

Data Science and Advanced Programming — Lecture 8

Supervised Machine Learning II (Classification)

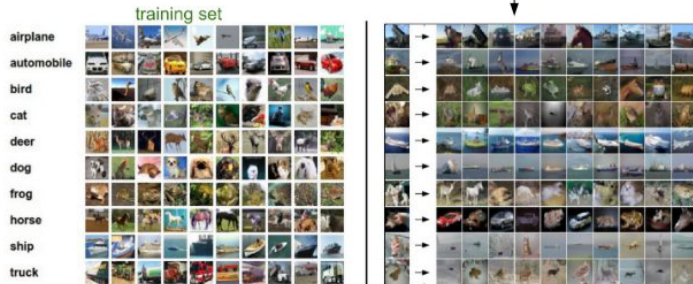
Simon Scheidegger
Department of Economics, University of Lausanne, Switzerland

November 3rd, 2025 | 12:30 - 16:00 | Internef 263

Today's Roadmap

1. Recall: Supervised Machine Learning - Classification
2. k-Nearest-Neighbors
3. Evaluating Classifiers
4. Naïve Bayes
5. Decision Trees
6. Combining Models (Boosting etc.; take-home materials)

Classification — Motivation

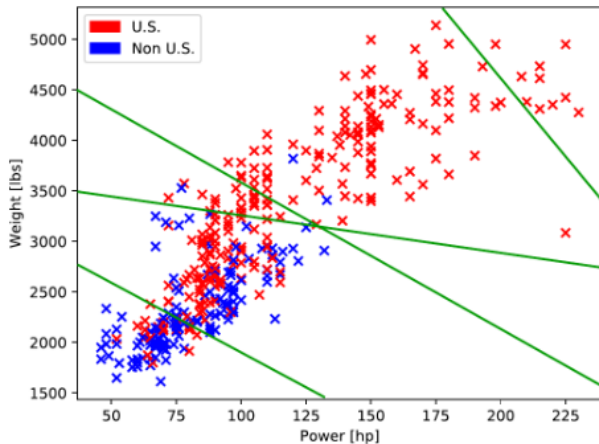


Recall: “our example”

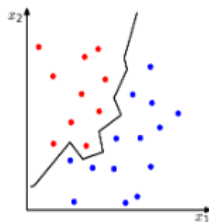
- ▶ **Classification** aims at **predicting a nominal target feature** based on one or **multiple other (numerical)** features.
- ▶ Example: **Predict the origin** (U.S. vs. Non-U.S.) of a car based on its power (in horsepower) and weight (in pounds).
- ▶ Since our **predictor $f(x)$ takes values in a discrete set G** , we can always **divide the input space** into a collection of regions labeled according to the classification.

Which country?

Data source: <https://archive.ics.uci.edu/ml/datasets/Auto+MPG>



Classification Boundaries



- ▶ Input vector $x \in \mathbb{R}^D$, assign it to one of K discrete classes $C_k, k = 1, \dots, K$.
- ▶ Assumption: classes are disjoint, i.e., input vectors are assigned to exactly one class.
- ▶ Idea: Divide input space into decision regions whose boundaries are called decision boundaries/surfaces.

Decision Theory

- ▶ We want a framework that allows us to make optimal decisions (in situations involving uncertainty).
- ▶ Consider, for example, a medical diagnosis problem in which we have taken an X-ray image of a patient, and we wish to determine whether the patient has cancer or not.
- ▶ In this case, the input vector \mathbf{x} is the set of pixel intensities in the image, and output variable \mathbf{t} will represent the presence of cancer, which we denote by the class C_1 , or the absence of cancer, which we denote by the class C_2 .
- ▶ We might, for instance, choose t to be a binary variable such that $\mathbf{t} = \mathbf{0}$ corresponds to class C_1 and $\mathbf{t} = \mathbf{1}$ corresponds to class C_2 .

Decision Theory — more abstract

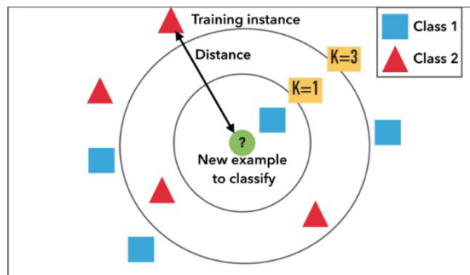
- ▶ **For a sample x , decide which class (C_k) it is from.**
- ▶ Ideas to do so:
 1. Maximum Likelihood
 2. Minimum Loss/Cost (e.g. misclassification rate)
 3. Maximum A posteriori (MAP)

2. k-Nearest-Neighbors (kNN)

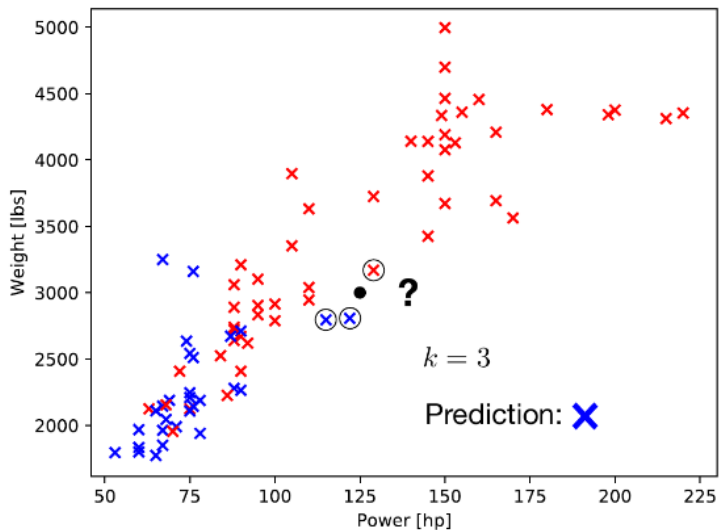
- ▶ **k-Nearest-Neighbors (kNN)** is another simple-yet-popular classification method that often serves as a baseline.
- ▶ kNN is a so-called **lazy learning method**, which reflects that **it does not actually learn the parameters of a model**, but always looks at the training data.
- ▶ **no cost for training a model**, i.e., learning parameters.
- ▶ cost at runtime (i.e., when classifying a data point).
- ▶ depends on the amount of training data available.

k-Nearest-Neighbors (kNN)

- ▶ To classify a previously unseen data point, kNN identifies the k closest data points in the training data according to a suitable distance measure.
- ▶ predicts the nominal target feature (class) as the most frequent value among the k closest data points.



Example



Minkowski Distance

For your personal entertainment: https://de.wikipedia.org/wiki/Hermann_Minkowski

- ▶ Minkowski Distance as a suitable distance measure

$$d(x, x') = \left(\sum_{i=1}^m |x_i - x'_i|^p \right)^{1/p}$$

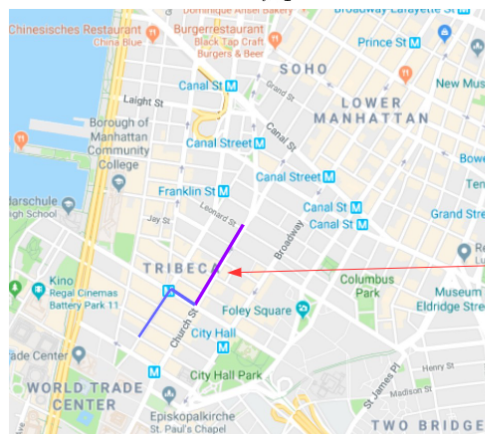
with p as a constant.

- ▶ Minkowski Distance is a metric, i.e., it is
 - ▶ non-negative.
 - ▶ symmetric $\forall x, x' : d(x, x') \geq 0$.
 - ▶ subadditive $\forall x, x', x'' : d(x, x'') \leq d(x, x') + d(x', x'')$ (triangle inequality).

Manhattan Metric

Manhattan Distance as Minkowski Distance with $p = 1$

$$d(x, x') = \sum_{i=1}^m |x_i - x'_i|$$

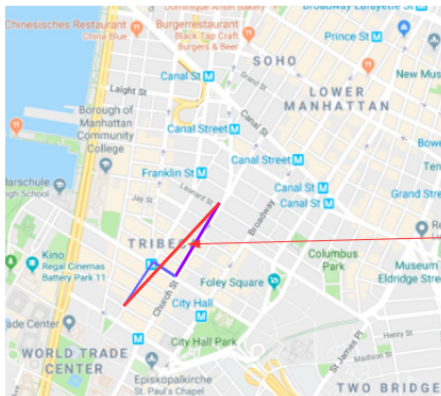


Distance:
9 Blocks!!

Euclidean Metric

Euclidean Distance as Minkowski Distance with $p = 2$

$$d(x, x') = \sqrt{\sum_{i=1}^m (x_i - x'_i)^2}$$



Distance:

$$\text{Sqrt}(1 + 8) = 3$$

Normalization and Standardization

- ▶ When computing Minkowski Distances, the magnitudes of features matter, e.g.:
 - ▶ Car 1 with 100hp weighting 2000lbs.
 - ▶ Car 2 with 100hp weighting 2200lbs.
 - ▶ Car 3 with 300hp weighting 2000lbs.

Car 1 has the same distance from Car 2 as from Car 3.

- ▶ To avoid features with larger magnitude (i.e., generally larger values) dominating the distances computed, **it makes sense to normalize or standardize features upfront.**

Normalization

- ▶ **Min-Max Normalization** maps feature values onto $[0, 1]$.
- ▶ let z_1, \dots, z_n be the feature values observed in the data.
- ▶ the transformed feature value is then obtained as

$$z'_i = \frac{z_i - \min_j(z_j)}{\max_j(z_j) - \min_j(z_j)}$$

- ▶ Minimum is mapped to 0; maximum is mapped to 1.
- ▶ Min-Max Normalization is sensitive to outliers in the data.

