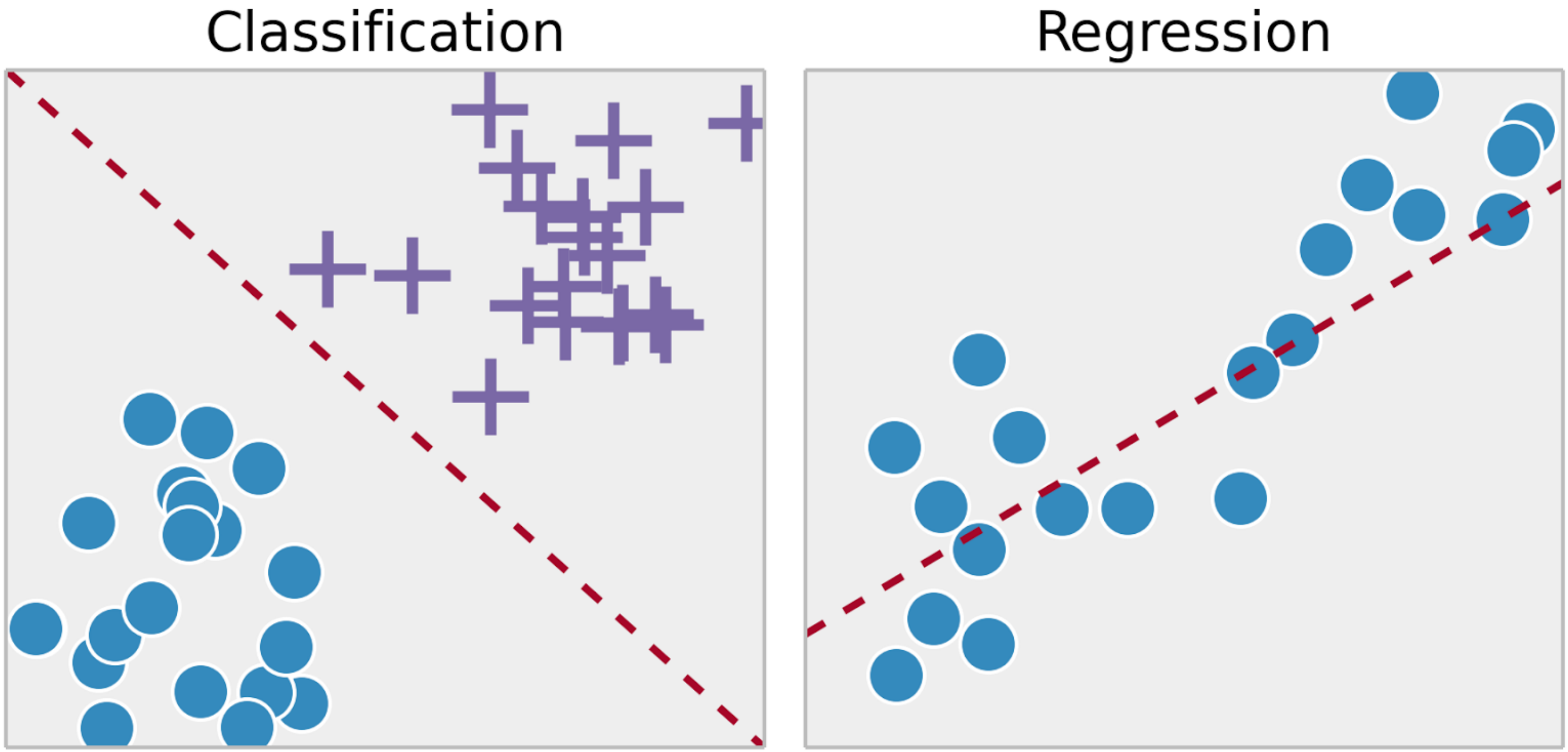
Lecture 2

Logistic Regression

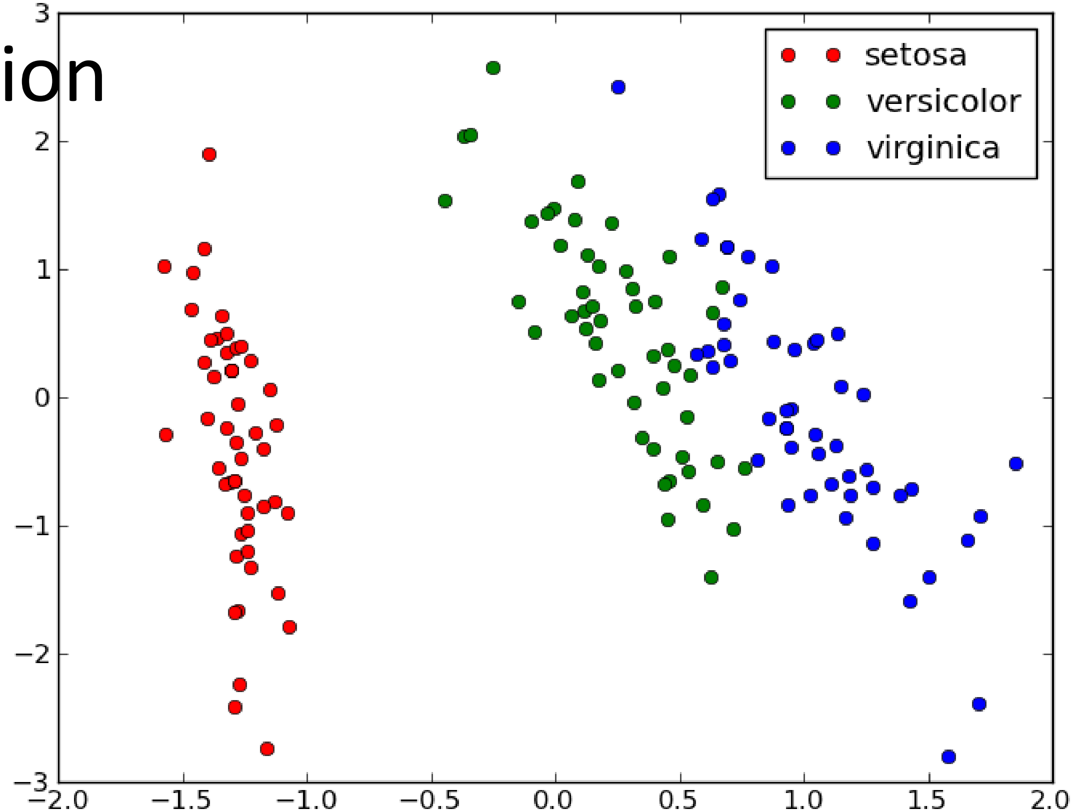
By: Sultanova Nazerke

Supervised Machine Learning Algorithms



