Supervised Learning

Leading Bootcamp

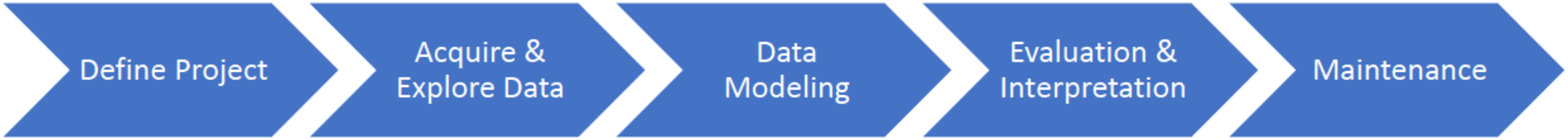
What is machine learning?

- Machine Learning is the science (and art) of programming computers so they can learn from data

- More general definition (Arthur Samuel, 1959):
  - Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed.

- More engineering-oriented definition (Tom Mitchell, 1997):
  - Machine Learning is the study of algorithms that
    - Improve their performance P
    - At some task T
    - With experience E
  - A well-defined learning task is given by <P, T, E>
  - Examples: classify emails as spam or not

Data Science Lifecycle

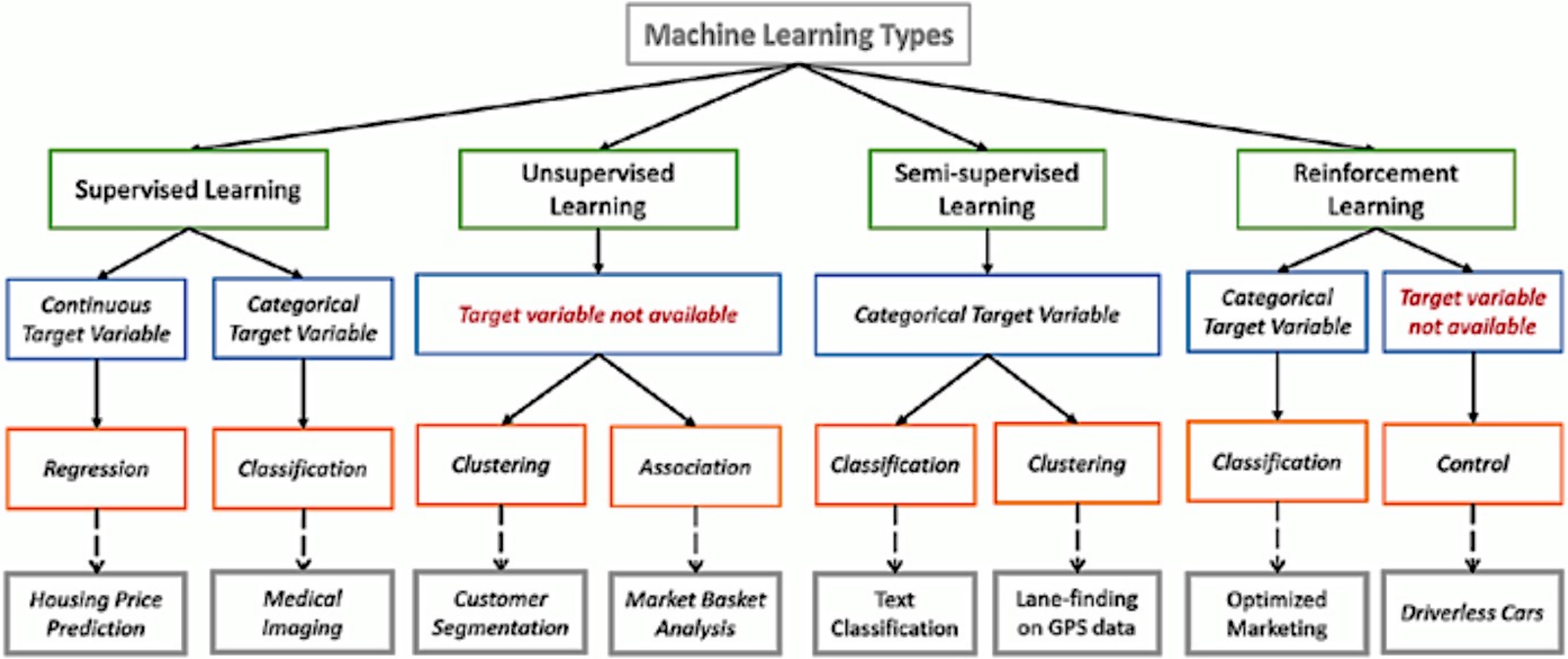


- **Define Projects**
  - What problem you are trying to solve?
  - What is the evaluation criterion?
  - What is the baseline?

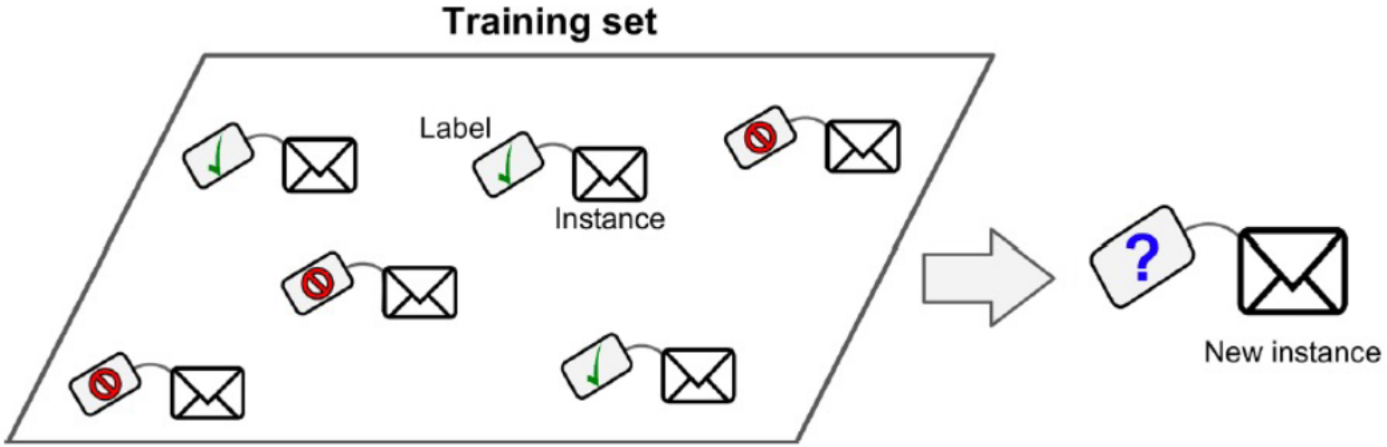
- **Acquire & Explore Data**
  - Collecting appropriate data
  - Cleaning and EDA
  - Feature engineering

