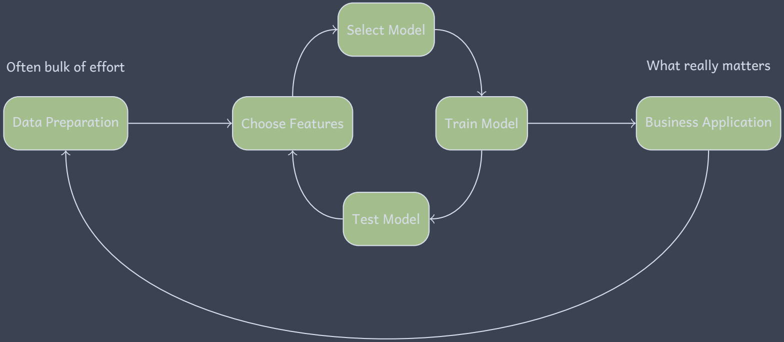
» Machine Learning Workflow



» Models

- Model: $\hat{y} = h_\theta(x)$
  - $\hat{y}$ is prediction
  - x are input features
  - $h_\theta(x)$ is the model $\rightarrow$ a model; is a *function* mapping from input features x to a prediction.
    - Input features x is a vector of real numbers
    - Prediction $h_\theta(x)$ is a real number (regression) or an integer (classification)
  - θ are model parameters
- So far we've looked at linear models
  - $\hat{y} = \theta^T x$ (regression)
  - $\hat{y} = sign(\theta^T x)$ (classification)
- ... but other sorts of model are possible.

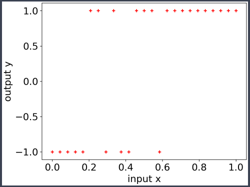
» Linear and Logistic Regression Summary

When first looking at a machine learning task linear and logistic regression should generally be your first port of call:

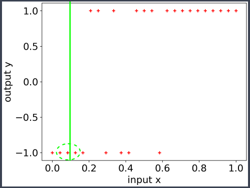
* Use linear models
* Add nonlinearity via feature engineering
* Easy and fast to train $\rightarrow$ cost function is convex in parameters.
* Scale well, can be used with pretty big data.
* Interpretable, sort of:
    * Magnitude of parameter θ_j tells how important the j'th input feature is (if θ_j v small maybe can delete j'th feature)
    * Sign of θ_j tells whether prediction tends to increase/decrease with j'th feature.
    * ... this assumes j'th feature itself has a reasonable interpretation
* SVMs and logistic regression generally perform much the same

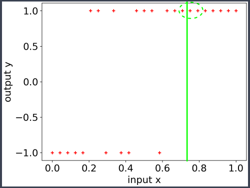
» *k*-Nearest Neighbour (*k*NN) Model

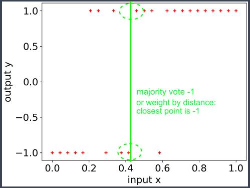
We can directly use the training data to make predictions:



training data







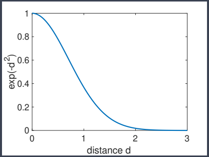
» k-Nearest Neighbour (kNN) Model

- Training data $(x^{(i)}, y^{(i)})$, $i = 1, 2, \ldots, m$
- Given feature vector x:
  1. For each training data point i calculate the distance $d(x^{(i)}, x)$ between feature vector $x^{(i)}$ and x
  2. Select the k training data points that are closest to x i.e. for which $d(x^{(i)}, x)$ is smallest
     - ... the k nearest neighbours
  3. Predict output y using the outputs $y^{(i)}$ for these k closest training points.
     - In a classification problem e.g. take majority vote (if $k = 3$ and two closest training points have label $+1$ and other has label -1 then predict $+1$).
     - In a regression problem e.g. calculate the average of the $y^{(i)}$ for the k closest training points and use that as prediction
- kNN makes predictions based directly on the training data $\rightarrow$ an example of an *instance-based* model

» *k*-Nearest Neighbour (*k*NN) Model

A *k*NN model needs four things to be specified:

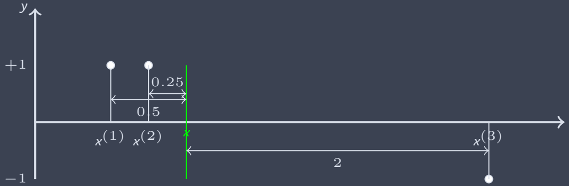
1. A distance metric. Typically Euclidean: $d(x^{(i)}, x) = \sqrt{\sum_{j=1}^{n} (x_j^{(i)} - x_j)^2}$
2. Number k of neighbours to use. E.g. $k = 5$ (select this using cross-validation)
3. Weighting of neighbour points. E.g. *uniform* $w^{(i)} = 1$ or *Gaussian* $w^{(i)} = e^{-\gamma d(x^{(i)}, x)^2}$ (attach less weight to training points that are further away from query point x).