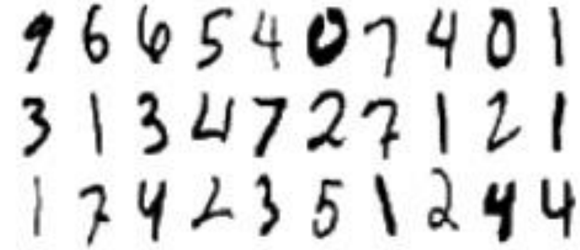
Classification Problems:

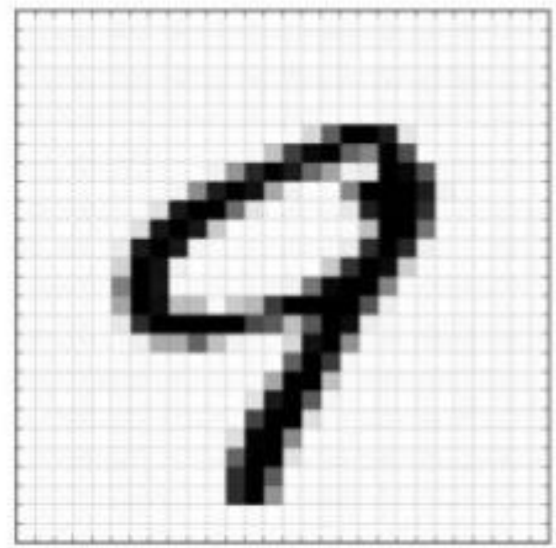
- Spam/non-spam
- Cancer/non-cancer
- Object classification



