CSE7850/CX4803 Machine Learning in Computational Biology

Lecture 6: Classification & ML Toolbox

Yunan Luo

Acknowledgements: Some of the slides were adapted from lectures of Jian Peng and Volodymyr Kuleshov

Classification

Predicting diabetes risk: Regression

Let's start with a simple example of machine learning problem: **predicting diabetes risk**.

We start with a dataset of diabetes patients.

- For each patient we have a access to their **BMI** (body mass index) and an estimate of **diabetes risk** (from 0-400).
- We are interested if we can ***predict*** an individual's diabetes risk based on the BMI

Individual	BMI	Risk
1	27	233
2	24	91
3	25	111
…	…	…

Predicting diabetes risk: Classification

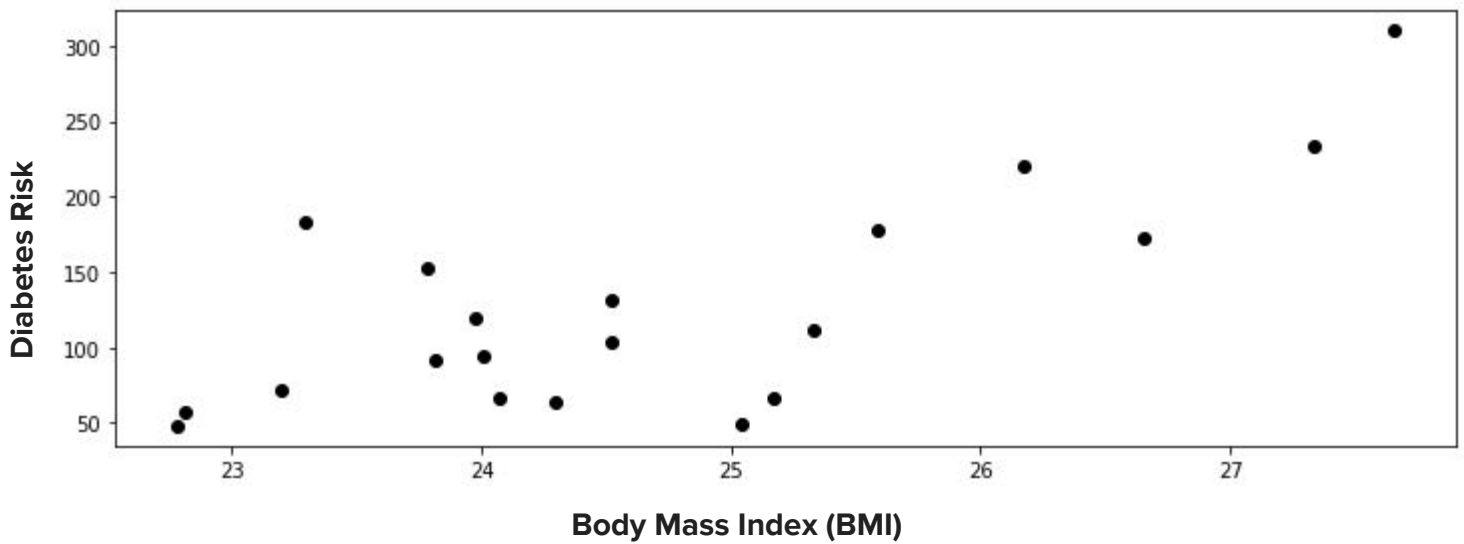
Let's start with a simple example of machine learning problem: **predicting diabetes risk**.

We start with a dataset of diabetes patients.

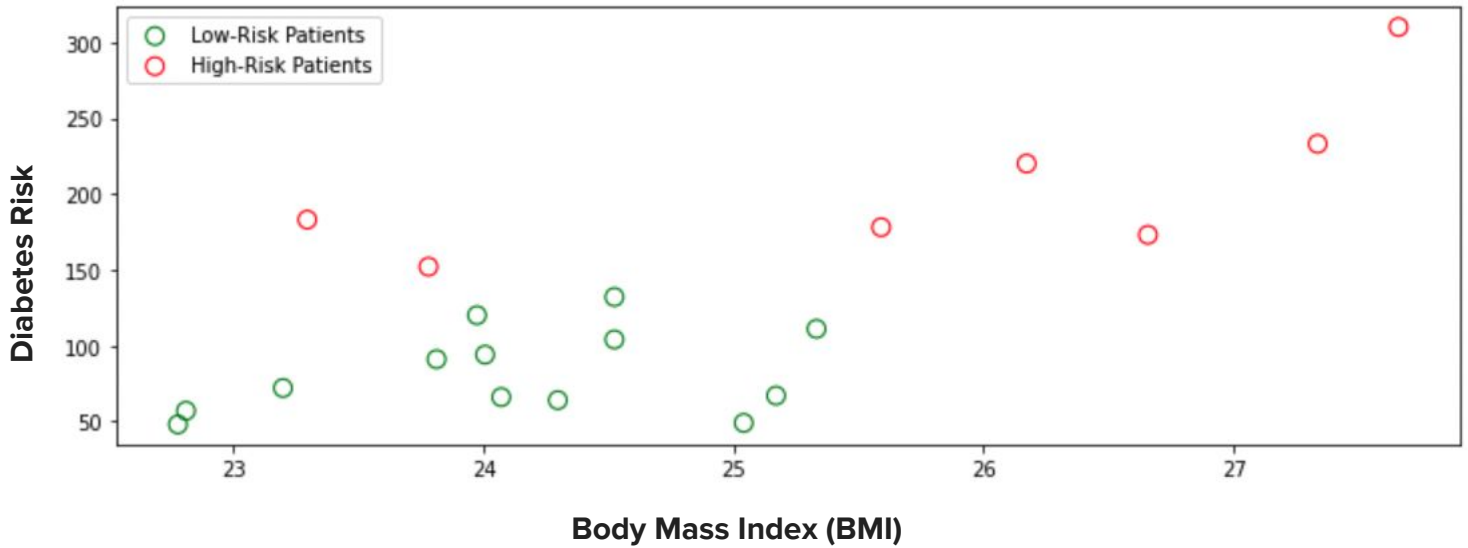
- For each patient we have a access to their **BMI** (body mass index) and an estimate of **diabetes risk** ~~(from 0-400).~~ Low (0-150) or High (150-400).
- We are interested if we can ***predict*** an individual's diabetes risk based on the BMI

Individual	BMI	Risk	Risk
1	27	233	High
2	24	91	Low
3	25	111	Low
…	…	…	…

Regression


Diabetes Risk
300
250
200
150
100
50
23
24
25
26
27
Body Mass Index (BMI)

Classification



Regression vs. Classification

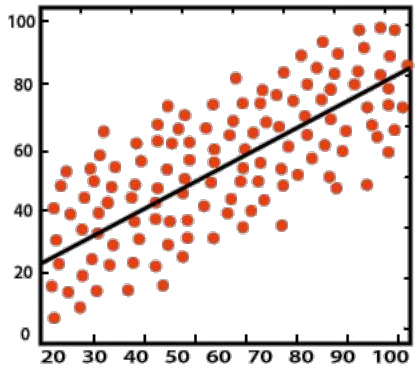
Consider a training dataset $\mathcal{D} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})\}$.

We distinguish between two types of supervised learning problems depnding on the targets $y^{(i)}$.

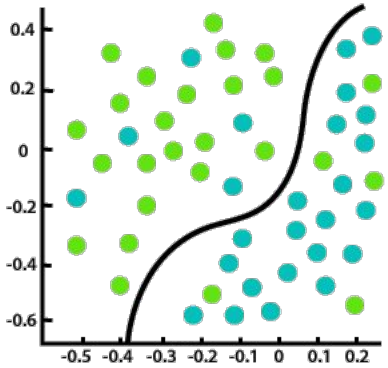
1. **Regression**: The target variable $y \in \mathcal{Y}$ is continuous: $\mathcal{Y} \subseteq \mathbb{R}$.

2. **Classification**: The target variable y is discrete and takes on one of K possible values: $\mathcal{Y} = \{y_1, y_2, \ldots y_K\}$.
   Each discrete value corresponds to a *class* that we want to predict.

Goal of Regression vs. Classification



Regression

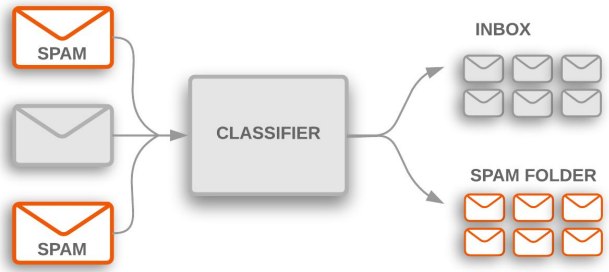


Classification

- **Regression**: Fitting a **curve** in a high-dimensional feature space that approximates the shape of the dataset.
- **Classification**: Finding the **boundaries** that separate different classes in the feature space

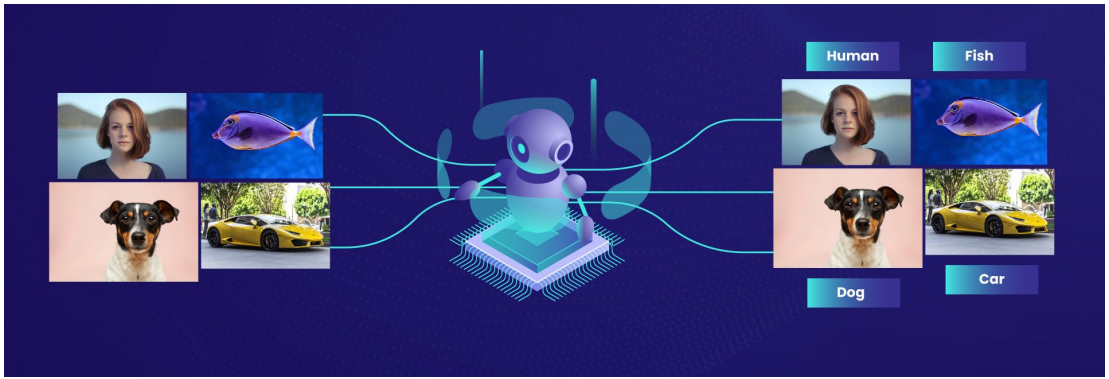
More examples of classification problem

Spam classification



Binary classification: two categories

Image classification



Multi-class classification: >2 categories

image source

More examples of classification problem

Binary Classification

- Spam
- Not spam

Multiclass Classification

- Dog
- Cat
- Horse
- Fish
- Bird
- ...

Multi-label Classification

- Dog
- Cat
- Horse
- Fish
- Bird
- ...

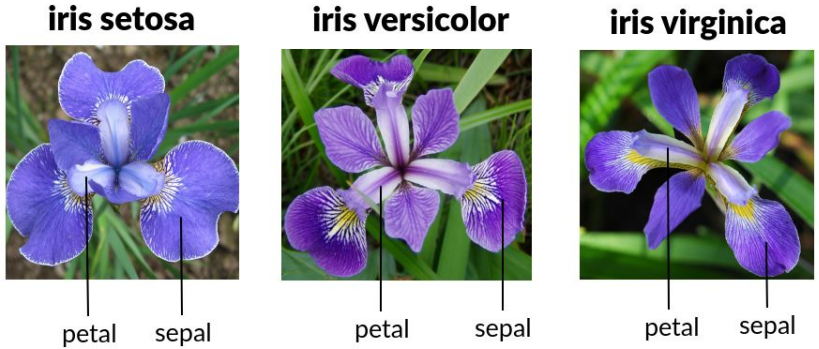
image source

Example: Iris Flowers Classification

Ronald Fisher
(1890-1962)

Mathematician, statistician, biologist, and geneticist

- To demonstrate classification algorithms, we are going to use the Iris flower dataset.
- It's a classical dataset originally published by R. A. Fisher in 1936. Nowadays, it's widely used for demonstrating machine learning algorithms.



	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

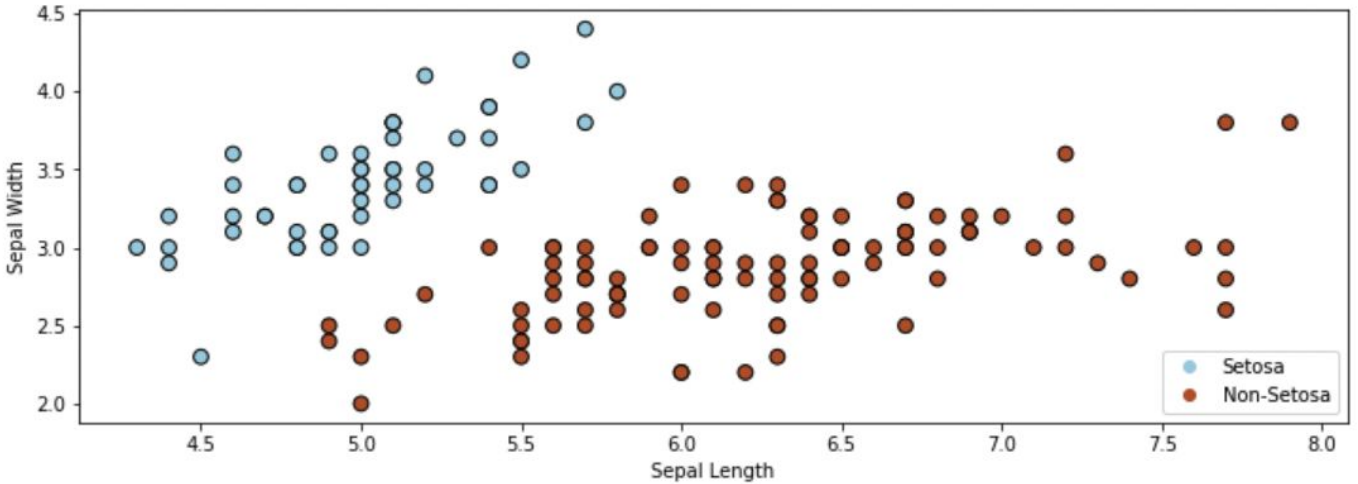
- Goal: Predict iris subtypes based on features

Binary classification

We will to start by looking at binary (two-class) classification.

To keep things simple, we will use the Iris dataset. We will attempt to distinguish class 0 (Iris Setosa) from the other two classes.

We only use the first two features in the dataset. Our task is to tell apart Setosa flowers from non-Setosa flowers.