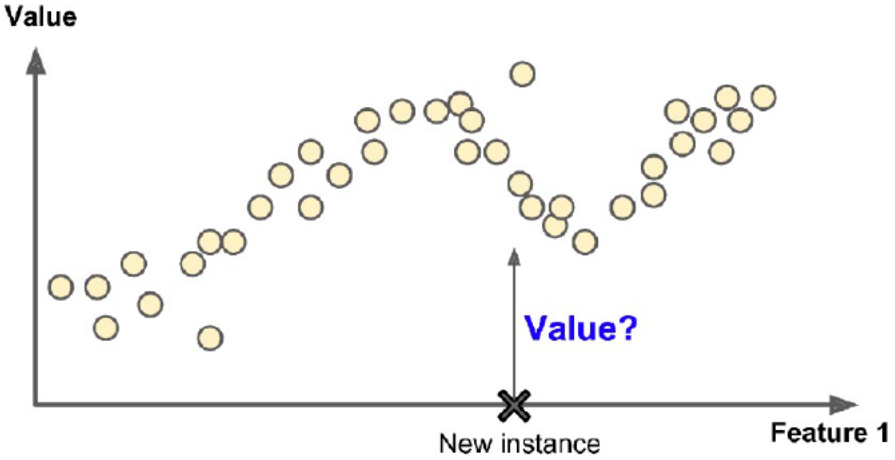
- **Data Modelling**
  - Build candidate models
  - Variable selection
  - Hyperparameter tuning
- **Evaluation & Interpretation**
  - Performance exceeds the baseline?
  - Interpret the model
- **Maintenance**
  - How to monitor performance in the future?
  - Frequency to re-train the model

Types of Machine Learning



Supervised Learning



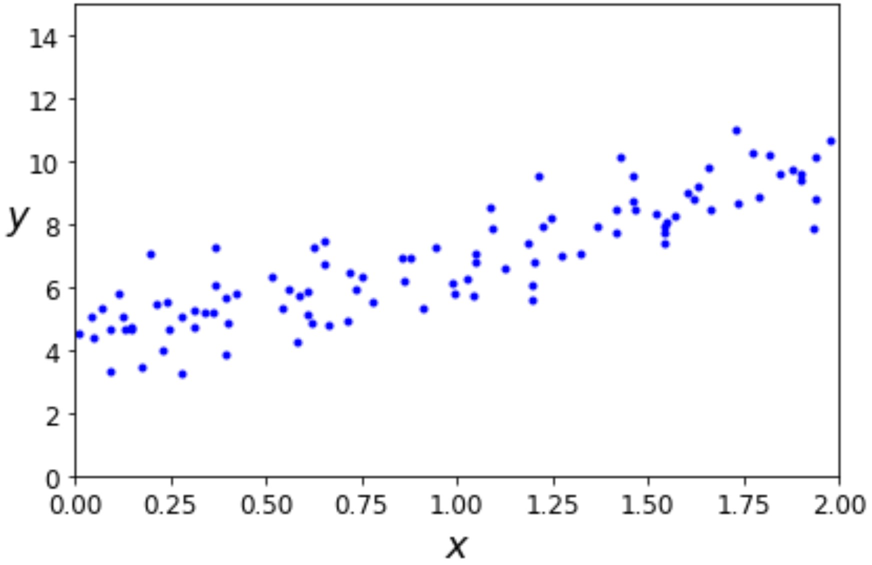


Supervised Learning

- Nearest Neighbor
- Naïve Bayes
- Decision Trees
- Linear Regression
- Logistic Regression
- Support Vector Machines
- Neural Networks

Linear Regression

- Given Data
  - Observations: $X = \{x^{(1)}, x^{(2)}, x^{(3)}, \dots, x^{(n)}\}$
  - Corresponding labels: $Y = \{y^{(1)}, y^{(2)}, y^{(3)}, \dots, y^{(n)}\}$



Linear Regression

- Hypothesized model

$$
\begin{aligned}
\hat{y} &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_d x_d \\
&= \sum \theta_j \, x_j = h_\theta(x)
\end{aligned}
$$