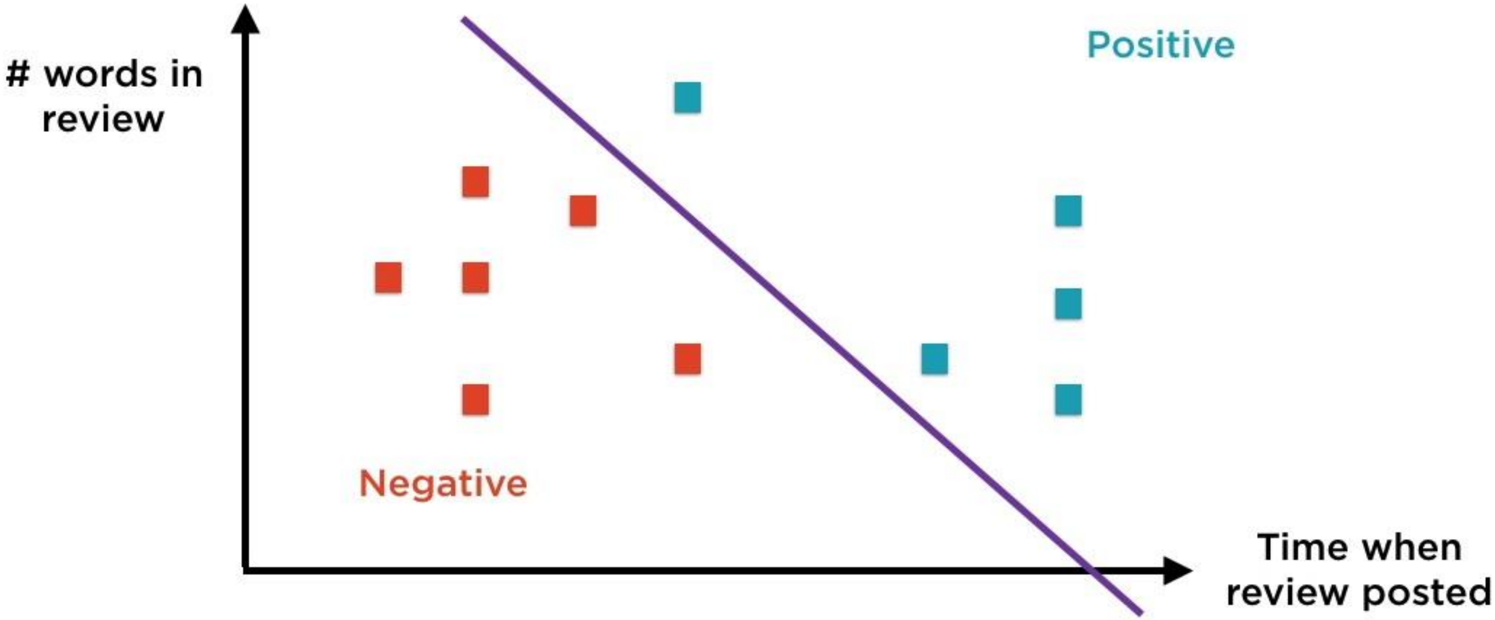
Example: Image Classification

- Classify handwritten digits with ML models
- Each input is an entire image
- Output is digit in the image





Classification



Ideally, data is linearly separable - hard decision boundary

ML Classification algorithms:

- Perceptron
- Logistic Regression
- SVM
- Naïve Bayes
- Decision Trees
- kNN
- Random Forests
- And much more..

Binary classification

- Spam/non spam:

$y \in \{0, 1\}$

0: "negative class" (e.g. not spam)

1: "positive class" (e.g. spam)

Logistic Regression

- Classification Algorithm
- Designed for binary classification, but can be extend to multiclass
- Uses odds ratio and logit function

Odds ratio and Logit function

- Odds ratio = p/(1-p)
- p is probability of positive event
- Logit function is log(p/(1-p))

Logistic function - sigmoid

$$S(z) = \frac{1}{1 + e^{-z}}$$

Note

- $s(z)$ = output between 0 and 1 (probability estimate)
- z = input to the function (your algorithm's prediction e.g. mx + b)
- e = base of natural log

