Data Science: Principles and Practice
Lecture 3: Classification

Ekaterina Kochmar

13 November 2019

Recap: Supervised Learning

Dataset: $\{< x^{(1)}, y^{(1)} >, < x^{(2)}, y^{(2)} >, ..., < x^{(m)}, y^{(m)} >\}$

Input features: $(x_1^{(i)}, x_2^{(i)}, ..., x_n^{(i)})$

Known (desired) outputs: $y^{(1)}, y^{(2)}, ..., y^{(m)}$

Our goal: Learn the mapping $f : X \rightarrow Y$
such that $y^{(i)} = f(x^{(i)})$ for all $i = 1, 2, ..., m$

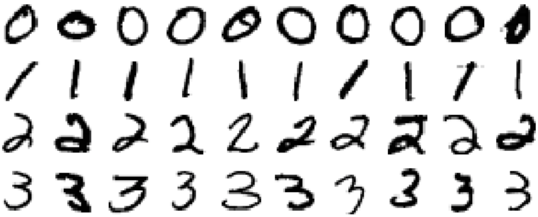
Strategy: Learn the function on the training set,
use to predict $\hat{y}^{(j)} = f(x^{(j)})$ for all x_j in the test set

Last time we looked into regression tasks, today – **classification**

Recap: Regression vs. Classification

Regression tasks: the desired labels are continuous
Examples: House size, age, income → price
Weather conditions, time → number of rented bikes

Classification tasks: the desired labels are discrete
Examples: Pixel distribution in the image → digit label
Word distribution in movie reviews → sentiment (pos/neg/neut) label



Outline

Binary classification

Case study

Let's start with a simpler case – binary classification
Task: Sentiment analysis in movie reviews (Part IA CST Machine Learning and Real-world Data)
Data: $m \times n$ matrix X with m reviews and n features (words)
Labels: $y \in (0, 1)$ with 0 for *neg* and 1 for *pos*

Approach

`Naive Bayes` classifier:

- relies on probabilistic assumptions about the data
- makes "naive" independence assumption about the features
- fast and scalable compared to more sophisticated methods
- competitive results on a number of real-world tasks, despite over-simplistic assumptions

Binary classification with Naive Bayes

Prediction

$$\hat{y}^{(i)} = argmax_{c \in (0,1)} p(y = c|x^{(i)}) = \begin{cases} 1, & \text{if } p(y = 1|x^{(i)}) > p(y = 0|x^{(i)}) \\ 0, & \text{otherwise} \end{cases}$$

where $x^{(i)} = (f_1^{(i)}, ..., f_n^{(i)})$

Flipping the conditions

$$\hat{p}(y = c|x^{(i)}) = \frac{p(c)p(x^{(i)}|c)}{p(x^{(i)})}$$

where $p(c)$ is the *prior*, $p(x^{(i)}|c)$ is *likelihood*, $p(x^{(i)})$ is *evidence* (note: it's irrelevant for the *argmax* estimation), and $p(y = c|x^{(i)})$ is the *posterior*

Binary classification with Naive Bayes

"Naive" independence assumption

$$p(f_1^{(i)}, ..., f_n^{(i)}|y) \approx \prod_{k=1}^{n} p(f_k^{(i)}|y)$$

Revised estimation

$$\hat{y}^{(i)} = argmax_y p(y|x^{(i)}) = argmax_y p(y) \prod_{k=1}^{n} p(f_k^{(i)}|y)$$

where probabilities can be estimated from the training data using *maximum a posteriori* estimate

`Naive Bayes` models typically differ with respect to the assumptions about the distribution of features $p(x^{(i)}|y)$. Commonly used models: Gaussian NB, Multinomial NB, Bernoulli NB.[a]

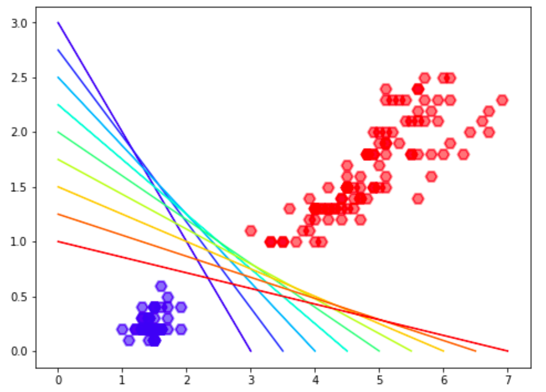
---

[a] Recommended reading: A. McCallum and K. Nigam (1998). *A comparison of event models for Naive Bayes text classification.* `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.46.1529`

Linearly separable data

Example

Linear ML models, or the models that try to build a linear separation boundary between the classes, are well-suited for such data. Examples: Logistic Regression, `Perceptron`, `Support Vector Machines`



Logistic Regression

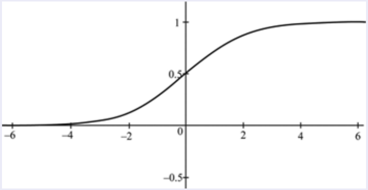
Logistic Regression vs Linear Regression

