MACHINE LEARNING FOR DATA MINING
LECTURE 2-2: NAIVE BAYES CLASSIFICATION

Siamak Sarmady (Urmia University of Technology)
Sebastian Thrun, Katie Malone (Google)

Naive Bayes method and Bayes Rule

- Bayes rule is the base of probabilistic inference.
- The rule was created by Rev. Thomas Bayes. He intended to use the principle to prove the existence of God.
- The word "Naive" refers to the simplistic nature of the algorithm.
- The rule has influenced vast areas of science including statistics and AI.

Classification using Bayes' Rule

- In classification, Bayes' rule is used to calculate the probabilities of the classes.
- Based on the training data (seen items), we select a class that minimizes the expected risk (of selecting the wrong class)

Probability

- **Coin toss:** If we had enough knowledge and time, we could predict the exact outcome.
  - Since we do not have enough knowledge and time, we use probability to make predictions.
- **Unobservable variables:** the piece(s) of knowledge that we do not have access to…
  - **In the case of coin:**
    - The only **observable** variable: the outcome of the toss (a dependent var too)
    - The **unobservable** variables: the amount of force, angel of force, the point of force
- Denoting the unobservable by z and the observable as x:

$$x = f(z)$$

where f(.) is the deterministic function (such as mechanical equation) that defines the outcome from the unobservable pieces of knowledge.

Minimizing Error

- Since we don't have enough knowledge (the unobservable variables) and we cannot model the process this way, we consider the outcome X as a random variable drawn from a probability distribution P(X=x).
- The outcome of the coin toss is either head or tail denoted by X=1 and X=0 so: $X \in \{1,0\}$
- These X are Bernoulli distributed where the parameter of the distribution p_0 is the probability of the outcome "head":

$$P_{(X=1)} = p_0 \quad \text{and} \quad P_{(X=0)} = 1 - P_{(X=1)} = 1 - p_0$$

- **Minimizing error:** if "$p_0 > 0.5$" (e.g. 0.8), our prediction for the next toss will be head. That is because if we choose the more probable case, the probability of error, which is 1 minus the probability of our choice, will be minimum.
  - If the coin is fair (p_0=0.5), we can choose Heads every time or use a fair coin to predict the outcome.

Estimating from Samples

- If we do not know the probability distribution, P(X) and we want to estimate it from a given sample, then we are doing statistics.
- We have a sample X, from the probability distribution of the observables (x^t i.e. the outcome of toss t). The aim is to build an approximator to it, $\hat{p}(x)$, using the sample X. The sample contains the outcome of previous N tosses.

$$\hat{p}_0 = \frac{\#\{tosses\ with\ outcome\ "heads"\}}{\#\{tosses\}}$$

- x^t is 1 if the outcome of toss t is heads and 0 if it is tail
- Given the sample X={1, 1, 1, 0, 1, 0, 0, 1, 1} the estimate will be

$$\hat{p}_0 = \frac{\sum_{t=1}^{N} x^t}{N} = \frac{6}{9}$$

Classification (with Example)

- We want to classify customers as "High risk" and "Low risk" based on previous examples of people who paid back or defaulted a loan.
  - **Observable variables:** let's say we observe "yearly income" and "savings" of customers represented by two random variables X_1 and X_2 (and of course the outcome of loan payback)
  - **Non-observable variables:** other variables that could help us determine the class deterministically
- **The credibility of customer:** is a Bernouli random variable C (i.e. class) conditioned on observable variables $X=[X_1, X_2]^T$
  - C=1 indicates a "High risk" customer
  - C=0 indicates a "Low risk" customer

Classification

- If we know P(C|X1,X2) when a new application arrives with X1=x1 and X2=x2, we can choose or predict the class using:

$$\begin{cases} C = 1, & if\ P(C=1|x_1,x_2) > 0.5 \\ C = 0, & otherwise \end{cases}$$

- Or:

$$\begin{cases} C = 1, & if\ P(C=1|x_1,x_2) > P(C=0|x_1,x_2) \\ C = 0, & otherwise \end{cases}$$

- The probability of error is $1 - \max(P(C=1|x_1,x_2), P(C=0|x_1,x_2))$

- If the observed variables are x=$[x_1, x_2]^T$, then the problem is to calculate P(C|x) using Bayes' rule:

$$P(C|x) = \frac{P(C)\,p(x|C)}{p(x)}$$

Classification using Bayes' Rule

- **Prior probability:** is the knowledge (the probability estimate) we have of the value of C even before looking at the observable x.
  - P(C=1) is called the prior probability that C takes the value 1, regardless of the x value.
  - The prior probabilities of two classes satisfy the following:

$$P(C=0) + P(C=1) = 1$$

- **Class likelihood:** p(x|C) is called the class likelihood and is the conditional probability that an event belonging to C has the associated observation value x
  - In our case p(x1, x2|C=1) is the probability that a "high risk" customer has his X1=x1 and X2=x2. This is obtained from the example data…
- **Evidence probability:** p(x) is the marginal probability that an observation x is seen, regardless of its class, i.e. positive or negative (in the example data)

$$p(x) = \sum p(x,C) = p(x|C=1)\,P(C=1) + p(x|C=0)\,P(C=0)$$

Classification using Bayes' Rule

- **Posterior Probability:** combining the prior and what data tells us using Bayes' rule, we calculate the posterior probability of the concept, P(C|x), after seeing the observation x

$$posterior = \frac{prior \times likelihood}{evidence}$$

- Since posterior is normalized by the evidence, the posteriors sum up to 1

$$P(C=0|x) + P(C=1|x) = 1$$

- After we have the posteriors, we can decide which class is more probable

- The prior and likelihoods are calculated by counting in the example data

Multi-class Classification using Bayes' Rule

- In the general case, we have K mutually exclusive classes; C_i , i=1,…,K

- For example in optical digit recognition the input is the bitmap image (pixel values), and there are 10 classes. The probabilities satisfy:

$$P(C_i) \geq 0 \;\; and \;\; \sum_{i=1}^{K} P(C_i) = 1$$

- $P(x|C_i)$ is the probability of seeing x as the input when it is known to belong to class C_i
- The posterior probability of class C_i can be calculated as

$$P(C_i|x) = \frac{p(x|C_i)P(C_i)}{p(x)} = \frac{p(x|C_i)P(C_i)}{\sum_{k=1}^{K} p(x|C_k)P(C_k)}$$

Multi-class Classification using Bayes' Rule

- For the minimum error, the Bayes' classifier chooses the class with the highest posterior probability; that is, we choose:

$$C_i \quad \text{if} \quad P(C_i|x) = \max_k P(C_k|x)$$

i.e. the k that maximizes $P(C_k|x)$

Example 1:

- There is a cancer type who is observed in 1% of the population i.e. P(c) = 0.01

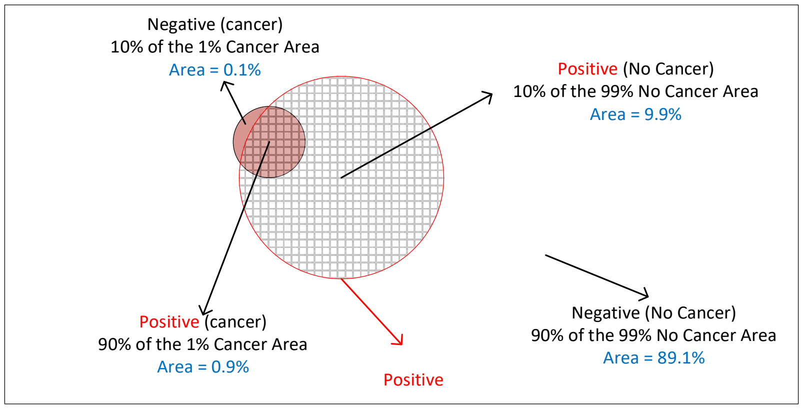
- There is a test for the cancer:

  - **If the person has cancer:** 90% chance that test is positive (sensitivity).
    So there is 10% chance that it won't be detected (false negative).
  - **If the person does not have cancer:** 90% chance that the test is negative (specificity).
    So there is 10% chance that it wrongly detects cancer (false poitive).

- Question: A person takes the test. The answer comes positive. How much is the probability that he has cancer?

Example 1:

- If the person has cancer: 90% chance that test is positive (sensitivity).
  So there is 10% chance that it won't be detected.
- If the person does not have cancer: 90% chance that the test is negative (specificity).
  So there is 10% chance that it wrongly detects cancer.

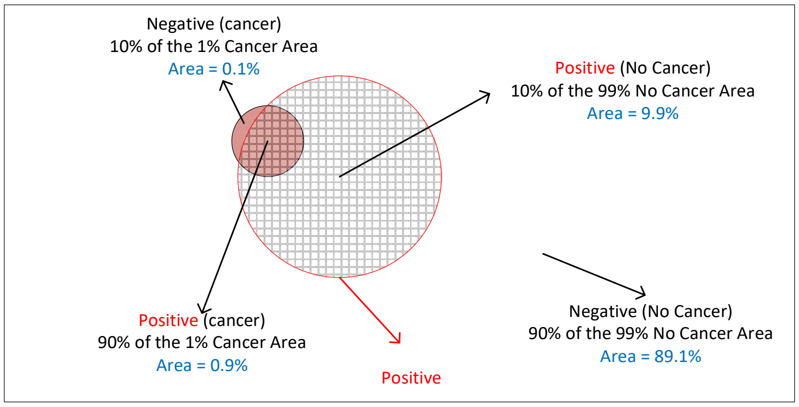


🟥 1% Cancer

Example 1:

- Question: A person takes the test. The answer comes positive. How much is the probability that he has cancer?