Standardization

- ▶ Standardization transforms feature values, so that they reflect by how many standard deviations a value deviates from the mean observed in the data.
- ▶ let z_1, \dots, z_n be the feature values observed in the data.
- ▶ the transformed feature value is then obtained as

$$z'_i = \frac{z_i - \mu}{\sigma}$$

with

$$\mu = \frac{1}{n} \sum_{i=1}^n z_i \quad \sigma = \sqrt{\frac{1}{n} \sum_{i=1}^n (z_i - \mu)^2}$$

Predicting Origin from Power and Weight

<https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler

# load data
cars = pd.read_csv('auto-mpg.data.txt', header=None, sep='\s+')

# extract power and weight as data matrix X
X = cars.iloc[:, [3,4]].values

# extract origin (0:Non-U.S. / 1:U.S.) as target vector y
y = cars.iloc[:, 7].values

# split into training data (80%) and test data (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2 random_state=0)

# use kNN with k = 3
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_predicted = knn.predict(X_test)

# compute accuracy
print(accuracy_score(y_true=y_test, y_pred=y_predicted))
```

Predicting Origin from Power and Weight

<https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>

```
# load data
cars = pd.read_csv('auto-mpg.data.txt', header=None, sep='\s+')

# extract power and weight as data matrix X
X = cars.iloc[:, [3,4]].values

# extract origin (0:Non-U.S. / 1:U.S.) as target vector y
y = cars.iloc[:, 7].values

# split into training data (80%) and test data (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2 random_state=0)

# normalize data
min_max_scaler = MinMaxScaler()
min_max_scaler.fit(X_train) # determine min and max
X_train_normalized = min_max_scaler.transform(X_train)
X_test_normalized = min_max_scaler.transform(X_test)

# use kNN with k = 3
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_predicted = knn.predict(X_test)

# compute accuracy
print(accuracy_score(y_true=y_test, y_pred=y_predicted))
```

Predicting Origin from Power and Weight

<https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>;

See demo/[auto_class_knn.py](#)

```
# load data
cars = pd.read_csv('auto-mpg.data.txt', header=None, sep='\s+')

# extract power and weight as data matrix X
X = cars.iloc[:, [3,4]].values

# extract origin (0:Non-U.S. / 1:U.S.) as target vector y
y = cars.iloc[:, 7].values

# split into training data (80%) and test data (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2 random_state=0)
# normalize data
scaler = StandardScaler()
scaler.fit(X_train) # determine mean and standard deviation
X_train_normalized = scaler.transform(X_train)
X_test_normalized = scaler.transform(X_test)

# use kNN with k = 3
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_predicted = knn.predict(X_test)

# compute accuracy
print(accuracy_score(y_true=y_test, y_pred=y_predicted))
```

Summary: kNN

k-Nearest-Neighbors as a **lazy classification method** that classifies a previously unseen data point based on the classes of the **k closest data points** from the training data.

3. Evaluating Classifiers

- ▶ **How can we assess the prediction quality of a classifier?**
- ▶ Initially, we'll consider the case of binary classification (and extend it later to multi-class classification).
- ▶ **Confusion matrix** shows the performance of a classifier.

		Predicted	
		0(No)	1 (Yes)
Actual	0(No)	True Negatives (TN)	False Positives (FP)
	1 (Yes)	False Negatives (FN)	True Positives (TP)

Accuracy and Error Rate

- **Accuracy** measures the classifier's ability to put data points into the right class.

$$\text{Accuracy} = \frac{\text{TN} + \text{TP}}{\text{TN} + \text{FP} + \text{FN} + \text{TP}}$$

- **Error rate**, as the counterpart to accuracy, reflects to what extent the classifier puts data points into the wrong class.

$$ER = (1 - \text{Accuracy}) = \frac{\text{FN} + \text{FP}}{\text{TN} + \text{FP} + \text{FN} + \text{TP}}$$

False-Positive Rate and True Positive Rate

- ▶ False-positive rate is the fraction of negative (0 / No) data points that is falsely classified as positive (1 / Yes)

$$FPR = \frac{FP}{TN + FP}$$

- ▶ True-positive rate is the fraction of positive (1 / Yes) points that is correctly classified as positive (1 / Yes)

$$TPR = \frac{TP}{FN + TP}$$

Precision and Recall

- Precision reflects the classifier's ability to correctly detect positive (1 / Yes) data points

$$\text{Precision} = \frac{TP}{FP + TP}$$

- Recall reflects the classifier's ability to detect all positive (1 / Yes) data points

$$\text{Recall} = \frac{TP}{FN + TP}$$

F1-measure

- ▶ Precision and Recall are widely used in Information Retrieval
- ▶ F1-Measure as the harmonic mean of precision and recall combines both measures in a single measure

$$F1 = 2 \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Accuracy and Error Rate

Confusion matrix for our binary car classifier (Non-U.S. = 0 vs. U.S. = 1) using logistic regression on power and weight

		Predicted	
		0 (No)	1 (Yes)
Actual	0 (No)	20(TN)	9(FP)
	1 (Yes)	6(FN)	44(TP)

$$\text{Accuracy} = \frac{20 + 44}{20 + 9 + 6 + 44} = \frac{64}{79}$$

$$ER = \frac{9 + 6}{20 + 9 + 6 + 44} = \frac{15}{79}$$

False-Positive Rate and True Positive Rate

Confusion matrix for our binary car classifier (Non-U.S. = 0 vs. U.S. =1) using logistic regression on power and weight.

		Predicted	
		0 (No)	1 (Yes)
Actual	0 (No)	20(TN)	9(FP)
	1 (Yes)	6(FN)	44(TP)

$$FPR = \frac{9}{20 + 9} = \frac{9}{29}$$

$$TPR = \frac{44}{6 + 44} = \frac{44}{50}$$

Precision and Recall

Confusion matrix for our binary car classifier (Non-U.S. = 0 vs. U.S. =1) using logistic regression on power and weight.

		Predicted	
		0 (No)	1 (Yes)
Actual	0 (No)	20(TN)	9(FP)
	1 (Yes)	6(FN)	44(TP)

$$\text{Precision} = \frac{44}{9 + 44} = \frac{44}{53}$$

$$\text{Recall} = \frac{44}{6 + 44} = \frac{44}{50}$$

Computing Quality Measures

```
# generate confusion matrix
conf = confusion_matrix(y_true=y_test, y_pred=y_predicted)
print("confusion matrix ", conf)

# compute accuracy
acc_score = accuracy_score(y_true=y_test, y_pred=y_predicted) # 0.7468354430379747
print("accuracy ", acc_score)

# compute precision
prec_score = precision_score(y_true=y_test, y_pred=y_predicted) # 0.77966101610491526
print("precision ", prec_score)