Logistic function - sigmoid

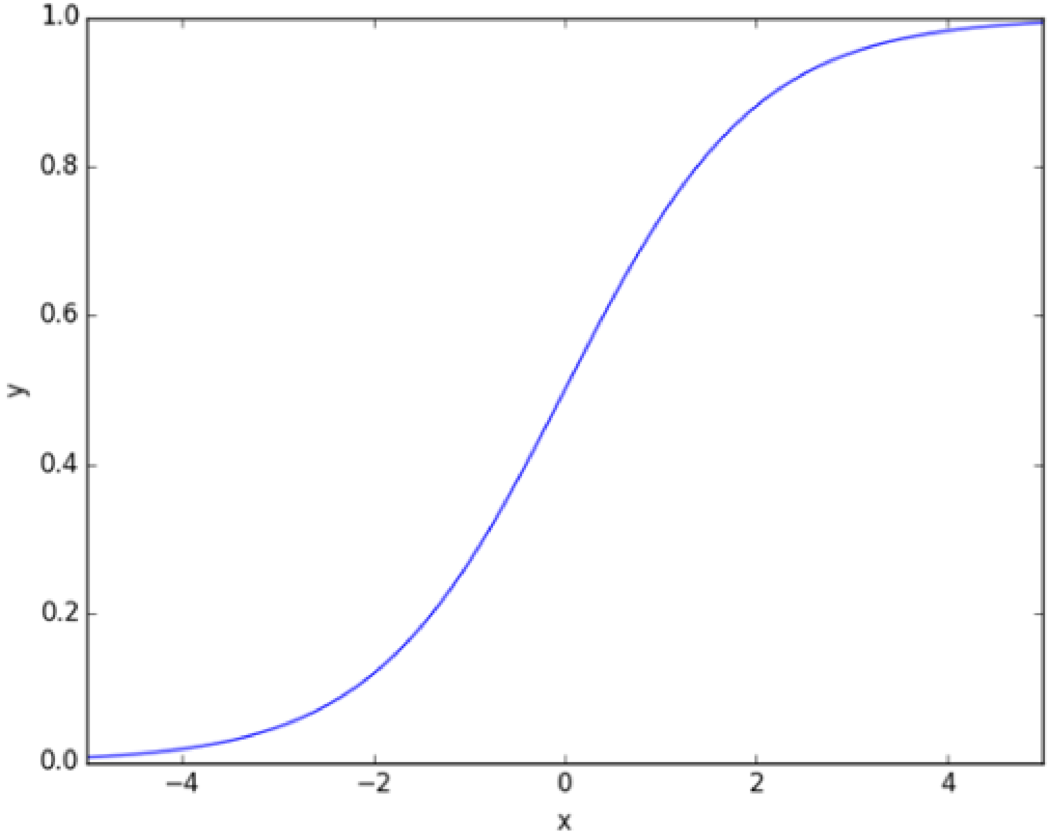
$$S(z) = \frac{1}{1 + e^{-z}}$$

Note

- $s(z)$ = output between 0 and 1 (probability estimate)
- z = input to the function (your algorithm's prediction e.g. mx + b)
- e = base of natural log

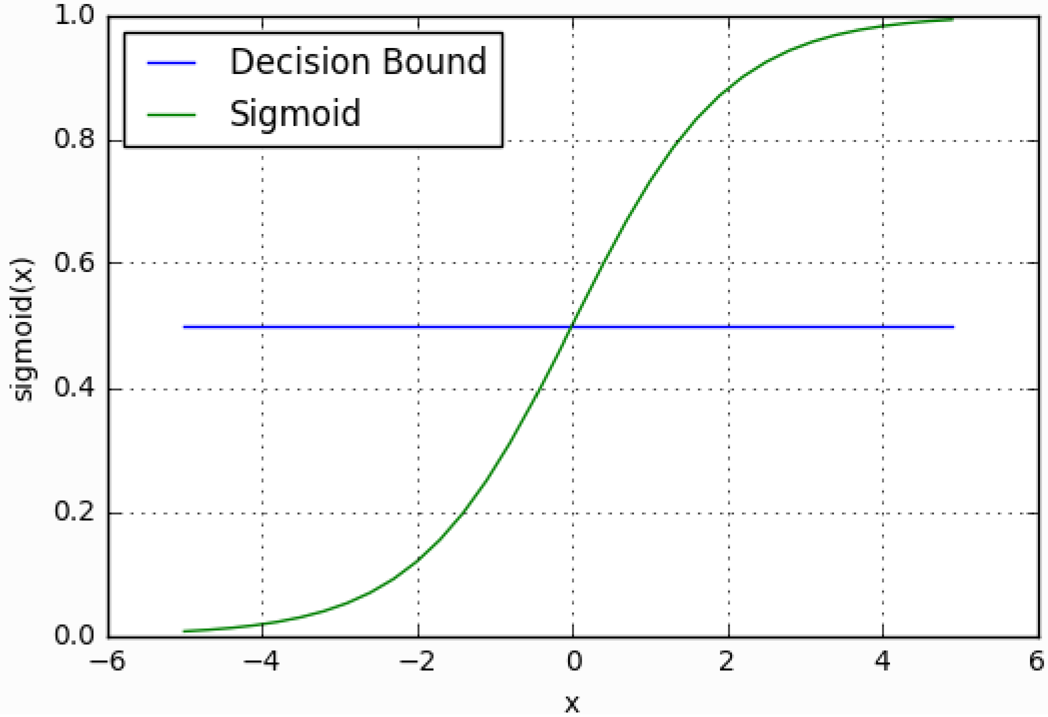
Our new hypothesis!!!