  $0.9 \ / \ (9.9 + 0.9) = 8.33$



🟥 1% Cancer

Bayes Rule

- Bayes rule:

$$\boxed{\text{Prior Probability}} \;,\; \boxed{\text{Test Evidence}} \longrightarrow \boxed{\text{Posterior Probability}}$$

- We have some prior probability. We then incorporate some evidence from a test, to arrive at posterior peobability.

- In fact the above rule is calculated like this:

Prior Probability . Test Evidence = Posterior Probability

Bayes Rule

- Bayes rule:

$$\boxed{\text{Prior Probability}} \;\cdot\; \boxed{\text{Test Evidence}} \longrightarrow \boxed{\text{Posterior Probability}}$$

$$P(a) \quad . \quad \underset{\text{(sensitivity)}}{P(b\,|\,a)} \quad = \quad P(a\,|\,b)$$

Bayes rule (the above one is not normalized):

$$P(a|b) = \frac{P(a) \times P(b|a)}{p(b)}$$

Or

$$P(b|a) = \frac{P(b) \times P(a|b)}{p(a)}$$

Example 1

- Prior: P(c) = 0.01 (1%)
  P(Pos | c) = 0.9 (90%)
  P(c') = 0.99
  P(Neg | c') = 0.9 => P(Pos | c') = 0.1

Bayes rule:
P(a|b) = P(a). P(b|a)

Normalizer: P(b)

1. P(c | Pos.) := P(c) . P(Pos. | c) = 0.01 * 0.9 = 0.009

but this is not quite right. It is not normalized. In order to normalize we should also calculate:

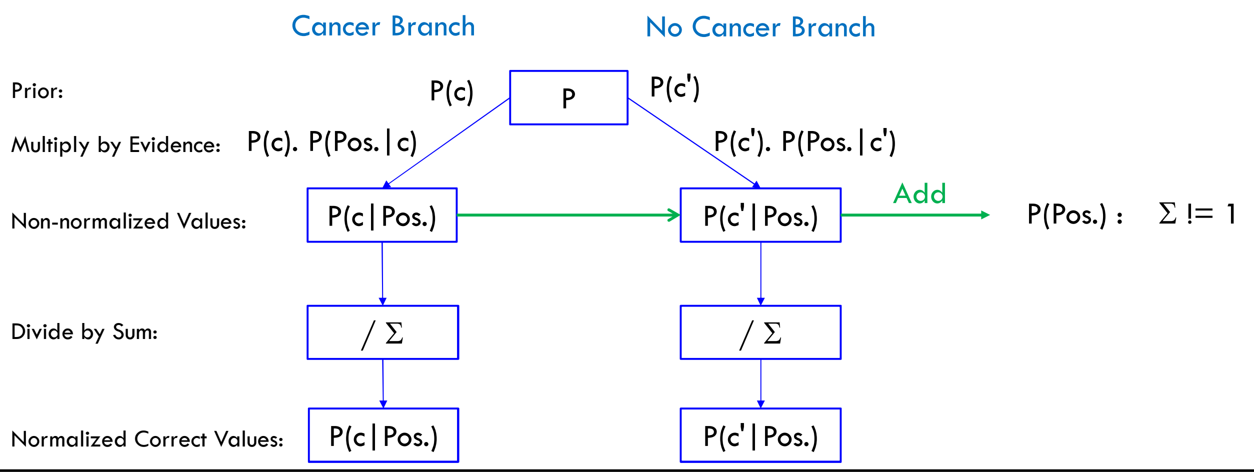
2. P(c' | Pos.) := P(c') . P(Pos. | c') = 0.99 * 0.1 = 0.099

The problem is that the above 1 & 2 do not add up to 1. So we need to normalize them to 1.

- **First:** We add them, the sum is 0.108, and this is in fact P(pos) = P(c | Pos.) + P(c' | Pos.) or the normalizer.
- **Second:** We find correct values for the above probabilities (which add up to 1):
  - P(c | Pos.) = 0.009 / 0.108 = 0.0833
  - P(c' | Pos.) = 0.099 / 0.108 = 0.9167

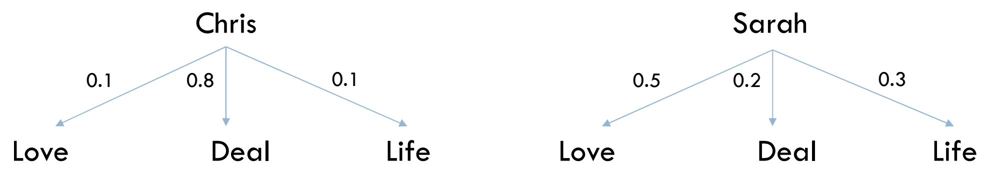
Example 1 (algorithm for bayes rule):