# compute recall
recall = recall_score(y_true=y_test, y_pred=y_predicted) # 0.8679
print("recall ", recall)

# compute f1
f1 = f1_score(y_true=y_test, y_pred=y_predicted) # 0.8214285714285715
print("f1 ", f1)
```

Micro- and Macro- Averages

- ▶ When dealing with more than two classes, the confusion matrix has one column and one row per class

		Predicted		
		0	1	2
Actual	0	49	0	1
	1	10	0	5
	2	8	0	6

- ▶ For each class i , we can now determine the numbers TN_i , FP_i , FN_i , TP_i , assuming that it is the positive class, and all others are treated as negative.

Micro- and Macro- Averages (II)

Micro-averages plug per-class numbers into the definitions

$$\text{Precision}_{\text{micro}} = \frac{\text{TP}_0 + \dots + \text{TP}_{k-1}}{\text{FP}_0 + \dots + \text{FP}_{k-1} + \text{TP}_0 + \dots + \text{TP}_{k-1}}$$

$$\text{Recall}_{\text{micro}} = \frac{\text{TP}_0 + \dots + \text{TP}_{k-1}}{\text{FN}_0 + \dots + \text{FN}_{k-1} + \text{TP}_0 + \dots + \text{TP}_{k-1}}$$

Macro-averages average per-class quality assessments

$$\text{Precision}_{\text{macro}} = \frac{1}{k} \sum_{i=0}^{k-1} \frac{\text{TP}_i}{\text{FP}_i + \text{TP}_i}$$

$$\text{Recall}_{\text{macro}} = \frac{1}{k} \sum_{i=0}^{k-1} \frac{\text{TP}_i}{\text{FN}_i + \text{TP}_i}$$

4. Naïve Bayes

- ▶ Naïve Bayes is a classification method that is often used for text classification (e.g., e-mails as SPAM or HAM), but variants for arbitrary data exist.
- ▶ Naïve Bayes supports an arbitrary number of classes.
- ▶ The name “Naïve Bayes” refers to the fact that Bayes’ theorem is used and that a (naïve) independence assumption is made about the data.



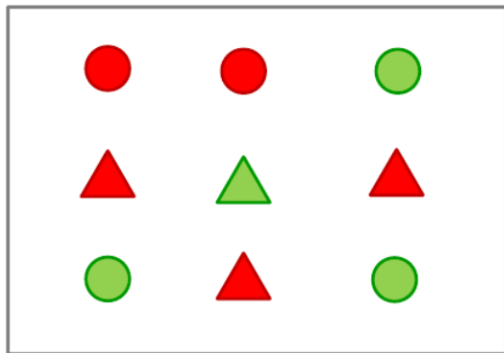
Recall: Events and Probabilities

- ▶ Let's consider two events A and B
 - ▶ **A** is the event that an object is a **circle**.
 - ▶ **B** is the event that an object is **green**

$$P[A] = \frac{5}{9} \quad P[B] = \frac{4}{9}$$

- ▶ We refer to $A \wedge B$ as the joint event that an object is a green circle

$$P[A \wedge B] = P[A, B] = \frac{3}{9}$$



Conditional Probabilities

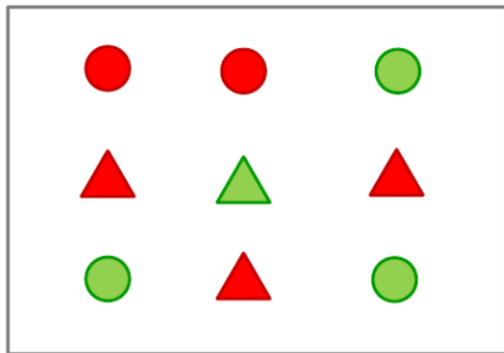
The conditional probability $P[B | A]$ (B given A) is the probability that the event B occurs if we already know that the event A has occurred

$$P[B | A] = \frac{P[A \wedge B]}{P[A]}$$

in our case

$$P[B | A] = \frac{3}{5}$$

$$P[A | B] = \frac{3}{4}$$



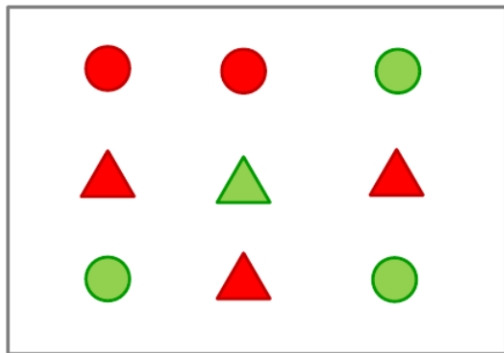
independence

- ▶ two events A and B are called (stochastically) independent, if the following holds for their joint probability

$$P[A \wedge B] = P[A]P[B]$$

- ▶ In our example, the events A and B are not independent

$$\frac{3}{9} \neq \frac{5}{9} \frac{4}{9}$$



Recall again: Bayes' Theorem

- ▶ Thomas Bayes (1701-1761) famously observed the following theorem regarding the conditional probabilities of events.

$$P[A \mid B] = \frac{P[B \mid A]P[A]}{P[B]}$$

- ▶ Bayes' theorem is particularly useful when, for two events A and B , one of the conditional probabilities is easy to estimate, but the other is hard to estimate.

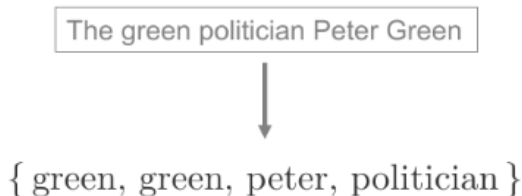
Bayes' Theorem in Action

- ▶ Example: Examining animals in the wild
 - ▶ A is the event that the animal is a fox.
 - ▶ B is the event that the animal has rabies.
 - Assume that we know the following probabilities
 - ▶ $P[A] = 0.1$ (e.g., estimated based on video surveillance)
 - ▶ $P[B] = 0.05$ (e.g., estimated based on hunted animals)
 - ▶ $P[A | B] = 0.25$ (e.g., estimated based on deceased animals)
- ▶ We can now estimate the probability that a fox has rabies

$$P[B | A] = \frac{0.25 \cdot 0.05}{0.1} = 0.125$$

Moving on: Bag-of-Words Model

- ▶ Common preprocessing steps for text documents
 - ▶ convert all letters to lower case
 - ▶ remove stop words (e.g., a, the, or, of)
 - ▶ split documents at white spaces (e.g., _, n, t) and punctuation marks (e.g., ?!.,;)
- ▶ The document is then viewed as a bag of words (i.e., a multi-set preserving frequency information)



Data Matrix

- Documents as bags-of-words with their respective class can be viewed as a data matrix.
- Example: Five documents d_1, \dots, d_5 consisting of words a, b, x, y and belonging to either class Spam or Ham.

	Words				Class
	a	b	x	y	-
d_1	2	1	0	0	H
d_2	0	1	2	2	S
d_3	2	1	1	1	H
d_4	1	0	2	2	S
d_5	1	1	0	0	H

Naïve Bayes for Text Classification

- For a previously unseen document d and any class c , we need to estimate the conditional probability

$$P[c \mid d]$$

that c is the correct class for document d .

- The document is then assigned to the class c , which has the highest probability.

Naïve Bayes for Text Classification

- ▶ Bayes' theorem allows rewriting the conditional probability as

$$P[c \mid d] = \frac{P[d \mid c] \cdot P[c]}{P[d]}$$

- ▶ since $P[d]$ is constant for any document

$$P[c \mid d] \propto P[d \mid c] \cdot P[c]$$

- ▶ with $P[c]$ as the so-called class prior.
- ▶ $P[d \mid c]$ as the probability that document d is from class $c \rightarrow$ How can we estimate these probabilities?

Class Priors in Naïve Bayes

Class priors can be estimated based on the training data as

$$P[c] = \frac{\# \text{ Documents from Class } c}{\# \text{ Documents}}$$

Example:

	Words			Class	-
	a	b	x	y	
d_1	2	1	0	0	H
d_2	0	1	2	2	S
d_3	2	1	1	1	H
d_4	1	0	2	2	S
d_5	1	1	0	0	H

$$P[\mathbf{H}] = 3/5$$

$$P[\mathbf{S}] = 2/5$$

Conditional Probabilities in Naïve Bayes

- ▶ The **conditional probability** that **document** d is from class c is estimated based on the contained words as

$$P[d | c] \propto \prod_{w \in d} P[w | c]^{f(w,d)}$$

- ▶ with **$f(w, d)$** as the **frequency** of **word w** in **document d** and

$$P[w | c] = \frac{\# \text{ Occurences of } w \text{ in Document from Class } c}{\# \text{ Word Occurrences in Documents from Class } c}$$

- ▶ as the probability that we randomly draw the word w from the documents in class c .

Conditional Probabilities in Naïve Bayes

- Intuitively, the conditional probability

$$P[d \mid c] \propto \prod_{w \in d} P[w \mid c]^{f(w,d)}$$

corresponds to the probability that we **randomly draw exactly the words in d** when drawing words from documents in c - Note that we make the simplifying assumption **that words occur independently from each other** (hence the product), which explains the label “**naïve**”

Example: Conditional Probabilities in Naïve Bayes

	Words			Class	
	a	b	x	y	-
d_1	2	1	0	0	H
d_2	0	1	2	2	S
d_3	2	1	1	1	H
d_4	1	0	2	2	S
d_5	1	1	0	0	H

$$P[\mathbf{a} \mid \mathbf{H}] = 5/10 \quad P[\mathbf{a} \mid \mathbf{S}] = 1/10$$

$$P[\mathbf{b} \mid \mathbf{H}] = 3/10 \quad P[\mathbf{b} \mid \mathbf{S}] = 1/10$$

$$P[\mathbf{x} \mid \mathbf{H}] = 1/10 \quad P[\mathbf{x} \mid \mathbf{S}] = 4/10$$

$$P[\mathbf{y} \mid \mathbf{H}] = 1/10 \quad P[\mathbf{y} \mid \mathbf{S}] = 4/10$$

Putting Naïve Bayes to Use

- Let's classify a previously unseen document

	a	b	x	y	
d_6	2	2	1	1	?

- We obtain the following probabilities

$$\begin{aligned}P[\mathbf{H} \mid d_6] &= P[d_6 \mid \mathbf{H}] \cdot P[\mathbf{H}] \\&= \left(\frac{5}{10} \cdot \frac{5}{10} \cdot \frac{3}{10} \cdot \frac{3}{10} \cdot \frac{1}{10} \cdot \frac{1}{10} \right) \cdot \frac{3}{5} = 135/10^6\end{aligned}$$

$$\begin{aligned}P[\mathbf{S} \mid d_6] &= P[d_6 \mid \mathbf{S}] \cdot P[\mathbf{S}] \\&= \left(\frac{1}{10} \cdot \frac{1}{10} \cdot \frac{1}{10} \cdot \frac{1}{10} \cdot \frac{4}{10} \cdot \frac{4}{10} \right) \cdot \frac{2}{5} = 6.4/10^6\end{aligned}$$

- and thus classify the document as Ham.

Working with Probabilities

When implementing probabilistic methods like Naïve Bayes, it is often useful to apply a logarithmic transformation to avoid underflows and other numerical issues.

$$\log P[c | d] \propto \log P[d | c] + \log P[c]$$

$$\log P[d | c] \propto \sum_{w \in d} f(w, d) \cdot \log P[w | c]$$

Classifying Reviews with Naïve Bayes

We'll work on a dataset of more than 400,000 reviews from Amazon about unlocked mobile phones: <https://www.kaggle.com/PromptCloudHQ/amazon-reviews-unlocked-mobile-phones/version/1>

```
"" "CLEAR CLEAN ESN" " Sprint EPIC 4G Galaxy SPH-  
D700*FRONT CAMERA*ANDROID*SLIDER*QWERTY  
KEYBOARD*TOUCH SCREEN", Samsung, 199.99, 4, "nice  
phone, nice up grade from my pantach revue. Very  
clean set up and easy set up. never had an android  
phone but they are fantastic to say the least.  
perfect size for surfing and social media. great  
phone samsung", 0
```

We want for instance to predict the rating (1-5) from the content

Amazon Reviews: Unlocked Mobile Phones

176

New Notebook

Download (34 MB)

Data Card Code (118) Discussion (6)

Amazon_Unlocked_Mobile.csv (131.88 MB)






Detail Compact Column

6 of 6 columns

About this file

PromptCloud extracted 400 thousand reviews of unlocked mobile phones sold on Amazon.com to find out insights with respect to reviews, ratings, price and their relationships.

A Product Name	A Brand Name	# Price	# Rating	A Reviews	# Review
The name of the Product. e.g. Sprint EPIC 4G Galaxy SPH-D7	Name of the parent company. e.g. Samsung	Price of the product. (Max: 2598, Min: 1.73, Mean: 226.86)	Rating of the product ranging between 1-5	Description of the user experience	Number of the review 645, Mean
4410 unique values	Samsung 16% [null] Other (282922) 68%			162492 unique values	
"CLEAR CLEAN ESN" Sprint EPIC 4G Galaxy SPH-D700*FRONT CAMERA*ANDROID*SLIDE R*QWERTY KEYBOARD*TOUCH S...	Samsung	199.99	5	I feel so LUCKY to have found this used (phone to us & not used hard at all), phone on line from som...	1
"CLEAR CLEAN ESN" Sprint EPIC 4G Galaxy SPH-D700*FRONT CAMERA*ANDROID*SLIDE R*QWERTY KEYBOARD*TOUCH S...	Samsung	199.99	4	nice phone, nice up grade from my pantach revue. Very clean set up and easy set up, never had an and...	0

Data Explorer

Version 1 (131.88 MB)

Amazon_Unlocked_Mobile.csv

Classifying Reviews with Naïve Bayes

https://scikit-learn.org/stable/modules/naive_bayes.html; Code example here: [demo/Naive_B_kaggle.py](#)

```
import math
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
# load data
reviews = pd.read_csv("Amazon_Unlocked_Mobile.csv", encoding='utf-8')

X = reviews.iloc[:,4].values
X_clean = X[pd.notnull(X)]
y = reviews.iloc[:,3].values
y_clean = y[pd.notnull(X)]

# convert documents into bags-of-words
vectorizer = CountVectorizer()
X_cnt = vectorizer.fit_transform(X_clean)

# split into training data (80%) and test data (20%)
X_train, X_test, y_train, y_test = train_test_split(X_cnt, y_clean,
test_size=0.2, random_state=0)

# train naive bases classifier
nb = MultinomialNB(alpha=0.0)
nb.fit(X_train, y_train)
```

Classifying Reviews with Naïve Bayes

```
# predict labels
y_predicted = nb.predict(X_test)

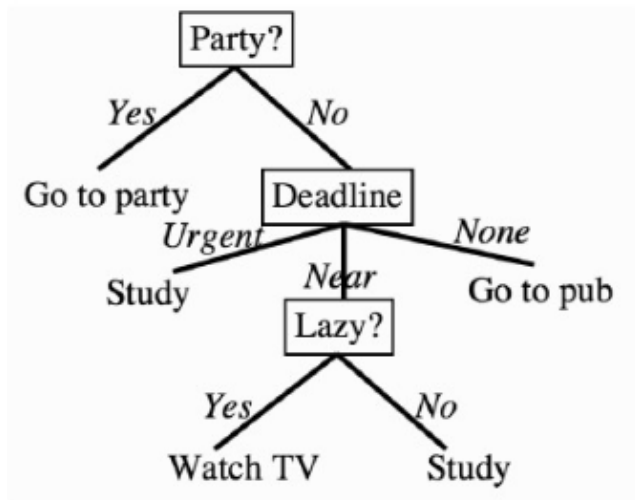
# compute confusion matrix
print(confusion_matrix(y_true=y_test, y_pred=y_predicted))