Sigmoid function

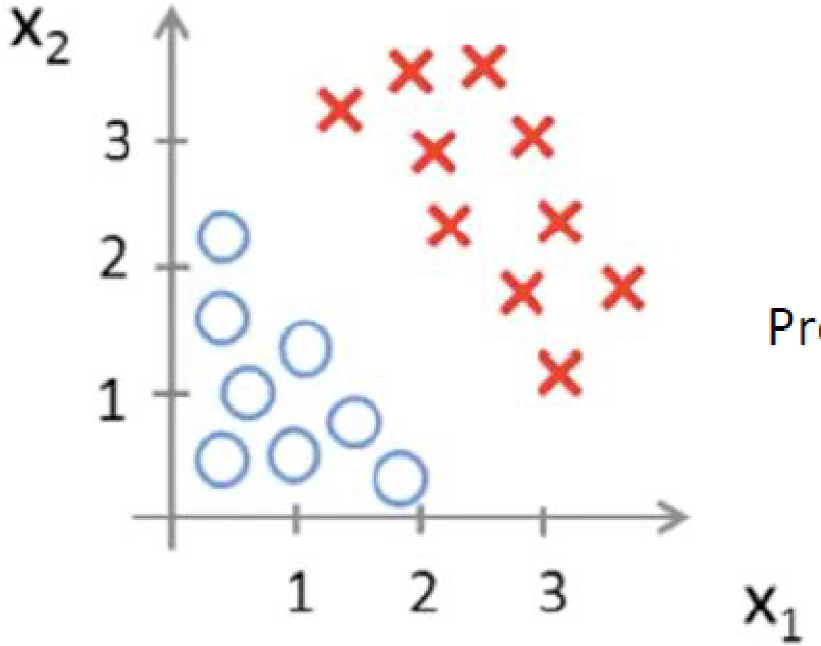


How to apply sigmoid function?

$$p \geq 0.5, class = 1$$
$$p < 0.5, class = 0$$



Decision boundary



$$z = w_0 + w_1 x_1 + w_2 x_2$$

Predict "y = 1" if $-3 + x_1 + x_2 \geq 0$

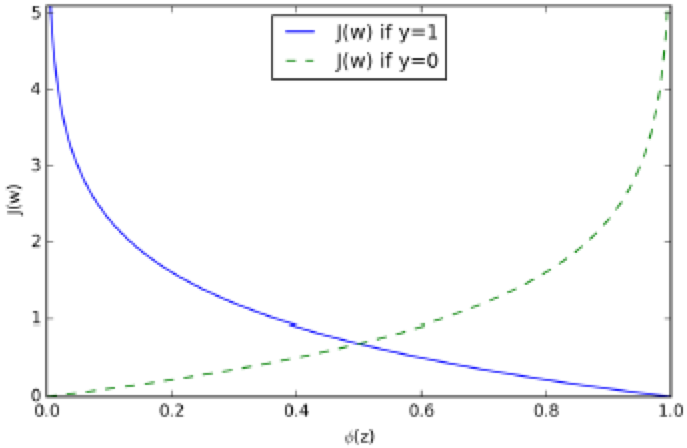
Cost function

Log-likelihood

$$J(w) = \sum_{i=1}^{n} \left[-y^i \log\left(\phi(z^i)\right) - \left(1 - y^i\right) \log(1 - \phi(z^i)) \right]$$