4. Method for aggregating the k neighbour points N_k
   * Classification: $sign(\frac{\sum_{i \in N_k} w^{(i)} y^{(i)}}{\sum_{i \in N_k} w^{(i)}})$ (majority vote when $w^{(i)} = 1$)
   * Regression: weighted mean $\hat{y} = \frac{\sum_{i \in N_k} w^{(i)} y^{(i)}}{\sum_{i \in N_k} w^{(i)}}$

» Recall kNN Model

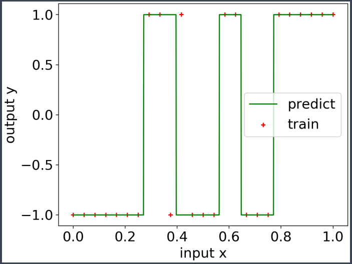
Impact of weighting:



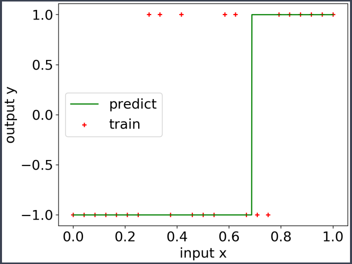
- Suppose input is $x = 1$ (just a scalar). Three nearest neighours are $x^{(1)} = 0.5$, $x^{(2)} = 0.75$, $x^{(1)} = 3$
- Distances are $d(x^{(1)}, x) = \sqrt{(0.5-1)^2} = 0.5$, $d(x^{(2)}, x) = \sqrt{(0.75-1)^2} = 0.25$, $d(x^{(3)}, x) = \sqrt{(3-1)^2} = 2$
- Uniform weights
  - $w^{(1)} = 1$, $w^{(2)} = 1$, $w^{(3)} = 1$
  - $\frac{\sum_{i \in N_k} w^{(i)} y^{(i)}}{\sum_{i \in N_k} w^{(i)}} = \frac{y^{(1)}+y^{(2)}+y^{(3)}}{3} = \frac{1+1-1}{3} = 0.66$. Predict $+1$
- Gaussian weights. Suppose $\gamma = 1$.
  - $w^{(1)} = e^{-0.5^2} = 0.78$, $w^{(2)} = e^{-0.25^2} = 0.94$, $w^{(1)} = e^{-2^2} = 0.02 \rightarrow$ higher weight on training points close to x.
  - $\frac{\sum_{i \in N_k} w^{(i)} y^{(i)}}{\sum_{i \in N_k} w^{(i)}} = \frac{0.78\times1+0.94\times1+0.02\times(-1)}{1.74} = 0.98$. Predict $+1$

» k-Nearest Neighbour (kNN) Model

Classification example:



$k = 3$

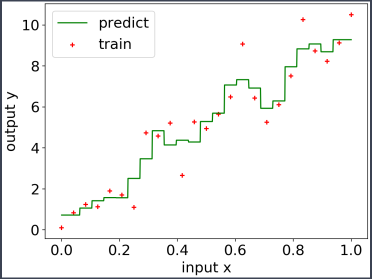


$k = 7$

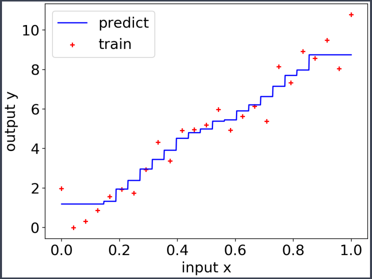
- 1) Euclidean distance, 2) (i) $k = 3$ and (ii) $k = 7$, 3) uniform weights, 4) majority vote
- Note: even though it's based on the training data, the kNN model is still just a function from input features x to prediction $+1$ or -1
- Increasing k will tend to smooth out the function, decreasing k to make it more complex
    - Increasing k tends to cause under-fitting, decreasing k to cause over-fitting. Choose k by cross-validation.

