its bias. Conversely, reducing a model's complexity increases its bias and reduces its variance. This is why it is called a trade-off.


underfitting zone
overfitting zone
generalization (test) error
bias squared
variance
error
irreducible error
training error
model complexity

# Let’s Ask ChatGPT !

- **1. Bias:**
- **Definition:** Bias is the error introduced by approximating a real-world problem, which may be complex, by a simplified model.
- **Characteristics:** High bias leads to underfitting. Models with high bias are too simple and may not capture the underlying patterns in the data. They make strong assumptions about the data that may not hold true.
- **Effects:** A high-bias model tends to be overly generalized and may not perform well on training data or new, unseen data. It may overlook important relationships in the data.

**Tradeoff:**
•**Bias-Variance Tradeoff:** The tradeoff suggests that as you increase the complexity of a model, bias decreases but variance increases, and vice versa.
•**Goal:** The goal in machine learning is to find the right level of model complexity that minimizes both bias and variance, leading to optimal model performance on new, unseen data.
•**Optimal Model:** An optimal model balances the tradeoff, capturing the underlying patterns in the data without being too simple (high bias) or too complex (high variance).

**2. Variance:**
•**Definition:** Variance is the error introduced by having a model that is too complex, capturing noise in the training data as if it were a real pattern.
•**Characteristics:** High variance leads to overfitting. Models with high variance are too flexible and may capture noise or random fluctuations in the training data, which don't generalize well to new data.
•**Effects:** A high-variance model may perform very well on the training data but fail to generalize to new data. It is sensitive to small changes in the training set and may exhibit poor performance on unseen examples.

**How to Address the Tradeoff:**
**1.Cross-Validation:** Use techniques like cross-validation to assess how well your model generalizes to new data and find the right level of complexity.
**2.Regularization:** Apply regularization techniques (e.g., Ridge, Lasso) to control the complexity of the model and prevent overfitting.
**3.Ensemble Methods:** Use ensemble methods (e.g., Random Forests, Gradient Boosting) that combine multiple models to achieve better performance by mitigating the weaknesses of individual models.
Understanding and managing the bias-variance tradeoff is crucial for building effective machine learning models that generalize well to new data and avoid overfitting or underfitting.

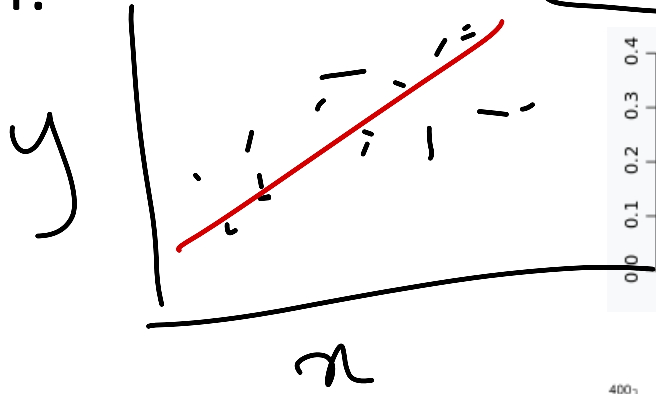
# Can we show how MSE depends on this Trade-off !

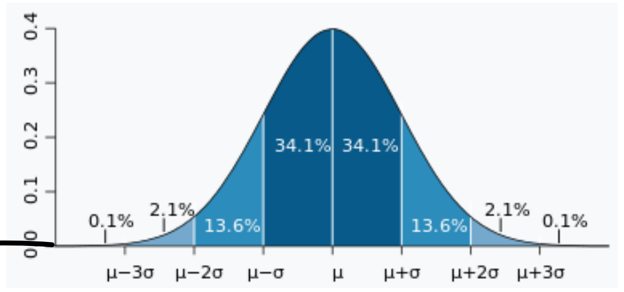
$$\boxed{\frac{1}{m}\sum_{i=1}^{m}\left(h\left(\mathbf{x}^{(i)}\right)-y^{(i)}\right)^2}$$

- Warning: Get your math caps on!
- Be attentive !

Independent variable ⇒ $x$

dependent var ⇒ $y$





$$\boxed{\underline{\underline{y}} = \underline{\underline{f(x)}} + \varepsilon}$$

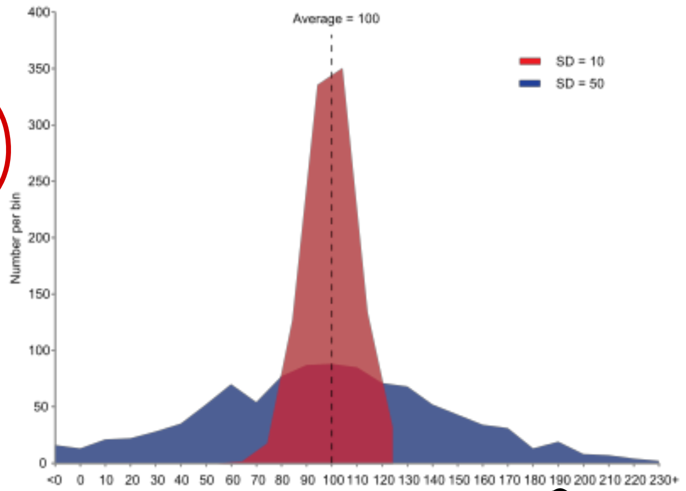
Observed

$\varepsilon \Rightarrow$ zero mean

$$\boxed{\underline{\underline{E(\varepsilon) = 0}}}$$

Expected value

$Var(\varepsilon) = E(\varepsilon^2) = \underline{\underline{\sigma_\varepsilon{}^2}}$   $Var \to \underline{\underline{\sigma_\varepsilon{}^2}}$



Find $\hat{f}$ (predicted) ⇒ s.t it is close to function $\underline{\underline{f}}$