Learning the weights of the logistic cost function

```
def cost_1(z):
    return - np.log(sigmoid(z))


def cost_0(z):
    return - np.log(1 - sigmoid(z))

z = np.arange(-10, 10, 0.1)
phi_z = sigmoid(z)

c1 = [cost_1(x) for x in z]
plt.plot(phi_z, c1, label='J(w) if y=1')

c0 = [cost_0(x) for x in z]
plt.plot(phi_z, c0, linestyle='--', label='J(w) if y=0')

plt.ylim(0.0, 5.1)
plt.xlim([0, 1])
plt.xlabel('$\phi$(z)')
plt.ylabel('J(w)')
plt.legend(loc='best')
plt.tight_layout()
# plt.savefig('./figures/log_cost.png', dpi=300)
plt.show()
```



Gradient Descent

$$s'(z) = s(z)(1 - s(z))$$

Which leads to an equally beautiful and convenient cost function derivative:

$$C' = x(s(z) - y)$$

Note

- C' is the derivative of cost with respect to weights
- y is the actual class label (0 or 1)
- $s(z)$ is your model's prediction
- x is your feature or feature vector.

Gradient Descent pseudocode

Repeat until convergence {

1. calculate gradient average (just like in linear regression)
2. multiply by learning rate
3. subtract from weights (thetas)

}

How to know our algorithm works well?

- Test it!
- Calculate accuracy

accuracy = (number of correctly labeled)/(all)

Overfitting

- Sometimes model performs well on training data but does not generalize well to unseen data (test data)
- This is overfitting
- If a model suffers from overfitting, the model has a high variance
- This is often caused by a model that's too complex
- Underfitting can also occur (high bias)
- Underfitting is caused by a model's not being complex enough
- Both suffer from low performance on unseen data