Binary classification

- In this example, we have two classes, i.e., $\mathcal{Y} = \{0, 1\}$

- Can we solve it using linear regression?

$$f(x) = \sum_{j=0}^{d} \theta_j \cdot x_j = \theta^\top x.$$

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} (y^{(i)} - \theta^\top x^{(i)})^2$$

- It is easy to construct a classification task where linear regression performs very poorly
  - Intuitively, it also doesn't make sense for f(x) to take values larger than 1 or smaller than 0 when we know that $y \in \{0, 1\}$

The Logistic Function

To address the fact that the output of linear regression is not in $[0, 1]$, we will instead attempt to *squeeze* it into that range using

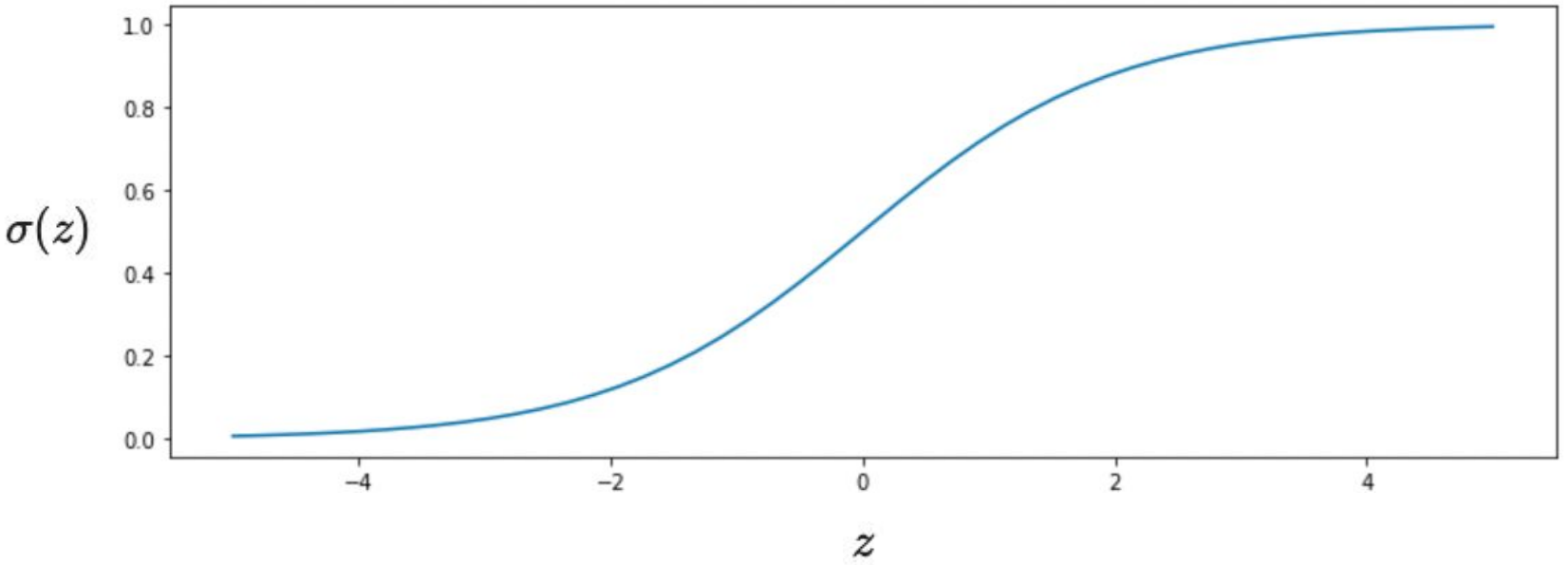
$$\sigma(z) = \frac{1}{1 + \exp(-z)}.$$

This is known as the *sigmoid* or *logistic* function.

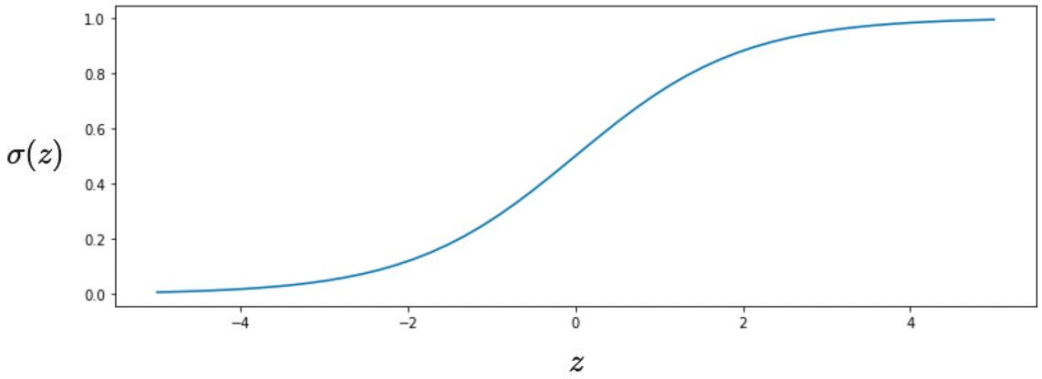
The logistic function $\sigma : \mathbb{R} \to [0, 1]$ "squeezes" points from the real line into $[0, 1]$.

The Logistic Function

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



Important properties



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

- The function tends to 1 as $z \to \infty$ and tends to 0 as $z \to -\infty$.
- Thus, models of the form $\sigma(\theta^\top x)$ output values between 0 and 1, which is suitable for binary classification.
- It is easy to show that the derivative of $\sigma(z)$ has a simple form: $\frac{d\sigma}{dz} = \sigma(z)(1 - \sigma(z))$.

A classification model: Logistic Regression

Logistic regression is a classification algorithm which uses a model f_θ of the form

$$f_\theta(x) = \sigma(\theta^\top x) = \frac{1}{1 + \exp(-\theta^\top x)},$$

where

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

is the *sigmoid* or *logistic* function.

** Although it is called logistic "regression", it is actually a **binary classification** algorithm*

Probabilistic Interpretations

$$f_\theta(x) = \sigma(\theta^\top x) = \frac{1}{1 + \exp(-\theta^\top x)}$$

The logistic model can be interpreted to output a probability, and defines a conditional probability distribution as follows:

$$P_\theta(y = 1|x) = \sigma(\theta^\top x)$$
$$P_\theta(y = 0|x) = 1 - \sigma(\theta^\top x).$$

Probabilistic Interpretations: $f_\theta(x)$ gives the the probability that x belongs to class y=1

How to optimize the logistic model?

Model: $f_\theta(x) = \sigma(\theta^\top x) = \dfrac{1}{1+\exp(-\theta^\top x)}$

Probabilistic Interpretations: $f_\theta(x)$ gives the the probability that x belongs to class y=1

- Intuitively, for a *training* data sample (x, y)
  - If true label **y = 1**, we want to *maximize* $\sigma(\theta^\top x)$
  - If true label **y = 0**, we want to *minimize* $\sigma(\theta^\top x)$, or equivalently, *maximize* $1 - \sigma(\theta^\top x)$
- We can write the two cases more compactly in one term:
  - We want to *maximize* $\sigma(\theta^\top x)^y \cdot (1 - \sigma(\theta^\top x))^{1-y}$

Objective function

- For *a* training data sample $(x^{(i)}, y^{(i)})$

  maximize $\sigma(\theta^\top x^{(i)})^{y^{(i)}} \cdot (1 - \sigma(\theta^\top x^{(i)}))^{1-y^{(i)}}$

- Or equivalently (taking log for mathematical reasons)

  maximize $\log \sigma(\theta^\top x^{(i)})^{y^{(i)}} \cdot (1 - \sigma(\theta^\top x^{(i)}))^{1-y^{(i)}}$

  = $y^{(i)} \cdot \log \sigma(\theta^\top x^{(i)}) + (1 - y^{(i)}) \cdot \log(1 - \sigma(\theta^\top x^{(i)}))$

- For *all* training samples, we maximize the average over all samples

  maximize $\frac{1}{n} \sum_{i=1}^{n} y^{(i)} \cdot \log \sigma(\theta^\top x^{(i)}) + (1 - y^{(i)}) \cdot \log(1 - \sigma(\theta^\top x^{(i)}))$

Loss function for gradient descent

maximize $$\frac{1}{n}\sum_{i=1}^{n} y^{(i)} \cdot \log \sigma(\theta^\top x^{(i)}) + (1 - y^{(i)}) \cdot \log(1 - \sigma(\theta^\top x^{(i)}))$$

- In gradient descent, we are *minimizing* the loss function.
  So we define the loss function as

minimize $$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n} y^{(i)} \cdot \log \sigma(\theta^\top x^{(i)}) + (1 - y^{(i)}) \cdot \log(1 - \sigma(\theta^\top x^{(i)}))$$

This loss function is known as the **cross-entropy loss**

The gradient

minimize $J(\theta) = -\frac{1}{n}\sum_{i=1}^{n} y^{(i)} \cdot \log \sigma(\theta^\top x^{(i)}) + (1 - y^{(i)}) \cdot \log(1 - \sigma(\theta^\top x^{(i)}))$

- It can be showed that the partial derivative of J is

$$\frac{\partial J(\theta)}{\partial \theta_j} = (f_\theta(x) - y)x_j$$

- The gradient of J can be computed accordingly
  - Recall that

$$\nabla J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \frac{\partial J(\theta)}{\partial \theta_2} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_d} \end{bmatrix}$$

Gradient descent

We start with an initial guess $\theta^{(0)}$ for the parameters and update the parameters at the *t*-th iteration ($\theta^{(t)}$) based on the parameters at the (*t*-1)-th iteration ($\theta^{(t-1)}$), until is no longer changing:

$$\theta^{(t)} = \theta^{(t-1)} - \alpha \cdot \nabla J(\theta^{(t-1)})$$

where a is the step size

Logistic Regression in Scikit-Learn (sklearn)

```
from sklearn.linear_model import LogisticRegression

# Data: x_train, y_train, x_test

# Define the logistic regression model
logisticRegr = LogisticRegression()

# Training the model on training data
logisticRegr.fit(x_train, y_train)

# Making predictions on new test data
predictions = logisticRegr.predict(x_test)
```

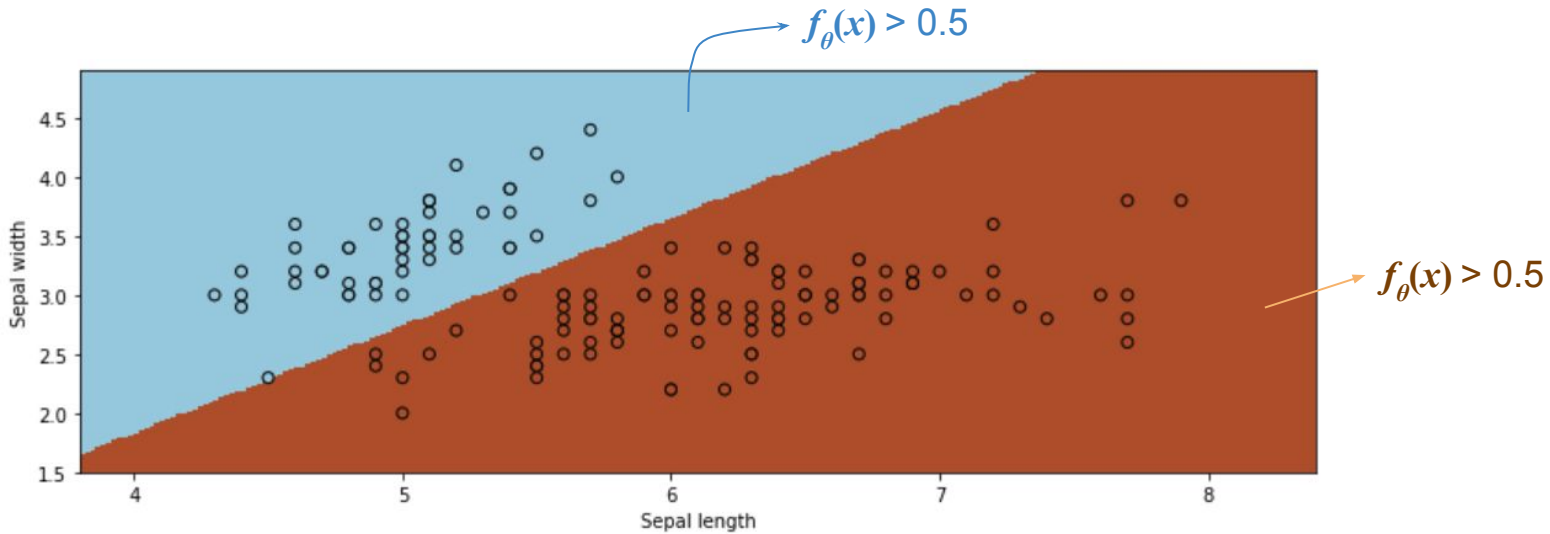
scikit-learn is a free software machine learning library for the Python programming language

https://scikit-learn.org/

Prediction

$f_\theta(x)$ gives the the probability that x belongs to class y=1

- Predict **y = 1**, if $f_\theta(x) > 0.5$
- Predict **y = 0**, otherwise



Prediction

$f_\theta(x)$ gives the the probability that x belongs to class y=1

- Predict **y = 1**, if $f_\theta(x) > 0.5$
- Predict **y = 0**, otherwise

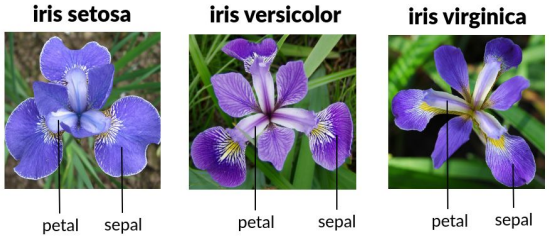
Can use other threshold: $f_\theta(x) > t$

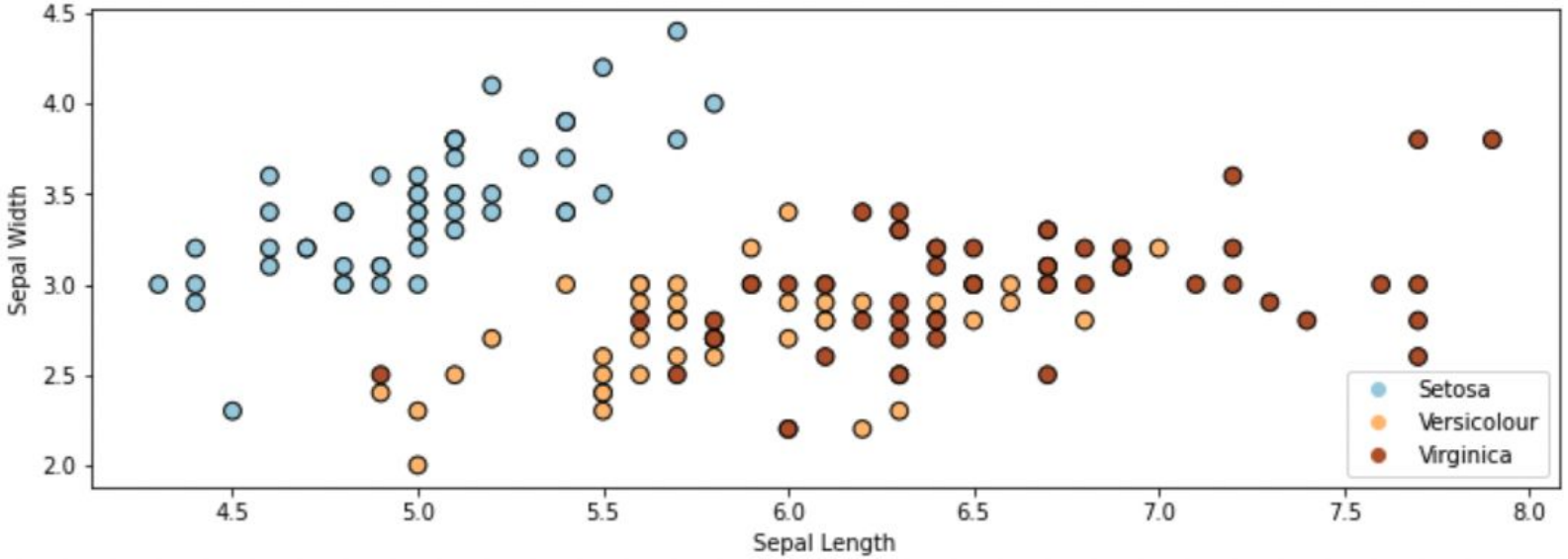
- Large t: more conservative, the model predicts y=1 when very confident
- Small t: more aggressive, the model predicts y=1 when there is a chance

To evaluate a model, we often look at its overall performance under all possible thresholds t

- Metrics: AUROC, AUPR (will be introduced later)

What if we have more than two classes?





