# Training a logistic regression model with scikit-learn

- how to use scikit-learn's more optimized implementation of logistic regression, which also supports multiclass settings off the shelf.
  - optimization algorithms – newton-cg, lbfgs, liblinear, sag, and saga, in addition to SGD
    - lbfgs – limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm
    - input parameter: solver = 'lbfgs'
  - multiclass classification - multinomial or OvR can be chosen
    - input parameter: multi_class = 'ovr', or 'multinomial'

# Training a logistic regression model with scikit-learn

- tackling overfitting via regularization
  - introduce additional information (bias) to penalize extreme parameter (weight) values.
  - L2 regularization (shrinkage or weight decay)

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

$\lambda$ – regularization parameter

  - For regularization to work properly, ensure that all features are on comparable scales.

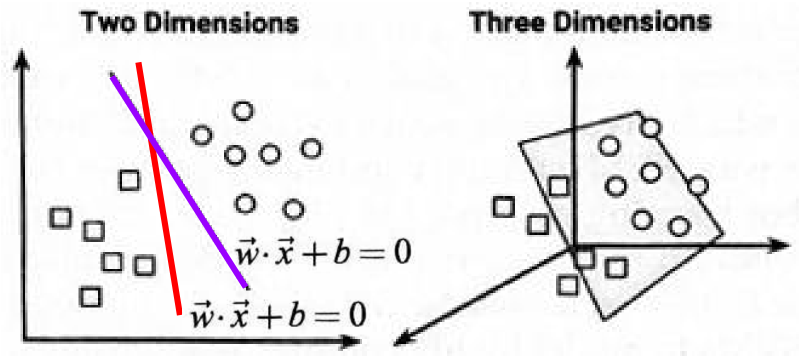
regularization strength

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

  - via the regularization parameter $\lambda$, we can control how well we fit the training data, while keeping the weights small

# support vector machines

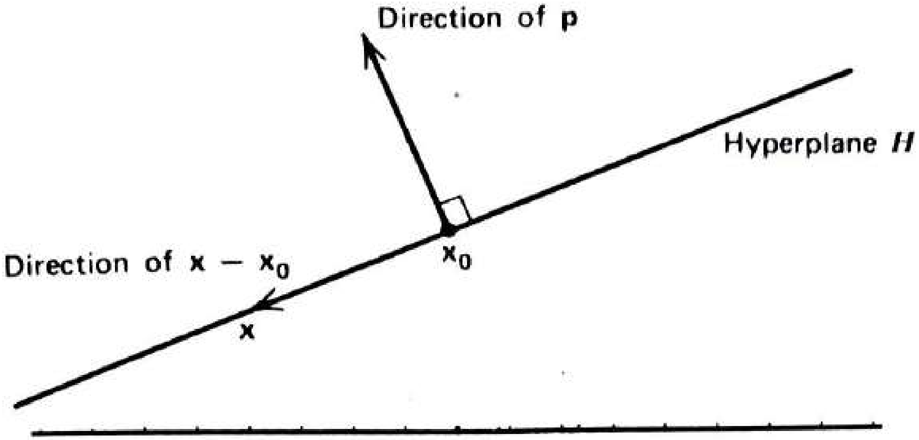
- SVMs use a boundary called a hyperplane to partition data into two groups of similar class values.




case: samples are linearly separable.

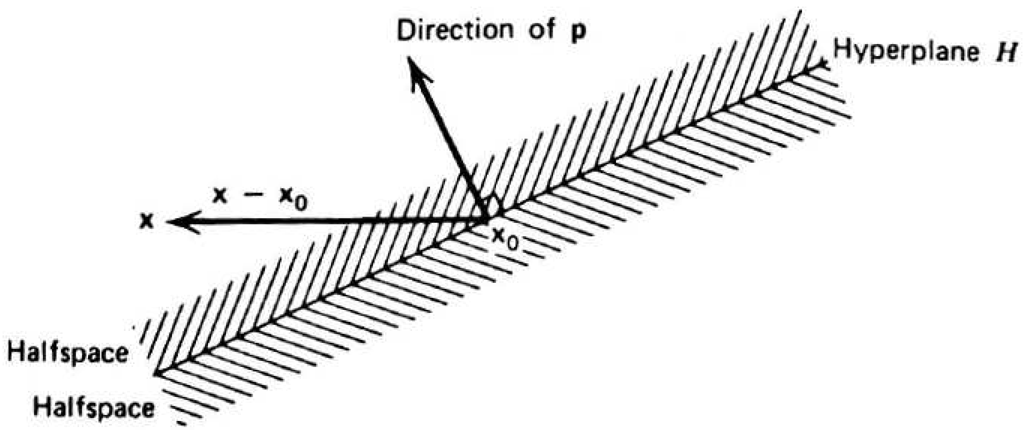
# Hyperplane and halfspace

- hyperplane H in $E^n$ is a set of x such that px = k where p is a nonzero vector in $E^n$ and normal to the hyperplane, and k is scalar



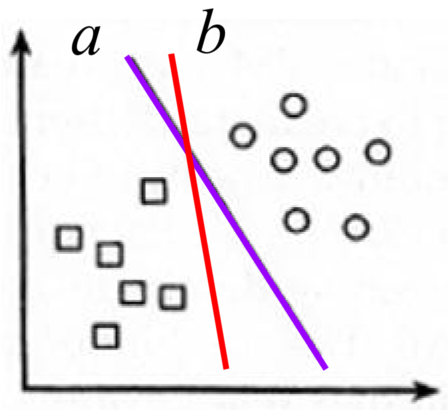
# Hyperplane and halfspace

- hyperplane divides $E^n$ into two regions, called halfspaces.
- halfspace is a collection of points of the form $\{x: px \geq k\}$
- another halfspace is a collection of points of the form $\{x: Px \leq k\}$



# support vector machines

- In two dimensions, the task of the SVM algorithm is to identify a line that separates the two classes.