- P(c) = 0.01 Prior
- P(Pos. | c) = 0.9 Sensitivity
- P(Neg.| c')= 0.9 Specificity



Example 2: Text Learning (Learning from Documents) using Naive Bayes

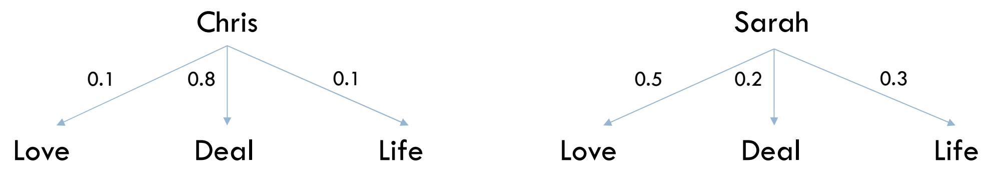
- Suppose we have large number of emails from two person, Chris and Sarah. For the simplicity assume these emails contain only 3 type of words (in reality it could be 30,000).
- The difference between these two series of emails is the frequencies of the words. They use these words with different frequencies.
- Naive Bayes allows to guess who is more likely to be the writer of an email.



- Chris likes to talk more about "Deal" while Sarah mostly likes to talk about "Love"…

Example 2-1:

- Assume these priories: P(Chris) = 0.5, P(Sarah)= 0.5

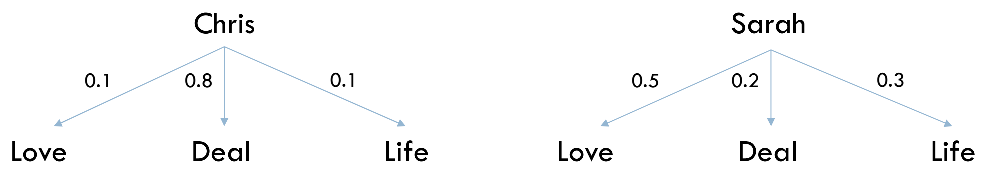


- We have received an email containing this text: "Love Life"
- Who is more likely to be the sender?
- This is relatively easy to guess but…

Example 2-2:

Bayes rule:
P(a|b,c) = P(a). P(b|a). P(c|a)
Normalizer: P(b,c)

- Priories: P(Chris) = 0.5, P(Sarah)= 0.5



- Now assume we have received another email: "Life Deal"
- Who is more likely to be the sender this time? Not as easy as previous case.

- P(Chris| Life, Deal) := P(Chris) * P(Life|Chris) * P(Deal|Chris) = 0.5 * 0.1 * 0.8 = 0.04
- P(Sarah| Life, Deal) := P(Sarah) * P(Life|Sarah) * P(Deal|Sarah) = 0.5 * 0.2 * 0.3 = 0.03

(so higher probability that Chris sent the letter)

Example 2-2:

- However notice that above probabilities are not normalized (do not sum up to 1) and therefore cannot be treated like probability. In order to get correct probabilities we need to normalize them.

- P(Chris| Life, Deal) = 0.04
- P(Sarah| Life, Deal) = 0.03

- Σ = 0.04 + 0.03 = 0.07

- P(Chris| Life, Deal) = 0.04 / 0.07 = 0.571
- P(Sarah| Life, Deal) = 0.03 / 0.07 = 0.428

- Above values are proper probabilities now.

Example 2-3:

- Now, let's assume we received another email with the text "Love Deal". Who is the likely sender.

- Non-normalized Probabilities:
  - P(Chris | Love, Deal) = 0.5 * 0.1 * 0.8 = 0.04
  - P(Sarah | Love, Deal) = 0.5 * 0.5 * 0.2 = 0.05

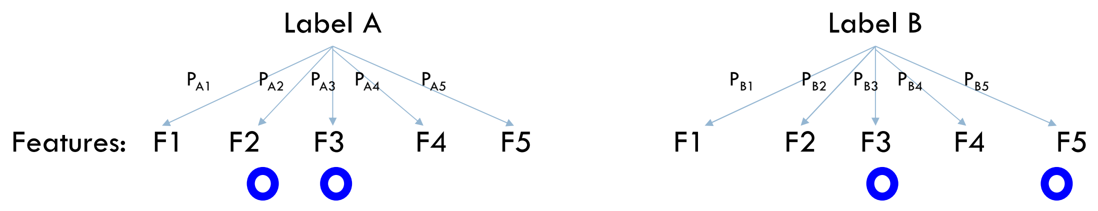
- $\Sigma = 0.04 + 0.05 = 0.09$

- Normalized correct probabilities:
  - P(Chris | Life, Deal) = 0.04 / 0.09 = 0.444
  - P(Sarah | Life, Deal) = 0.05 / 0.09 = 0.555