Softmax function

The logistic function $\sigma : \mathbb{R} \to [0, 1]$ "squeezes" the score $z \in \mathbb{R}$ of a class into a probability in $[0, 1]$.

The *softmax* function $\vec{\sigma} : \mathbb{R}^K \to [0, 1]^K$ is a multi-class version of σ

- It takes in a K-dimensional *vector* of class scores $\vec{z} \in \mathbb{R}$
- It "squeezes" $\vec{z}$ into a length K *vector* of probabilities in $[0, 1]^K$

The k-th component of the output of the softmax function $\vec{\sigma}$ is defined as

$$\sigma(\vec{z})_k = \frac{\exp(z_k)}{\sum_{l=1}^{K} \exp(z_l)}.$$

Softmax takes a vector of scores $\vec{z}$, exponentiates each score z_k, and normalizes the exponentiated scores such that they sum to one.

Multi-class classification: Softmax Regression

Softmax regression is a multi-class classification algorithm which uses a model $f_\theta : \mathcal{X} \to [0,1]^K$ that generalizes logistic regression.

Softmax regression works as follows:

1. Given an input x, we compute K scores, one per class. The score

$$z_k = \theta_k^\top x$$

of class k is a linear function of x and parameters θ_k for class k

2. We "squeeze" the vector of scores $\vec{z}$ into $[0,1]^K$ using the softmax function $\vec{\sigma}$ and we output $\vec{\sigma}(\vec{z})$, a vector of K probabilities.

Multi-class classification: Softmax Regression

Softmax regression is a multi-class classification algorithm which uses a model $f_\theta : \mathcal{X} \to [0,1]^K$ that generalizes logistic regression.

The parameters of this model are $\theta = (\theta_1, \theta_2, \ldots, \theta_K)$, and the parameter space is $\Theta = \mathbb{R}^{K \times d}$.

The output of the model is a *vector* of class membership probabilities, whose k-th component $f_\theta(x)_k$ is

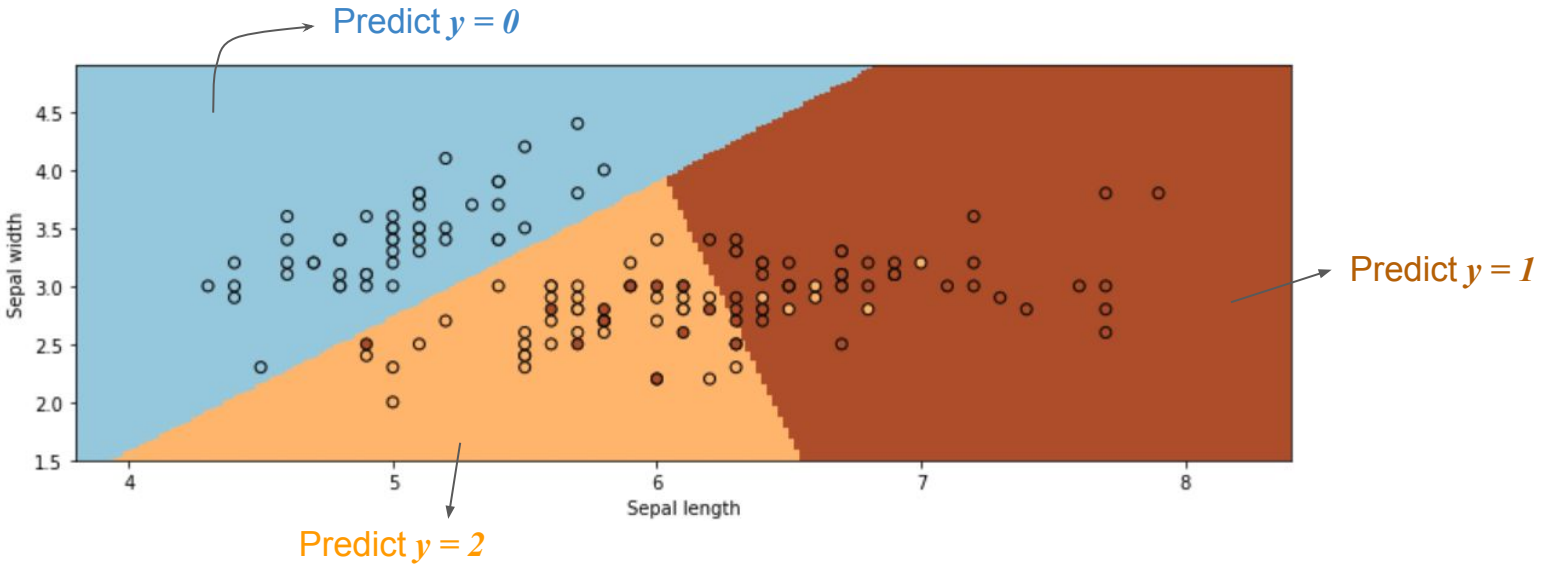
$$f_\theta(x)_k = \sigma(\theta_k^\top x)_k = \frac{\exp(\theta_k^\top x)}{\sum_{l=1}^{K} \exp(\theta_l^\top x)},$$

where each $\theta_l \in \mathbb{R}^d$ is the vector of parameters for class ℓ and $\theta = (\theta_1, \theta_2, \ldots, \theta_K)$.

Prediction

$f_\theta(x)_k$ gives the the probability that x belongs to class $y = k$

- Predict **y = *k***, where $k = \textit{argmax}\, f_\theta(x)_k$



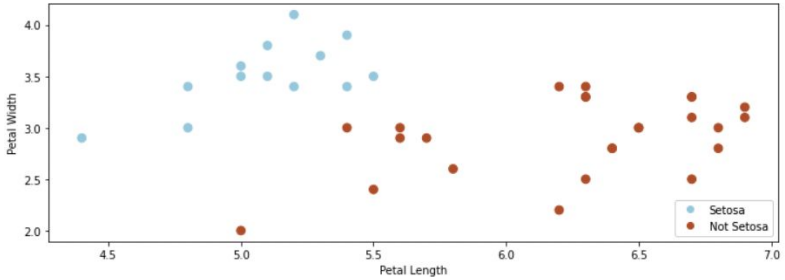
Softmax Regression in sklearn

```
from sklearn.linear_model import LogisticRegression

# Data: x_train, y_train, x_test

# Define the logistic regression model
logisticRegr = LogisticRegression(multi_class='multinomial')

# Training the model on training data
logisticRegr.fit(x_train, y_train)

# Making predictions on new test data
predictions = logisticRegr.predict(x_test)
```

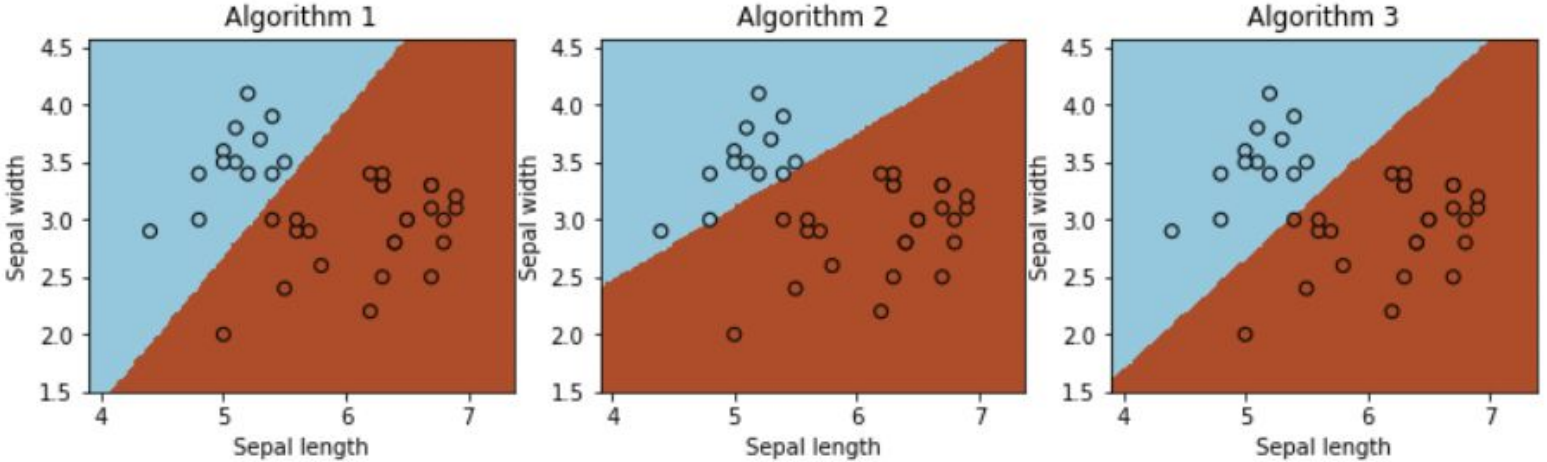
Overview of other classification models

Support Vector Machine (SVM)

Support Vector Machine (SVM)



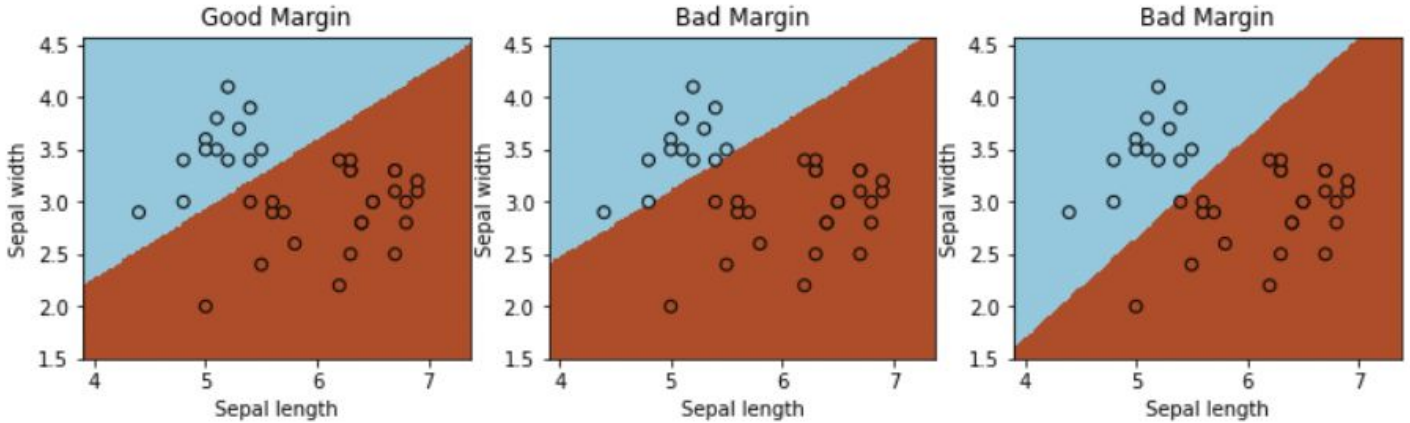
- When fitting a model, there may be many valid decision boundaries.



- How do we select one of them?

Support Vector Machine (SVM)

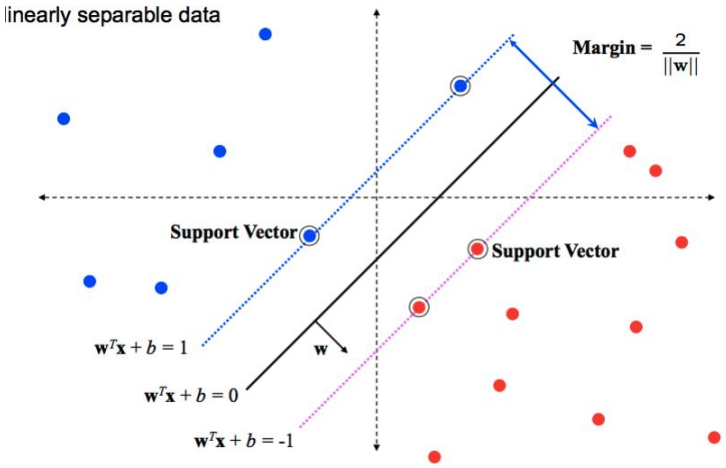
- Intuitively, we want to select boundaries with the ***maximum margin***.
- This means that we are as confident as possible for every point and we are as far as possible from the decision boundary.



- Several of the separating boundaries in our previous example had low margin: they came too close to the boundary.

Support Vector Machine (SVM)

- SVM: maximize the margin between two classes.
  - In this figure, the two dashed lines that lie at the margin.
  - A good decision boundary is as far away as possible from the points at the margin.



- Learning the SVM can be formulated as an optimization:

$$\max_{\mathbf{w}} \frac{2}{||\mathbf{w}||} \text{ subject to } \mathbf{w}^\top \mathbf{x}_i + b \begin{array}{ll} \geq 1 & \text{if } y_i = +1 \\ \leq -1 & \text{if } y_i = -1 \end{array} \text{ for } i = 1 \ldots N$$

- Or equivalently

$$\min_{\mathbf{w}} ||\mathbf{w}||^2 \text{ subject to } y_i \left(\mathbf{w}^\top \mathbf{x}_i + b\right) \geq 1 \text{ for } i = 1 \ldots N$$

- This is a quadratic optimization problem subject to linear constraints and there is a unique minimum

Decision Trees

Decision Trees

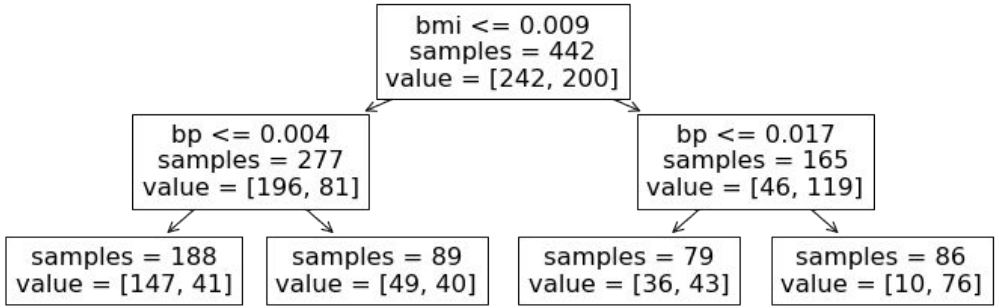
- Recall our diabetes risk dataset

	age	sex	bmi	bp
0	0.038076	0.050680	0.061696	0.021872
1	-0.001882	-0.044642	-0.051474	-0.026328
2	0.085299	0.050680	0.044451	-0.005671
3	-0.089063	-0.044642	-0.011595	-0.036656
4	0.005383	-0.044642	-0.036385	0.021872

- Decision trees are machine learning models that mimic how a human would approach this problem.