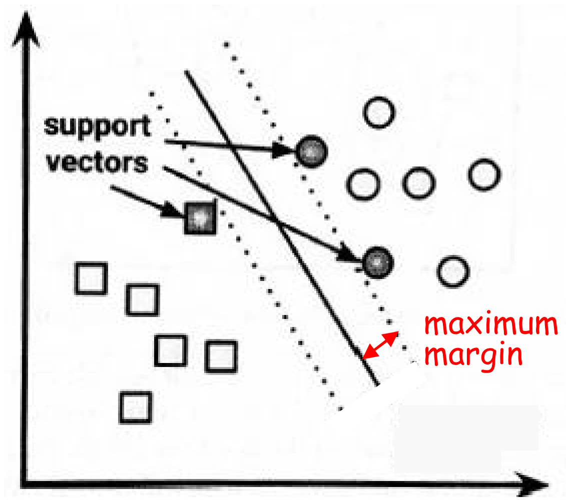

case: samples are linearly separable.

- Question is how does the algorithm choose the most appropriate one.

  Answer – find the maximum margin hyperplane

# support vectors and MMH

- maximum margin hyperplane (MMH) creates the greatest separation between the two classes.
- MMH will generalize best to future data.
- support vectors are the points from each class that are the closest to the MMH.
- support vectors alone define the MMH.
  - support vectors provide a very compact way to store a classification model.

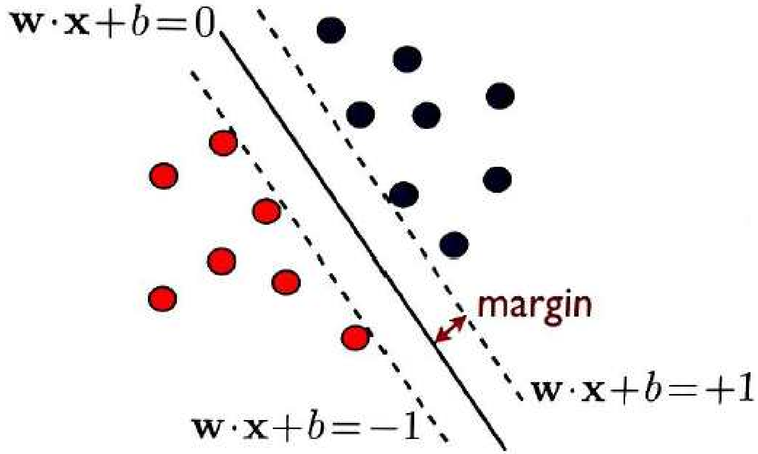


# How to determine the MMH?

- general equation of a hyperplane in $\mathbb{R}^N$

- scale w and b such that min |wx + b| = 1 over all training data.
  - The corresponding hyperplane is called canonical hyperplane.

$$w \cdot x + b = +1 \text{ or } w \cdot x + b = -1$$




- For a canonical hyperplane, margin $\rho$ is given by $1/\|w\|$

# How to determine the MMH?

- case: classes are linearly separable

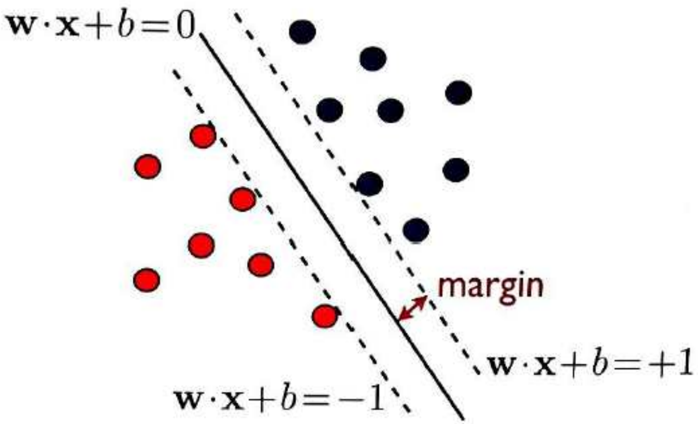
hyperplane:

$$\vec{w} \cdot \vec{x} + b = 0$$

w = {$w_1$, $w_2$, … ,$w_n$}
b: intercept, scalar

For a canonical hyperplane, we have

Thus $x^{(i)}$ is correctly classified when

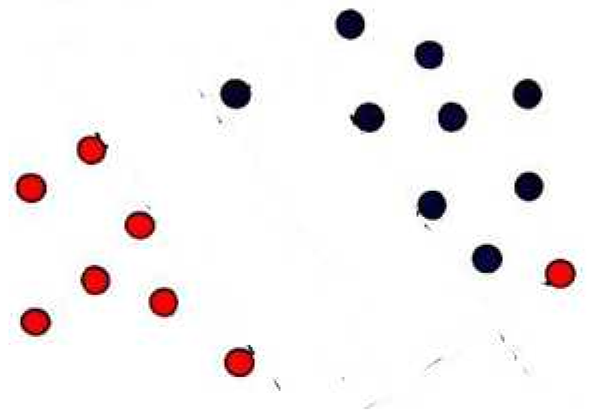



Find a hyperplane satisfying these with the biggest margin.