- So the new email is likely to be sent by Sarah

Naive Bayes

- So, to sum up



- **A:** $P_{(A)} \cdot P_{A2} \cdot P_{A3}$ this is in fact non-normalized of P(A | F2, F3)
- **B:** $P_{(B)} \cdot P_{B3} \cdot P_{B5}$ this is in fact non-normalized of P(B | F3, F5)

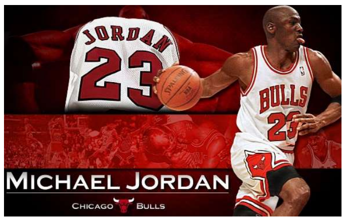
- What comes out of above calculations, is the ratio between the probabilities of Label A and Label B
- If we need probabilities, we need to normalize them (i.e. Divide each to the sum of both).
- So Naive Bayes lets you identify the label (class) which is more likely, by giving it the features (evidence)

Naive Bayes – Why Naive

- Naive Bayes is a powerful method. You can identify the writer of a book, email or text using it.
- One of the reasons it is called "Naive" (simplistic) is that, it ignores one of below things. Guess which one:
  - Words
  - Word Orders
  - Length of Message
- The probability calculation ignores the order of words. If you change the order of words, the meanings change or the text may not make sense at all.
- So the method considers occurrence and frequencies of words, but not the order of them. But still it works well in some situations.
- Another reason is that it always selects by minimizing risk.

Naive Bayes

- Advantages:
  - It is easy to implement and efficient.
  - Can work with a really big feature space (e.g. 20000 words)
- Disadvantages:
- It can break in funny ways. For example when google intially started to work, when people searched for "Chicago Bulls" which is a basketball team, google would show results about "Bulls" (cows) and the city of "Chicago". So things sensitive to order of the features are not very suitable for Naive Bayes method.

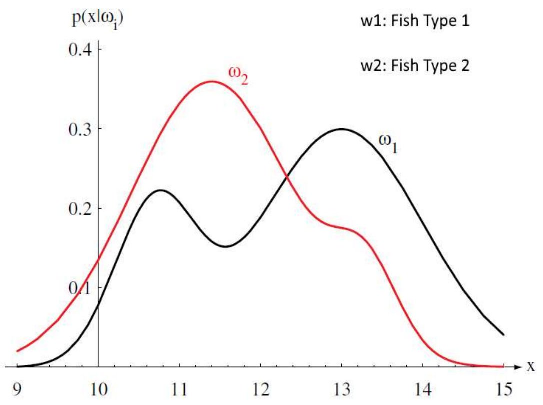


!=

or

Example 3 – Separating fishes based on their length

- An automatic machine should separate 2 types of fish on a conveyer belt. Probability distribution of each fish being of different lengths has been provided in below graph.
- If we know that 70% of fishes are of type 1 and 30% of type 2 (priori probability)
  - if the length of a fish is 11.5, what is the type?
  - If the length is 13, what is the type?



- Assumptions:
  - P(11.5|w1) = 0.17
  - P(11.5|w2) = 0.35
  - P(13|w1) = 0.3
  - P(13|w2) = 0.2

Example 3 – Separating fishes based on their length

- P(w1|11.5) = P(w1) * P(11.5|w1) = 0.7 * 0.17 = 0.12
- P(w2|11.5) = P(w2) * P(11.5|w2) = 0.3 * 0.35 = 0.105

- So the probability of a 11.5 cm fish of being a Type 1 is higher.

- P(w1|13) = P(w1) * P(13|w1) = 0.7 * 0.3 = 0.21
- P(w2|13) = P(w2) * P(13|w2) = 0.3 * 0.2 = 0.06

- Again the probability of a 13 cm fish of being a Type 1 is higher.

Example 4 – Spam Email Identification

- Assume 10% of all emails are spam (priori probability).
- Also assume the distribution of words in spam and non-spam emails are as follows (the actual distribution has much more words)
  - Spam: Rolex 0.4, Winner 0.4, Send 0.2
  - Non-spam: Send 0.4, Exercise 0.3, Winner 0.2, Rolex 0.1
- If we have received 2 emails with the following word distributions, which one is likely to be spam:
  - Email 1: Rolex winner
  - Email 2: Send winner

Example 4 – Spam Email Identification

- Email1:
  - P(spam|Rolex, winner) = P(spam)*P(rolex|spam)*P(winner|spam)
    = 0.1*0.4*0.4 = 0.016 -> Normlaized = 0.47
  - P(!spam|Rolex, winner) = P(!spam)*P(rolex|!spam)*P(winner|!spam)
    = 0.9*0.1*0.2 = 0.018 -> Normlaized = 0.52
- Email2:
  - P(spam|send, winner) = P(spam)*P(send|spam)*P(winner|spam)
    = 0.1*0.2*0.4 = 0.008 -> Normlaized = 0.1
  - P(!spam|send, winner) = P(!spam)*P(send|!spam)*P(winner|!spam)
    = 0.9*0.4*0.2 = 0.072 -> Normlaized = 0.9