1. We start by picking a feature (e.g., age)
2. Then we *branch* on the feature based on its value (e.g, age > 65?)
3. We select and branch on one or more features (e.g., is it a man?)
4. Then we return an output that depends on all the features we've seen (e.g., a man over 65)

Decision Trees

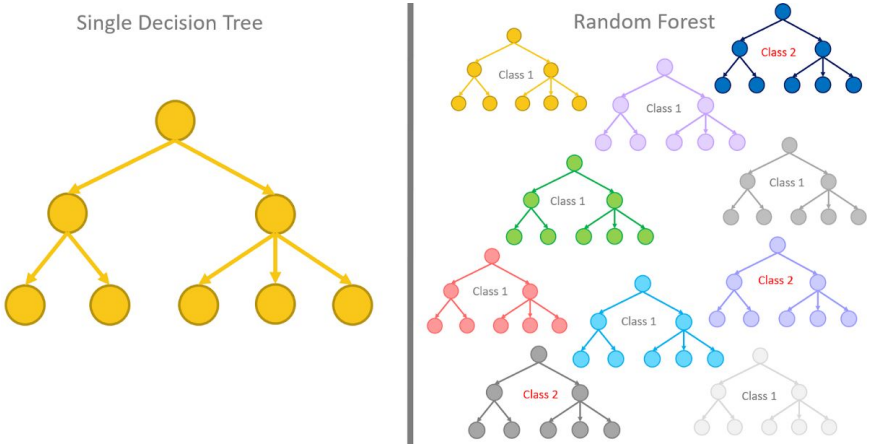


A decision tree is a (usually binary) tree, where:

- Each internal node n corresponds to a rule r_n
- The j-th edge out of n is associated with a rule value v_j, and we follow the j-th edge if $r_n(x) = v_j$
- Each leaf node l contains a prediction $f(x)$
- Given input x, we start at the root, apply its rule, follow the eddge that corresponds to the outcome, and repeat recursively.

Decision trees -> Random forest

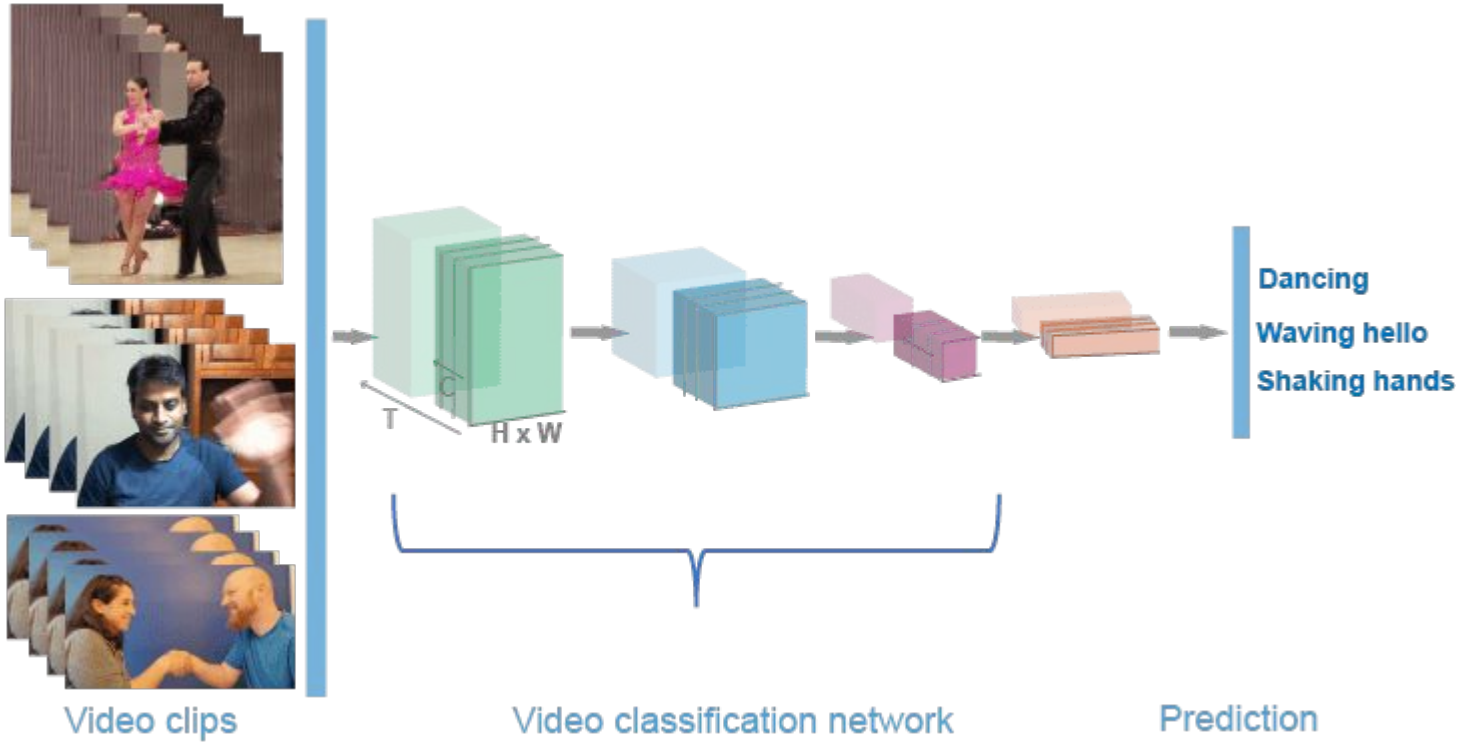
- Random forest: combining many decision trees to give more robust predictions




```
for i in range(n_models):
    # collect data samples and fit models
    X_i, y_i = sample_with_replacement(X, y, n_samples)
    model = DecisionTree().fit(X_i, y_i)
    random_forest.append(model)

# output average prediction at test time:
y_test = random_forest.average_prediction(y_test)
```

Neural networks

Neural networks



https://www.mathworks.com/help/vision/ug/video-classification-using-deep-learning.html

Summary of Classification

- Logistic regression
- Softmax function
- Other classification models
  - SVM
  - Random forest
  - …

“Toolbox” for applied ML

Outline

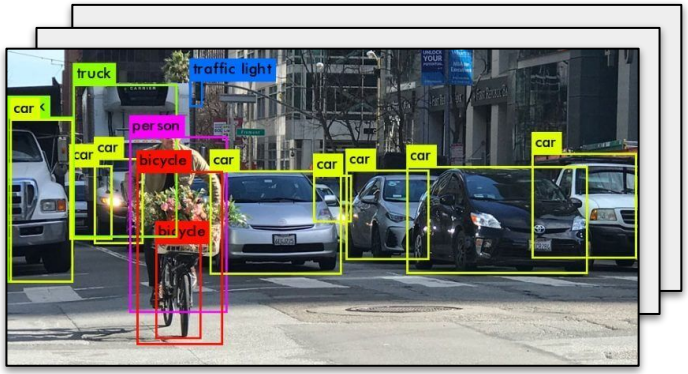
- Model Evaluation
- Feature engineering
- Model selection
- Regularization

Outline

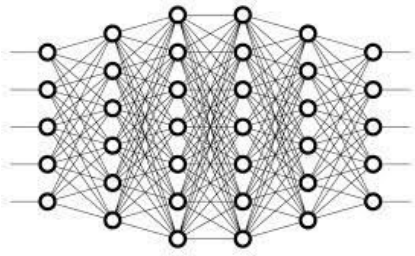
- **Model Evaluation**
- Feature engineering
- Model selection
- Regularization

Recall: “Recipe” of (supervised) machine learning

1. Dataset



2. Model



More complex model

3. Optimization

- Find the “best” value for every parameter in the model

4. Predictions

New scenes

What is a good (supervised) learning model?

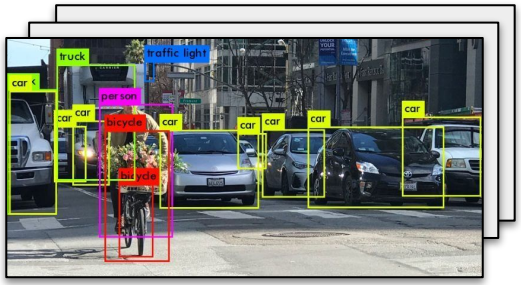
- A good predictive model is one that makes ***accurate predictions*** on ***new data*** that it has not seen at training time.
    - Accurate object detection in new scenes
    - Accurate prediction of diabetes for a new individual

** Note that other definitions exist, e.g., does the model discover useful structure in the data?*

Evaluating a machine learning model

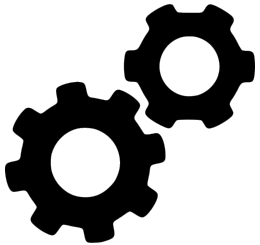
1. First, we collect a dataset of labeled training examples (**training data**).
2. We train a **ML model** to output accurate predictions on this dataset.
3. Evaluate the model on a held out dataset (**test data**)
    - Test data must contain data samples ***distinct*** from the training samples

Training data



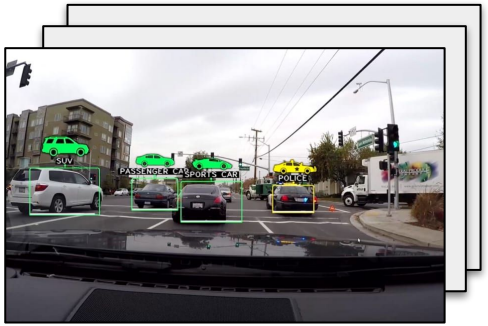
Development

ML model

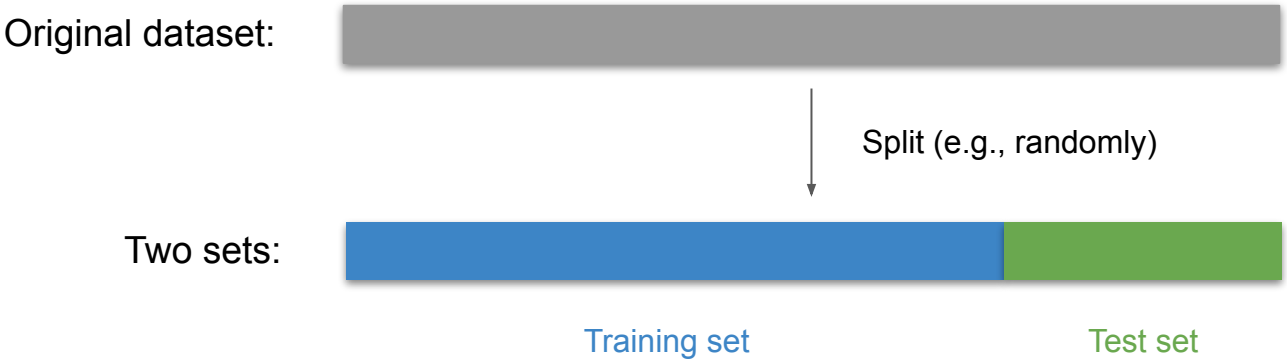


Evaluation

Test data

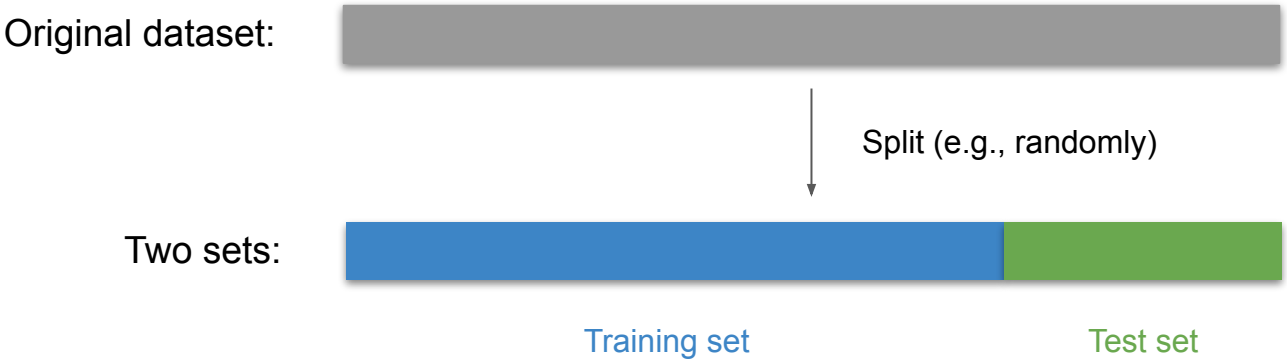


What if we do not have a hold out test data?



- Randomly split the original dataset into two sets: training set & test set
- Train the ML model using the training set
- Evaluate the ML model on the test set

What if we do not have a hold out test data?



- Randomly split the original dataset into two sets: training set & test set
- Train the ML model using the training set
- Evaluate the ML model on the test set

Only a portion of samples are used as test data. Is this representative?

K-fold Cross-Validation

Given a dataset, split it into *K* equally sized sets (or folds). For example, for *K*=5:

We train the model *K* times, each time using a different fold for testing, and the rest for training.

K-fold Cross-Validation

Given a dataset, split it into *K* equally sized sets (or folds). For example, for *K*=5:

We train the model *K* times, each time using a different fold for testing, and the rest for training.

- Iteration 1: Training set {2, 3, 4, 5}, Test set {1}

K-fold Cross-Validation

Given a dataset, split it into *K* equally sized sets (or folds). For example, for *K*=5:

We train the model *K* times, each time using a different fold for testing, and the rest for training.

- Iteration 1: Training set {2, 3, 4, 5}, Test set {1}
- Iteration 2: Training set {1, 3, 4, 5}, Test set {2}

K-fold Cross-Validation

Given a dataset, split it into *K* equally sized sets (or folds). For example, for *K*=5:

We train the model *K* times, each time using a different fold for testing, and the rest for training.

- Iteration 1: Training set {2, 3, 4, 5}, Test set {1}
- Iteration 2: Training set {1, 3, 4, 5}, Test set {2}
- Iteration 3: Training set {1, 2, 4, 5}, Test set {3}

K-fold Cross-Validation

Given a dataset, split it into *K* equally sized sets (or folds). For example, for *K*=5:

We train the model *K* times, each time using a different fold for testing, and the rest for training.