# compute accuracy
print(accuracy_score(y_true=y_test, y_pred=y_predicted))

# print class priors
for c in range(0, len(nb.classes_)):
    print('Class: ' + str(c))
    print(str(math.exp(nb.class_log_prior_[c])))

# print probabilities per class for words in {android, apple, ... , error, crash}
feature_names = vectorizer.get_feature_names()
for w in ['android', 'apple', 'good', 'bad', 'terrible', 'error', 'crash']:
    print('Word: ' + w)
    for c in range(0, len(nb.classes_)):
        print(str(c) + " : " + str(math.exp(nb.coef_[c][feature_names.index(w)])))
```

5. Decision Trees



A simple decision tree to decide how you will spend the evening.

Decision Trees — what is it about

- ▶ Decision trees: Supervised Learning.
- ▶ Decision trees are a family of classification methods that support an arbitrary number of classes.
- ▶ A decision tree provides a sequence of (binary) decisions which have to be made in order to decide which class a data point belongs to.
- ▶ Decision trees are often praised for their interoperability:
 - ▶ we can see why a data point ended up in a specific class, and in principle- the classification could be performed by a human user not knowing anything about ML.

Benefits of Decision Trees

- ▶ The computational cost of making the tree is fairly low.
- ▶ The cost of using it is even lower: $O(\log N)$, where N is the number of data points (cf. Bisection methods).
- ▶ Important for machine learning:
 - ▶ **querying** the trained algorithm **is fast**.
 - ▶ the result is immediately **easy to understand**.
 - ▶ makes people **trust** it more than getting an answer from a black box.
- ▶ e.g.: phone a helpline for computer faults. The phone operators are guided through the decision tree by your answers to their questions.

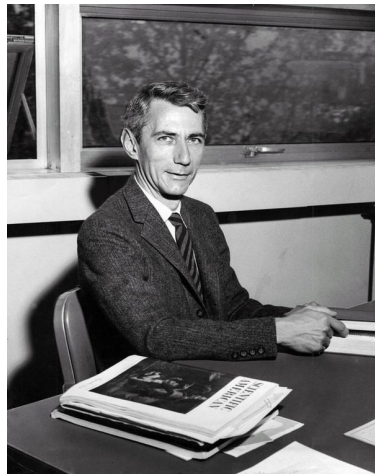
Idea of the Tree

- ▶ We start at the **root (base)** of the tree and progress down to the **leaves**, where we receive the classification decision.
- ▶ At each stage we **choose a question that gives us the most information** given what we know already. Encoding this mathematically is the task **of information theory**.
- ▶ The trees can even be turned into a set of if-then rules, suitable for use in a **rule induction system**.

Information Theory

Claude E. Shannon (1916-2001) proposed **Information Theory**, which is important for, e.g.:

- ▶ encoding and compression.
- ▶ data transmission (e.g., in networks) and has applications in.
- ▶ Machine Learning.
- ▶ Information Retrieval.
- ▶ Natural Language Processing.



Information Theory: Claude E. Shannon



Claude Shannon
1916–2001

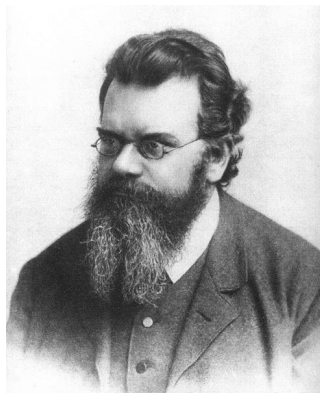
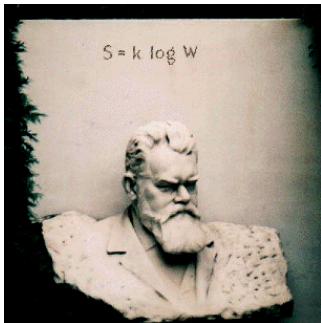
After graduating from Michigan and MIT, Shannon joined the AT&T Bell Telephone laboratories in 1941. His paper 'A Mathematical Theory of Communication' published in the *Bell System Technical Journal* in 1948 laid the foundations for modern information the-

ory. This paper introduced the word 'bit', and his concept that information could be sent as a stream of 1s and 0s paved the way for the communications revolution. It is said that von Neumann recommended to Shannon that he use the term entropy, not only because of its similarity to the quantity used in physics, but also because "nobody knows what entropy really is, so in any discussion you will always have an advantage".

Ludwig Boltzmann

* 20. Februar 1844 in Wien;

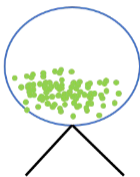
† 5. September 1906 in Duino, Austria-Hungary



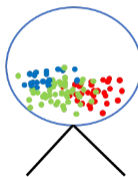
Information Theory

- ▶ We ask **how much information is received** when we observe a specific value for this variable.
- ▶ The amount of information can be viewed as the '**degree of surprise**' on **learning the value of x** .
- ▶ If we are told that **a highly improbable event has just occurred**, we will have **received more information** than if we were told that some very likely event has just occurred.
- ▶ If we knew that the event was certain to happen we would receive no information.

Totally pure



More impure



Information Theory

- ▶ Our measure of **information content** will therefore depend on the probability distribution $p(x)$.
- ▶ We therefore look for a **quantity $h(x)$** that is a monotonic function of the probability $p(x)$ and that expresses the information content.
- ▶ The form of $h(\cdot)$ can be found by noting that if **we have two events** x and y that are **unrelated**, then the information gain from observing both of them should be the sum of the information gained from each of them separately, so that **$h(x, y) = h(x) + h(y)$** .
- ▶ Two unrelated events will be statistically independent and so **$p(x, y) = p(x)p(y)$** .
- ▶ From these two relationships, it is easily shown that $h(x)$ must be given **by the logarithm of** $p(x) \rightarrow h(x) = -\log(p(x))$.

Entropy

- ▶ Entropy quantifies the amount of uncertainty of a random variable Y .
- ▶ “Average amount of information”

$$H(Y) = - \sum_y P[y] \log_2 P[y]$$

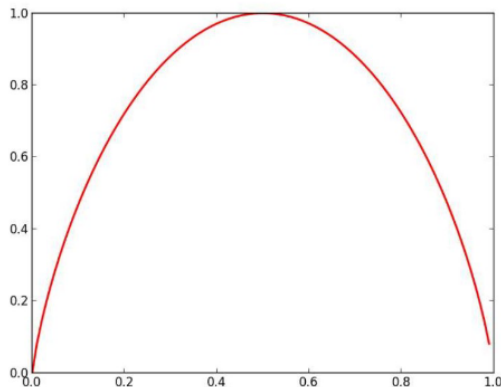
- ▶ Negative sign: ensures that information is positive or zero.
- ▶ Choice of Basis because of “bits” (binary integers).
- ▶ Example: Let's consider a fair coin

$$P(Y = \text{head}) = P(Y = \text{tail}) = \frac{1}{2}$$
$$H(Y) = - \left(\frac{1}{2} \log \frac{1}{2} + \frac{1}{2} \log \frac{1}{2} \right) = 1$$



Binary Decision Theory

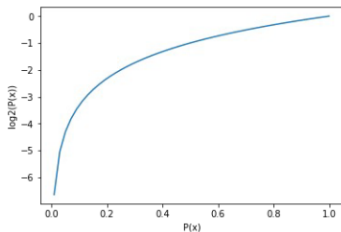
- ▶ Binary decision : “positive” with probability p , “negative” with probability $1 - p$
- ▶ Maximal entropy of 1 lies by $p = 0.5$



Entropy — Python Code

demo/entropy.py