Training and Testing Classifiers

Training and Test Data Sets

- Over fitting

Training and Testing Classifiers

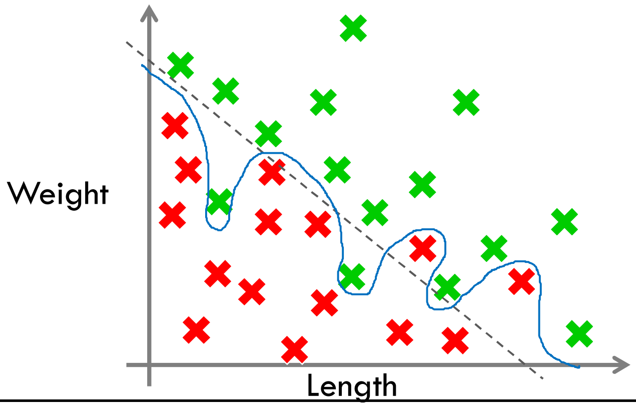
- We do training and evaluation (testing) on two separate data sets. Using an independent "test set" allows to test and evaluate the performance independent from training data.

1. Randomly divide the data (features and labels) to two groups, namely "training set" and "test set".
   - The training set will normally be 80%-90% of data items while test set could be 20%-10% of them.
2. Train the classifier with "training set".
3. Use the features of the "test set" to predict labels using classifier.
4. Now compare the predicted values and the test set labels.
5. Calculate the percentage of correct predictions. The percentage of the correct predictions, determines the performance of the classifier.

Overfitting

Over-fitting: sometimes we might over-fit the classifier to the training data. In that case the classifier will no more generalize properly.

- The classifier has in fact memorized the training set instead of finding the patterns in it.
- It will get 100% accuracy with the training data but it has no clue how to generalize to new data. It will have very poor results with other unseen data (like test data set).



Conclusion

- As we saw, we should not look at classifiers as black boxes.
  - We should understand how they work, so that we can decide whether they are suitable for a problem.
  - In addition knowledge of inner works of a method helps in adjustment of settings and better use of the method and therefore better performance.
  - And also we should test and see how a specific classifier works for a specific problem.
  - Testing should be done with a separate data set which was not seen by the classifier during test phase.

Python and Scikit-Learn Library

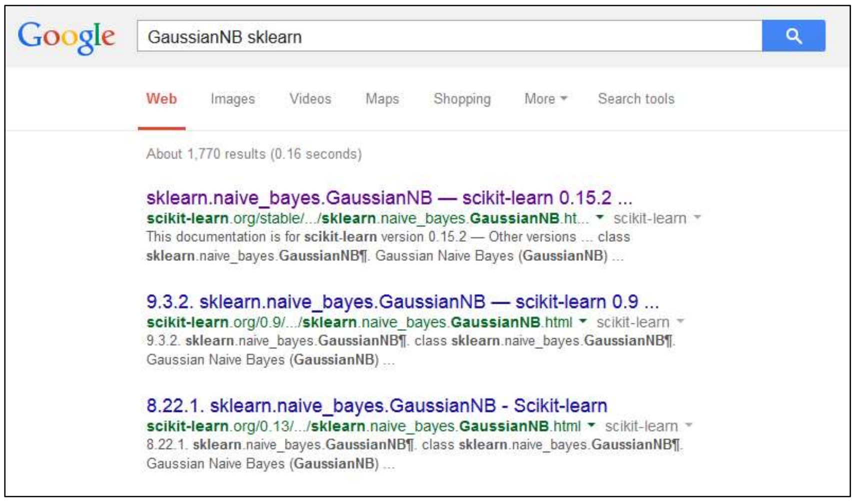
Using Python and Scikit-Learn library

- We use Python version 3.5 because most of the libraries in the field of machine learning have become available for this version.
- We should either install Python and then add sickit-learn (and a few other libraries) to it, or use a Python-Bundle which puts together python and several libraries.
- We will use Winpython which includes several useful libraries…
- The library which we will use for Machine-Learning is called Scikit-Learn

Using Python and Scikit-Learn library

- The website of Scikit-Learn is: http://scikit-learn.org
- The documentation can be found at: http://scikit-learn.org/stable/documentation.html
- Scikit-Learn is also known as "sklearn".
- If you want examples of a specific function just search for "sklearn" and the name of the function.