# How to determine the MMH?

## case: nonlinearly separable data

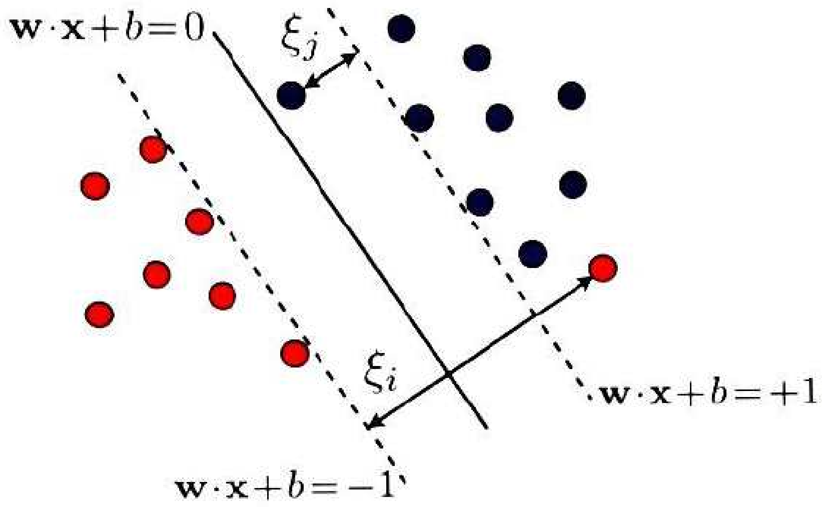
- What happens in the case that the data are not linearly separable?
- the constraints imposed in the linearly separable case cannot be met.



# How to determine the MMH?

## case: nonlinearly separable data

- Solution : creates a soft margin that allows some points to fall on the incorrect side of the margin

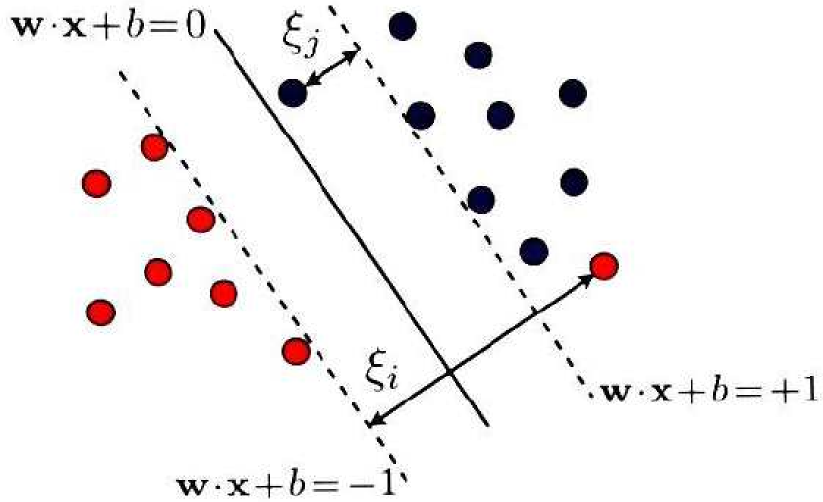


# How to determine the MMH?

## case: nonlinearly separable data

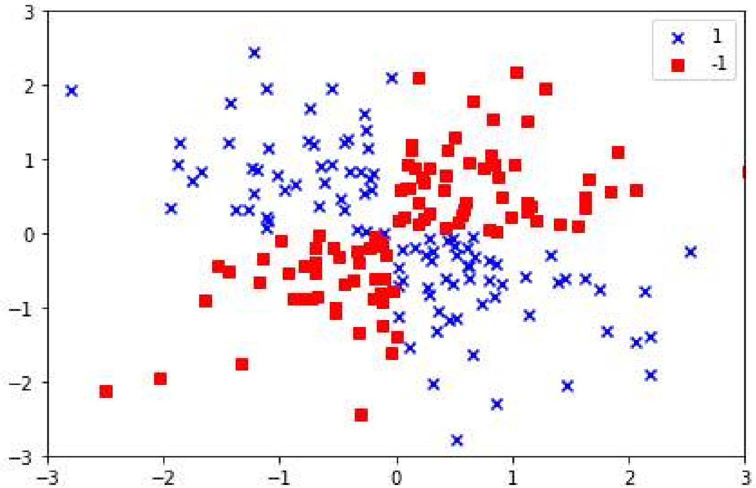
- problem formulation

$$\min \ \frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$
$$s.t.\ \ y_i(\vec{w}\cdot\vec{x}_i + b) \geq 1 - \xi_i, \forall\, \xi_i \geq 0$$



# Kernel methods for linearly inseparable data

- In many real-world datasets, relationship between variables are nonlinear. Using some mapping technique, a nonlinear relationship can be made quite linear.
- example dataset not linearly separable, XOR data