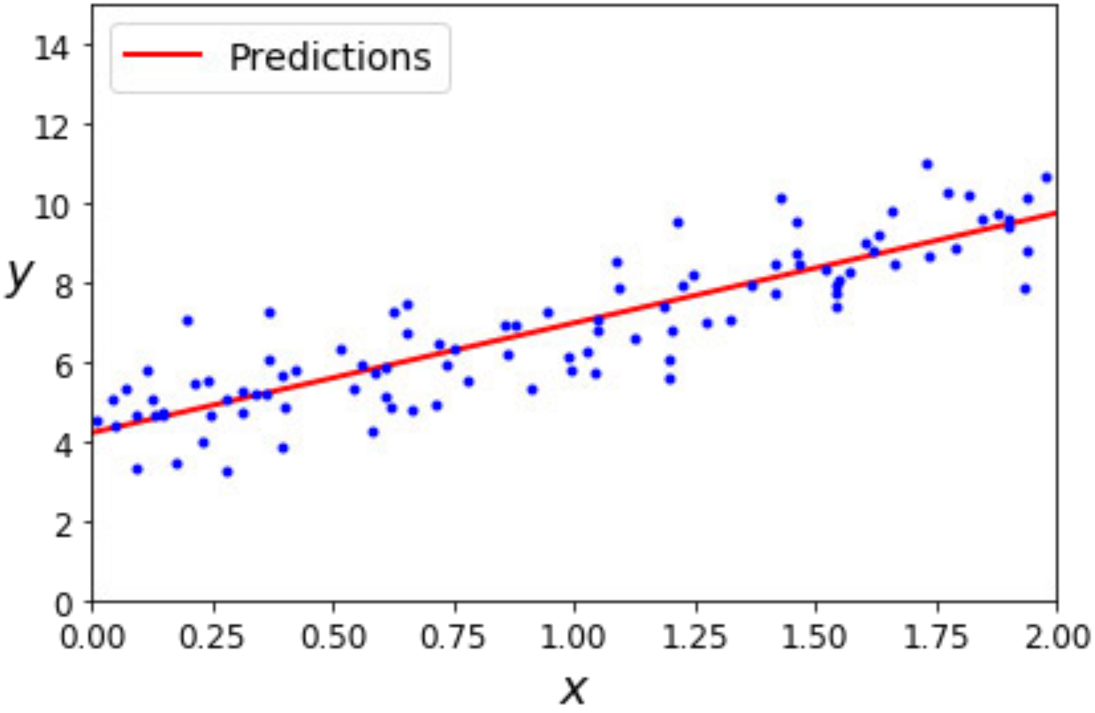
- Vectorized form

$$\hat{y} = h_\theta(\mathbf{x}) = \theta^T \cdot \mathbf{x}$$

Linear Regression

- Closed form solution (normal equation)

$$\hat{\theta} = \left(\mathbf{X}^T \cdot \mathbf{X}\right)^{-1} \cdot \mathbf{X}^T \cdot \mathbf{y}$$



Evaluating metric for regression

- Root Mean Square Error (RMSE)

$$\mathrm{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right)^2}$$

- Sum of squared error (SSE)

$$SSE(X, h_\theta) = \sum(h_\theta\left(x^{(i)}\right) - y^{(i)})^2$$

Linear Regression

- Implement on Scikit-Learn

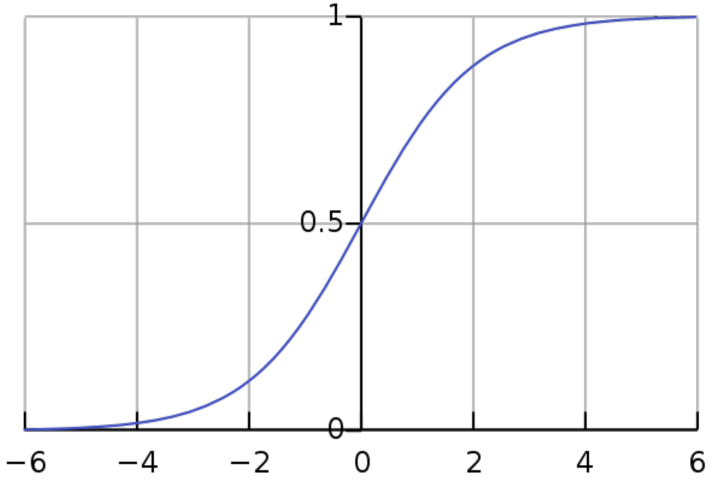
```
>>> from sklearn.linear_model import LinearRegression
>>> lin_reg = LinearRegression()
>>> lin_reg.fit(X, y)
>>> lin_reg.intercept_, lin_reg.coef_
>>> lin_reg.predict(X_new)
```

Logistic Regression

- Spam (prob = 0.6), not spam (prob = 0.4)
- Probability estimation

$$\hat{p} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma(\mathbf{x}^{\mathsf{T}}\boldsymbol{\theta})$$

- Logistic function: $\sigma(t) = \dfrac{1}{1 + \exp(-t)}$



Logistic Regression

- Implement on Scikit-Learn

```
>>> from sklearn.linear_model import LogisticRegression
>>> logR = LogisticRegression()
>>> logR.fit(X_train, y_train)
>>> logR.predict(X_test)
```

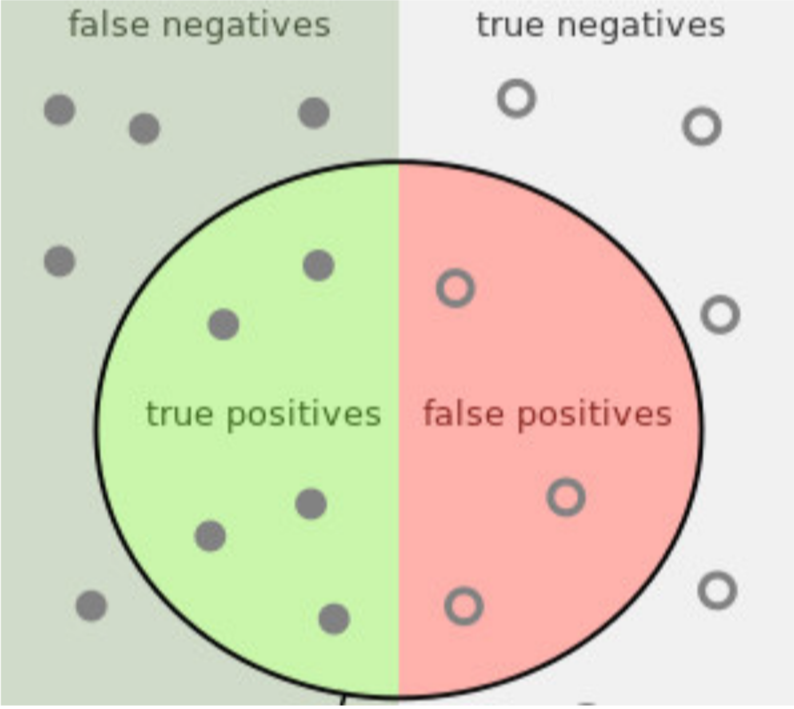
Performance evaluation

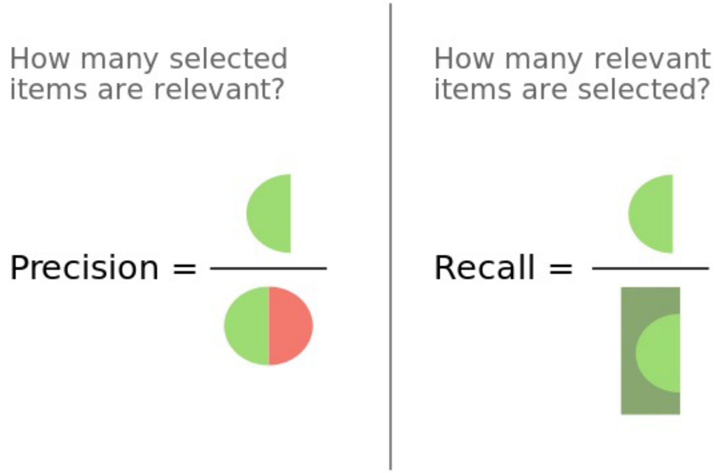
- **Accuracy**

```
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_pred=y_prediction, y_true=y_train)
```