- Iteration 1: Training set {2, 3, 4, 5}, Test set {1}
- Iteration 2: Training set {1, 3, 4, 5}, Test set {2}
- Iteration 3: Training set {1, 2, 4, 5}, Test set {3}
- Iteration 4: Training set {1, 2, 3, 5}, Test set {4}

K-fold Cross-Validation

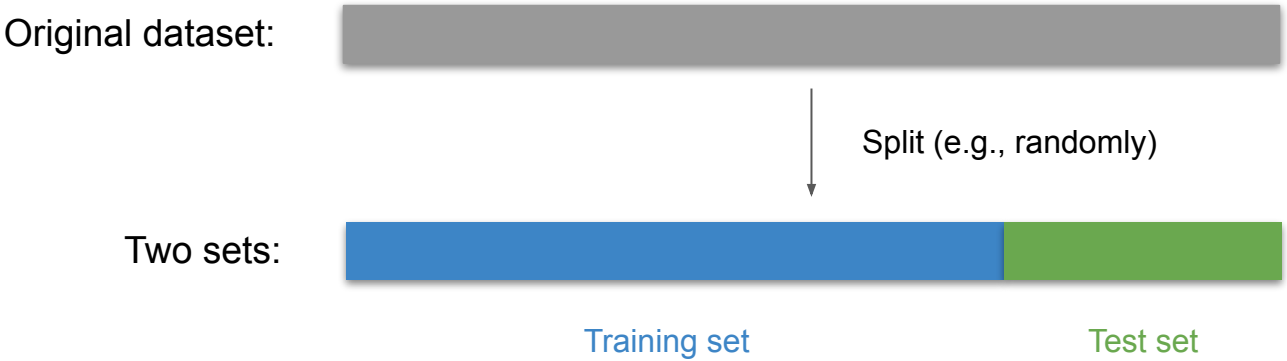
Given a dataset, split it into *K* equally sized sets (or folds). For example, for *K*=5:

We train the model *K* times, each time using a different fold for testing, and the rest for training.

- Iteration 1: Training set {2, 3, 4, 5}, Test set {1}
- Iteration 2: Training set {1, 3, 4, 5}, Test set {2}
- Iteration 3: Training set {1, 2, 4, 5}, Test set {3}
- Iteration 4: Training set {1, 2, 3, 5}, Test set {4}
- Iteration 5: Training set {1, 2, 3, 4}, Test set {5}

Finally, report the average evaluation results (e.g., MSE) over all *K* folds

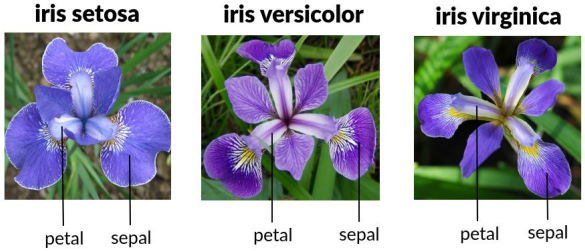
Train/test split



- Randomly split the original dataset into two sets: training set & test set
- Train the ML model using the training set
- Evaluate the ML model on the test set

What is the criterion used to evaluate a model?

Evaluation metrics



- Consider a classification problem, e.g., Iris subtype classification

No	Target	Prediction
1	0	0
2	1	2
3	2	2
4	2	1
5	1	1
6	0	0
7	1	1

- How good are the model's predictions?

Classification Accuracy

The simplest and most natural metric for classification algorithms on a dataset $\mathcal{D} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})\}$ is accuracy:

$$\text{acc}(f) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}\{f(x^{(i)}) = y^{(i)}\},$$

where $\mathbb{I}\{\cdot\}$ is an indicator function (equals 1 if its input is true and zero otherwise).

- In other words,

$$\text{Accuracy} = \frac{\text{\# of correct predictions}}{\text{\# of total predictions}}$$

No	Target	Prediction	
1	0	0	✅
2	1	2	
3	2	2	✅
4	2	1	
5	1	1	✅
6	0	0	✅
7	1	1	✅

Accuracy = 5/7

Is accuracy a good classification metric?

- Could be problematic
  - Because the ML model can “cheat”

- Consider a binary classification problem

  Test set: 9 positive samples, 1 negative samples

  \+ + + + + + + + + -

  ML model: always predicting + for all test samples

  \+ + + + + + + + + +

- Accuracy = 9/10 = 0.9
- Is this a good model?

Sensitivity and Specificity

- Consider a binary classification problem

	Predicted positive $\hat{y} = 1$	**Predicted negative $\hat{y} = 0$**
Positive class $y = 1$	True positive (TP)	False negative (FN)
Negative class $y = 0$	False positive (FP)	True negative (TN)

We can also look at "accuracy" on each class separately. This reveals problems with imbalanced classes.

$$\underset{\text{(recall, true positive rate)}}{\text{sensitivity}} = \frac{\text{TP}}{\text{positive class}} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\underset{\text{(true negative rate)}}{\text{specificity}} = \frac{\text{TN}}{\text{negative class}} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

- Example:

True labels + + + + + + + + + -

Predictions + + + + + + + + + +

Sensitivity = 9/9 = 1.0

Specificity = 0/1 = 0.0

Receiver Operating Characteristic (ROC) curve

	Predicted positive $\hat{y} = 1$	**Predicted negative** $\hat{y} = 0$
Positive class $y = 1$	True positive (TP)	False negative (FN)
Negative class $y = 0$	False positive (FP)	True negative (TN)

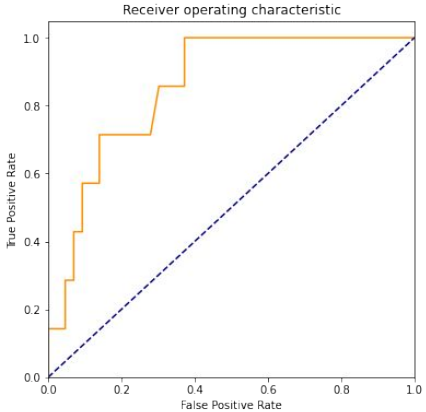
In binary classification, the Receiver Operating Characteristic (ROC) curve plots the true positive rate and the false positive rate (FPR) as we vary the threshold for labeling a positive example.

$$\text{TPR} = \underset{\text{(recall, sensitivity)}}{\text{true positive rate}} = \frac{\text{TP}}{\text{positive class}} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

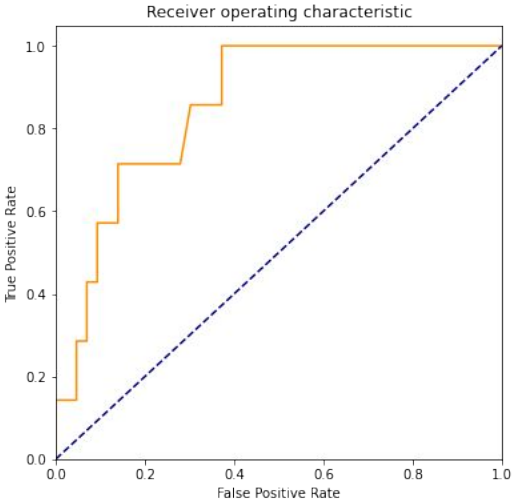
$$\text{FPR} = 1 - \underset{\text{(true negative rate)}}{\text{specificity}} = 1 - \frac{\text{TN}}{\text{negative class}} = \frac{\text{FP}}{\text{TP} + \text{FP}}$$

Suppose we want to improve sensitivity for Class 2 on the Iris dataset. We first compute the probability $p(y = 2|x)$ for each input x. For any threshold $t > 0$, we label x as Class 2 if $p(y = 2|x) > t$.

- Small t result in a high TPR (we identify all positives) and high FPR (many are false).
- High t result in a low TPR (we identify few positives) and low FPR (we only label the most confident inputs).



Receiver Operating Characteristic (ROC) curve



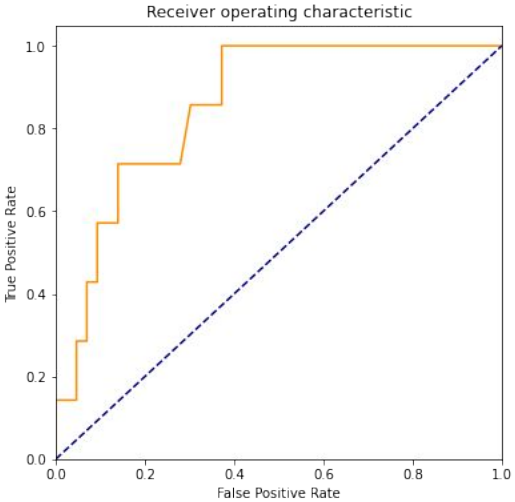
$$\text{TPR} = \underset{\text{(recall, sensitivity)}}{\text{true positive rate}} = \frac{\text{TP}}{\text{positive class}} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{FPR} = 1 - \underset{\text{(true negative rate)}}{\text{specificity}} = 1 - \frac{\text{TN}}{\text{negative class}} = \frac{\text{FP}}{\text{TP} + \text{FP}}$$

We highlight the following properties of the ROC curve:

- In the bottom left corner, we predict only negatives. TPR = FPR = 0.
- In the top right corner, we predict only positives. TPR = FPR = 1.
- The ideal classifier is in the top left: TPR = 1, FPR = 0.
- The blue diagonal corresponds to randomly guessing "positive" with $p = \text{TPR}$.
- The ROC curve lies between ideal and random.

Area Under the ROC Curve



We can use the area under the curve (AUC) as a single measure of classifier performance.

- The ideal classifier (ROC curve reaches top left corner) has an AUC-ROC of 1.0
- A random classifier (diagonal dashed line) has an AUC-ROC of 0.5
- What if AUC-ROC = 0.1?

Precision and Recall (after-class exercise)

- Consider a binary classification problem

	Predicted positive $\hat{y} = 1$	**Predicted negative $\hat{y} = 0$**
Positive class $y = 1$	True positive (TP)	False negative (FN)
Negative class $y = 0$	False positive (FP)	True negative (TN)

An alternative set of measures is precision and recall.

$$\underset{\text{(positive predictive value)}}{\text{precision}} = \frac{\text{TP}}{\text{predicted positive}} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\underset{\text{(sensitivity, true positive rate)}}{\text{recall}} = \frac{\text{TP}}{\text{positive class}} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- Example:

True labels + + + + + + + + + -

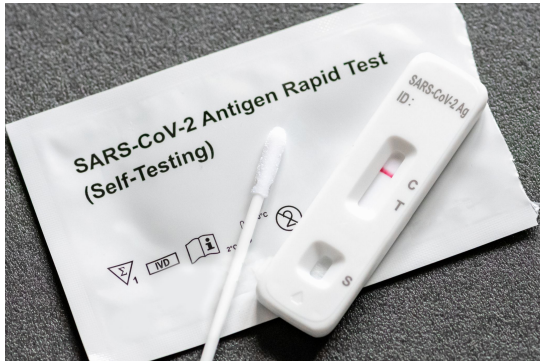
Predictions + - - - - - - - - -

precision = 1/1 = 1.0

recall = 1/9 = 0.11

An example of COVID-19 test (after-class exercise)

		Test positive	Test negative
		Predicted positive $\hat{y} = 1$	**Predicted negative** $\hat{y} = 0$
Infected	Positive class $y = 1$	True positive (TP)	False negative (FN)
Not infected	Negative class $y = 0$	False positive (FP)	True negative (TN)



$$\underset{\text{(true negative rate)}}{\text{specificity}} = \frac{\text{TN}}{\text{negative class}} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

Among all non-infected people, what % got a negative test result?

$$\underset{\text{(positive predictive value)}}{\text{precision}} = \frac{\text{TP}}{\text{predicted positive}} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

?

$$\underset{\text{(sensitivity, true positive rate)}}{\text{recall}} = \frac{\text{TP}}{\text{positive class}} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

?

Precision-recall curve (after-class exercise)

Regression metrics

Most standard regression losses can be used as evaluation metrics.

- Mean squared error:

$$\frac{1}{2n}\sum_{i=1}^{n}\left(f(x^{(i)})-y^{(i)}\right)^2$$

- Absolute (L1) error:

$$\frac{1}{n}\sum_{i=1}^{n}\left|f(x^{(i)})-y^{(i)}\right|$$

- Correlation coefficients
  - Spearman's correlation
  - Pearson correlation
  - R^2

Summary of metrics

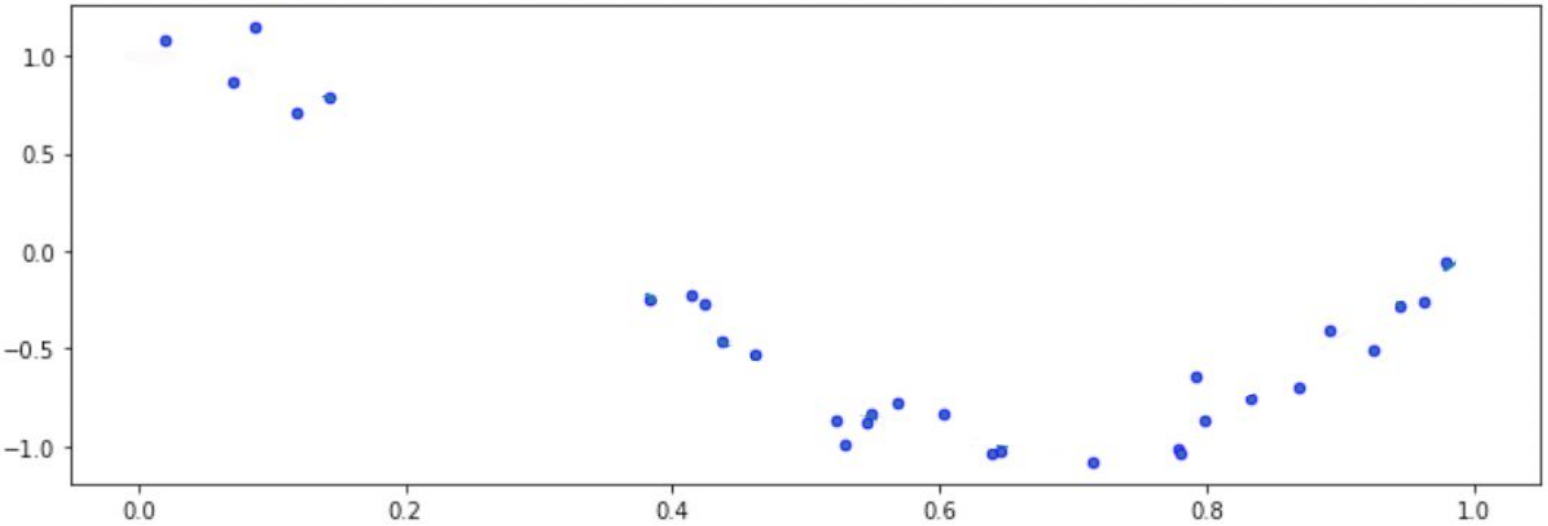
- Classification
  - Accuracy
  - Specificity
  - Sensitivity (Recall)
  - Precision
- Regression
  - Mean squared error
  - Absolute error (L1 error)
  - Correlations

Outline

- Model Evaluation
- **Feature engineering**
- Model selection
- Regularization

Regression

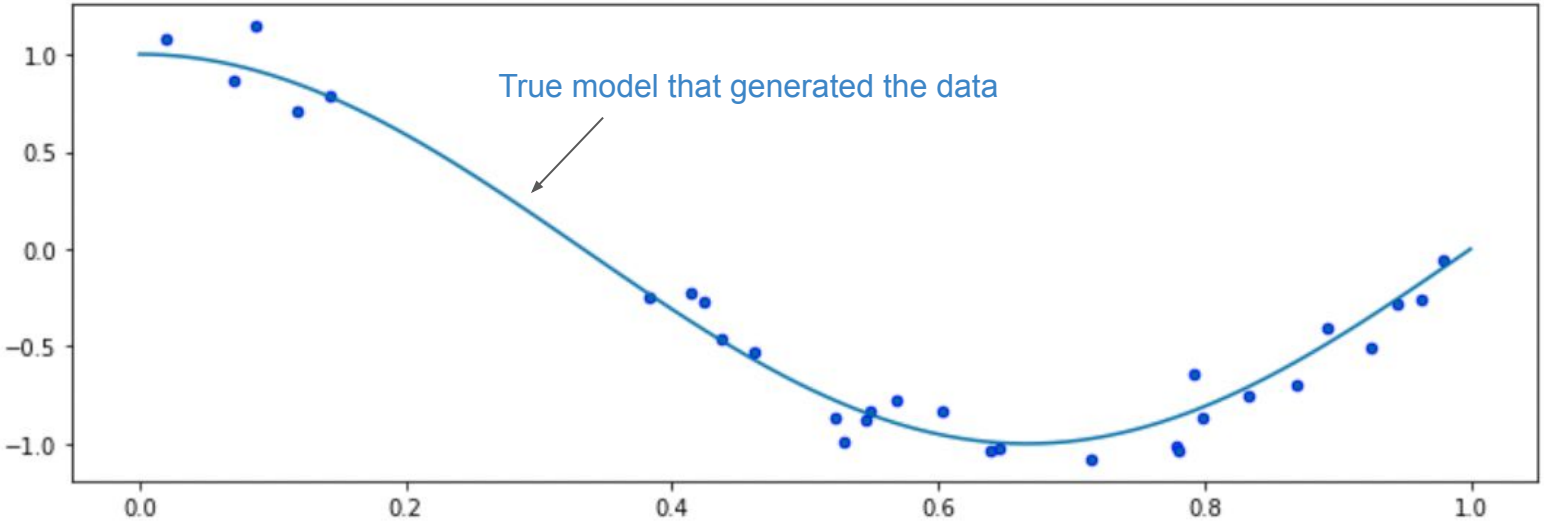
- Consider fitting a linear regression model on the following data $y = \theta_1 \cdot x + \theta_0$



- Will this model have good prediction ability?