Bias-variance tradeoff

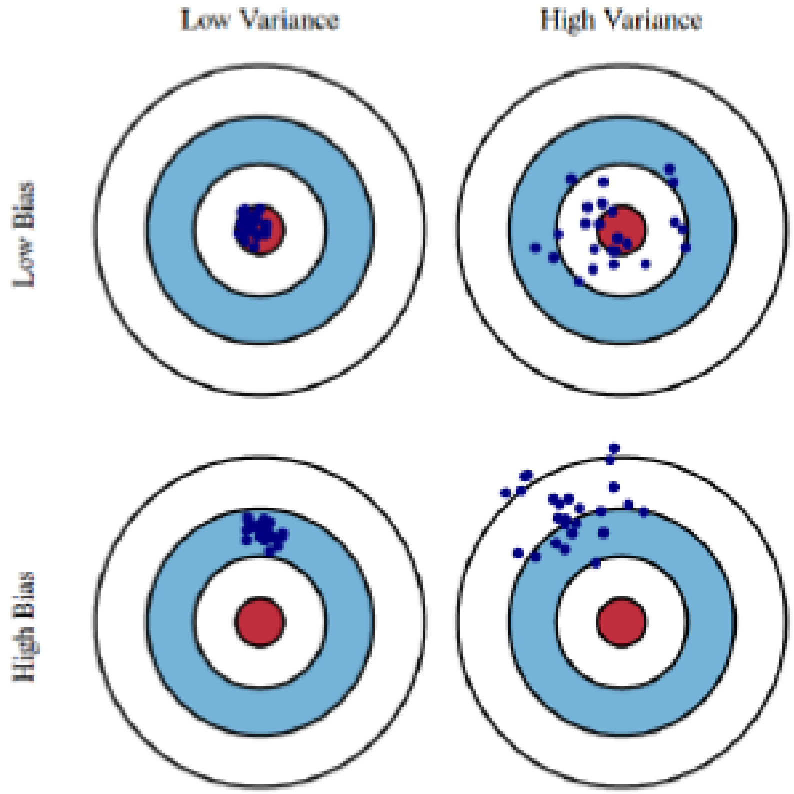
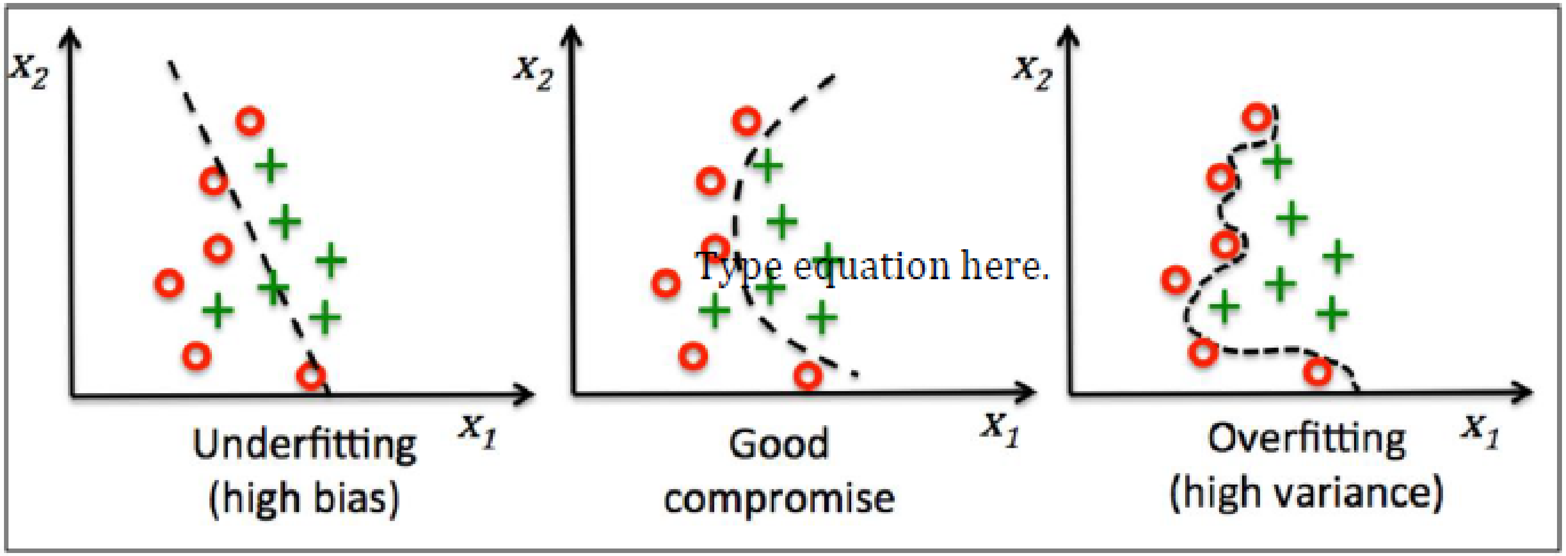


Fig. 1 Graphical illustration of bias and variance.

How to prevent overfitting??

Regularization



- Regularization is a way to tune the complexity of the model
- Regularization helps to filter out noise from training data
- As a result, regularization prevents overtting

Regularization

- Regularization is a very useful method to handle collinearity (high correlation among features), filter out noise from data, and eventually prevent overfitting.
- The most common form of regularization is the so-called L2 regularization:

$$\frac{\lambda}{2}\|\boldsymbol{w}\|^2 = \frac{\lambda}{2}\sum_{j=1}^{m} w_j^2$$

Regularization

- Where λ is the so-called regularization parameter. To apply regularization, we add the regularization term to the cost function, which shrinks the weights:

$$J(\boldsymbol{w}) = \sum_{i=1}^{n} \left[-y^i \log\left(\phi(z^i)\right) - (1 - \phi(z^i)) \right] + \frac{\lambda}{2} \|\boldsymbol{w}\|^2$$

Regularization parameter

- We control how well we fit the training data via the regularization parameter λ
- By increasing λ, we increase the strength of regularization
- Sometimes (e.g in scikit-learn), SVM terminology is used

$$C = \frac{1}{\lambda}$$

- i.e. we rewrite the regularized cost function of logistic regression:

