ML as Function Approximation

Problem Setting
- Set \( S \) possible inputs, \( \mathbf{X} \) (all possible feature vectors)
- Set \( S \) possible outputs, \( \mathbf{Y} \) (all possible labels)
- Unknown target function, \( \mathbf{c}^* : \mathbf{X} \rightarrow \mathbf{Y} \)
- Set \( S \) candidate hypotheses
  \[ \mathcal{H} = \{ \mathbf{h} : \mathbf{X} \rightarrow \mathbf{Y} \} \]

Learner is given:
- Training examples \( \mathcal{D}_{\text{train}} = \{ (\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots, (\mathbf{x}^{(m)}, y^{(m)}) \} \)
  of unknown target function \( y^{(i)} = c^*(\mathbf{x}^{(i)}) \)
- \( N = \# \) \( \mathcal{D}_{\text{train}} \) examples, \( M = \# \) \( \mathcal{H} \) features = \( |\mathbf{x}^{(i)}| \)

Learner produces:
- Hypothesis \( \mathbf{h} \in \mathcal{H} \) that "best" approximates \( \mathbf{c}^* \)

To Evaluate:
- Loss function \( L : \mathbf{Y} \times \mathbf{Y} \rightarrow \mathbb{R} \) measures how "bad"
  predictions \( \hat{y} = h(\mathbf{x}) \) are compared to \( c^*(\mathbf{x}) \)
- Another dataset \( \mathcal{D}_{\text{test}} = \{ (\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(n)}, y^{(n)}) \} \)
- Evaluate the average loss of \( h(\mathbf{x}) \) on \( \mathcal{D}_{\text{test}} \)

---

Algorithms for Classification

**Alg 1.** Majority Vote Algorithm

```python
def train(D):
    store v = majority vote(D)
    = the class y \( \in \mathbf{Y} \) that appears most often in D

def h(x):
    return v

def predict(D_test):
    for (x(i), y(i)) in D_test:
        y(i) = h(x(i))  # reuse this for any classifier
```
for $(x_i, y_i) \in D_{best}$:  
\[ y_i = h(x_i) \]

**Algorithm 2** Memorizer

```python
def train(D):
    store dataset D

def h(x):
    if $\exists x_i \in D$ s.t. $x_i = x$:
        return $y_i$
    else
        return $y \in Y$ randomly
```

**Algorithm 3** Decision Stump

```python
def train(D):
    1. Pick an attribute, \( m \)
    2. Divide dataset \( D \) on \( x_m \)
       \[ D^{(0)} = \{(x_i, y_i) \in D \mid x_m = 0\} \]
       \[ D^{(1)} = \{(x_i, y_i) \in D \mid x_m = 1\} \]
    3. Two votes
       \[ v^{(0)} = \text{majority_vote}(D^{(0)}) \]
       \[ v^{(1)} = \text{majority_vote}(D^{(1)}) \]

    def h(x):
        if $x_m = 0$: return $v^{(0)}$
        if $x_m = 1$: return $v^{(1)}$
```

**Algorithm 4** Decision Tree