$(\hat{y}) \Rightarrow \hat{f}$

$$\boxed{\text{MSE} \Rightarrow E[(y-\hat{f}(x))^2]}$$

$$\boxed{\text{bias } \hat{f}(x) = E[\hat{f}(x)] - f(x)}$$

Difference between the average value of prediction to the true underlying function $f(x)$ for a given unseen test point. $x$

$$\boxed{\text{Var}[\hat{f}(x)] = E\left[\left(\hat{f}(x) - E[\hat{f}(x)]\right)^2\right]}$$

mean squared deviation of $\hat{f}(x)$ from its expected value $E(\hat{f}(x))$

Prove

$$E\left[E\left[y - \hat{f}(x)\right]^2\right] = E\left[\text{bias}\left[\hat{f}(x)\right]^2\right] + E\left[\text{var}\left(\hat{f}(x)\right)\right] + \sigma_\varepsilon^2$$

Bias

Var

Irr. error

$$E\left[\left(f(x) + \varepsilon - \hat{f}(x)\right)^2\right] \quad \left[f(x) - \hat{f}(x) \quad + \varepsilon\right]$$

$$= E\left[\left(f(x) - \hat{f}(x)\right)^2\right] + E(\varepsilon^2) + 2E\left[f(x) - \hat{f}(x)\right] \cdot E(\varepsilon)$$

$E(XY) = E(X) \cdot E(Y)$

$$= \text{—} \;\; \text{॥} \;\; \text{——} + \sigma_\varepsilon^2 + 0$$

$$= \boxed{E\left(\left(f(x) - \hat{f}(x)\right)^2\right) + \sigma_\varepsilon^2}$$

A

$$E\left[(f(x)-\hat{f}(x))^2\right] = E\left[\left(\left(f(x)-\boxed{E\hat{f}(x)}\right) - \left(\hat{f}(x) - E[\hat{f}(x)]\right)\right)^2\right]$$

$$= E\left(\left(E(\hat{f}(x)) - f(x)\right)^2\right) \quad \text{bias}^2$$

$$+ E\left[\left(\hat{f}(x) - E(\hat{f}(x))\right)^2\right] \Rightarrow \boxed{\text{var.}}$$

$$- 2E\left[\cancel{f(x) - E(\hat{f}(x))}\right] \times \left[E\left[\hat{f}(x) - E[\hat{f}(x)]\right]\right]$$

$$0$$

$$\downarrow$$

$$E(\hat{f}(x)) - E(E(\hat{f}(x)))$$

$$\downarrow$$

$$E(\hat{f}(x)) - E(\hat{f}(x)) = 0$$

$$MSE = (\text{bias})^2 + (\text{var}) + \sigma_\varepsilon$$

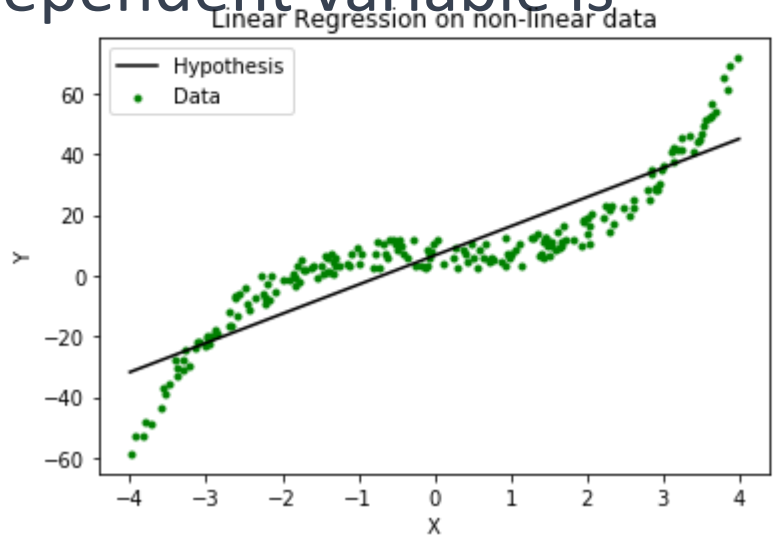
$$\underline{E\left[\left(y-\hat{f}(x)\right)^2\right]} = \underline{\text{bias}\left(\hat{f}(x)\right)^2} + \underline{\text{var}\left(\hat{f}(x)\right)} + \underline{\sigma_\varepsilon^2}$$

$\downarrow$ lot of points

$$E(\quad) = E(\quad) + E(\quad) + \sigma_\varepsilon^2$$

# Locally weighted regression

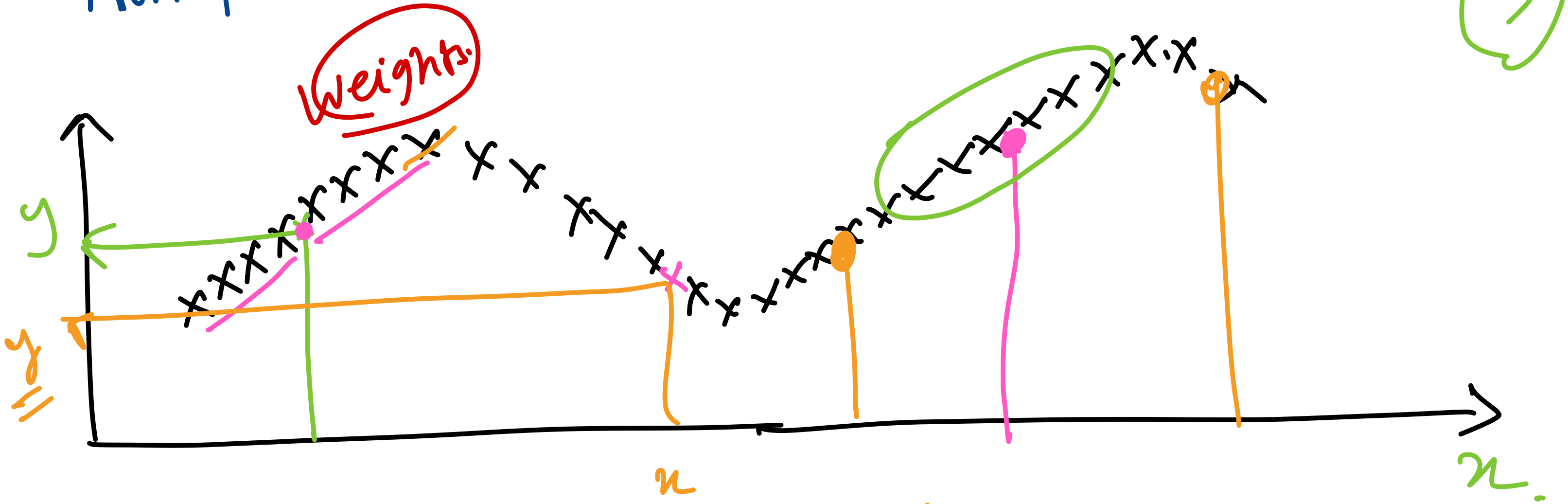
- Locally Weighted Regression (LWR), also known as Locally Weighted Scatterplot Smoothing (LOWESS), is a non-parametric regression method that combines aspects of linear regression with a weighting function to give different weights to different data points when making predictions. It is particularly useful when the relationship between the independent variable and the dependent variable is expected to vary across the dataset.