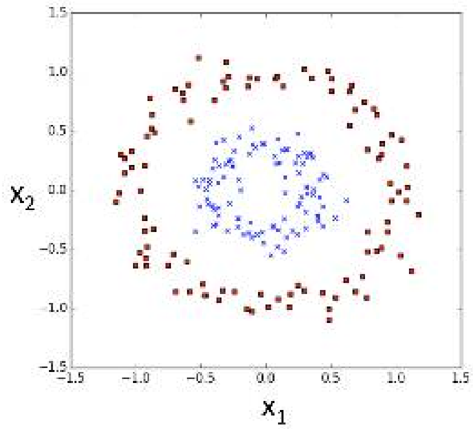


# Kernel methods for linearly inseparable data

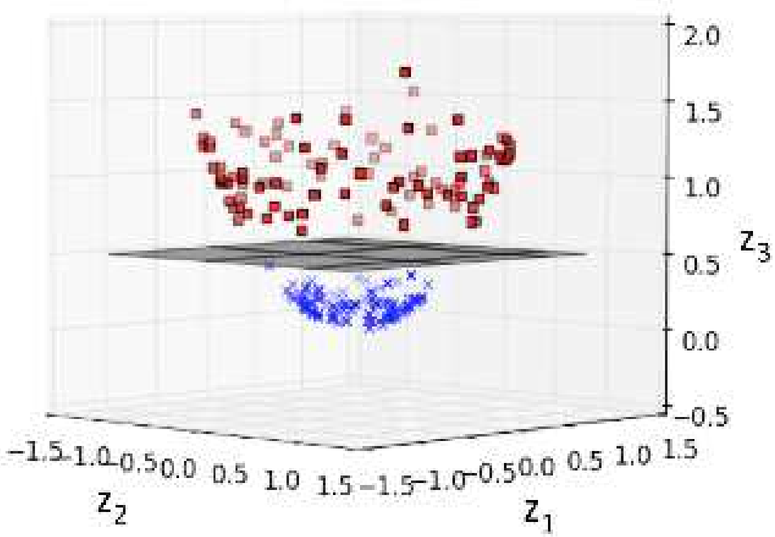
- transform a linearly inseparable data to the data linearly separable
- uses nonlinear combinations of the original features
- original space is transformed to a higher-dimensional space via a mapping function, $\phi$
- example – two dimensional dataset is transformed into a new three-dimensional feature space, where the classes become separable

$$\phi(x_1, x_2) = (z_1, z_2, z_3) = (x_1, x_2, x_1^2 + x_2^2)$$

# Kernel methods for linearly inseparable data




$\xrightarrow{\phi}$

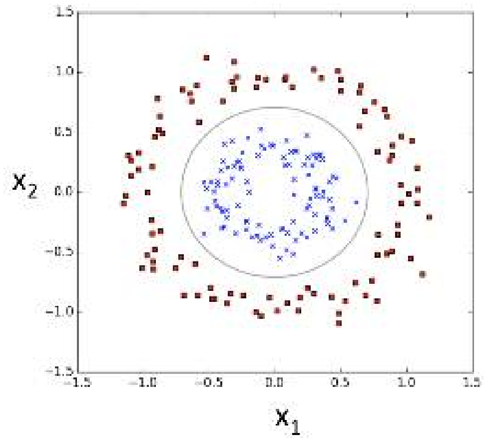



$x_1 \quad x_2$

$\downarrow \phi^{-1}$

$$\phi(x_1, x_2) = (z_1, z_2, z_3) = (x_1, x_2, x_1^2 + x_2^2)$$

nonlinear function of
the features $x_1$ and $x_2$



# Kernel SVM using kernel tricks to find separating hyperplanes in a higher-dimensional space

- to save the expensive step of calculating the dot product $\phi(\boldsymbol{x}^{(i)})^T \phi(\boldsymbol{x}^{(j)})$ between two points explicitly, kernel function is defined.

$$\kappa(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) \triangleq \phi(\boldsymbol{x}^{(i)})^T \phi(\boldsymbol{x}^{(j)})$$

- radial basis function (RBF) kernel
  - widely used, a.k.a Gaussian kernel
  - often

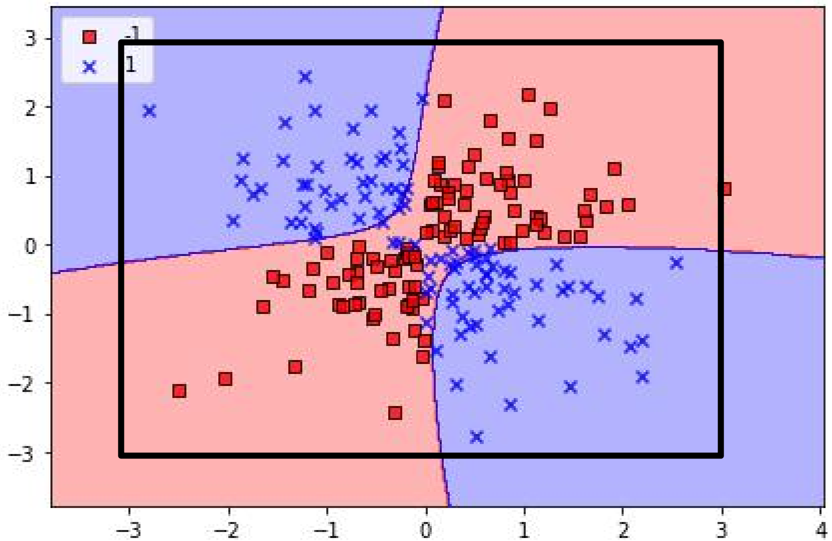
$$\kappa(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \exp\left(-\gamma \left\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\right\|^2\right)$$