```
import numpy as np
import matplotlib.pyplot as plt
fig=plt.figure()
ax=fig.add_subplot(111)
ax.plot(np.linspace(0.01,1),np.log2(np.linspace(0.01,1)))
ax.set_xlabel("P(x)")
ax.set_ylabel("log2(P(x))")
plt.show()
```



Entropy (II)

- Entropy quantifies the amount of uncertainty of a random variable Y .

$$H(Y) = - \sum_y P[y] \log_2 P[y]$$

- Example: Let's consider a **biased coin**

$$P(Y = \text{head}) = \frac{3}{4} \quad P(Y = \text{tail}) = \frac{1}{4}$$

$$H(Y) = - \left(\frac{3}{4} \log \frac{3}{4} + \frac{1}{4} \log \frac{1}{4} \right) \approx 0.6579$$

- Less uncertainty for the biased coin, or put differently, **knowing the outcome of fair coin provides more information.**



Entropy (III)

- Entropy quantifies the amount of uncertainty of a random variable Y .

$$H(Y) = - \sum_y P[y] \log_2 P[y]$$

- Example: Let's consider a **fair dice**

$$P(Y = n) = \frac{1}{6} \quad \text{with} \quad n \in \{1, \dots, 6\}$$

$$H(Y) = - \left(\frac{1}{6} \log \frac{1}{6} + \dots + \frac{1}{6} \log \frac{1}{6} \right) \approx 2.5849$$



Entropy (IV)

- Entropy quantifies the amount of uncertainty of a random variable Y .

$$H(Y) = - \sum_y P[y] \log_2 P[y]$$

- Example: Let's consider a biased dice

$$P(Y=1) = \frac{1}{2}$$

$$P(Y=n) = \frac{1}{10} \quad \text{with} \quad n \in \{2, \dots, 6\}$$

$$H(Y) = - \left(\frac{1}{2} \log \frac{1}{2} + \frac{1}{10} \log \frac{1}{10} + \dots + \frac{1}{10} \log \frac{1}{10} \right) \approx 2.1609$$



Conditional Entropy

- ▶ **Conditional entropy** quantifies the **amount of uncertainty** of a random variable Y if the value of another random variable X is known.

$$H(Y | X) = \sum_x P[x] H(Y | x)$$

- ▶ Example: Let's again consider a fair dice
 - ▶ random variable Y indicates **face of dice** $\{1, \dots, 6\}$
 - ▶ random variable X indicates whether **face is odd or even** $\{o, e\}$

$$P(Y = n) = \frac{1}{6} \quad \text{with} \quad n \in \{1, \dots, 6\}$$

$$P(X = o) = P(X = e) = \frac{1}{2}$$

Conditional Entropy (II)

Example (cont'd): Let's look at the conditional probabilities

y	x	$P[y x]$
1	o	$1/3$
1	e	0
2	o	0
2	e	$1/3$
3	o	$1/3$
3	e	0
4	o	0
4	e	$1/3$
5	o	$1/3$
5	e	0
6	o	0
6	e	$1/3$

$$H(Y | o) = - \left(\frac{1}{3} \log \frac{1}{3} + 0 + \frac{1}{3} \log \frac{1}{3} + 0 + \dots \frac{1}{3} \log \frac{1}{3} + 0 \right) \\ \approx 1.5849$$

$$H(Y | e) = - \left(0 + \frac{1}{3} \log \frac{1}{3} + 0 + \frac{1}{3} \log \frac{1}{3} + 0 + \dots \frac{1}{3} \log \frac{1}{3} \right) \\ \approx 1.5849$$

$$H(Y | X) = \frac{1}{2} H(Y | o) + \frac{1}{2} H(Y | e) \\ \approx 1.5849$$

Intuition: Knowing whether an odd or even face is shown reduces the amount of uncertainty.

Information Gain

- **Information gain** (also: mutual information) measures how much information is gained about a random variable Y if the value of another random variable X is known.

$$I(X, Y) = \sum_x \sum_y P[x \wedge y] \log \left(\frac{P[x \wedge y]}{P[x]P[y]} \right)$$

- which can be rewritten as

$$I(X, Y) = H(X) - H(X | Y)$$

$$I(X, Y) = H(Y) - H(Y | X)$$

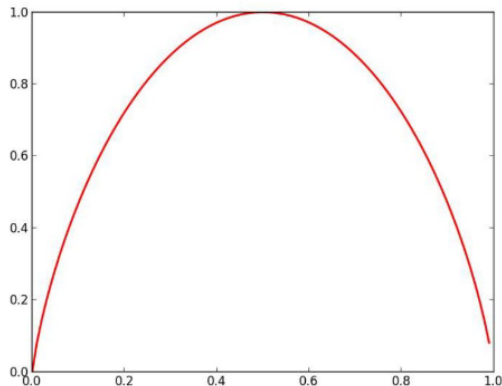
Information Gain (II)

- ▶ Information Gain (II)- Example: Let's again consider a fair dice.
- ▶ random variable Y indicates face of dice $\{1, \dots, 6\}$.
- ▶ random variable X indicates whether face is odd or even $\{o, e\}$

$$\begin{aligned} I(X, Y) &= H(Y) - H(Y | X) \\ &\approx 2.5849 - 1.5849 = 1 \end{aligned}$$

Construct a Decision Tree

- ▶ If the feature separates the example into 50% positive and 50% negative, then the amount of entropy is at a maximum, and knowing about that feature is very useful to us.
- ▶ For our decision tree, the best feature to pick as the one to classify on now is the one that gives you **most information**, i.e., the one with the highest entropy.
- ▶ After using that feature, we **re-evaluate the entropy of each feature and again pick the one with the highest entropy.**



Recursively Growing a Decision Tree

- ▶ We'll now learn to generate a decision tree recursively.
 - ▶ random variable Y indicates class labels (e.g., Europe, U.S.)
 - ▶ random variable X indicates the outcome of a binary decision (e.g., does the car weight more than 2000 lbs).
- ▶ In each step, we need to determine the best possible decision criterion to split the remaining data points (hint: we'll use information gain to this end).

Recursively Growing a Decision Tree (II)

- ▶ **Recursive top-down generation** of a decision tree (i.e., from root to leaves).
- ▶ if only a few data points are left or most of them belong to a single class, then generate a leaf with majority class.
- ▶ otherwise, **identify the best possible decision criterion**, split the data accordingly, and recursively generate left and right decision subtree.

Recursively Growing a Decision Tree: pseudo-code snippet

```
Node decisionTree(Data d) {  
    // Generate leaf node, if data is pure  
    if (isPure(d)) {  
        return Leaf(majorityClass(d));  
    }  
  
    // Determine best decision criterion  
    Condition c = bestSplit(d);  
  
    // Split the data accordingly  
    Data ld, rd = split(d, c)  
  
    // Recursively generate subtree  
    Node lc = decisionTree(ld);  
    Node rc = decisionTree(rd);  
  
    return new Node(c, lc, rc);  
}
```

Splits for Numerical and Ordinal Features

- ▶ For numerical and ordinal features we can compare the value of the feature x against a threshold a

$$X: x \leq a$$

- ▶ The random variable X thus indicates whether the feature value x of a data point is smaller or equal than the threshold a (true) or larger (false).
- ▶ In theory, there can be an infinite number of possible thresholds a to consider; in practice, it is good enough to consider the feature values that appear in the data.

Splits for Numerical and Ordinal Features

- Example: Let's assume that we have observed the following values for a feature x in our training data

$$x \in \{2, 4, 5, 6, 8, 10, 12\}$$

- We need to consider the following decision criteria

$$X: x \leq a \text{ with } a \in \{2, 4, 5, 6, 8, 10\}$$

- Note: Some implementations consider thresholds midway between values observed in the data, i.e.:

$$X: x \leq a \text{ with } a \in \{3.5, 4.5, 5.5, 7, 9, 11\}$$

Splits for Nominal Features

- For **nominal features** we can check whether the value of the feature **x is in a subset A** of observed feature values

$$X: x \in A$$

- The random variable X thus indicates whether the feature value x of a data point is contained in the subset A (true) or not (false). If there are n observed values for feature x in our data, we need to consider an exponential number of $2^{(n-1)} - 1$ subsets A .

Splits for Nominal Features

- Example: Let's assume that we have observed the following values for a feature x in our training data

$$x \in \{r, g, b, w\}$$

- We need to consider the following decision criteria

$$X: x \in A \text{ with}$$

$$A \in \{\{r\}, \{g\}, \{b\}, \{w\}, \{r, g\}, \{r, b\}, \{r, w\}, \dots\}$$

- Note: There is no need to consider all 2^n subsets, because (i) the empty set $\{\}$ does not split the data and (ii) a subset and its complementary set result in the same split (e.g., $\{r\}$ and $\{g, b, w\}$ in our example).

If we grow the decision tree until all leaves contain only data points from a single class, we're prone to overfitting.

It is better to **prune the decision tree** by stopping the recursive generation, once less than **minSize** data points are seen or at least **minPercentage** percent of data points belong to the majority class.

Decision Trees (Example)

Example: Predict the risk of an insurance customer based on **age** and **type of car**.

Age	Type of Car	Risk
25	Roadster	Low
20	Oldtimer	High
25	Roadster	Low
45	SUV	High
20	Roadster	High
25	SUV	High

$$H(\text{Risk}) = - \left(\left(\frac{1}{3} \log \frac{1}{3} \right) + \left(\frac{2}{3} \log \frac{2}{3} \right) \right) = 0.9183$$

Decision Trees (Example)

Step 1: Determine the root of the decision tree

Decision criterion Age : 20

$$\begin{aligned}H(\text{Risk} \mid \text{Age} : 20) &= \frac{1}{3}H(\text{Risk} \mid \text{Age} \leq 20) + \frac{2}{3}H(\text{Risk} \mid \text{Age} > 20) \\&= \frac{1}{3}(0) + \frac{2}{3}(1) \\&= 0.6667 \\I(\text{Risk}, \text{Age} : 20) &= 0.9183 - 0.6667 \\&= 0.2516\end{aligned}$$

Age	Type of Car	Risk
25	Roadster	Low
20	Oldtimer	High
25	Roadster	Low
45	SUV	High
20	Roadster	High
25	SUV	High

Decision Trees (Example)

Decision criterion Age : 25

$$\begin{aligned}H(\text{Risk} \mid \text{Age} : 25) &= \frac{5}{6}H(\text{Risk} \mid \text{Age} \leq 25) + \frac{1}{6}H(\text{Risk} \mid \text{Age} > 25) \\&= \frac{5}{6} \left(- \left(\frac{2}{5} \log \frac{2}{5} + \frac{3}{5} \log \frac{3}{5} \right) \right) + \frac{1}{6}(0) \\&= 0.8091 \\I(\text{Risk}, \text{Age} : 25) &= 0.9183 - 0.8091 \\&= 0.1092\end{aligned}$$

Age	Type of Car	Risk
25	Roadster	Low
20	Oldtimer	High
25	Roadster	Low
45	SUV	High
20	Roadster	High
25	SUV	High

Decision Trees (Example)

Decision criterion Type : {Oldtimer}

$$\begin{aligned}
 H(\text{Risk} \mid \text{Type} : \{O\}) &= \frac{1}{6} H(\text{Risk} \mid \text{Type} \in \{O\}) + \frac{5}{6} H(\text{Risk} \mid \text{Type} = \{O\}) \\
 &= \frac{1}{6}(0) + \frac{5}{6} \left(- \left(\frac{2}{5} \log \frac{2}{5} + \frac{3}{5} \log \frac{3}{5} \right) \right) \\
 &= 0.8091 \\
 I(\text{Risk}, \text{Type} : \{O\}) &= 0.9183 - 0.8091 \\
 &= 0.1092
 \end{aligned}$$

Age	Type of Car	Risk
25	Roadster	Low
30	Oldtimer	High
20	Roadster	Low
45	SUV	High
20	Roadster	High
25	SUV	High

Decision Trees (Example)

Decision criterion Type : {Roadster}

$$H(\text{Risk} \mid \text{Type} : \{R\}) = \frac{3}{6} H(\text{Risk} \mid \text{Type} \in \{R\}) + \frac{3}{6} H(\text{Risk} \mid \text{Type} \in \{R\})$$

$$= \frac{3}{6} \left(- \left(\frac{2}{3} \log \frac{2}{3} + \frac{1}{3} \log \frac{1}{3} \right) \right) + \frac{3}{6} (0)$$

$$= 0.4591$$

$$I(\text{Risk}, \text{Type} : \{R\}) = 0.9183 - 0.4591$$

$$= 0.4592$$

Age	Type of Car	Risk
25	Roadster	Low
20	Oldtimer	High
25	Roadster	Low
45	SUV	High
20	Roadster	High
25	SUV	High

Decision Trees (Example)

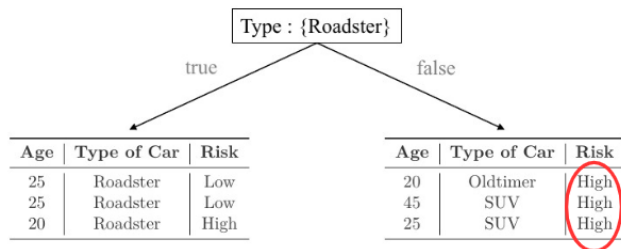
Decision criterion Type : {SUV}

$$\begin{aligned}
 H(\text{Risk} \mid \text{Type} : \{S\}) &= \frac{2}{6} H(\text{Risk} \mid \text{Type} \in \{S\}) + \frac{4}{6} H(\text{Risk} \mid \text{Type} \notin \{S\}) \\
 &= \frac{2}{6}(0) + \frac{4}{6} \left(- \left(\frac{1}{2} \log \frac{1}{2} + \frac{1}{2} \log \frac{1}{2} \right) \right) \\
 &= 0.6667 \\
 I(\text{Risk}, \text{Type} : \{S\}) &= 0.9183 - 0.6667 \\
 &= 0.2516
 \end{aligned}$$

Age	Type of Car	Risk
25	Roadster	Low
20	Oldtimer	High
25	Roadster	Low
45	SUV	High
20	Roadster	High
25	SUV	High

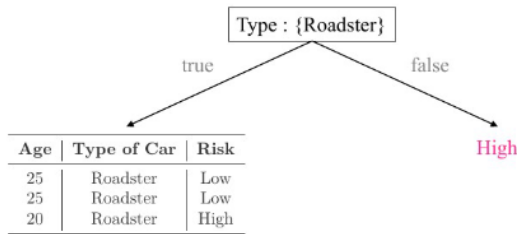
Decision Trees (Example)

- Pick Type : {Roadster} as the **decision criterion** for the root node, because it achieves the **highest information gain**.
- Split the data accordingly and recursively grow decision trees for the obtained subsets of the data.



Decision Trees (Example)

- For the right subtree, there is **nothing to do**, since all data points belong to a single class



Age	Type of Car	Risk
20	Oldtimer	High
45	SUV	High
25	SUV	High

Decision Trees (Example)

- For the left subtree, there is only one decision criterion to consider, namely Age : 20

$$H(\text{Risk}) = - \left(\frac{1}{3} \log \frac{1}{3} + \frac{2}{3} \log \frac{2}{3} \right)$$

$$= 0.9183$$

$$H(\text{Risk} \mid \text{Age} : 20) = \frac{1}{3} H(\text{Risk} \mid \text{Age} \leq 20) + \frac{2}{3} H(\text{Risk} \mid \text{Age} > 20)$$

$$= \frac{1}{3}(0) + \frac{2}{3}(1)$$

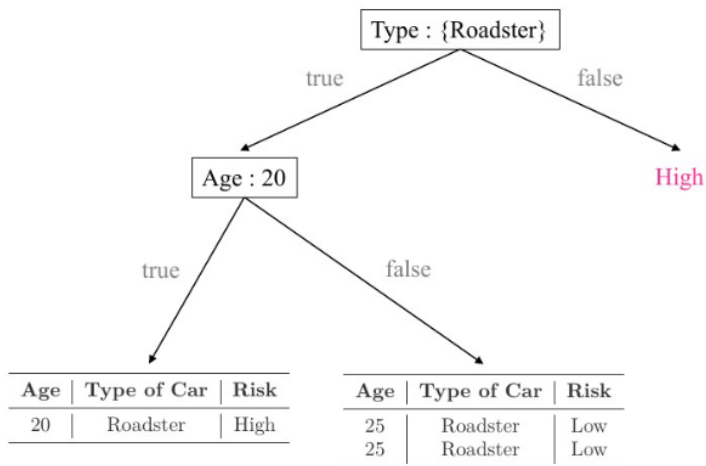
$$= 0.6667$$

$$I(\text{Risk}, \text{Age} : 20) = 0.9183 - 0.6667$$

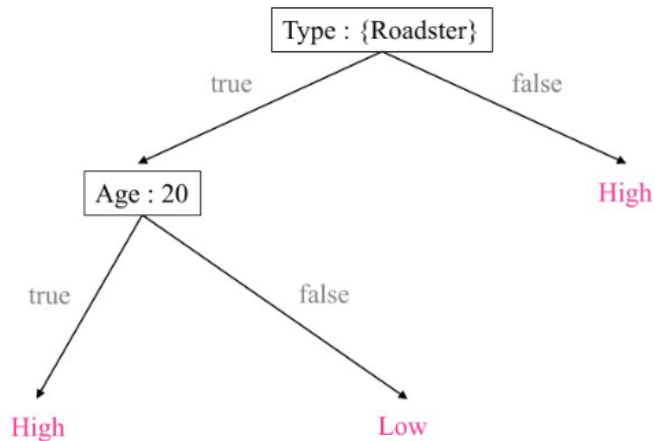
$$= 0.2516$$

Age	Type of Car	Risk
25	Roadster	Low
25	Roadster	Low
20	Roadster	High

Decision Trees (Example)



Decision Trees (Example)