Performance evaluation

- **Precision**: accuracy of the positive prediction
- **Recall** (sensitivity/True Positive Rate, TPR): ratio of positive instances that are correctly detected by the classifier




How many selected items are relevant?
Precision =
How many relevant items are selected?
Recall =

$$\text{precision} = \frac{TP}{TP + FP} \qquad \text{recall} = \frac{TP}{TP + FN}$$

Performance evaluation

- Precision and Recall

```
>>> from sklearn.metrics import precision_score, recall_score
>>> precision_score(y_pred=y_prediction, y_true=y_train)
>>> recall_score(y_pred=y_prediction, y_true=y_train)
```

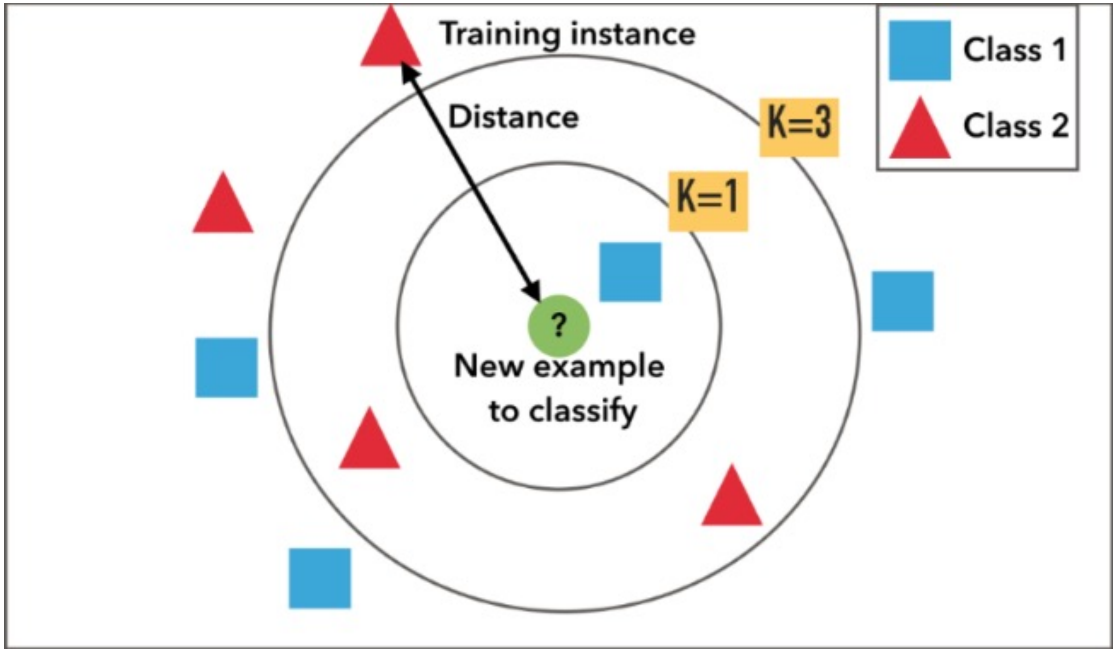
- F1 score: combination of Precision and Recall

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{TP}{TP + \frac{FN + FP}{2}}$$

```
>>> from sklearn.metrics import f1_score
>>> f1_score(y_pred=y_prediction, y_true=y_train)
```

K Nearest Neighbors

- Use k “closet” (nearest neighbors) for performing classification
- Instance based classifier



K Nearest Neighbors

- Required:
  - Stored instances
  - Distance metric to compute between instances
  - Value of k, number of nearest neighbors to retrieve

- To classify a new instance:
  - Compute distances to other training instances
  - Identify k nearest neighbors
  - Determine the label of new instances based on k nearest neighbors

K Nearest Neighbors

- Implement on Scikit-Learn
  - Classification

```
>>> from sklearn.neighbors import KNeighborsClassifier
>>> KNN_classifier=KNeighborsClassifier(n_neighbors=5)
>>> KNN_classifier.fit(X_train, y_train)
>>> KNN_classifier.predict(X_test)
```

  - Regression

```
>>> from sklearn.neighbors import KNeighborsRegressor
>>> KNN_regressor=KNeighborsRegressor(n_neighbors=5)
>>> KNN_regressor.fit(X_train, y_train)
>>> KNN_regressor.predict(X_test)
```

Naïve Bayes

- Simple, probabilistic classifier
- Assumptions:
    - Assume all features are independent (Naïve)
    - Features have no correlations with each other
- High efficiency
- Performance may not be good
- Could be competitive for some tasks

Bayes Rule

$$\text{posterior probability} = \frac{\text{conditional probability} \cdot \text{prior probability}}{\text{evidence}}$$

$$P(y \mid d) = \frac{P(d \mid y)P(y)}{P(d)}$$

- $P(d)$: independent probability of d. can be treated as constant or independent of hypothesis
- $P(y)$: prior probability
- $P(d|y)$: conditional probability, likelihood
- $P(y|d)$: posterior probability

$$P(d \mid y) = P(w_1, \cdots, w_{M_d} \mid y) = \prod_{k=1}^{M_d} P(w_k \mid y)$$

Naïve Bayes

- **Multinomial Naive Bayes:** This is mostly used for document classification problem. Like whether a document belongs to the category of sports, politics, technology etc. The features/predictors used by the classifier are the frequency of the words present in the document.
- **Bernoulli Naive Bayes:** This is similar to the multinomial Naïve Bayes but the predictors are Boolean variables. The parameters that we use to predict the class variable take up only values yes or no, for example if a word occurs in the text or not.
- **Gaussian Naive Bayes:** When the predictors take up a continuous value and are not discrete, we assume that these values are sampled from a gaussian distribution.