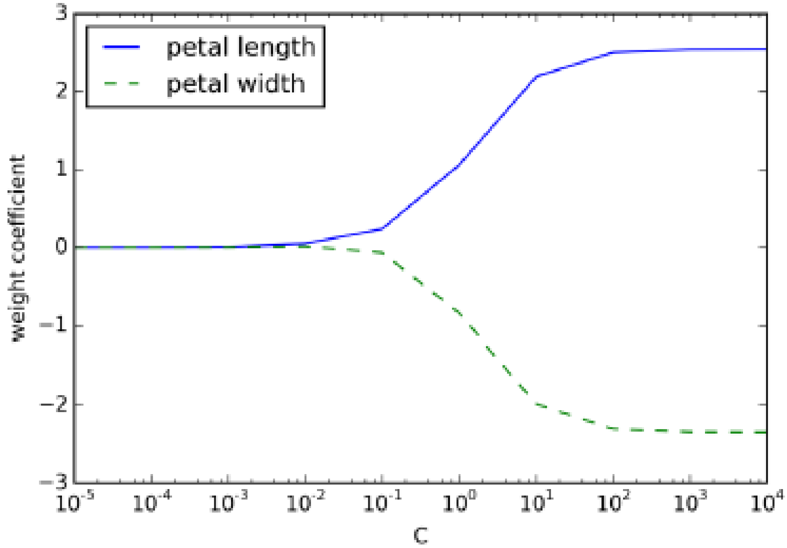
$$C\left[\sum_{i=1}^{n}\left(-y^{i}\log\left(\phi(z^{i}) - (1-y^{i})\right)\right)\log(1-\phi(z^{i}))\right] + \frac{\lambda}{2}\|\boldsymbol{w}\|^{2}$$

Visualization of regularization

- Decreasing the value of C means increasing the regularization strength
- Can be visualized by plotting L2 regularization path for two weights
- Display weights across multiple C values
- As you see, weights shrink to zero as C decreased

Visualization of regularization

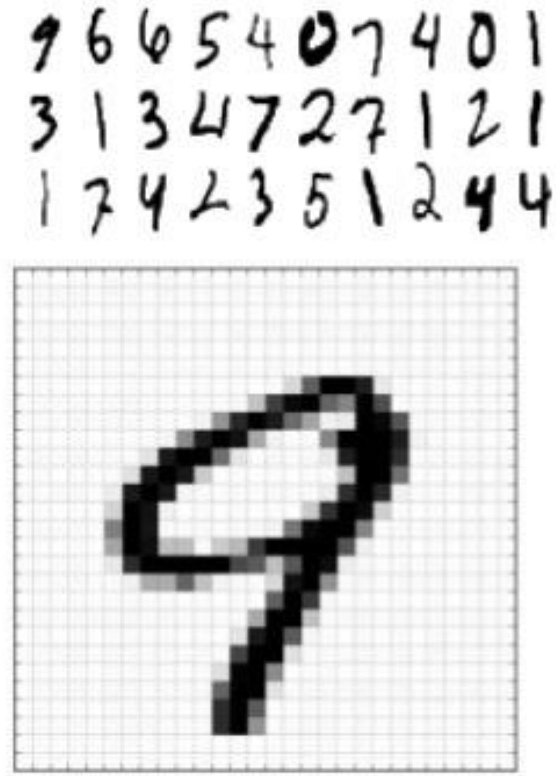
```
weights, params = [], []
for c in np.arange(-5., 5.):
    lr = LogisticRegression(C=10.**c, random_state=0)
    lr.fit(X_train_std, y_train)
    weights.append(lr.coef_[1])
    params.append(10**c)

weights = np.array(weights)
plt.plot(params, weights[:, 0],
         label='petal length')
plt.plot(params, weights[:, 1], linestyle='--',
         label='petal width')
plt.ylabel('weight coefficient')
plt.xlabel('C')
plt.legend(loc='upper left')
plt.xscale('log')
# plt.savefig('./figures/regression_path.png', dpi=300)
plt.show()
```



Assignment 3

Example: Image Classification

- Classify handwritten digits with ML models
- Each input is an entire image
- Output is digit in the image



Assignment Description

For a full grade you need:

1. Read dataset and visualize one randomly
2. Hypothesis function
3. Cost function
4. Gradient Descent
5. One vs All (Multiclass classification)
6. Calculate accuracy

Notes for Assignment

- Use vectorized forms –np.arrays because here you have 401 features
- Avoid loops
- To read dataset use:

  from scipy.io import loadmat

  x = loadmat('test.mat')

- **Start earlier!!! No late assignments**