```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.tree import export_graphviz

# load data
cars = pd.read_csv('auto-mpg.data.txt', header=None, sep='\s+')

# extract power and weight as data matrix X
X = cars.iloc[:, [3,4]].values

# extract origin as target vector y
y = cars.iloc[:, 7].values

# split into training data (80%) and test data (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# learn decision tree
tree = DecisionTreeClassifier(criterion='entropy')
tree.fit(X_train, y_train)
y_predicted = tree.predict(X_test)

# compute confusion matrix
print(confusion_matrix(y_true=y_test, y_pred=y_predicted))

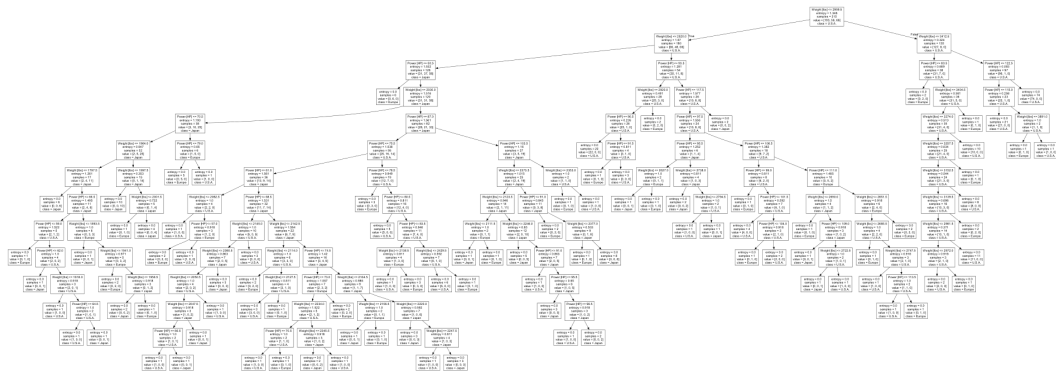
# compute accuracy
print(accuracy_score(y_true=y_test, y_pred=y_predicted)) # 0.620253164556962
```

Plotting the Decision Tree

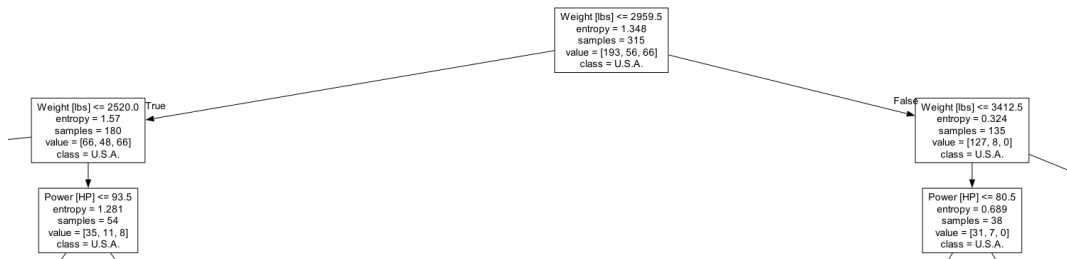
```
# Plotting the Decision Tree  
export_graphviz(tree, out_file='tree.dot',  
feature_names=['Power [HP]', 'Weight [lbs]'],  
class_names=['U.S.A.', 'Europe', 'Japan'])  
# you need to have Graphviz (graphviz.org) installed to open the generated file  
# to generate a PDF from the .dot file, run: dot -Tpdf tree.dot -o tree.pdf
```

→ **See next slide...**

Plotting the Decision Tree: Result



Plotting the Decision Tree: Close-up



→ can we generate a simpler decision tree?

Predicting Origin from Power and Weight

demo/auto_tree_simple.py

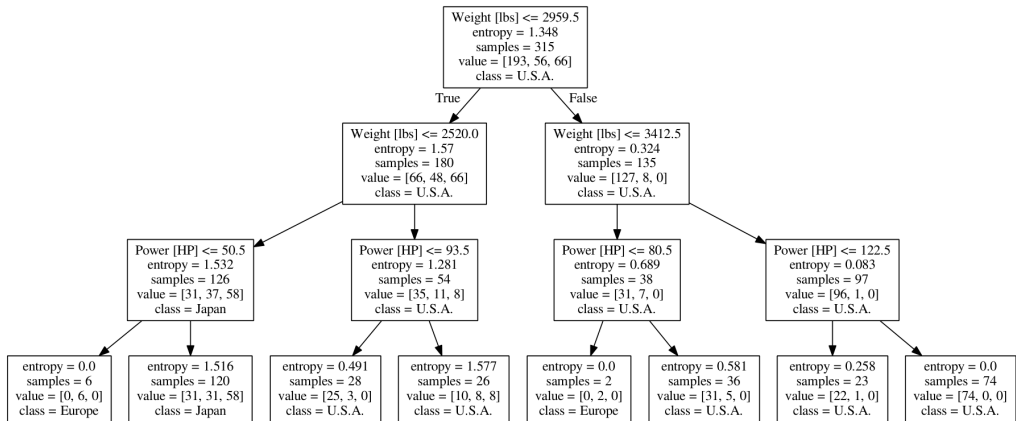
The beginning is the same; but this time, we will limit the depth of the tree to 3:

```
# learn decision tree of maximal depth 3
tree = DecisionTreeClassifier(criterion='entropy',max_depth=3)
tree.fit(X_train,y_train)
y_predicted = tree.predict(X_test)