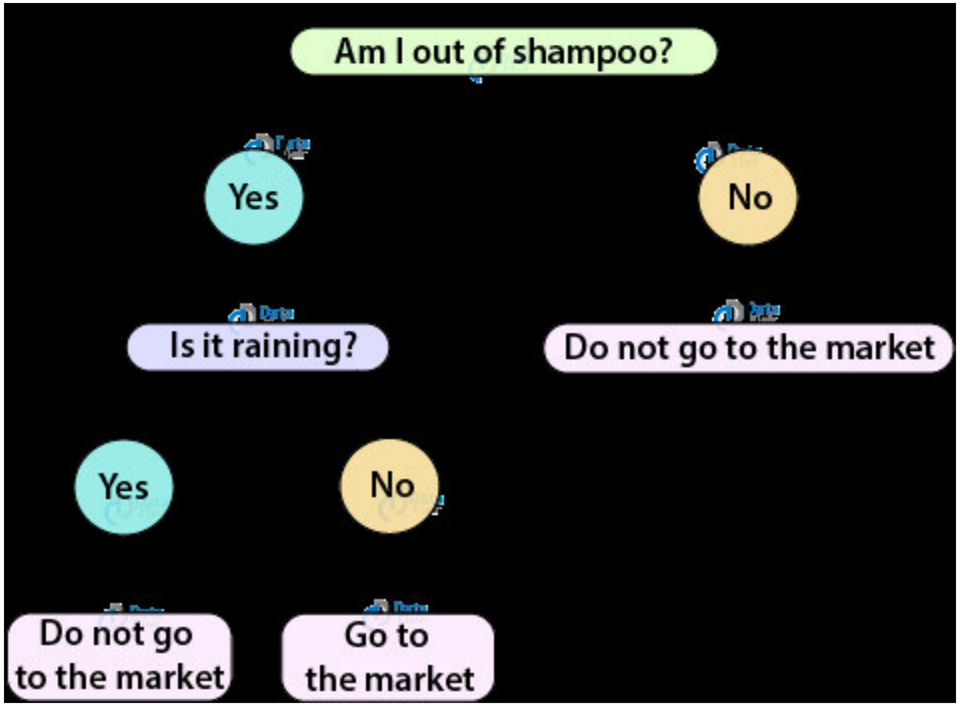
Naïve Bayes

- Implement on Scikit-Learn
  - Classification

```
>>> from sklearn.naive_bayes import GaussianNB
>>> from sklearn.naive_bayes import BernoulliNB
>>> from sklearn.naive_bayes import MultinomialNB
>>> GaussianNB = GaussianNB()
>>> GaussianNB.fit(X_train, y_train)
>>> GaussianNB.predict(X_test)
```

Decision Trees

- A series of binary questions



Decision Tree

- Implement on Scikit-Learn
  - Classification

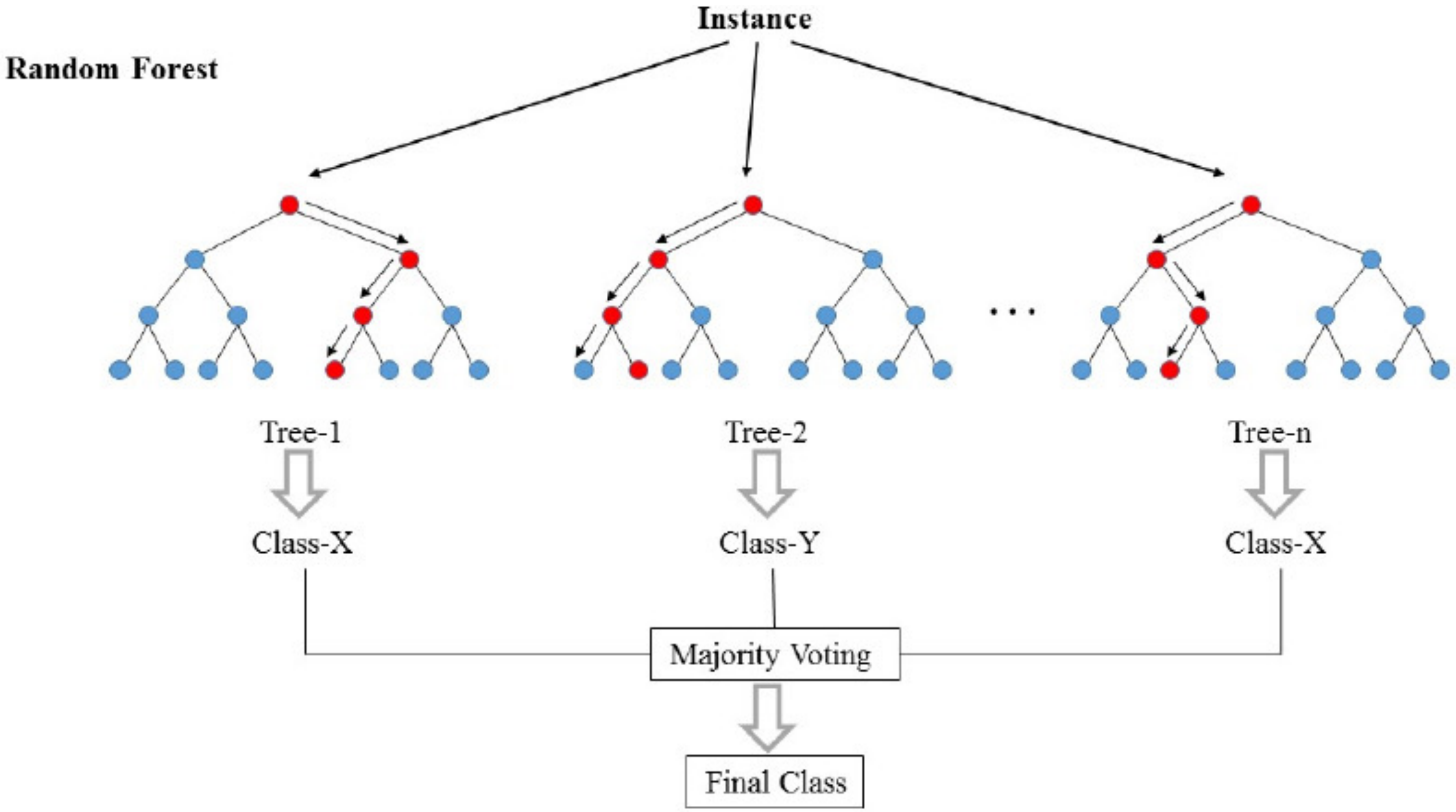
```
>>> from sklearn.tree import DecisionTreeClassifier
>>> DT_classifier = DecisionTreeClassifier()
>>> DT_classifier.fit(X_train, y_train)
>>> DT_classifier.predict(X_test)
```