Regression

- Likely NO, because the true model is a nonlinear curve



Polynomial Regression

- Although fitting a linear model does not work well, quadratic or cubic polynomials improve the fit.
- Instead of fitting a linear line

$$y = \theta_1 \cdot x + \theta_0$$

- We fit a polynomial of degree *p*

$$y = \theta_p \cdot x^p + \theta_{p-1} \cdot x^{p-1} + \cdots + \theta_1 x \ + \theta_0$$

- How to fit the polynomial regression?

Linear Regression vs Polynomial Regression

- Linear regression with multiple features

$$f(x) = \theta_d \cdot x_d + \theta_{d-1} \cdot x_{d-1} + \cdots + \theta_1 x_1 + \theta_0$$

- Polynomial regression

$$f(x) = \theta_p \cdot x^p + \theta_{p-1} \cdot x^{p-1} + \cdots + \theta_1 x \ + \theta_0$$

Linear Regression vs Polynomial Regression

- Linear regression with multiple features

$$f(x) = \theta_d \cdot x_d + \theta_{d-1} \cdot x_{d-1} + \cdots + \theta_1 x_1 + \theta_0$$

- Polynomial regression

$$f(x) = \theta_p \cdot x^p + \theta_{p-1} \cdot x^{p-1} + \cdots + \theta_1 x \ + \theta_0$$

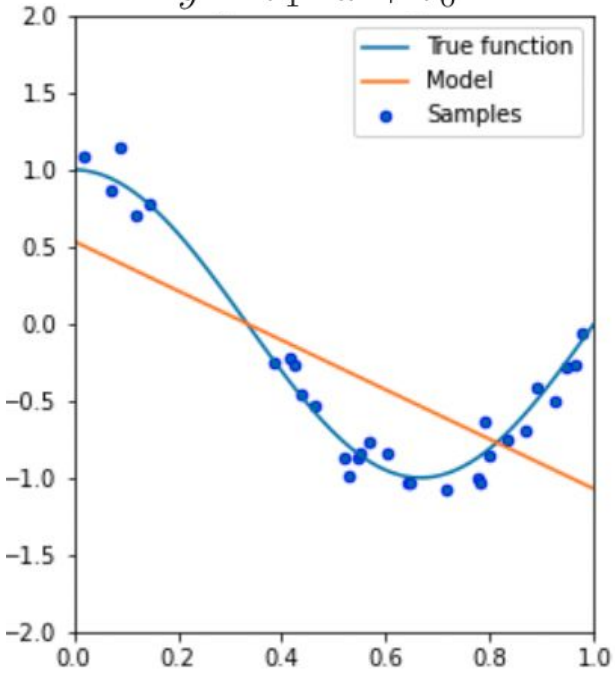
- This model is non-linear in the input variable x, meaning that we can model complex data relationships.

- It is a linear model as a function of the parameters θ, meaning that we can use our familiar ordinary least squares algorithm to learn these features.

Polynomial Regression

Polynomial of Degree 1

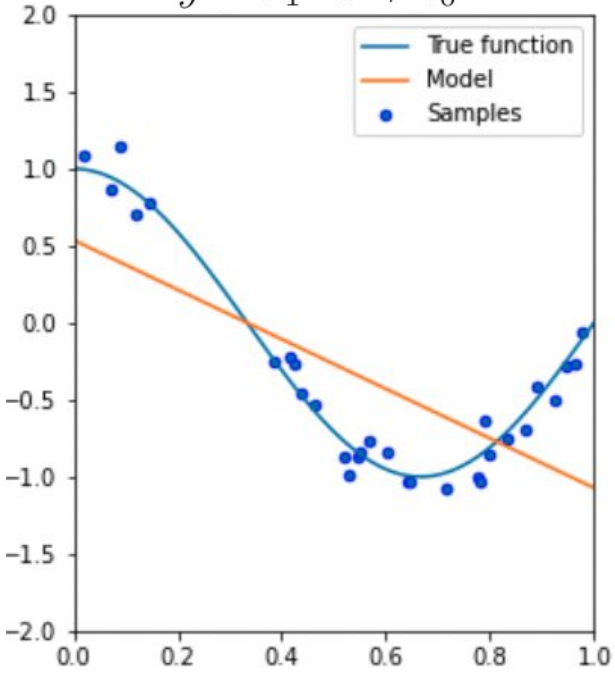
$$y = \theta_1 \cdot x + \theta_0$$



Polynomial Regression

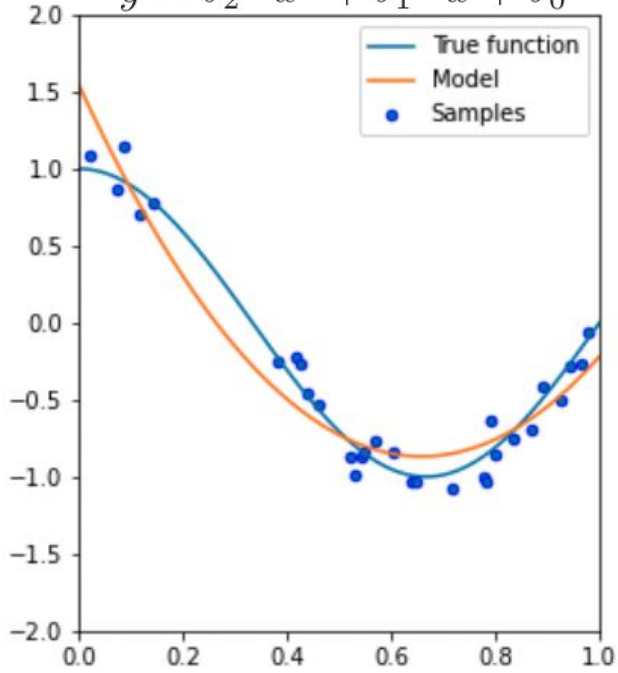
Polynomial of Degree 1

$$y = \theta_1 \cdot x + \theta_0$$



Polynomial of Degree 2

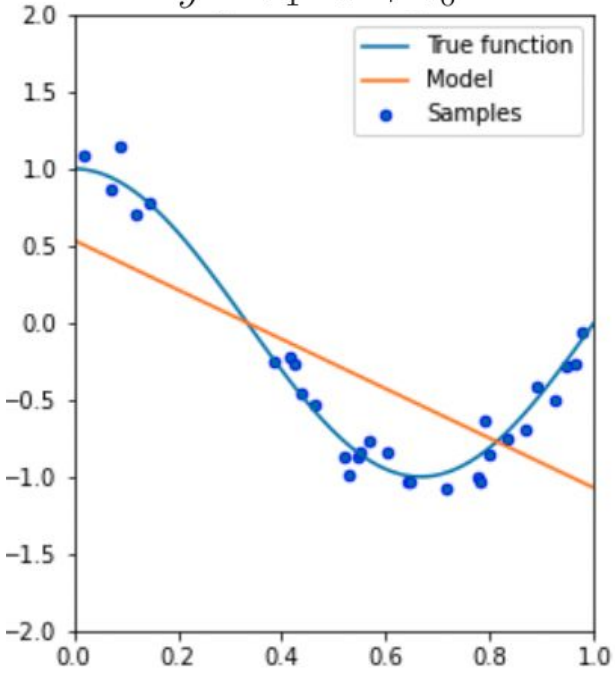
$$y = \theta_2 \cdot x^2 + \theta_1 \cdot x + \theta_0$$



Polynomial Regression

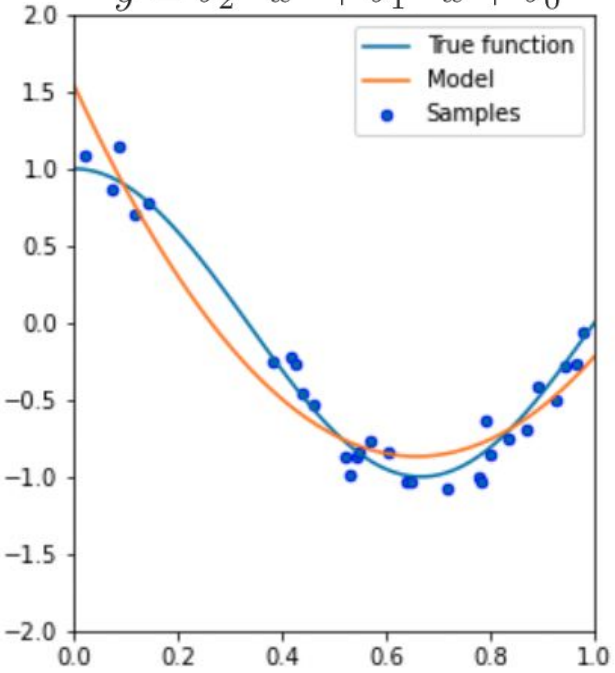
Polynomial of Degree 1

$$y = \theta_1 \cdot x + \theta_0$$



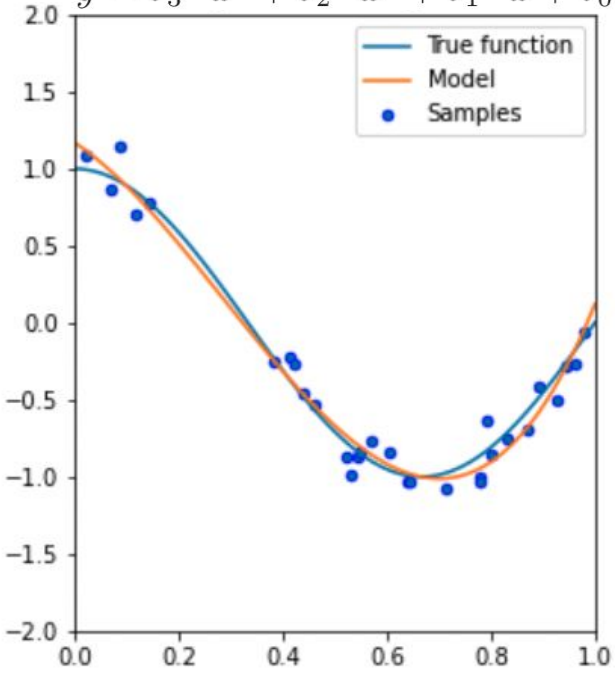
Polynomial of Degree 2

$$y = \theta_2 \cdot x^2 + \theta_1 \cdot x + \theta_0$$



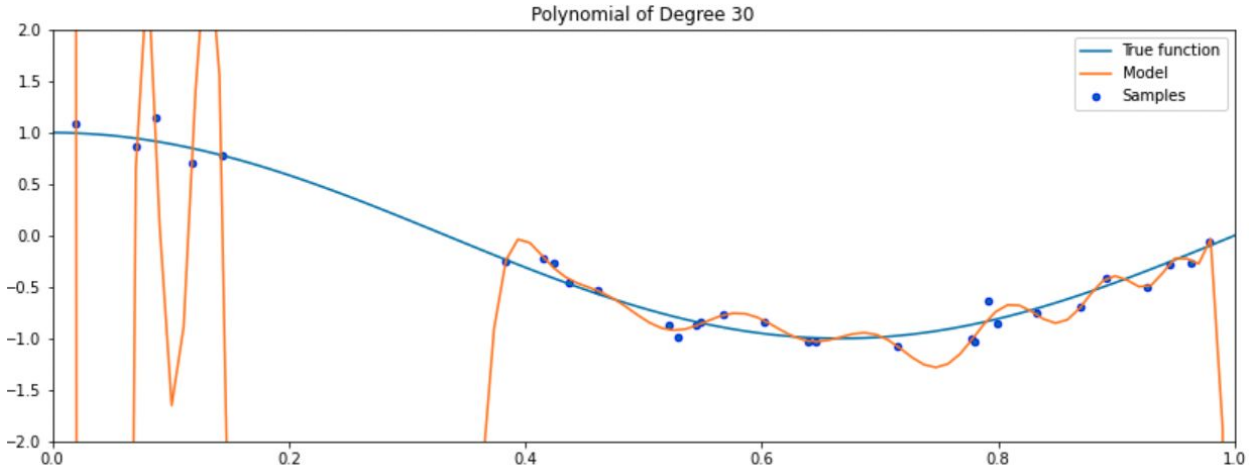
Polynomial of Degree 3

$$y = \theta_3 \cdot x^3 + \theta_2 \cdot x^2 + \theta_1 \cdot x + \theta_0$$



Should we go to even higher order polynomial?

Polynomial of Degree 30

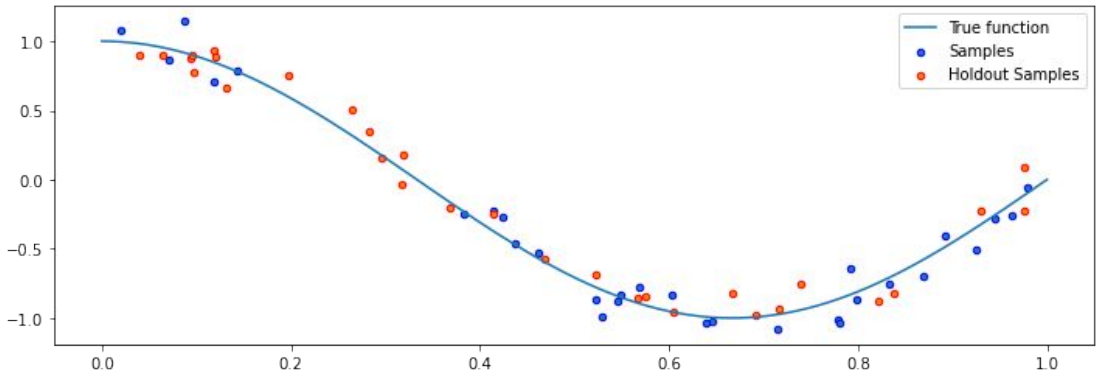


- Is this a good model? Why?