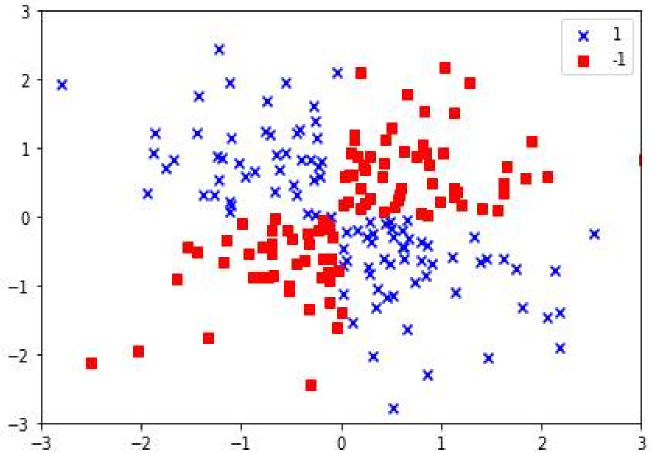
    - $\gamma$ a free parameter to be optimized

# Kernel SVM using kernel tricks to find separating hyperplanes in a higher-dimensional space

```
svm = SVC(kernel='rbf', random_state=1, gamma=0.10, C=10.0)
svm.fit(X_xor, y_xor)
plot_decision_regions(X_xor, y_xor,
                      classifier=svm)

plt.legend(loc='upper left')
plt.tight_layout()
#plt.savefig('images/03_14.png', dpi=300)
plt.show()
```

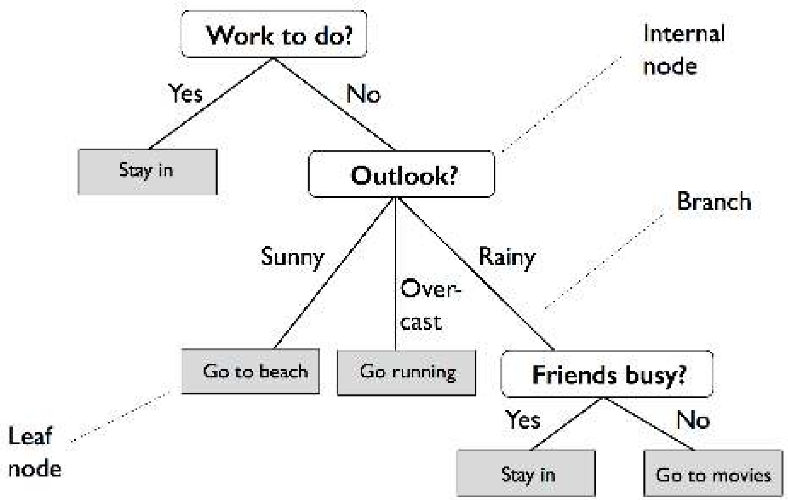



XOR data

# Decision tree learning

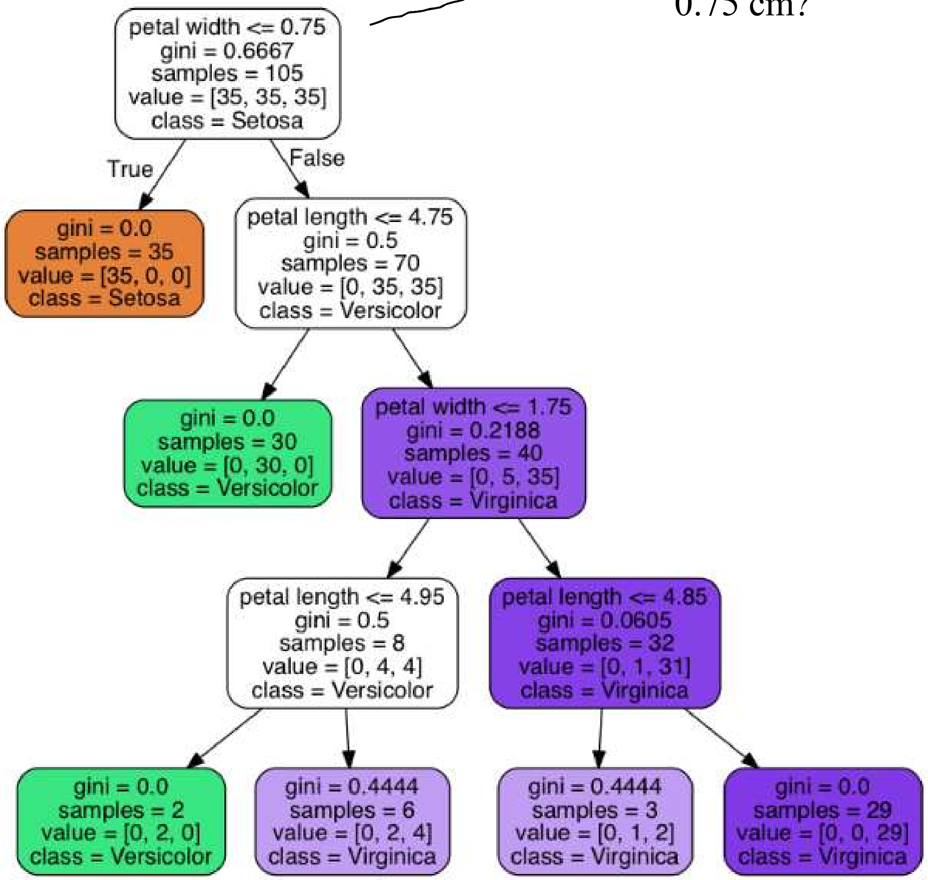
- Decision tree classifiers are attractive model if we are about interpretability




- Question: which feature to split upon?
  - degree to which a subset of examples contains only a single class is known as purity.

p. 132, machine learning with R, 3rd ed.