  - Regression

```
>>> from sklearn.tree import DecisionTreeRegressor
>>> DT_regressor = DecisionTreeRegressor()
>>> DT_regressor.fit(X_train, y_train)
>>> DT_regressor.predict(X_test)
```

Random Forest

- An ensemble method using many decision trees



Random Forest

- Implement on Scikit-Learn
  - Classification

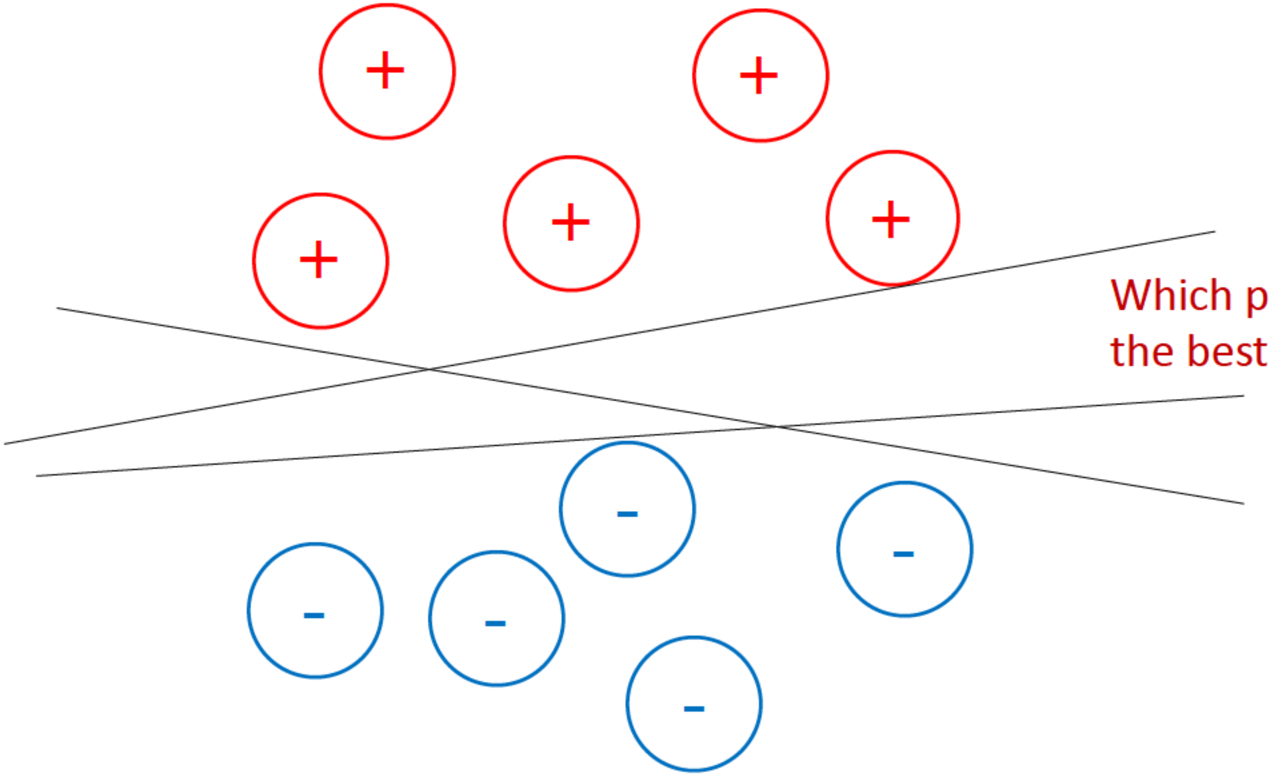
```
>>> from sklearn.ensemble import RandomForestClassifier
>>> RF_classifier = RandomForestClassifier()
>>> RF_classifier.fit(X_train, y_train)
>>> RF_classifier.predict(X_test)
```

  - Regression

```
>>> from sklearn.ensemble import RandomForestRegressor
>>> RF_regressor = RandomForestRegressor()
>>> RF_regressor.fit(X_train, y_train)
>>> RF_regressor.predict(X_test)
```

Support Vector Machine (SVM)

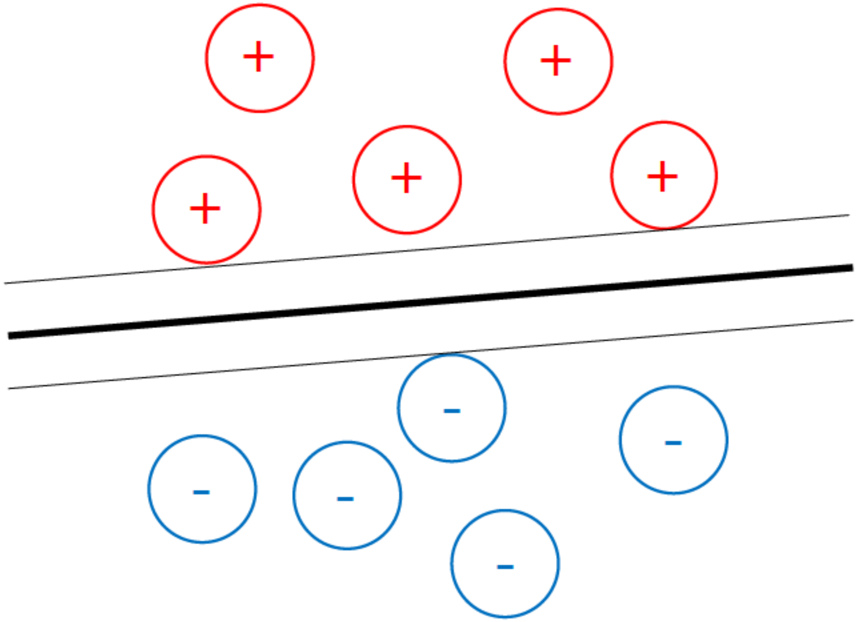
- A linearly separable binary classification dataset
- There are a number of hyper-planes could separate the classes