# compute confusion matrix
print(confusion_matrix(y_true=y_test,y_pred=y_predicted))

# compute accuracy
print(accuracy_score(y_true=y_test,y_pred=y_predicted)) # 0.6582278481012658
#plot tree
export_graphviz(tree, out_file='tree.dot',
                 feature_names=['Power [HP]', 'Weight [lbs]'],
                 class_names=['U.S.A.', 'Europe', 'Japan'])
```

A Simpler Tree



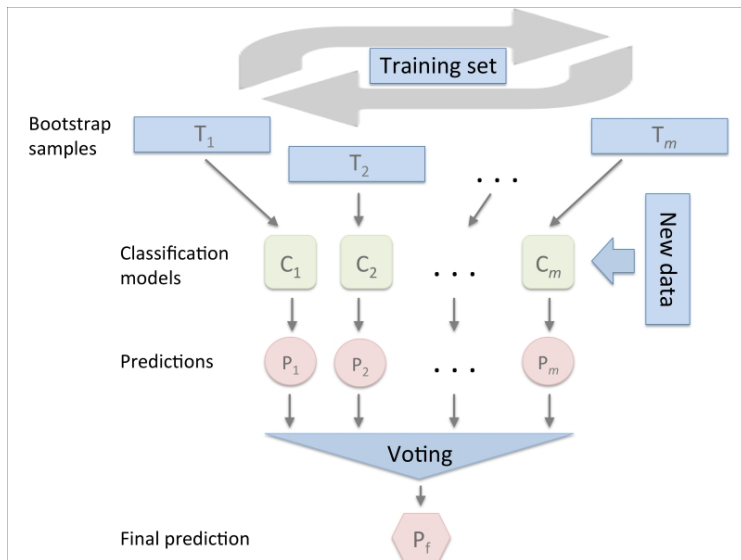
Classification with the Zoo Data set

`demo/zoo_predict_aux.py`

!! ACTION REQUIRED !!

- ▶ Run the decision tree method yourself.
- ▶ Go to here: <http://archive.ics.uci.edu/ml/datasets/zoo>
- ▶ This dataset consists of 101 rows and 17 categorically valued attributes defining whether an animal has a specific property or not (e.g.hairs, feathers,...).
- ▶ The first attribute represents the name of the animal and will be removed.
- ▶ The target feature consist of 7 integer values [1:7] which represents [1:Mammal,2:Bird,3:Reptile,4:Fish,5:Amphibian,6:Bug,7:Invertebrate]

6. Combining Models



Combining Models

- ▶ Motivation: let's say we have a number of (predictive) models for a problem.
- ▶ e.g. Regression with polynomials (different degree).
- ▶ e.g. Classification with support vector machines (kernel type, parameters, see later in this lecture/course).
- ▶ Often, **improved performance can be obtained by combining different models.**
- ▶ But **how can we combine them together?**

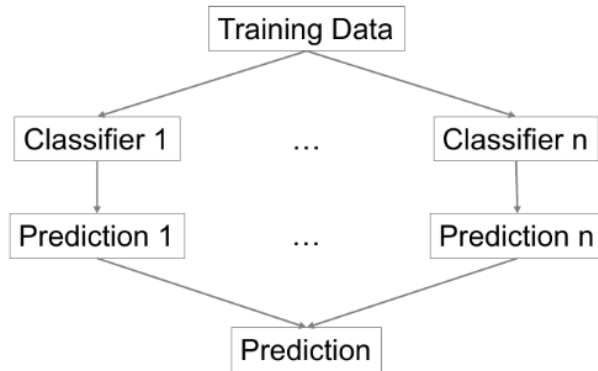
Ensemble Learning — what is it about?

See Bishop Chapter 14

- ▶ Ensemble learning determines multiple classification models (e.g., single method on different samples or multiple methods) and aggregates their predictions.
- ▶ Ensembles are often more robust than individual classifiers.

Example: Majority Voting

- The simplest ensemble method trains **different classifiers** on the same training data and uses **majority voting to determine the class of an unseen data point.**



Majority Voting — example

demo/majority.py

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import accuracy_score

# load data
cars = pd.read_csv('auto-mpg.data.txt', header=None, sep='\s+')

# extract power and weight as data matrix X
X = cars.iloc[:, [3,4]].values
# extract origin as target vector y
y = cars.iloc[:, 7].values
# split into training data (80%) and test data (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

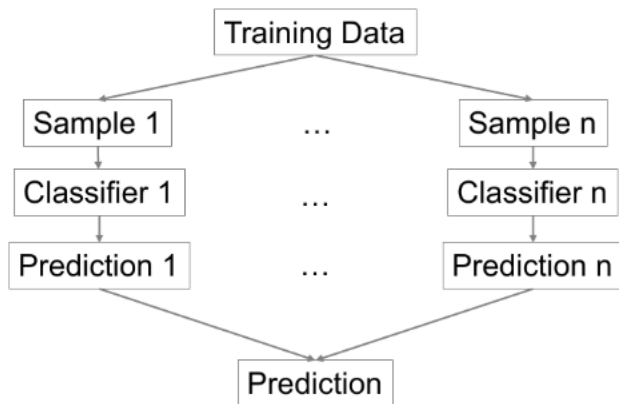
# fit logistic regression model on training data
lr = LogisticRegression()
# use kNN with k = 3
knn = KNeighborsClassifier(n_neighbors=3)

# learn decision tree
tree = DecisionTreeClassifier(criterion='entropy')

# voting classifier
vc = VotingClassifier(estimators=[('lr', lr), ('knn', knn), ('tree', tree)], voting='hard')
vc.fit(X_train, y_train)
vc_y_predicted = vc.predict(X_test)
print(accuracy_score(y_true=y_test, y_pred=vc_y_predicted)) # 0.683544303797
```


Bagging

- **Bagging (bootstrap aggregation)** trains different classifiers on samples of the training data and uses majority voting to determine the class of an unseen data point.



Bagging

demo/bagging.py

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.metrics import accuracy_score
# load data
cars = pd.read_csv('auto-mpg.data.txt', header=None, sep='\s+')

# extract power and weight as data matrix X
X = cars.iloc[:, [3,4]].values

# extract origin as target vector y
y = cars.iloc[:, 7].values

# split into training data (80%) and test data (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# learn decision tree of maximal depth 2
tree = DecisionTreeClassifier(criterion='entropy', max_depth=2)