As the degree of the polynomial increases to the size of the dataset, we are increasingly able to fit every point in the dataset.

However, this results in a highly irregular curve: its behavior outside the training set is wildly inaccurate.

Let’s evaluate the polynomial regression models



- Use training samples to fit the regression models and use holdout samples (test data) to evaluate those models

Overfitting & Underfitting

- **Overfitting**
  - A very expressive model (e.g., a high degree polynomial) fits the training dataset perfectly.
  - But the model makes highly incorrect predictions outside this dataset, and doesn't generalize.

Overfitting is one of the most common failure modes of machine learning

- **Underfitting**
  - A small model (e.g. a straight line), will not fit the training data well.
  - Therefore, it will also not be accurate on new data.

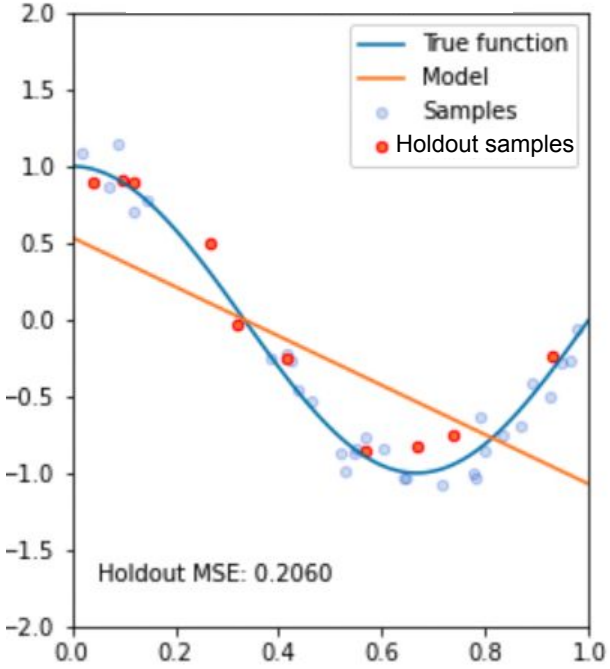
Finding the tradeoff between overfitting and underfitting is one of the main challenges in applying machine learning.

Diagnose overfitting & underfitting

- We can diagnose overfitting and underfitting by measuring performance on a separate held out dataset (not used for training)
    - If training performance is **high** but holdout performance is **low**, we are overfitting.
    - If training performance is **low** and holdout performance is **low**, we are underfitting.

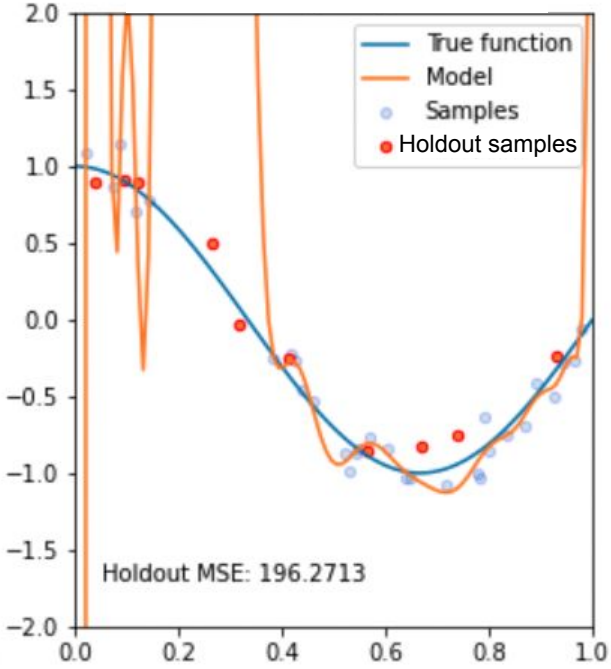
Examples

Polynomial of Degree 1



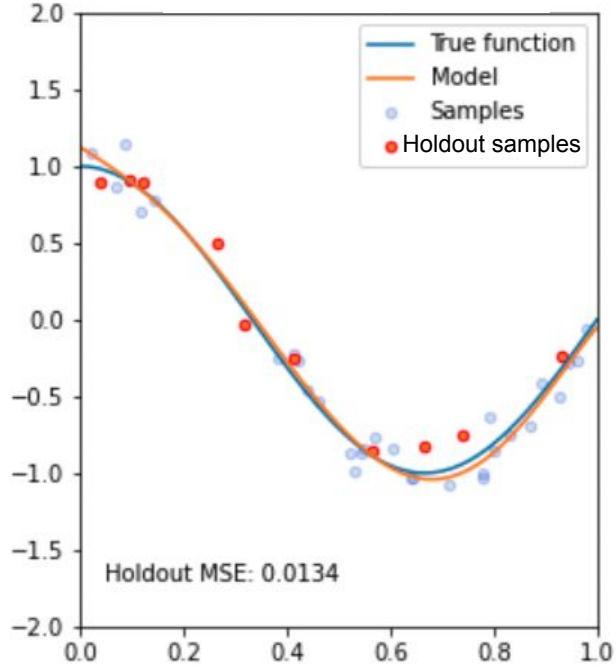
Underfitting

Polynomial of Degree 20



Overfitting

Polynomial of Degree 5

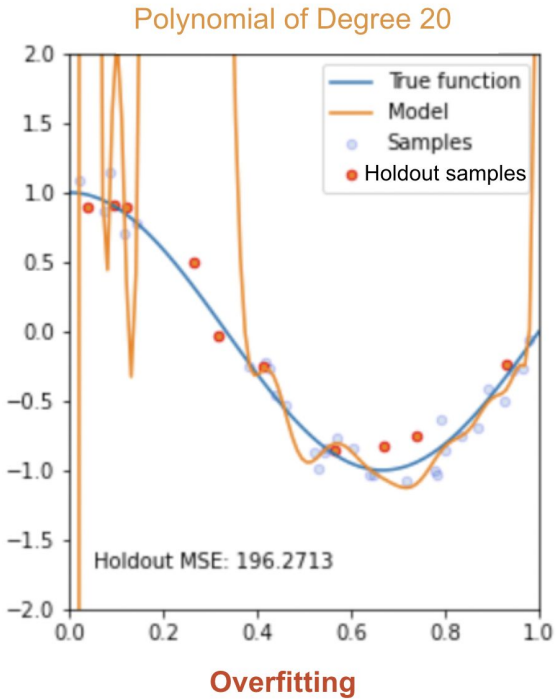


A good fit

How to fix overfitting?

- **Overfitting**
  - A very expressive model (e.g., a high degree polynomial) fits the training dataset perfectly.
  - But the model makes highly incorrect predictions outside this dataset, and doesn't generalize.
  - Training performance is high but holdout performance is low

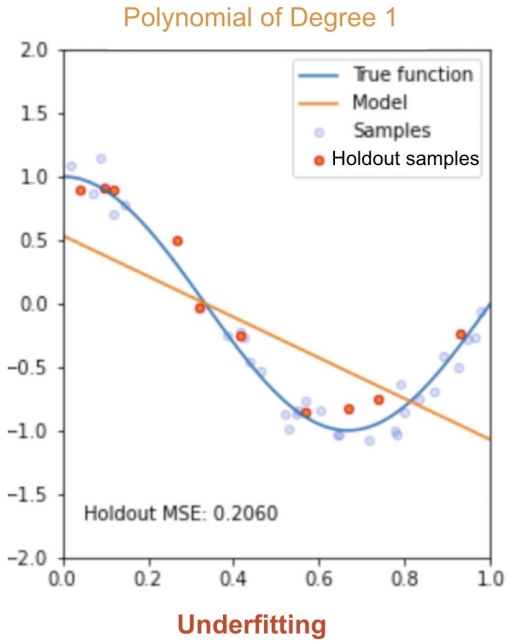
Overfitting is one of the most common failure modes of machine learning



- **Many ways to fix. Some examples:**
  - Use a simpler model family (linear models vs. neural nets)
  - Keep the same model, but collect more training data
  - Modify the training process to penalize overly complex models.

How to fix underfitting?

- **Underfitting**
  - A small model (e.g. a straight line), will not fit the training data well.
  - Therefore, it will also not be accurate on new data.
  - Training performance is low and holdout performance is low



- **Many ways to fix. Some examples:**
  - Create richer features that will make the dataset easier to fit.
  - Use a more expressive model family (neural nets vs. linear models)
  - Try a better optimization procedure

Finding the tradeoff between overfitting and underfitting is one of the main challenges in applying machine learning.

Outline

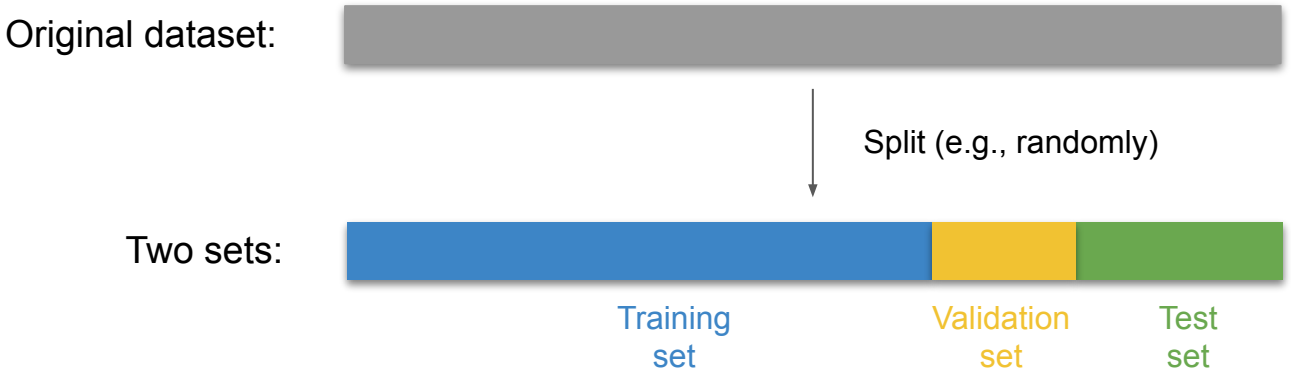
- Model Evaluation
- Feature engineering
- **Model selection**
- Regularization

Polynomial Regression

$$f(x) = \theta_p \cdot x^p + \theta_{p-1} \cdot x^{p-1} + \cdots + \theta_1 x \ + \theta_0$$

- A particular value of *p* defines a polynomial regression model
- Some of them may be overfitting, underfitting, and good fits
- How to choose the value of *p*?

Train/validation/test splits



- When developing machine learning models, we typically split into **three** sets:
  - **Training set**: Data on which we train our algorithms.
  - **Validation set** (development or holdout set): Data used for tuning algorithms.
  - **Test set**: Data used to evaluate the final performance of the model.
- Split ratio varies: e.g., train : validation : test = 70% : 10% : 20%

Use validation set to tune hyperparameters

- **Example**: how to select the degree *p* in a polynomial regression?

$$y = \theta_p \cdot x^p + \theta_{p-1} \cdot x^{p-1} + \cdots + \theta_1 x \ + \theta_0$$

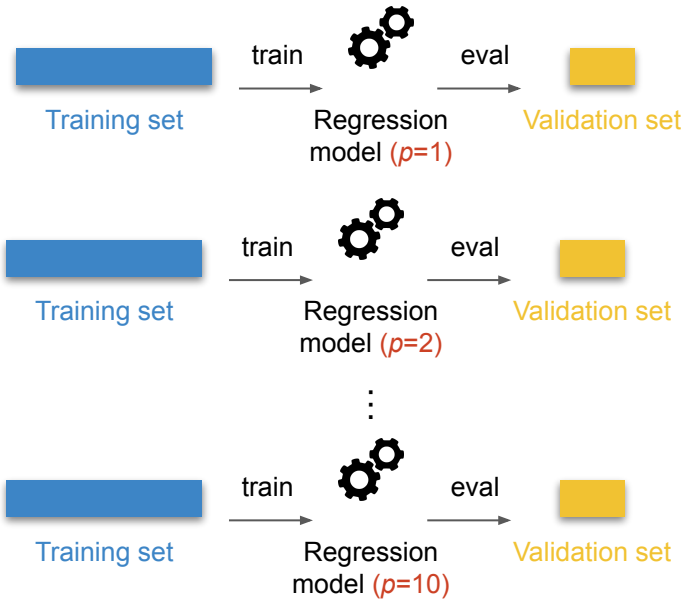
- **Hyperparameter tuning**: train multiple models with different values of *p* (e.g., 1, 2, 5, 10) and evaluate on the validation set

Use validation set to tune hyperparameters

- **Example**: how to select the degree *p* in a polynomial regression?

$$y = \theta_p \cdot x^p + \theta_{p-1} \cdot x^{p-1} + \cdots + \theta_1 x_1 + \theta_0$$

- **Hyperparameter tuning**: train multiple models with different values of *p* (e.g., 1, 2, 5, 10) and evaluate on the validation set



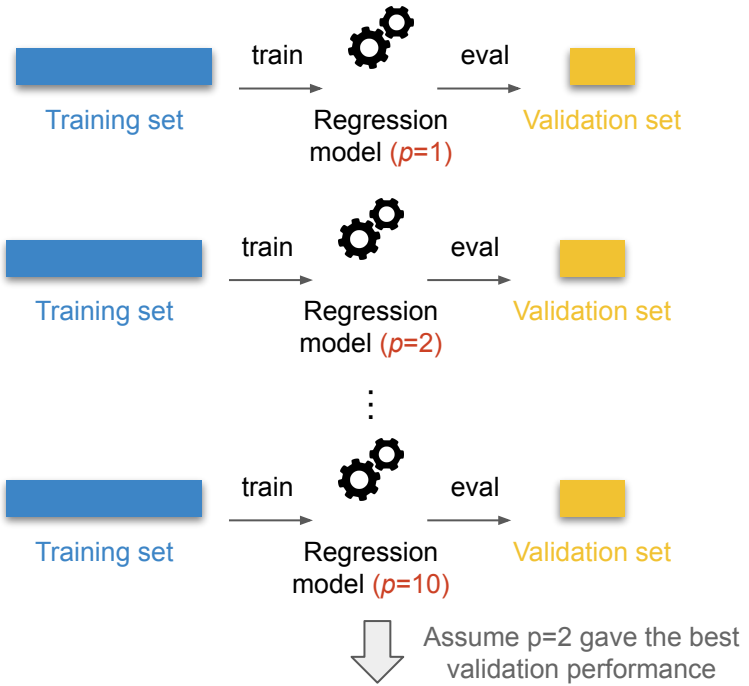
Use validation set to tune hyperparameters

- **Example**: how to select the degree *p* in a polynomial regression?

$$y = \theta_p \cdot x^p + \theta_{p-1} \cdot x^{p-1} + \cdots + \theta_1 x_1 + \theta_0$$