» k-Nearest Neighbour (kNN) Model

Regression example:



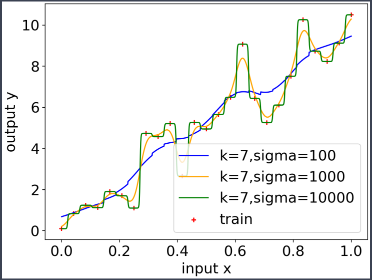
$k = 3$



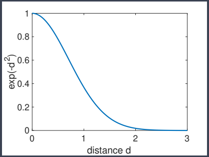
$k = 7$

- 1) Euclidean distance, 2) (i) $k = 3$ and (ii) $k = 7$, 3) uniform weights, 4) weighted average
- Increasing k will tend to smooth out the function, decreasing k to make it track the training data points more closely (i.e. fit the "noise").

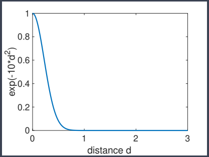
» k-Nearest Neighbour (kNN) Model



- 1) Euclidean distance, 2) $k = 7$, 3) Gaussian weights $e^{-\gamma d(x^{(i)},x)^2}$, 4) weighted average
- Decreasing σ tends to smooth out the function, increasing γ to make it rougher. Choose γ and k using cross-validation.



$\gamma = 1$



$\gamma = 10$

» *k*-Nearest Neighbour (*k*NN) Classifier Code

```
import numpy as np
m = 25
Xtrain = np.linspace(0.0,1.0,num=m)
ytrain = np.sign(Xtrain−0.5+np.random.normal(0,0.2,m))
Xtrain = Xtrain.reshape(−1, 1)
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=3,weights='uniform').fit(Xtrain, ytrain)

Xtest=np.linspace(0.0,1.0,num=1000).reshape(−1, 1)
ypred = model.predict(Xtest)
import matplotlib.pyplot as plt
plt.rc('font', size=18); plt.rcParams['figure.constrained_layout.use'] = True
plt.scatter(Xtrain, ytrain, color='red', marker='+')
plt.plot(Xtest, ypred, color='green')
plt.xlabel("input x"); plt.ylabel("output y")
plt.legend(["predict","train"])
plt.show()

model = KNeighborsClassifier(n_neighbors=7,weights='uniform').fit(Xtrain, ytrain)
ypred = model.predict(Xtest)
plt.scatter(Xtrain, ytrain, color='red', marker='+')
plt.plot(Xtest, ypred, color='green')
plt.xlabel("input x"); plt.ylabel("output y")
plt.legend(["predict","train"])
plt.show()
```

» *k*-Nearest Neighbour (*k*NN) Regression Code

```
import numpy as np
m = 25
Xtrain = np.linspace(0.0,1.0,num=m)
ytrain = 10*Xtrain + np.random.normal(0.0,1.0,m)
Xtrain = Xtrain.reshape(−1, 1)
from sklearn.neighbors import KNeighborsRegressor
model = KNeighborsRegressor(n_neighbors=3,weights='uniform').fit(Xtrain, ytrain)

Xtest=np.linspace(0.0,1.0,num=1000).reshape(−1, 1)
ypred = model.predict(Xtest)
import matplotlib.pyplot as plt
plt.rc('font', size=18); plt.rcParams['figure.constrained_layout.use'] = True
plt.scatter(Xtrain, ytrain, color='red', marker='+')
plt.plot(Xtest, ypred, color='green')
plt.xlabel("input x"); plt.ylabel("output y"); plt.legend(["predict","train"])
plt.show()

model2 = KNeighborsRegressor(n_neighbors=7,weights='uniform').fit(Xtrain, ytrain)
ypred2 = model2.predict(Xtest)
plt.scatter(Xtrain, ytrain, color='red', marker='+')
plt.plot(Xtest, ypred2, color='blue')
plt.xlabel("input x"); plt.ylabel("output y"); plt.legend(["predict","train"])
plt.show()
```

» *k*-Nearest Neighbour (*k*NN) Regression Code (cont)

```
def gaussian_kernel100(distances):
        weights = np.exp(−100∗(distances∗∗2))
        return weights/np.sum(weights)

def gaussian_kernel1000(distances):
        weights = np.exp(−1000∗(distances∗∗2))
        return weights/np.sum(weights)

def gaussian_kernel10000(distances):
        weights = np.exp(−10000∗(distances∗∗2))
        return weights/np.sum(weights)

model2 = KNeighborsRegressor(n_neighbors=7,weights=gaussian_kernel100).fit(Xtrain, ytrain)
ypred2 = model2.predict(Xtest)
model3 = KNeighborsRegressor(n_neighbors=7,weights=gaussian_kernel1000).fit(Xtrain, ytrain)
ypred3 = model3.predict(Xtest)
model4 = KNeighborsRegressor(n_neighbors=7,weights=gaussian_kernel10000).fit(Xtrain, ytrain)
ypred4 = model4.predict(Xtest)
plt.scatter(Xtrain, ytrain, color='red', marker='+')
plt.plot(Xtest, ypred2, color='blue')
plt.plot(Xtest, ypred3, color='orange')
plt.plot(Xtest, ypred4, color='green')
plt.xlabel("input x"); plt.ylabel("output y")
plt.legend(["k=7,sigma=100","k=7,sigma=1000","k=7,sigma=10000","train"])
plt.show()
```