# Decision tree learning

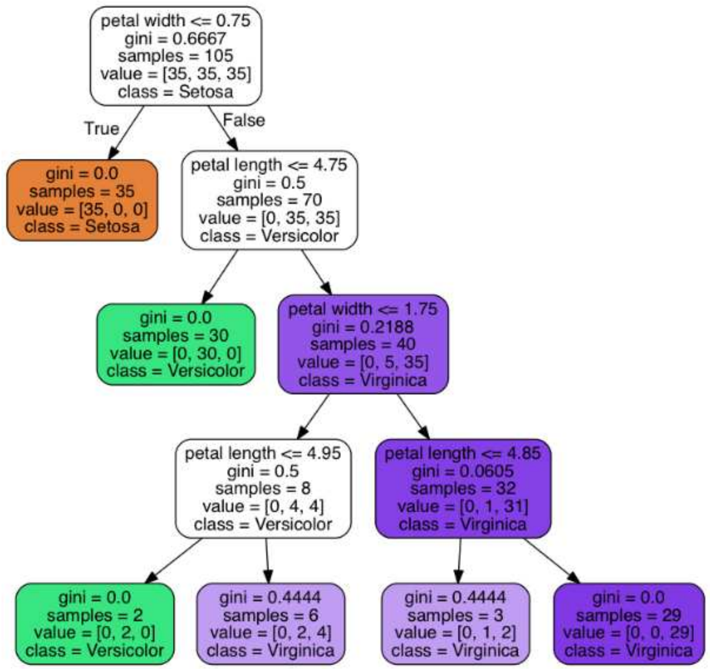



binary question: petal width smaller than or equal to 0.75 cm?

- splits data on the feature that results in the largest information gain (IG)
- this can result in a very deep tree with many nodes
  - this can lead to overfitting
  - typically want to prune the tree by setting a limit for the maximal depth of the tree.

# maximizing information gain

- objective: to split the nodes at the most informative features
- objective function
  - maximizes the IG at each split



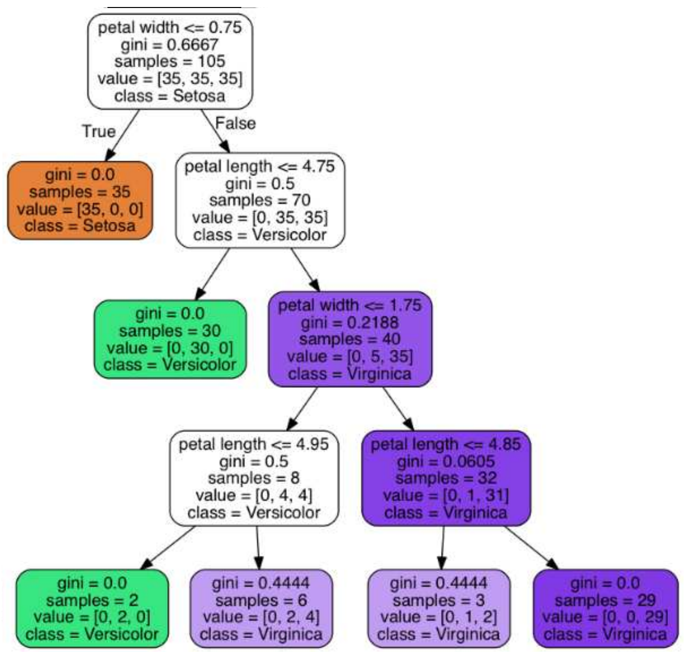

- f: feature to perform the split
- $D_p$: data set of parent
- $D_j$: data set of j-th child node
- I( ): impurity measure
- $N_p$: total number of training examples at parent node
- $N_j$: number of examples in j-th child node

# maximizing information gain

- The lower the impurities of the child nodes, the larger the information gain

$$IG(D_p, f) = I(D_p) - \sum_{j=1}^{m} \frac{N_j}{N_p} I(D_j)$$

- for simplicity and to reduce combinatorial search space, most libraries implement *binary* decision trees.
  - each parent node is split into two child nodes, $D_{left}$ and $D_{right}$




- common impurity measures
  - Gini impurity, entropy, classification error

# Impurity measures

- p(i|t) ~ proportion of the examples that belong to the class i for a particular node t.
- entropy for all <u>non-empty</u> classes (c: number of classes)

  - entropy is *zero* if all examples at a node belong to <u>the *same* class</u>
  - entropy is maximal if we have a uniform class distribution
  - entropy criterion attempts to maximize the mutual information in the tree (minimize entropy)

# Impurity measures

- Gini impurity : criterion to minimize the <u>probability of misclassification</u>

  - similar to entropy, Gini impurity is maximal if the classes are perfectly mixed.