- Last time we looked into `Linear Regression` and learned how to use it to output a continuous value
- Despite the name, `Logistic Regression` outputs a *discrete value*, i.e. it is used for classification
- `Logistic Regression` estimates whether the probability of an instance i belonging to class c is greater than 0.5. If it is, the item is classified a c; otherwise as $\neg c$

Logistic Regression

- Estimate $w \cdot X$ as before, where w is the weight vector $(w_0, w_1, ..., w_n)$
- Apply a *sigmoid* function to the result: $\hat{p} = \sigma(w \cdot X)$, where $\sigma(t) = \frac{1}{1+exp(-t)}$
- Prediction step:

$$\hat{y} = \begin{cases} 1, & \text{if } \hat{p} \geq 0.5 \\ 0, & \text{otherwise} \end{cases} \qquad \text{or: } \hat{y} = \begin{cases} 1, & \text{if } t \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

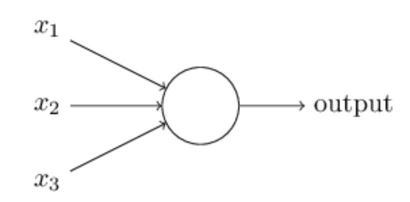


Logistic Regression

Training

- Learning objective: learn weights w such that prediction $\hat{p}$ has a high positive value for $y = 1$ and high negative value for $y = 0$
- The following cost function answers this objective:
  $$c(w) = \begin{cases} -log(\hat{p}), & \text{if } y = 1 \\ -log(1 - \hat{p}), & \text{if } y = 0 \end{cases}$$
- Log-loss cost function:
  $J(w) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}log(\hat{p}^{(i)}) + (1 - y^{(i)})log(1 - \hat{p}^{(i)})]$
- No closed form solution for w that minimises the cost function, but since the function is convex, Gradient Descent (refer to the previous lecture) can be used to find the optimal weights

Single-layer perceptron



$$\hat{y}^{(i)} = \begin{cases} 1, & \text{if } w \cdot x^{(i)} + b > 0 \\ 0, & \text{otherwise} \end{cases}$$

where $w \cdot x^{(i)}$ is the dot product of weight vector w and the feature vector $x^{(i)}$ for the instance i, $\sum_{j=1}^{n} w_j x_j^{(i)}$, and b is the bias term

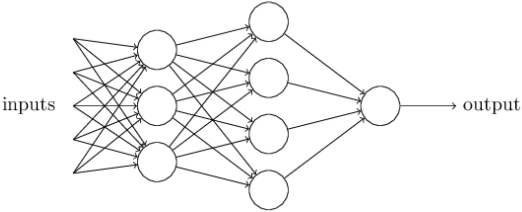
Single-layer perceptron

Training

1. **Initialisation**: Initialise the weights $w = (w_1, ..., w_j)$ and the bias $b = w_0$ to some value (e.g., 0 or some other small value)
2. **Estimation** at time t for each instance i:
$\hat{y}^{(i)} = f(w(t) \cdot x^{(i)}) = f(w_0(t) + w_1(t)x_1^{(i)} + ... + w_n(t)x_n^{(i)})$
3. **Update** for the weights at time $(t+1)$ for instance i and each feature $0 \leq j \leq n$: $w_j(t+1) = w_j(t) + r(y^{(i)} - \hat{y}^{(i)})x_j^{(i)}$, where r is a predefined learning rate
4. **Stopping criteria**: convergence to an error below a predefined threshold γ, or after a predefined number of iterations $t \leq T$.

Single-layer perceptron

- If the data is linearly separable, the perceptron algorithm is guaranteed to converge
- If the data is not linearly separable, the perceptron will never be able to find a solution to separate the classes in the training data
- A single layer perceptron is a simple linear classifier. Often used to illustrate the simplest feedforward neural network. Multilayer perceptrons combine multiple layers and use non-linear activation functions, which makes them capable to classify data that is not linearly separable (more on this in later lectures)

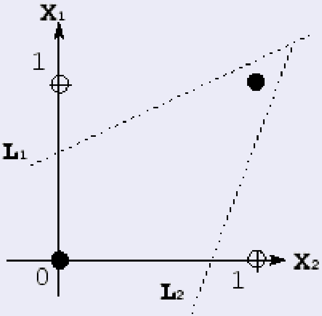


Non-linearly separable data

The classic example: XOR problem

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

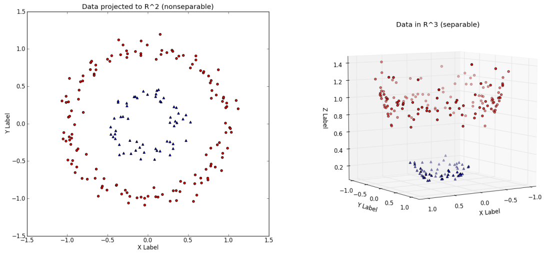
$$Y = X_1 \oplus X_2$$



Non-linearly separable data

Data transformations for non-linearly separable data

- **Actual (raw) data**: two classes non-linearly separable (on the left)
- **Objective**: transform the data using additional dimensions such that it becomes possible to separate the classes linearly (on the right)
- **Method**: data transformations / feature maps that transform the data into higher dimensional space (e.g., *kernel trick*)