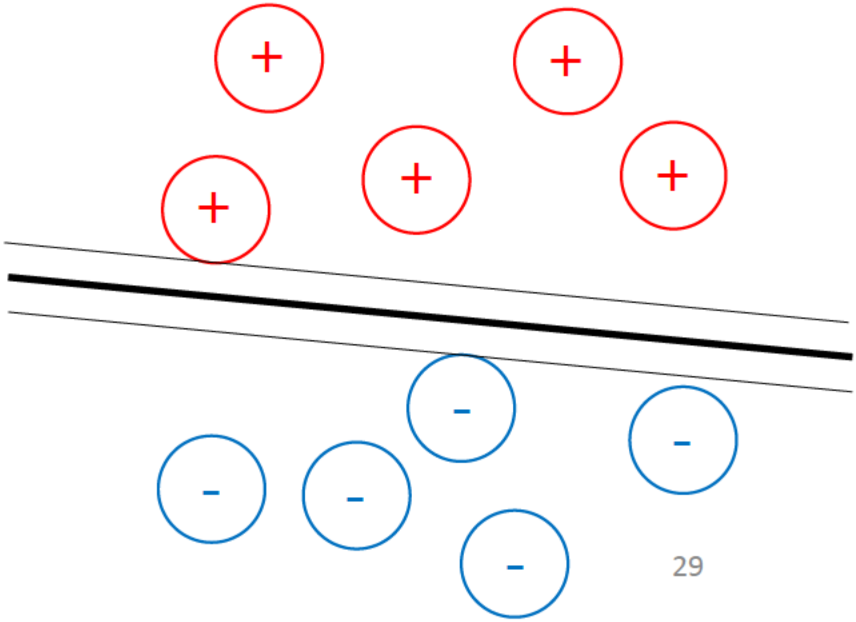


Which plane is the best?

Support Vector Machine (SVM)

- For a given separating hyper-plane, the **margin** is twice the (Euclidean) distance from hyper-plane to nearest training example.
- Why maximal margin?