Parametric algo ⇒ Fixed for set of parameters $\theta_i$

Non parameter learning algo:

3



looks @ neighbourhood to fit a straight line

Fit $\theta$ to minimise:

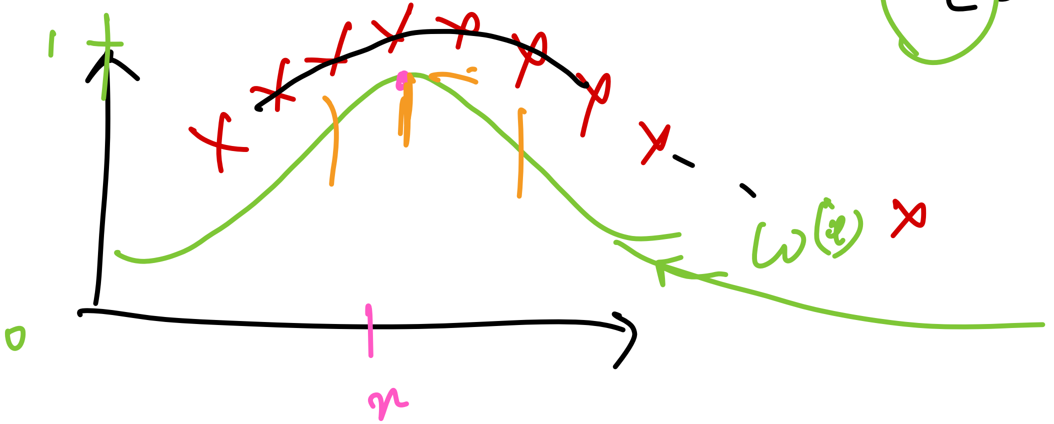
$$\left( \sum_{i=1}^{m} w^{(i)} \left( y^{(i)} - \theta^T x^{(i)} \right)^2 \right)$$

$$w(i) = \exp\left( - \frac{(x^{(i)} - x_0)^2}{2\tau^2} \right)$$

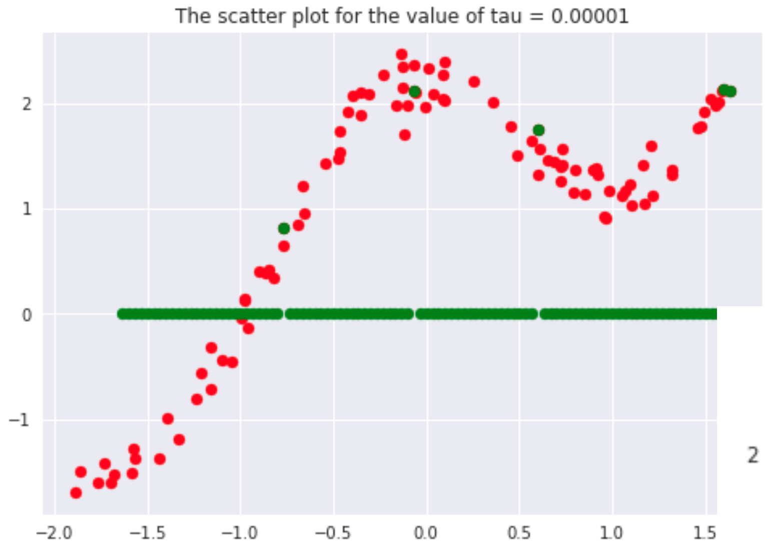
if $\underline{|x^i - x_0|}$ is small $\quad w^{(i)} \simeq 1$

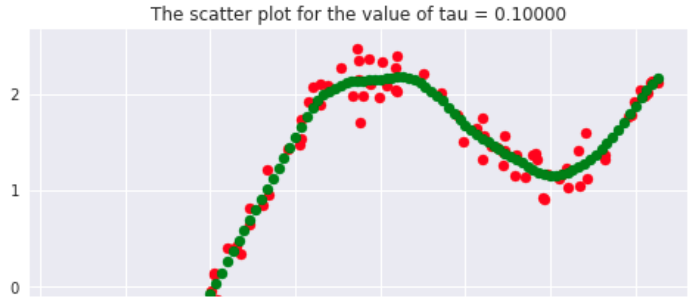
if $|x^i - x_0|$ is large $\quad w^{(i)} \simeq 0$