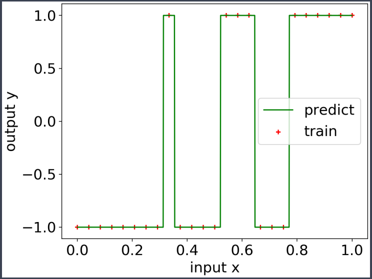
» *k*NN Summary

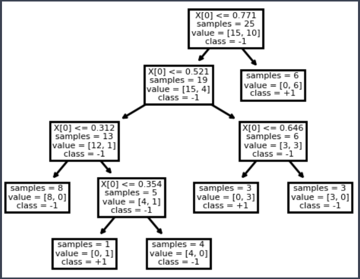
- Easy to use $\rightarrow$ only parameter is k.
- Need to choose distance function and any weighting with distance, but standard choices (Euclidean distance, uniform or Gaussian weighting) often work well
- Small data only $\rightarrow$ each prediction requires a search over training data to find k nearest neighbours, this becomes expensive/slow when there is a lot of training data

Decision Tree Classfiers

- Model uses if-then rules e.g. if X>0.771 then predict class +1
- Constructs tree of rules, leaves of tree are the $+1$ or -1 predictions.

Example:

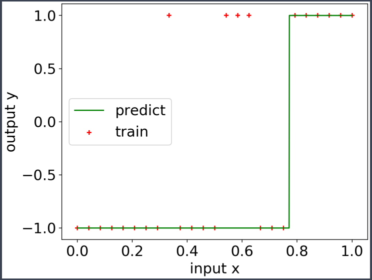


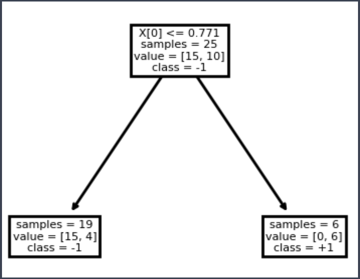


- Note: even though it's based on if-then rules model is still just a function from input features *x* to prediction $+1$ or -1

» Decision Tree Classfiers

- Control complexity of model by limiting tree depth e.g. if depth restricted to be 1 then will get a single transition between $+1$ and -1
- $\rightarrow$ choose tree depth using cross-validation





- Easy to understand when tree is small, but quickly becomes hard as tree gets large (as it usually does).
- Learning tree rules is NP-hard in general, special-purpose algorithms are used (not gradient decent)
- Decision trees often used as an ensemble (a "forest") since hard to control complexity using just tree depth.

» Decision Tree Code

```
import numpy as np
m = 25
Xtrain = np.linspace(0.0,1.0,num=m)
ytrain = np.sign(Xtrain−0.5+np.random.normal(0,0.2,m))
Xtrain = Xtrain.reshape(−1, 1)
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier().fit(Xtrain, ytrain)

Xtest=np.linspace(0.0,1.0,num=1000).reshape(−1, 1)
ypred = model.predict(Xtest)

import matplotlib.pyplot as plt
from sklearn.tree import export_text
print(export_text(model))
from sklearn.tree import plot_tree
plot_tree(model, fontsize=4, impurity=False, class_names=['−1','+1'])
plt.show()

plt.rc('font', size=18); plt.rcParams['figure.constrained_layout.use'] = True
plt.scatter(Xtrain, ytrain, color='red', marker='+')
plt.plot(Xtest, ypred, color='green')
plt.xlabel("input x"); plt.ylabel("output y")
plt.legend(["predict","train"])
plt.show()

model = DecisionTreeClassifier(max_depth=1).fit(Xtrain, ytrain)
ypred = model.predict(Xtest)

plot_tree(model, fontsize=4, impurity=False, class_names=['−1','+1'])
plt.show()

plt.scatter(Xtrain, ytrain, color='red', marker='+')
plt.plot(Xtest, ypred, color='green')
plt.xlabel("input x"); plt.ylabel("output y")
plt.legend(["predict","train"])
plt.show()
```