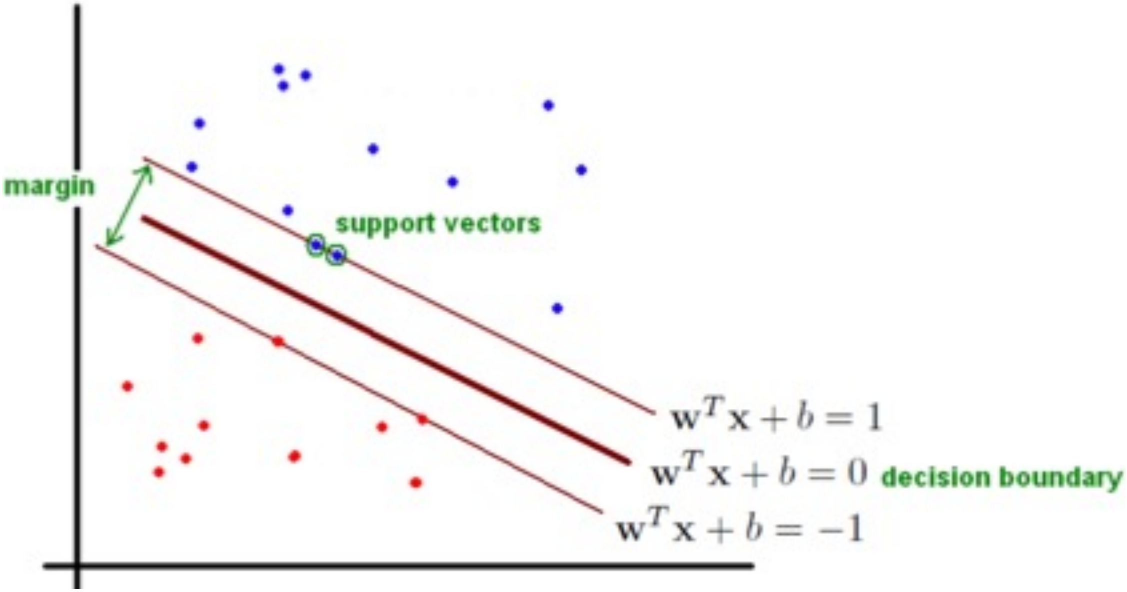


Support Vector Machine

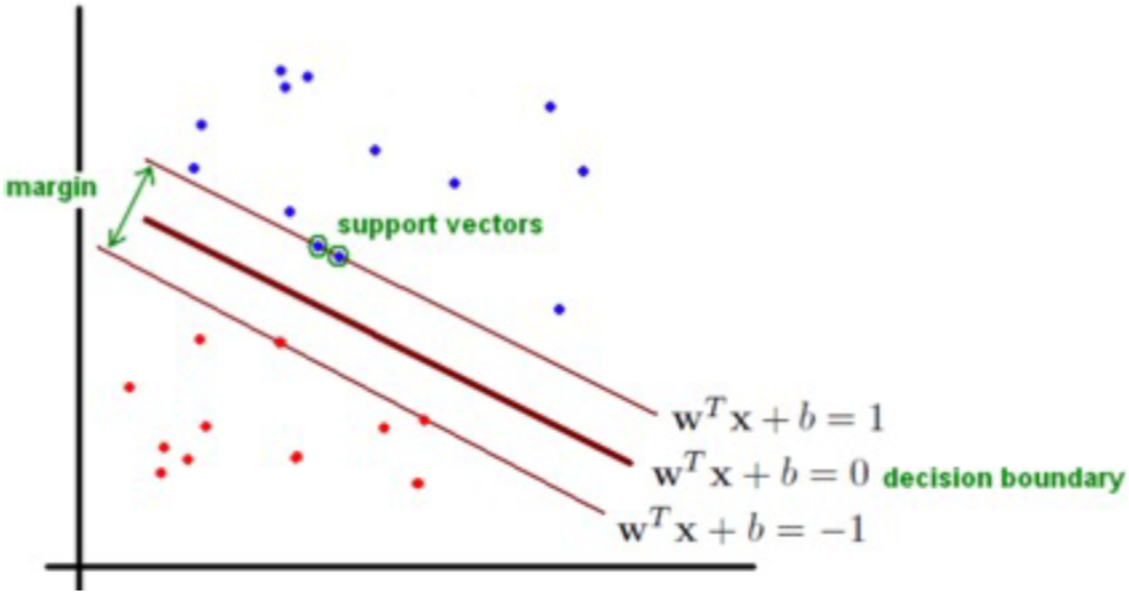
- Linear SVM decision function

$$\mathbf{w}^{\mathsf{T}}\,\mathbf{x} + b = w_1\,x_1 + \cdots + w_n\,x_n + b$$

$$\hat{y} = \begin{cases} 0 & \text{if } \mathbf{w}^{\mathsf{T}}\mathbf{x} + b < 0, \\ 1 & \text{if } \mathbf{w}^{\mathsf{T}}\mathbf{x} + b \geq 0 \end{cases}$$


margin
support vectors
$\mathbf{w}^T\mathbf{x} + b = 1$
$\mathbf{w}^T\mathbf{x} + b = 0$ decision boundary
$\mathbf{w}^T\mathbf{x} + b = -1$

Support Vector Machine


margin
support vectors
$\mathbf{w}^T\mathbf{x} + b = 1$
$\mathbf{w}^T\mathbf{x} + b = 0$
decision boundary
$\mathbf{w}^T\mathbf{x} + b = -1$

$$w^T x_+ + b = 1$$

$$w^T x_- + b = -1$$

↓

$$w^T(x_+ - x_-) = 2$$

→

$$\hat{w}^T(x_+ - x_-) = \frac{2}{\|w\|}$$

Support Vector Machine

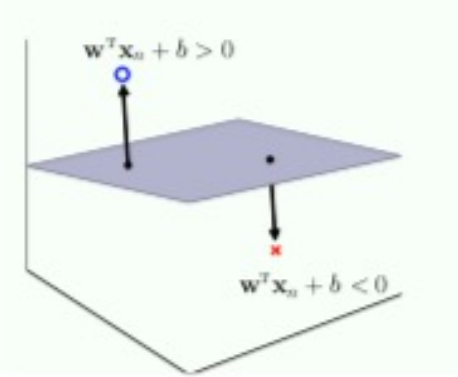
- Maximize the margin $\frac{2}{\|w\|} \Leftrightarrow$ minimize the $\|w\|$.

- Objective: maximize the margin while correctly classifying all data correctly.

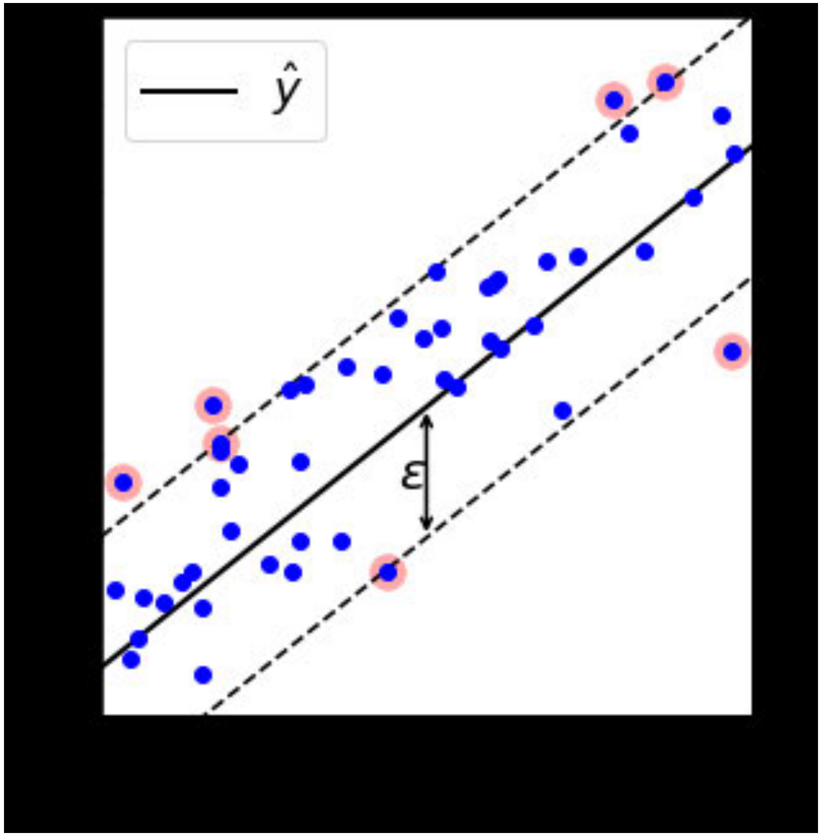
$$\underset{w,b}{minimize}\ \frac{1}{2}\|w\|^2$$

$$subject\ to\colon\ y_i(w^T x_i + b) \geq 1$$

$$(i = 1,2,3,\ldots,m)$$


$\mathbf{w}^T\mathbf{x}_n + b > 0$
$\mathbf{w}^T\mathbf{x}_n + b < 0$

SVM for regression



Support Vector Machine

- Implement on Scikit-Learn
    - Classification

```
>>> from sklearn.svm import LinearSVC
>>> SVM_classifier = LinearSVC()
>>> SVM_classifier.fit(X_train, y_train)
>>> SVM_classifier.predict(X_test)
```

- Regression

```
>>> from sklearn.svm import LinearSVR
>>> SVM_regressor = LinearSVR()
>>> SVM_regressor.fit(X_train, y_train)
>>> SVM_regressor.predict(X_test)
```