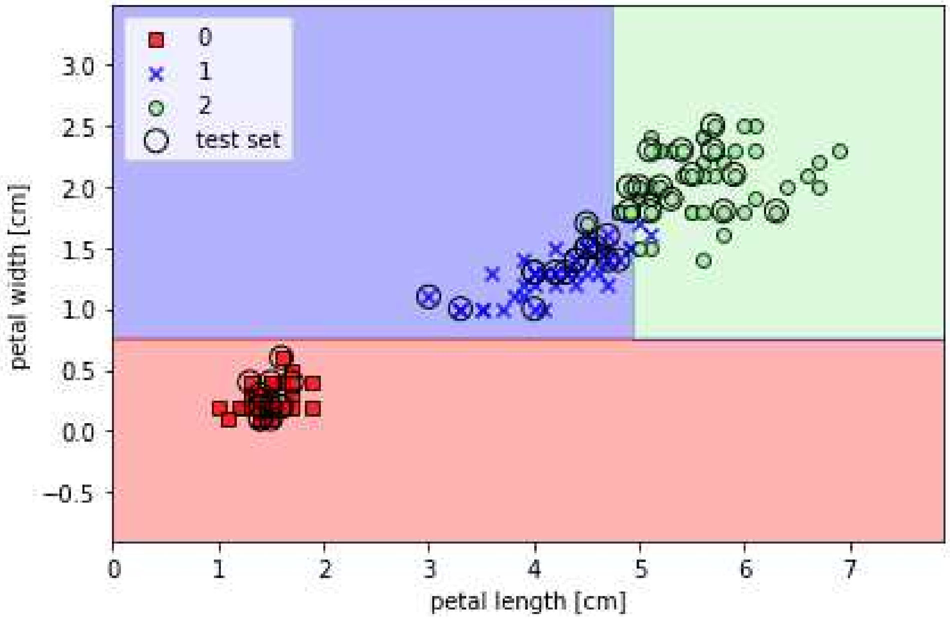
in a binary class setting ($c = 2$):

$$I_G(t) = 1 - \sum_{i=1}^{c} 0.5^2 = 0.5$$

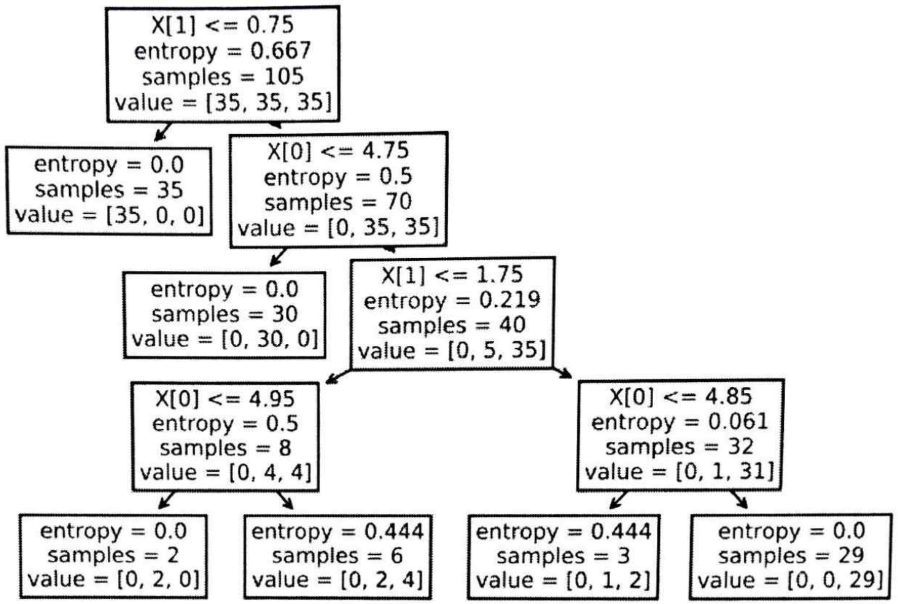
- classification error : useful criterion for pruning but not recommended for growing a decision tree, (less sensitive to changes in the class probabilities of the nodes)