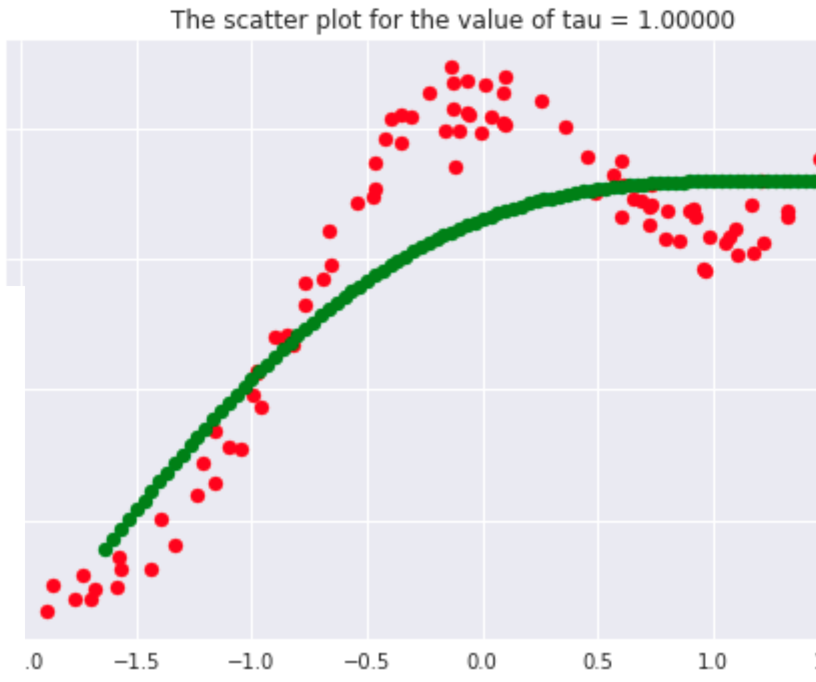
$\tau \Rightarrow$ hyperparameter

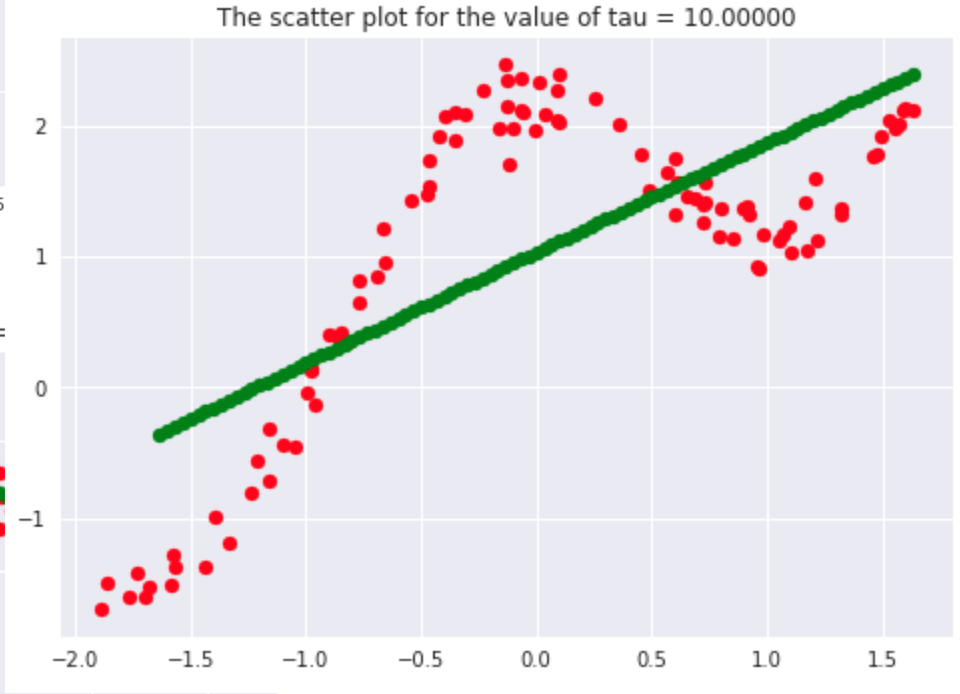


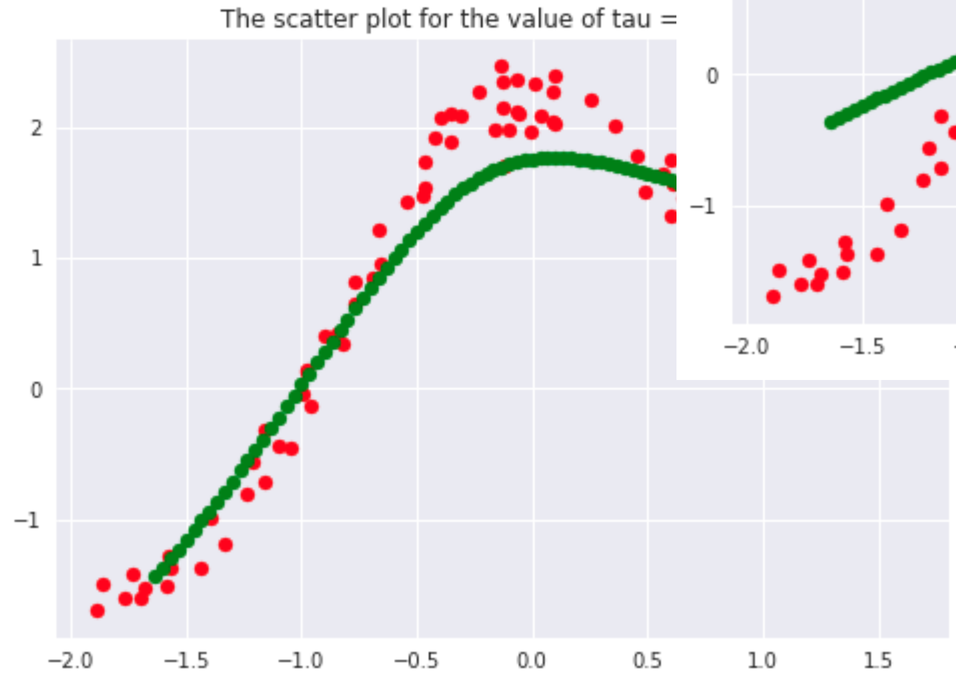
hyperparameter tuning


The scatter plot for the value of tau = 0.00001


The scatter plot for the value of tau = 0.10000

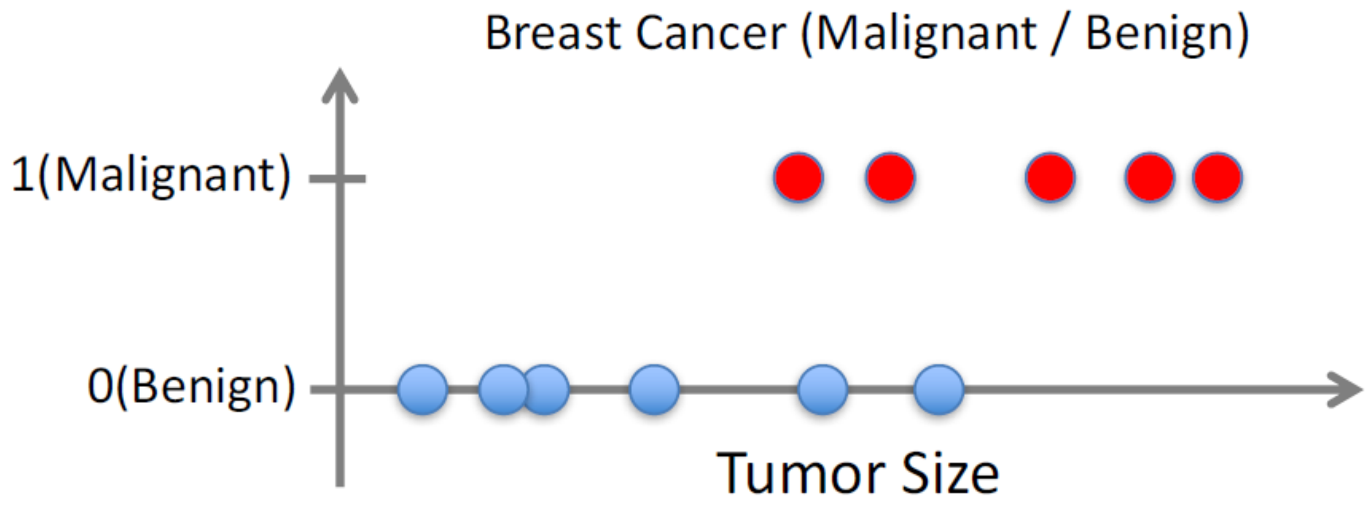

The scatter plot for the value of tau = 1.00000


The scatter plot for the value of tau = 10.00000


The scatter plot for the value of tau =

# Classification: Logistic Regression

- Given $(x_1, y_1)$, $(x_2, y_2)$, …, $(x_n, y_n)$
- Learn a function $f(x)$ to predict $y$ given $x$
  - $y$ is categorical == classification

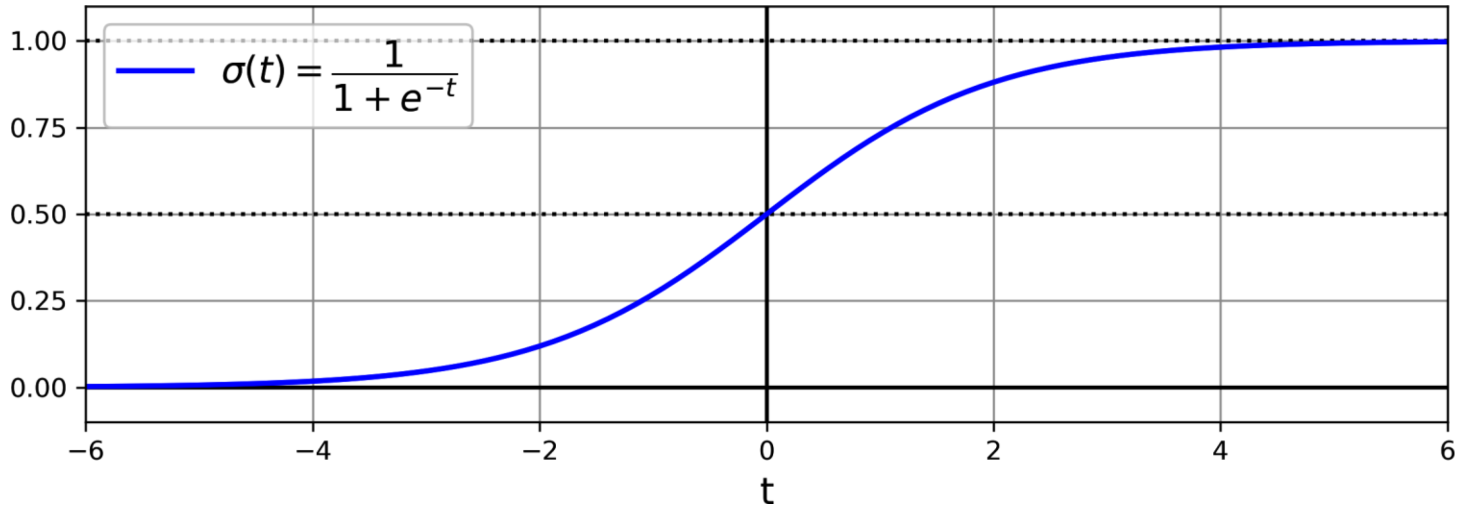


# Logistic Regression

- Classification Algorithm – Binary classifier
- If the probability is greater than 50% … then in Class A else Class B
- Logistic function

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$

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



# Logistic Regression Cost Function

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

- Cost function
- Cost function written as log loss
- To minimise log loss or cost function
- Maximise likelihood
- No closed form of the equation
- However, it is a convex equation with global minima

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

$$L(\theta) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(\frac{-(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)$$

likelihood of parameters; probability of data!

$$\ell(\theta) = \log(L(\theta))$$

$$L(\theta) = \boxed{m \log\left(\frac{1}{\sigma\sqrt{2\pi}}\right) + \underbrace{\boxed{\sum_{i=1}^{m} - \frac{(y^{(i)} - \theta^{(T)} x^{(i)})^2}{2\sigma^2}}}_{\text{MSE}}}$$