- **Hyperparameter tuning**: train multiple models with different values of *p* (e.g., 1, 2, 5, 10) and evaluate on the validation set

- Pick the value of *p** with best validation performance

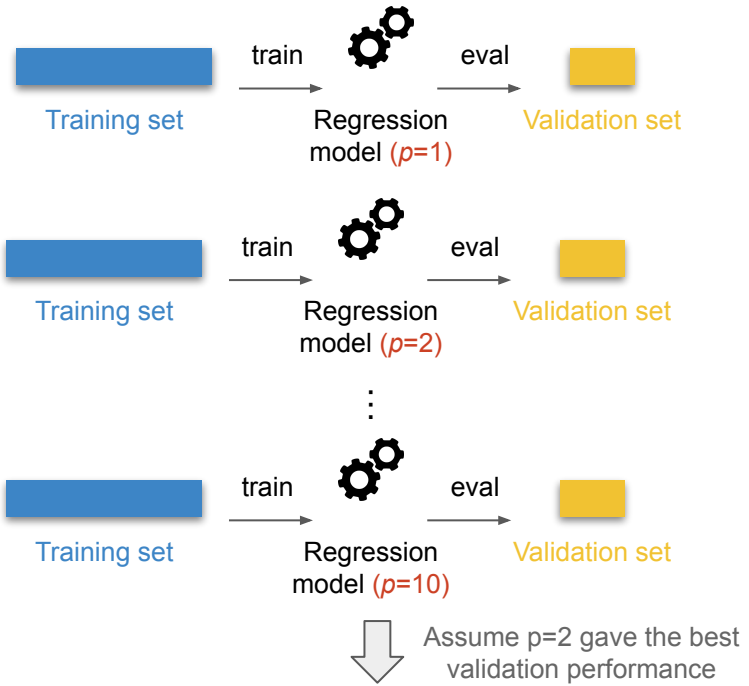


Use validation set to tune hyperparameters

- **Example**: how to select the degree *p* in a polynomial regression?

$$y = \theta_p \cdot x^p + \theta_{p-1} \cdot x^{p-1} + \cdots + \theta_1 x_1 + \theta_0$$

- **Hyperparameter tuning**: train multiple models with different values of *p* (e.g., 1, 2, 5, 10) and evaluate on the validation set
- Pick the value of *p** with best validation performance
- Train the final model using *p** on the training set + validation set
- Evaluate the final model on the test set

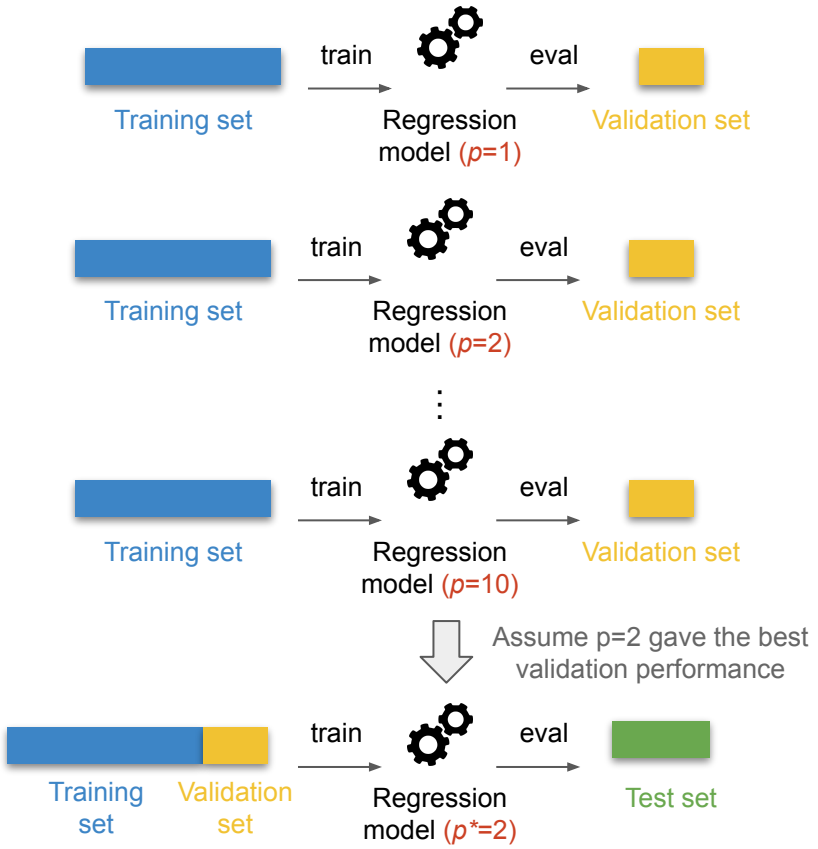


Use validation set to tune hyperparameters

- **Example**: how to select the degree *p* in a polynomial regression?

$$y = \theta_p \cdot x^p + \theta_{p-1} \cdot x^{p-1} + \cdots + \theta_1 x_1 + \theta_0$$

- **Hyperparameter tuning**: train multiple models with different values of *p* (e.g., 1, 2, 5, 10) and evaluate on the validation set
- Pick the value of *p** with best validation performance
- Train the final model using *p** on the training set + validation set
- Evaluate the final model on the test set



Use validation set to tune hyperparameters

- Only perform hyperparameter tuning on the validation set
- NEVER tune hyperparameter on the test set

How to fix overfitting?

- **Many ways to fix. Some examples:**
  - Use a simpler model family (linear models vs. neural nets)
  - Keep the same model, but collect more training data
  - Modify the training process to penalize overly complex models.

Regularization

Outline

- Model Evaluation
- Feature engineering
- Model selection
- **Regularization**

Regularization: Definition

- The idea of regularization is to train models with an augmented objective

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} L(y^{(i)}, f_\theta(x^{(i)}))$$

Regularization: Definition

- The idea of regularization is to train models with an augmented objective

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} L(y^{(i)}, f_\theta(x^{(i)})) + \lambda \cdot R(\theta)$$

Regularization: Definition

- The idea of regularization is to train models with an augmented objective

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(y^{(i)}, f_\theta(x^{(i)})) + \lambda \cdot R(\theta)$$

- Three components:
  - A loss function $L(y, f(x))$ such as the mean squared error.
  - A regularizer $R : \mathcal{M} \to \mathbb{R}$ that penalizes models that are overly complex.
  - A regularization parameter $\lambda > 0$, which controls the strength of the regularizer.

Example A: L2 Regularization

In the context of linear models $f_\theta(x) = \theta^\top x$, a widely used approach is L2 regularization, which defines the following objective:

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} L(y^{(i)}, \theta^\top x^{(i)}) + \frac{\lambda}{2} \cdot ||\theta||_2^2.$$

- The regularizer $R : \Theta \to \mathbb{R}$ is the function

$R(\theta) = ||\theta||_2^2 = \sum_{j=1}^{d} \theta_j^2$. This is also known as the L2 norm of θ.

Example A: L2 Regularization

In the context of linear models $f_\theta(x) = \theta^\top x$, a widely used approach is L2 regularization, which defines the following objective:

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} L(y^{(i)}, \theta^\top x^{(i)}) + \frac{\lambda}{2} \cdot ||\theta||_2^2.$$

- The regularizer $R : \Theta \to \mathbb{R}$ is the function

$R(\theta) = ||\theta||_2^2 = \sum_{j=1}^{d} \theta_j^2$. This is also known as the L2 norm of θ.

- The regularizer penalizes large parameters. This prevents us from relying on any single feature and penalizes very irregular solutions.

- L2 regularization can be used with most models (linear, neural, etc.)

L2 Regularization for Polynomial Regression

- The polynomial regression $f(x) = \theta_p \cdot x^p + \theta_{p-1} \cdot x^{p-1} + \cdots + \theta_1 x \ + \theta_0$

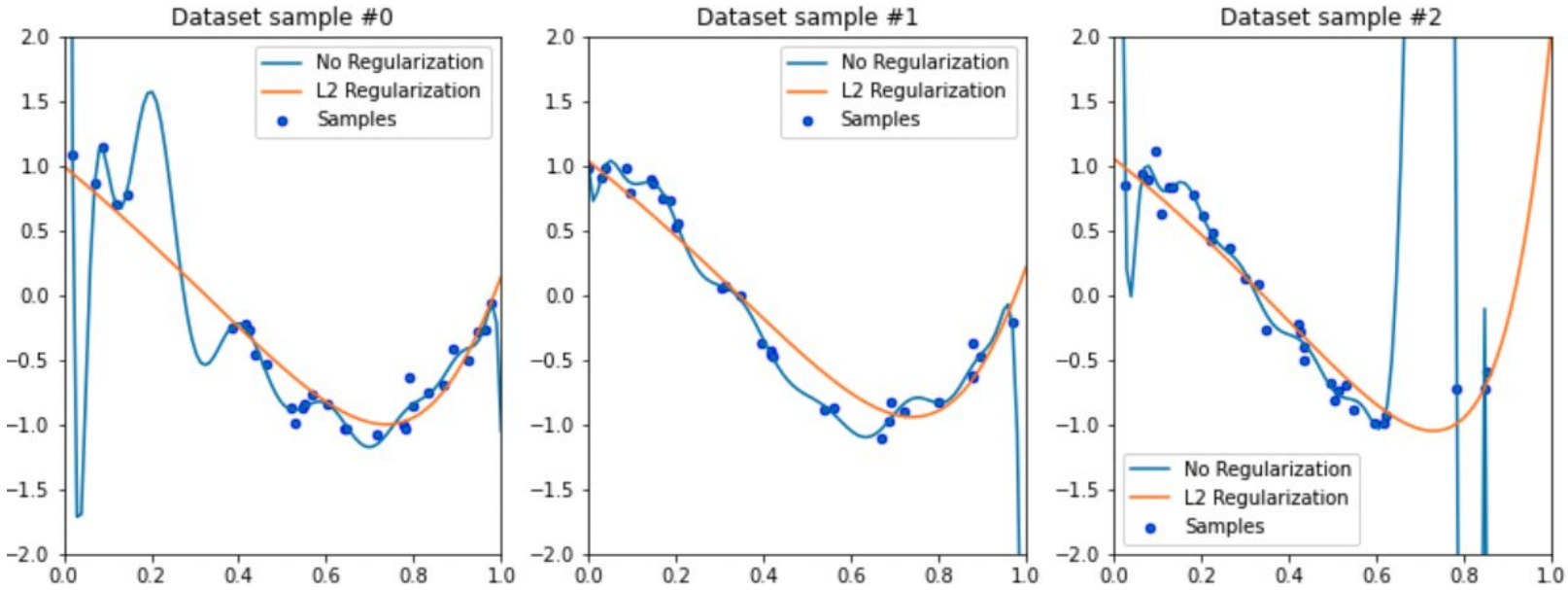
  can also be written in a vector form:

$$f_\theta(x) := \sum_{j=0}^{p} \theta_p x^p = \theta^\top \phi(x) \qquad \phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^p \end{bmatrix}$$

- The polynomial regression with L2 regularization has the following objective

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} \left(y^{(i)} - \theta^\top \phi(x^{(i)}) \right)^2 + \frac{\lambda}{2} \cdot ||\theta||_2^2.$$

Results



- In order to define a very irregular function, we need very large polynomial weights.
- Forcing the model to use small weights prevents it from learning irregular functions.

How to Choose λ? Hyperparameter Tuning

$$J(\theta) = \frac{1}{2n}\sum_{i=1}^{n}\left(y^{(i)} - \theta^{\top}\phi(x^{(i)})\right)^2 + \frac{\lambda}{2}\cdot||\theta||_2^2.$$

We refer to λ as a **hyperparameter**, because it's a high-level parameter that controls other parameters.

How do we choose λ?

- We select the λ with the best performance on the development set.
- If we don't have enough data, we select λ by cross-validation.

Example B: L1 Regularization

Another closely related approach to regularization is to penalize the size of the weights using the L1 norm.

In the context of linear models $f(x) = \theta^\top x$, L1 regularization yields the following objective:

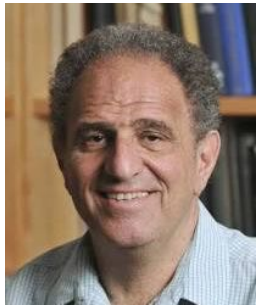
$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(y^{(i)}, \theta^\top x^{(i)}) + \lambda \cdot ||\theta||_1.$$

- The regularizer $R : \mathcal{M} \to \mathbb{R}$ is

$R(\theta) = ||\theta||_1 = \sum_{j=1}^{d} |\theta_j|$. This is known as the L1 norm of θ.

- This regularizer also penalizes large weights. It additionally forces most weights to decay to zero, as opposed to just being small.

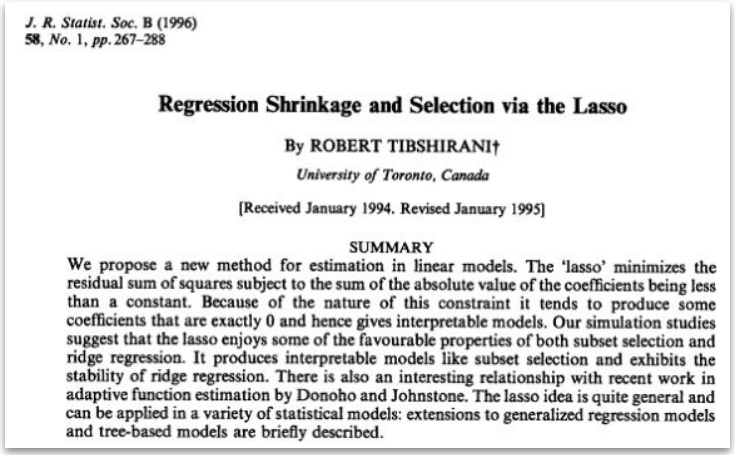
LASSO Regression



Robert Tibshirani

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(y^{(i)}, \theta^\top x^{(i)}) + \lambda \cdot ||\theta||_1$$

- L1-regularized linear regression is also known as the LASSO (least absolute shrinkage and selection operator)
- LASSO forces the weights of “unimportant” features to be **exactly zero** (not only being small)



J. R. Statist. Soc. B (1996)
58, *No.* 1, *pp.* 267–288

Regression Shrinkage and Selection via the Lasso

By ROBERT TIBSHIRANI†

University of Toronto, Canada

[Received January 1994. Revised January 1995]

SUMMARY

We propose a new method for estimation in linear models. The ‘lasso’ minimizes the residual sum of squares subject to the sum of the absolute value of the coefficients being less than a constant. Because of the nature of this constraint it tends to produce some coefficients that are exactly 0 and hence gives interpretable models. Our simulation studies suggest that the lasso enjoys some of the favourable properties of both subset selection and ridge regression. It produces interpretable models like subset selection and exhibits the stability of ridge regression. There is also an interesting relationship with recent work in adaptive function estimation by Donoho and Johnstone. The lasso idea is quite general and can be applied in a variety of statistical models: extensions to generalized regression models and tree-based models are briefly described.

The LASSO paper has been cited by 57k+ times

Conclusion: important principles in ML

- Model Evaluation
- Feature engineering
- Model selection
- Regularization