Using Python and Scikit-Learn library



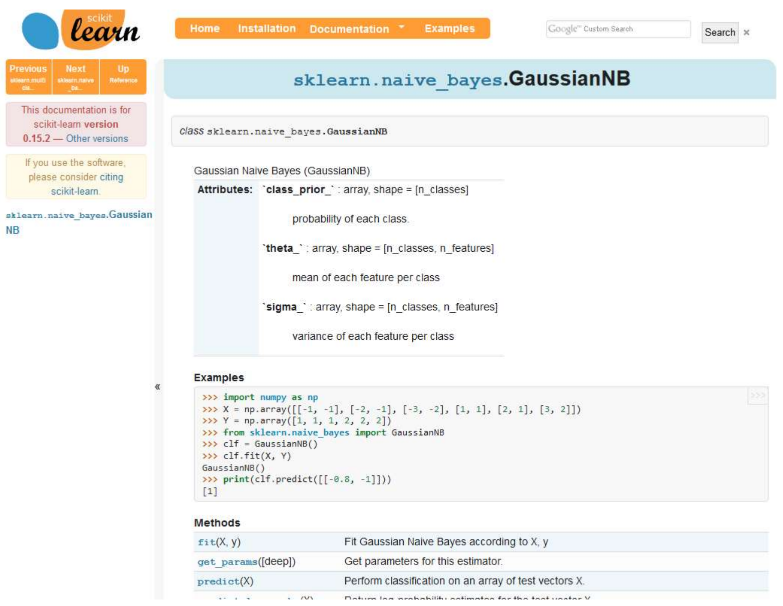
About 1,770 results (0.16 seconds)

sklearn.naive_bayes.GaussianNB — scikit-learn 0.15.2 ...
scikit-learn.org/stable/.../sklearn.naive_bayes.GaussianNB.ht... scikit-learn
This documentation is for scikit-learn version 0.15.2 — Other versions ... class sklearn.naive_bayes.GaussianNB¶. Gaussian Naive Bayes (GaussianNB) ...

9.3.2. sklearn.naive_bayes.GaussianNB — scikit-learn 0.9 ...
scikit-learn.org/0.9/.../sklearn.naive_bayes.GaussianNB.html scikit-learn
9.3.2. sklearn.naive_bayes.GaussianNB¶. class sklearn.naive_bayes.GaussianNB¶. Gaussian Naive Bayes (GaussianNB) ...

8.22.1. sklearn.naive_bayes.GaussianNB - Scikit-learn
scikit-learn.org/0.13/.../sklearn.naive_bayes.GaussianNB.html scikit-learn
8.22.1. sklearn.naive_bayes.GaussianNB¶. class sklearn.naive_bayes.GaussianNB¶. Gaussian Naive Bayes (GaussianNB) ...

Using Python and Scikit-Learn library



This documentation is for scikit-learn **version** 0.15.2 — Other versions

If you use the software, please consider citing scikit-learn.

sklearn.naive_bayes.GaussianNB

sklearn.naive_bayes.GaussianNB

class sklearn.naive_bayes.**GaussianNB**

Gaussian Naive Bayes (GaussianNB)

Attributes: `**class_prior_**` : array, shape = [n_classes]

probability of each class.

`**theta_**` : array, shape = [n_classes, n_features]

mean of each feature per class

`**sigma_**` : array, shape = [n_classes, n_features]

variance of each feature per class

Examples

```
>>> import numpy as np
>>> X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
>>> Y = np.array([1, 1, 1, 2, 2, 2])
>>> from sklearn.naive_bayes import GaussianNB
>>> clf = GaussianNB()
>>> clf.fit(X, Y)
GaussianNB()
>>> print(clf.predict([[-0.8, -1]]))
[1]
```

Methods

fit(X, y)	Fit Gaussian Naive Bayes according to X, y
get_params([deep])	Get parameters for this estimator.
predict(X)	Perform classification on an array of test vectors X.

Sklearn - Example 1

- The example code is as follows:

```
import numpy as np

features_train = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1],[3,3]])
labels_train   = np.array([1, 1, 1, 2, 2, 2])
features_test  = np.array([[-2, -2], [-2, -3], [2, 3], [1, 2]])
labels_test    = np.array([1, 1, 1, 2])

from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
clf.fit(features_train, labels_train)
pred = clf.predict(features_test)

print ("Test labels:       ", labels_test)
print ("Predicted labels: ", pred)

from sklearn.metrics import accuracy_score
print ("Accuracy:          ", accuracy_score(pred, labels_test))

print ("\nPredicted label for ", [-0.8, -1] ," is ", (clf.predict([[-0.8, -1]])))
```