Gaussian

MSE

Probability

$\sigma, \mu$

$\downarrow$

distribution

$P(X \mid \theta)$

Likelihood

$\downarrow$

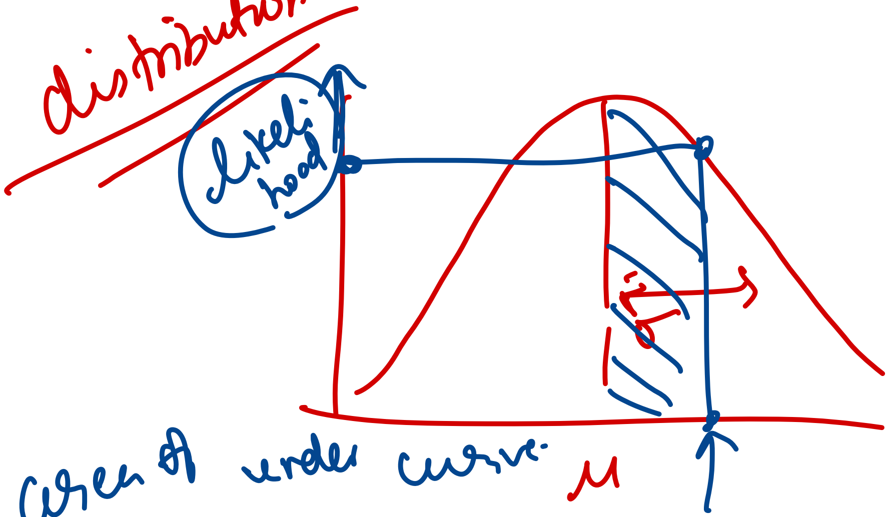
given data set

$\downarrow$

$\sigma, \mu$

estimate parameter

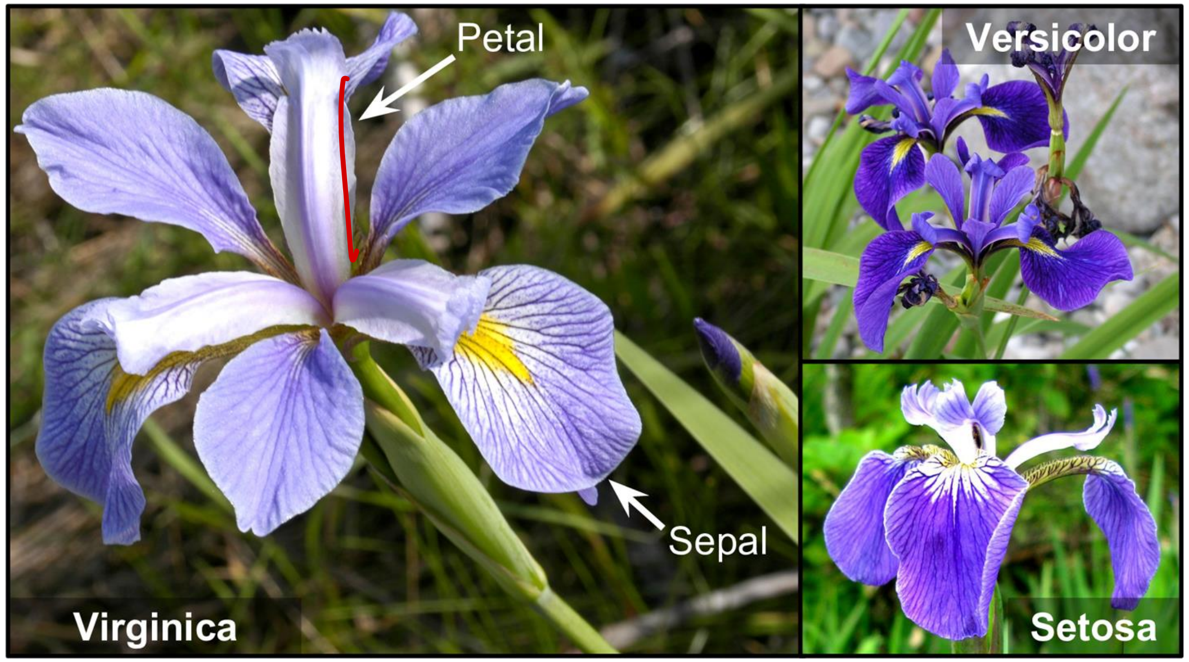
$L(\mu, \sigma \mid X)$

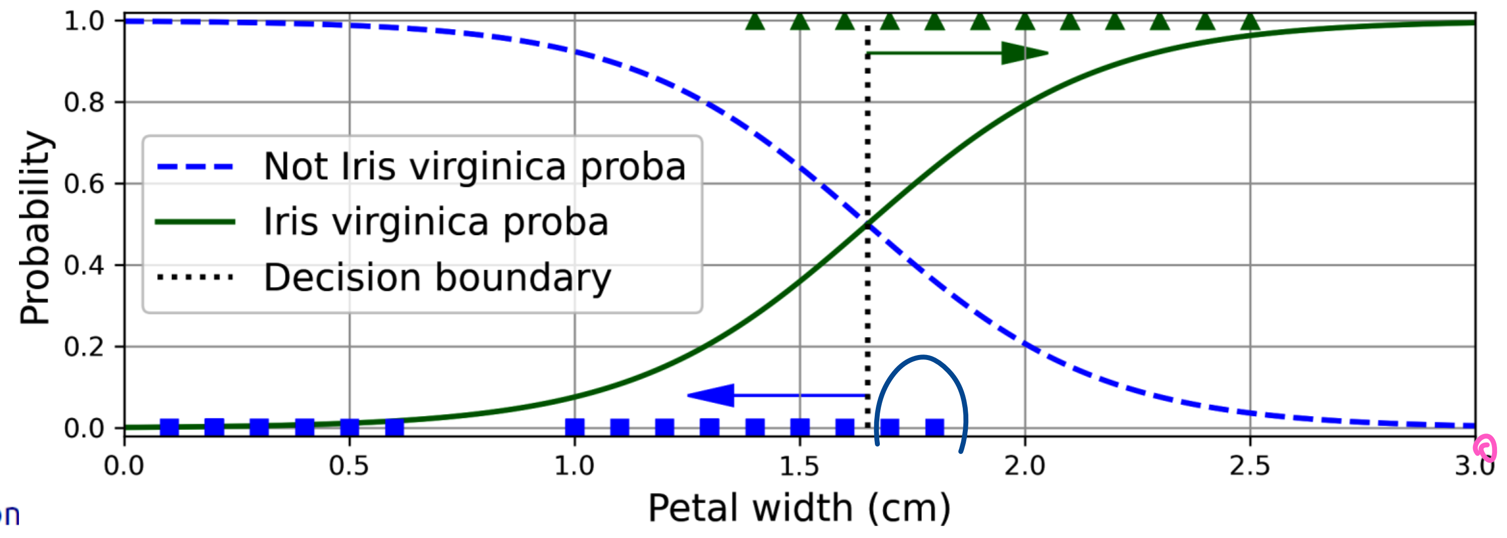


# Decision Boundaries

This is a famous dataset that contains the sepal and petal length and width of 150 iris flowers of three different species: *Iris setosa*, *Iris versicolor*, and *Iris virginica*

```
>>> from sklearn.datasets import load_iris
>>> iris = load_iris(as_frame=True)
>>> list(iris)
['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names',
 'filename', 'data_module']
>>> iris.data.head(3)
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
>>> iris.target.head(3)  # note that the instances are not shuffled
0    0
1    0
2    0
Name: target, dtype: int64
>>> iris.target_names
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```



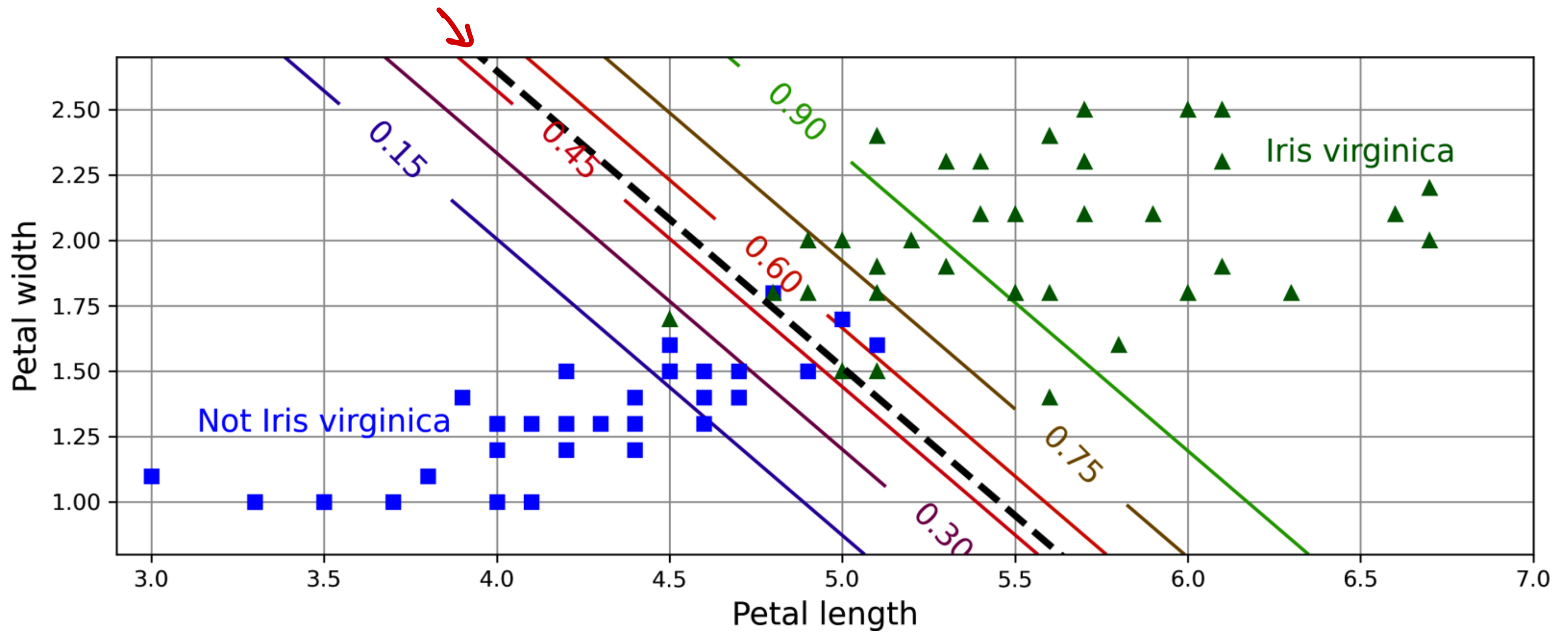


```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

X = iris.data[["petal width (cm)"]].values
y = iris.target_names[iris.target] == 'virginica'
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

log_reg = LogisticRegression(random_state=42)
log_reg.fit(X_train, y_train)
```

```python
X_new = np.linspace(0, 3, 1000).reshape(-1, 1)  # reshape to get a column vector
y_proba = log_reg.predict_proba(X_new)
decision_boundary = X_new[y_proba[:, 1] >= 0.5][0, 0]

plt.plot(X_new, y_proba[:, 0], "b--", linewidth=2,