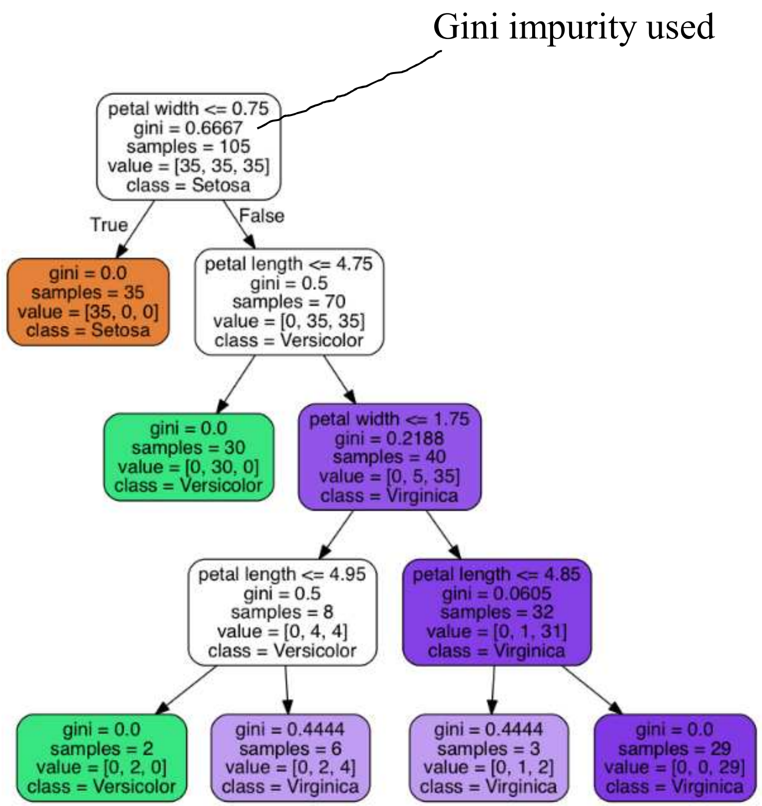
$$I_E(t) = 1 - \max\{ p(i|t)\}$$

# decision boundaries of decision tree model



# Visualization of the decision tree model

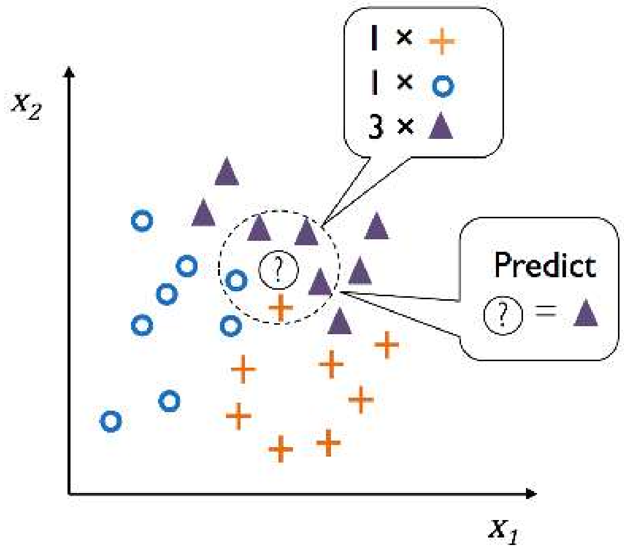


# K-nearest neighbors classifier

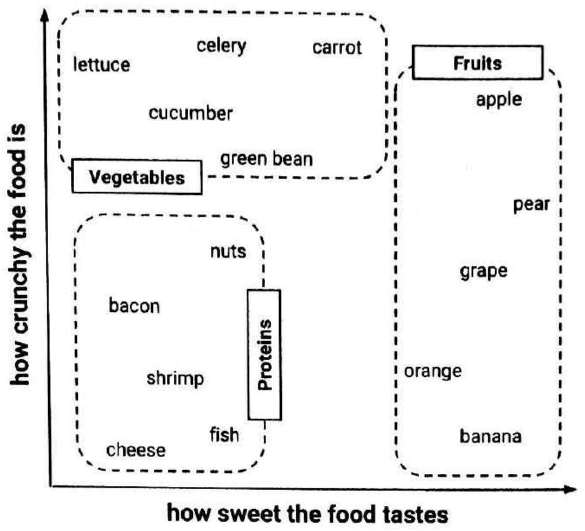
- KNN
- a typical example of a lazy learner
- decision steps
  - choose the number k and a distance metric
  - find the k-nearest neighbors of the training data
  - assign the class label by majority voting

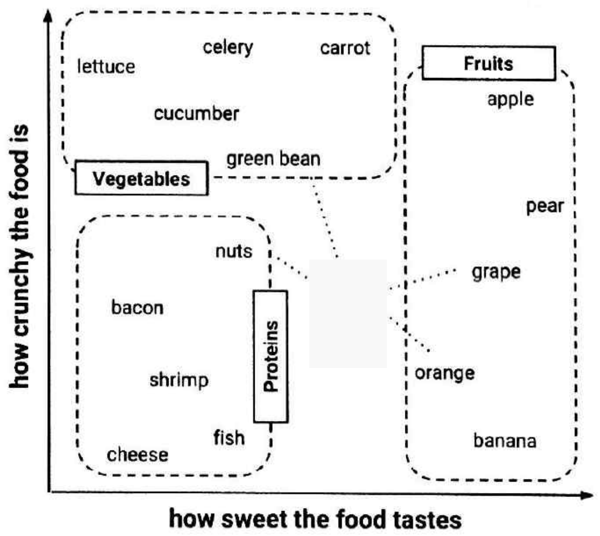
- following figure illustrates how a new data point(?) is assigned the triangle class label based on majority voting among its five nearest negibors.



# K-nearest neighbors classifier

| Ingredient | Sweetness | Crunchiness | Food type |
|---|---|---|---|
| Apple | 10 | 9 | Fruit |
| Bacon | 1 | 4 | Protein |
| Banana | 10 | 1 | Fruit |
| Carrot | 7 | 10 | Vegetable |
| Celery | 3 | 10 | Vegetable |
| Cheese | 1 | 1 | Protein |



# KNN model in scikit-learning

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=
                           p=
                           metric=
knn.fit(X_train_std, y_train)

plot_decision_regions(X_combined_std, y_combined,
                      classifier=knn, test_idx=range(105, 150))

plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
#plt.savefig('images/03_24.png', dpi=300)
plt.show()
```