Sklearn - Example 1

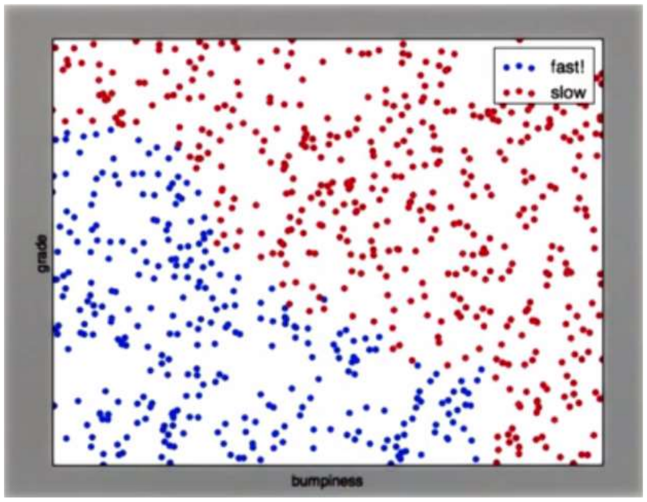
- Put the example code in a file with ".py" extension and run the code this way:

```
F:\mc-code\nb1>python nb11.py
Test labels:        [1 1 1 2]
Predicted labels:  [1 1 2 2]
Accuracy:           0.75

Predicted label for  [-0.8, -1]  is  [1]
F:\mc-code\nb1>
```

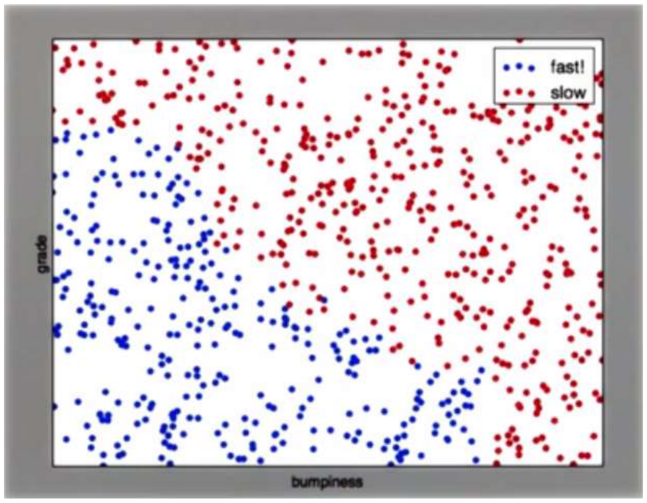
Sklearn - Example 2

- What we are going to do in the 2nd example, is to take a training data set which contains a few hundred data points for the car speed adjustment example.



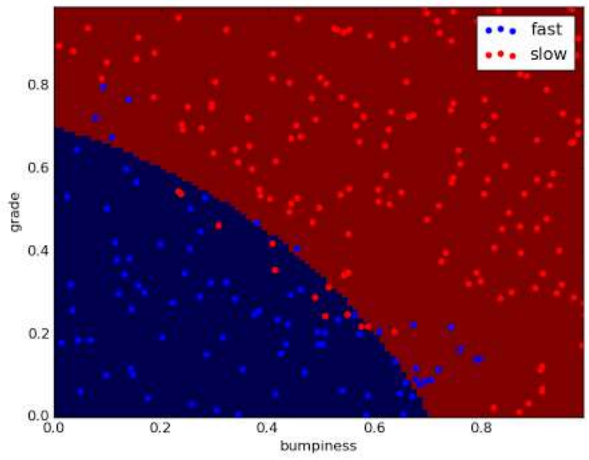
Using Scikit Learn Python Library

- We then try to find a decision boundry for the data (i.e. Learning phase)



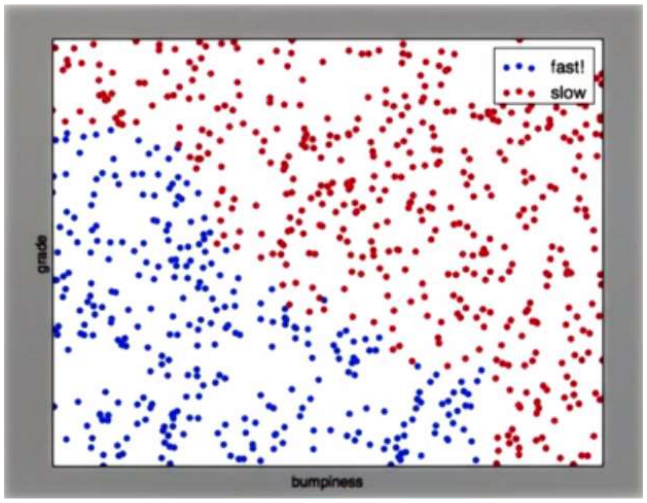
Using Scikit Learn Python Library

- The classes painted with different colors



Using Scikit Learn Python Library

- We can then use the boundry to decide which class an unseen arbitrary data belongs to (generalization step).


fast!
slow
grade
bumpiness