         label="Not Iris virginica proba")
plt.plot(X_new, y_proba[:, 1], "g-", linewidth=2, label="Iris virginica proba")
plt.plot([decision_boundary, decision_boundary], [0, 1], "k:", linewidth=2,
         label="Decision boundary")
[...] # beautify the figure: add grid, labels, axis, legend, arrows, and samples
plt.show()
```

```
>>> decision_boundary
1.6516516516516517
>>> log_reg.predict([[1.7], [1.5]])
array([ True, False])
```

## 2 Features



Once trained, the logistic regression classifier can, based on these two features, estimate the probability that a new flower is an *Iris virginica*. The dashed line represents the points where the model estimates a 50% probability: this is the model's decision boundary. Note that it is a linear boundary. Each parallel line represents the points where the model outputs a specific probability, from 15% (bottom left) to 90% (top right). All the flowers beyond the top-right line have over 90% chance of being *Iris virginica*, according to the model.

## Multi-class Classifier

$$\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp\left(s_k(\mathbf{x})\right)}{\sum_{j=1}^{K} \exp\left(s_j(\mathbf{x})\right)}$$

- Soft-max regression

In this equation:

- $K$ is the number of classes.
- $\mathbf{s}(\mathbf{x})$ is a vector containing the scores of each class for the instance $\mathbf{x}$.
- $\sigma(\mathbf{s}(\mathbf{x}))_k$ is the estimated probability that the instance $\mathbf{x}$ belongs to class $k$, given the scores of each class for that instance.