# bagging classifier based on 10 decision trees
bc = BaggingClassifier(base_estimator=tree, n_estimators=10)
bc.fit(X_train, y_train)
bc_y_predicted = bc.predict(X_test)
print(accuracy_score(y_true=y_test, y_pred=bc_y_predicted)) # 0.6455696202531646
```

More formally: Committees

- ▶ A **combination of models** is often called a **committee**.
- ▶ Simplest way to combine models is to just average them together:

$$y_{COM}(x) = \frac{1}{M} \sum_{m=1}^M y_m(x)$$

- ▶ It turns out this simple method is better than (or same as) the individual models on average (**in expectation**).
- ▶ And usually slightly better.
- ▶ But there are better methods, which we shall discuss.

Error of individual models

- ▶ Consider **individual models** $Y_m(x)$, assume they can be written as true value plus error:

$$y_m(x) = h(x) + \epsilon_m(x)$$

- ▶ The expected value of the error of an individual model is then:

$$\mathbb{E}_x \left[\{y_m(x) - h(x)\}^2 \right] = \mathbb{E}_x [\epsilon_m(x)^2]$$

- ▶ The average error made by an individual model is then:

$$E_{AV} = \frac{1}{M} \sum_{m=1}^M \mathbb{E}_x [\epsilon_m(x)^2]$$

Error of Committee

The committee

$$y_{COM}(x) = \frac{1}{M} \sum_{m=1}^M y_m(x)$$

has expected error

$$\begin{aligned} E_{COM} &= \mathbb{E}_x \left[\left\{ \left(\frac{1}{M} \sum_{m=1}^M y_m(x) \right) - h(x) \right\}^2 \right] \\ &= \mathbb{E}_x \left[\left\{ \left(\frac{1}{M} \sum_{m=1}^M h(x) + \epsilon_m(x) \right) - h(x) \right\}^2 \right] \\ &= \mathbb{E}_x \left[\left\{ \left(\frac{1}{M} \sum_{m=1}^M \epsilon_m(x) \right) + h(x) - h(x) \right\}^2 \right] = \mathbb{E}_x \left[\left\{ \frac{1}{M} \sum_{m=1}^M \epsilon_m(x) \right\}^2 \right] \end{aligned}$$

Committee Error vs. Individual Error

- So, the committee error is

$$E_{COM} = \mathbb{E}_x \left[\left\{ \frac{1}{M} \sum_{m=1}^M \epsilon_m(x) \right\}^2 \right] = \frac{1}{M^2} \sum_{m=1}^M \sum_{n=1}^M \mathbb{E}_x [\epsilon_m(x) \epsilon_n(x)]$$

- If we assume errors are uncorrelated, $\mathbb{E}_x [\epsilon_m(x) \epsilon_n(x)] = 0$ when $m \neq n$, then:

$$E_{COM} = \frac{1}{M^2} \sum_{m=1}^M \mathbb{E}_x [\epsilon_m(x)^2] = \frac{1}{M} E_{AV}$$

- However, **errors are rarely uncorrelated**
 - For example, if all errors are the same, $\epsilon_m(x) = \epsilon_n(x)$, then $E_{COM} = E_{AV}$
 - Using Jensen's inequality (convex functions), can show $E_{COM} \leq E_{AV}$

Boosting (advanced — to read at home)

- ▶ Boosting is a technique for combining classifiers into a committee.
- ▶ We describe **AdaBoost (adaptive boosting)**, the most commonly used variant.
- ▶ Boosting is a meta-learning technique.
- ▶ Combines a set of classifiers trained using their own learning algorithms.
- ▶ Magic: can work well even if those classifiers only perform slightly better than random!

The Boosting Model

- ▶ We consider two-class classification problems, training data (x_i, t_i) , with $t_i \in \{-1, 1\}$.
- ▶ In boosting we build a “linear” classifier of the form:

$$y(x) = \sum_{m=1}^M \alpha_m y_m(x)$$

- ▶ A committee of classifiers, with weights.
- ▶ In boosting terminology:
 - ▶ Each $y_m(x)$ is called a **weak learner** or **base classifier**.
 - ▶ Final classifier $y(x)$ is called a **strong learner**.
- ▶ **Learning problem:** how do we choose the weak learners $y_m(x)$ and weights α_m ?

Example — Thresholds

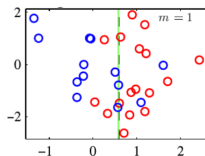
- ▶ Let's consider a simple example where weak learners are thresholds
- ▶ i.e. each $y_m(x)$ is of the form:

$$y_m(x) = x_i > \theta$$

- ▶ To allow different directions of threshold, include $p \in \{-1, +1\}$:

$$y_m(x) = px_i > p\theta$$

Choosing Weak Learners

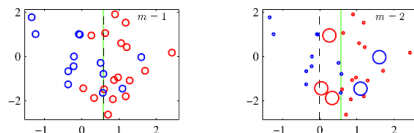


- ▶ Boosting is a greedy strategy for building the strong learner

$$y(x) = \sum_{m=1}^M \alpha_m y_m(x)$$

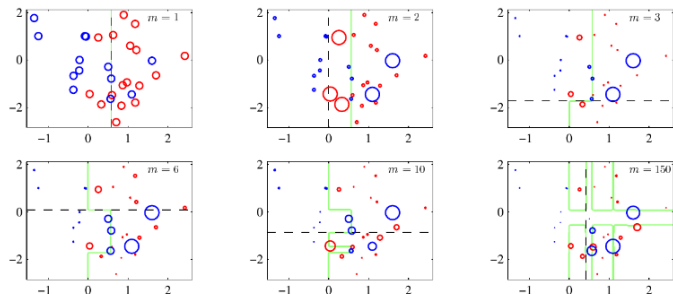
- ▶ Start by choosing the **best** weak learner, use it as $y_1(x)$.
- ▶ **Best** is defined as that which minimizes number of mistakes made (0-1 classification loss).
- ▶ i.e. search over all p, θ, i to find best $y_m(x) = px_i > p\theta$.

Choosing Weak Learners



- ▶ The first weak learner $Y_1(x)$ made some mistakes.
 - ▶ Choose the second weak learner $y_2(x)$ to try to get those ones correct
 - ▶ Best is now defined as that which minimizes weighted number of mistakes made
 - ▶ Higher weight given to those $y_1(x)$ got incorrect.
- Strong learner now $y(x) = \alpha_1 y_1(x) + \alpha_2 y_2(x)$.

Choosing Weak Learners



- Repeat: re-weight examples and choose new weak learner based on weights.
- Green line shows decision boundary of strong learner.

What About Those Weights?

- ▶ So exactly **how should we choose the weights** for the examples when classified incorrectly?
- ▶ And what should the α_m be for combining the weak learners $Y_m(x)$?
- ▶ As usual, **we define a loss function**, and choose these parameters to minimize it.

Exponential Loss

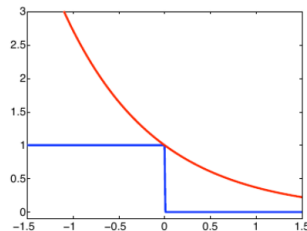
- ▶ Boosting attempts to minimize the exponential loss function

$$E_n = \exp \{ -t_n y(x_n) \}$$

error on n^{th} training example.

- ▶ Exponential loss is differentiable approximation to 0/1 loss
- ▶ Better for optimization
- ▶ Total error

$$E = \sum_{n=1}^N \exp \{ -t_n y(x_n) \}$$



Minimizing Exponential Loss

- ▶ Let's assume we've already chosen weak learners
- ▶ $Y_1(x), \dots, Y_{m-1}(x)$ and their weights $\alpha_1, \dots, \alpha_{m-1}$.
- ▶ Define $f_{m-1}(x) = \alpha_1 Y_1(x) + \dots + \alpha_{m-1} Y_{m-1}(x)$
- ▶ Just focus on choosing $y_m(x)$ and α_m .
- ▶ Greedy optimization strategy.
- ▶ Total error using exponential loss is:

$$\begin{aligned} E &= \sum_{n=1}^N \exp \{-t_n y(x_n)\} = \sum_{n=1}^N \exp \{-t_n [f_{m-1}(x_n) + \alpha_m y_m(x_n)]\} \\ &= \sum_{n=1}^N \exp \{-t_n f_{m-1}(x_n) - t_n \alpha_m y_m(x_n)\} \\ &= \sum_{n=1}^N \underbrace{\exp \{-t_n f_{m-1}(x_n)\}}_{\text{weight } w_n^{(m)}} \exp \{-t_n \alpha_m y_m(x_n)\} \end{aligned}$$

Weighted Loss

- ▶ On the m -th iteration of boosting, we are choosing Y_m and α_m to minimize the weighted loss:

$$E = \sum_{n=1}^N w_n^{(m)} \exp \{-t_n \alpha_m y_m(x_n)\}$$

- ▶ where $w_n^{(m)} = \exp \{-t_n f_{m-1}(x_n)\}$.
- ▶ Can define these as weights since they are constant wrt y_m and α_m .
- ▶ We'll see they're the right weights to use.

Minimization wrt y_{text}

- Consider the weighted loss

$$E = \sum_{n=1}^N w_n^{(m)} e^{-t_n \alpha_m y_m(x_n)} = e^{-\alpha_m} \sum_{n \in \mathcal{T}_m} w_n^{(m)} + e^{\alpha_m} \sum_{n \in \mathcal{M}_m} w_n^{(m)}$$

where \mathcal{T}_m is the set of points correctly classified by the choice of $y_m(x)$, and \mathcal{M}_m those that are not

$$\begin{aligned} E &= e^{\alpha_m} \sum_{n=1}^N w_n^{(m)} I(y_m(x_n) \neq t_n) + e^{-\alpha_m} \sum_{n=1}^N w_n^{(m)} (1 - I(y_m(x_n) \neq t_n)) \\ &= (e^{\alpha_m} - e^{-\alpha_m}) \sum_{n=1}^N w_n^{(m)} I(y_m(x_n) \neq t_n) + e^{-\alpha_m} \sum_{n=1}^N w_n^{(m)} \end{aligned}$$

- Since the second term is a constant wrt y_m and $e^{\alpha_m} - e^{-\alpha_m} > 0$ if $\alpha_m > 0$ best y_m minimizes weighted 0-1 loss.

Choosing α_m

- ▶ So best y_m minimizes weighted 0-1 loss .regardless of α_m .
- ▶ How should we set α_m given this best y_m ?
- ▶ Recall from above:

$$\begin{aligned} E &= e^{\alpha_m} \sum_{n=1}^N w_n^{(m)} I(y_m(x_n) \neq t_n) + e^{-\alpha_m} \sum_{n=1}^N w_n^{(m)} (1 - I(y_m(x_n) \\ &= e^{\alpha_m} \epsilon_m + e^{-\alpha_m} (1 - \epsilon_m) \end{aligned}$$

where we define ϵ_m to be the weighted error of Y_m .

- ▶ Calculus: $\alpha_m = \frac{1}{2} \log \frac{1-\epsilon_m}{\epsilon_m}$.

AdaBoost: Algorithm Summary

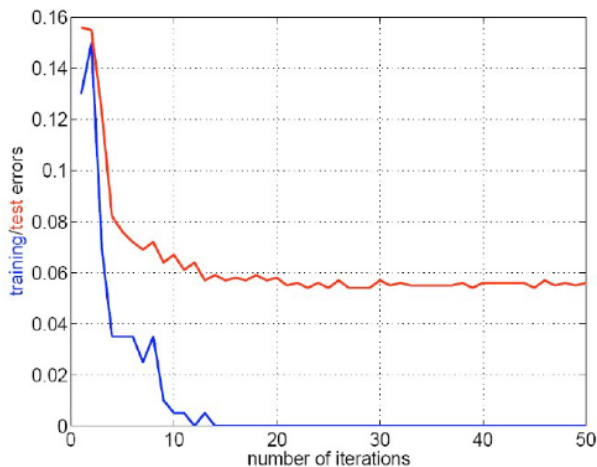
- ▶ Initialize weights $w_n^{(1)} = 1/N$
- ▶ For $m = 1, \dots, M$ (and while $\epsilon_m < 1/2$)
 - ▶ Find weak learner $y_m(x)$ with minimum weighted error

$$\epsilon_m = \sum_{n=1}^N w_n^{(m)} I(y_m(x_n) \neq t_n)$$

- ▶ Set $\alpha_m = \frac{1}{2} \log \frac{1-\epsilon_m}{\epsilon_m}$
 - ▶ Update weights $w_n^{(m+1)} = w_n^{(m)} \exp \{-\alpha_m t_n y_m(x_n)\}$
 - ▶ Normalize weights to sum to one
- ▶ Final classifier is

$$y(x) = \text{sign} \left(\sum_{m=1}^M \alpha_m y_m(x) \right)$$

AdaBoost behavior



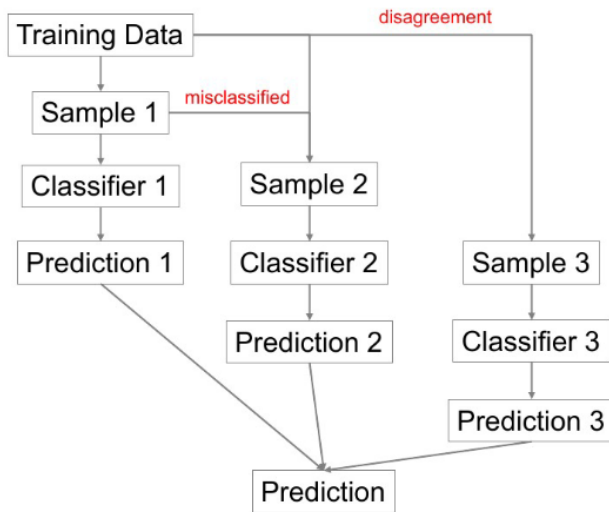
Typical behavior:

- ▶ Test test error error decreases even after training error is flat (even error zero!)
- ▶ **Tends to not over-fit!**

Example: Boosting

- ▶ Boosting trains a sequence of classifiers, so that later classifiers are trained on data points that have been misclassified by previous classifiers
 - ▶ Classifier 1 is trained on a random sample of training data
 - ▶ Classifier 2 is trained on a random sample of training data enriched by 50% of data points misclassified by Classifier 1
 - ▶ Classifier 3 is trained on a random sample of data points from the training data, for which Classifier 1 and Classifier 2 disagree, i.e., predict different classes
- ▶ Class of unseen data point is predict based on a majority vote of Classifiers 1, 2, and 3

Example: Boosting



Example: AdaBoost

- ▶ As we have seen before, AdaBoost, as a popular variant of boosting, trains a sequence of classifiers.
- ▶ It increases the weight of data points that have been misclassified by the ensemble consisting of the already-trained classifiers, i.e., later classifiers learn to correct the mistakes of earlier ones.

AdaBoost — Example

demo/AdaBoost.py

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score

# load data
cars = pd.read_csv('auto-mpg.data.txt', header=None, sep='\s+')

# extract power and weight as data matrix X
X = cars.iloc[:, [3,4]].values

# extract origin as target vector y
y = cars.iloc[:, 7].values

# split into training data (80%) and test data (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# learn decision tree of maximal depth 2
tree = DecisionTreeClassifier(criterion='entropy', max_depth=2)

# bagging classifier based on 10 decision trees
bc = AdaBoostClassifier(base_estimator=tree, n_estimators=30)
bc.fit(X_train, y_train)
bc_y_predicted = bc.predict(X_test)
print(accuracy_score(y_true=y_test, y_pred=bc_y_predicted)) # 0.620253164556962
```


Summary

- ▶ **Information theory** provides a foundation for choosing decision criteria for nodes in decision trees.
- ▶ Majority voting, bagging, and boosting as ensemble methods that **aggregate the predictions of multiple classifiers** to obtain a **more robust classifier**.