```python
X = iris.data[["petal length (cm)", "petal width (cm)"]].values
y = iris["target"]
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

softmax_reg = LogisticRegression(C=30, random_state=42)
softmax_reg.fit(X_train, y_train)
```

So the next time you find an iris with petals that are 5 cm long and 2 cm wide, you can ask your model to tell you what type of iris it is, and it will answer *Iris virginica* (class 2) with 96% probability (or *Iris versicolor* with 4% probability):

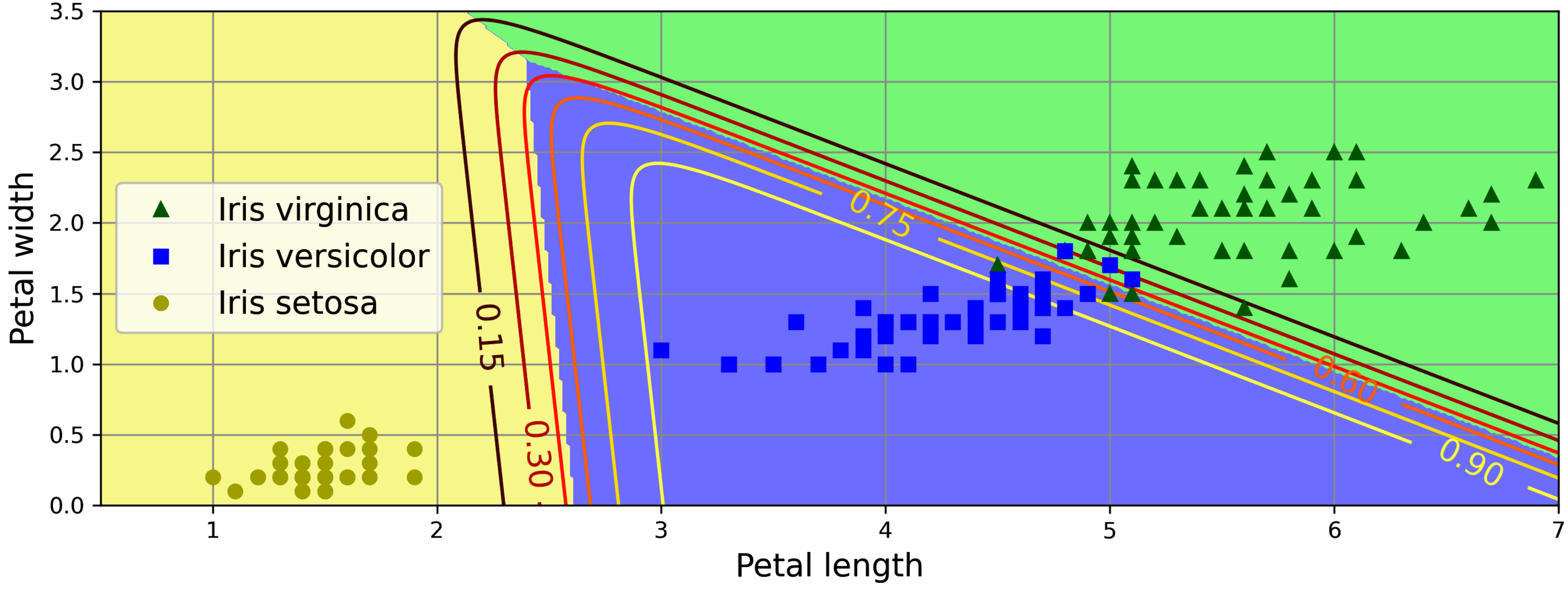
```python
>>> softmax_reg.predict([[5, 2]])
array([2])
>>> softmax_reg.predict_proba([[5, 2]]).round(2)
array([[0.  , 0.04, 0.96]])
```

$$\hat{y} = \underset{k}{\operatorname{argmax}}\ \sigma(\mathbf{s}(\mathbf{x}))_k = \underset{k}{\operatorname{argmax}}\ s_k(\mathbf{x}) = \underset{k}{\operatorname{argmax}}\ \left(\left(\boldsymbol{\theta}^{(k)}\right)^{\intercal}\mathbf{x}\right)$$

$$J(\boldsymbol{\Theta}) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right)$$

In this equation, $y_k^{(i)}$ is the target probability that the $i^{\text{th}}$ instance belongs to class $k$. In general, it is either equal to 1 or 0, depending on whether the instance belongs to the class or not.

Cross-entropy is a concept used in information theory and machine learning to measure the difference between two probability distributions or, more commonly, between the predicted probability distribution and the actual probability distribution of the target variable. It is widely used as a loss function in classification problems.



## Softmax Regression

Prediction

$$\hat{y} = \underset{k}{\operatorname{argmax}}\ \sigma(\mathbf{s}(\mathbf{x}))_k = \underset{k}{\operatorname{argmax}}\ s_k(\mathbf{x}) = \underset{k}{\operatorname{argmax}}\ \left(\left(\boldsymbol{\theta}^{(k)}\right)^\mathsf{T}\mathbf{x}\right)$$

- Cross Entropy Function

$$J(\boldsymbol{\Theta}) = -\frac{1}{m}\Sigma_{i=1}^{m}\Sigma_{k=1}^{K} y_k^{(i)}\log\left(\hat{p}_k^{(i)}\right)$$

In this equation, $y_k^{(i)}$ is the target probability that the $i^{\text{th}}$ instance belongs to class $k$. In general, it is either equal to 1 or 0, depending on whether the instance belongs to the class or not.

Cross Entropy gradient vector for class k

$$\nabla_{\boldsymbol{\theta}^{(k)}} J(\boldsymbol{\Theta}) = \frac{1}{m}\sum_{i=1}^{m}\left(\hat{p}_k^{(i)} - y_k^{(i)}\right)\mathbf{x}^{(i)}$$

Another way of looking at softmax regression is through neural networks … later on!