Non-linearly separable data

Toy example

- Suppose a non-linearly separable classes as above: e.g., instances $x^{(0)} = (0.5, 0.5)$ and $x^{(1)} = (-1, -1)$
- Consider using a square function: $x^{(0)} \rightarrow x'^{(0)} = (0.25, 0.25)$ and $x^{(1)} \rightarrow x'^{(1)} = (1, 1)$
- With the new data representation, the instances of class 0 (blue) end up on the left, and the instances of class 1 (red) end up on the right
- *Kernel trick* and feature maps allow us to cast the original data into a higher dimensional data: e.g. $(x, y) \rightarrow (x^2, xy, y^2)$

Performance measures

Accuracy

- **Task**: suppose you select a digit in the handwritten digits dataset (e.g., 5), and perform a binary classification task of detecting 5 vs. $\neg$5 in a balanced dataset of 10 digits
- **Evaluation**: the most straightforward way to evaluate is to calculate the proportion of correct predictions:
  $ACC = \frac{num(\hat{y}==y)}{num(\hat{y}==y)+num(\hat{y}!=y)}$
- **Results**: suppose that you get an accuracy of 91%. Is this a good accuracy score?

Performance measures

What accuracy score is missing

- If the classifier always predicts $\neg 5$ (i.e., does nothing), the accuracy will be $ACC = 90\%$
- It's unclear what exactly the classifier gets wrong

Confusion matrix

	predicted $\neg 5$	predicted 5
actual $\neg 5$	TN	FP
actual 5	FN	TP

- *True negatives* (TN) – actual instances of $\neg 5$ correctly classified as $\neg 5$
- *False negatives* (FN) – actual instances of 5 missed by the classifier
- *True positives* (TP) – actual instances of 5 correctly classified as 5
- *False positives* (FP) – actual instances of $\neg 5$ misclassified as 5

Performance measures

Measures

- Accuracy: $ACC = \frac{TP+TN}{TP+TN+FP+FN}$
- Precision: $P = \frac{TP}{TP+FP}$
- Recall: $R = \frac{TP}{TP+FN}$
- F_1-score: $F_1 = 2 \times \frac{P \times R}{P+R}$ $[F_\beta = (1+\beta^2) \times \frac{P \times R}{\beta^2 \times P + R}]$

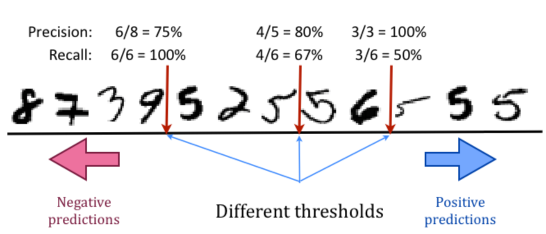
Precision-recall trade-off

Some tasks require higher recall and some higher precision, e.g.:

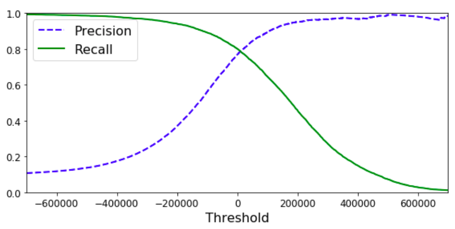
- Detection of a potentially cancerous case that needs further tests?
- Detection of suspicious activity on a credit card? Automated blocking?
- Automated change of drug dosage for a hospital patient?
- Automated spell/grammar checker correction?
- Search for related web-pages online?

Performance measures

Confidence threshold



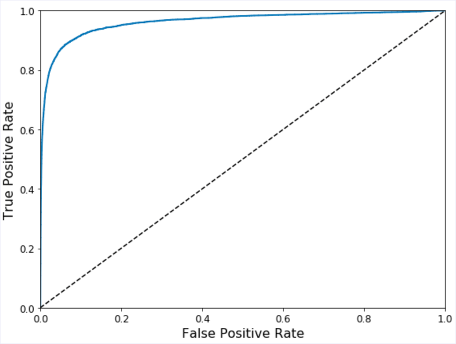
Precision-recall curve



Performance measures

Receiver Operating Characteristic (ROC)

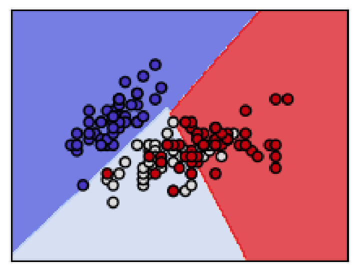
- *Specificity* $= \frac{TN}{TN+FP}$
- *False positive rate (FPR) / fall-out / probability of false alarm* $= (1 - \textit{specificity})$
- *True positive rate (TPR) / sensitivity / probability of detection = recall*



Multi-class classification

From binary to multi-class

- Directly classified with some algorithms: e.g., `Naive Bayes` – simply output the most probable class
- Linear classifiers: one of two strategies:
  1. *one-vs-all* (*OvA*) / *one-vs-rest* (*OvR*): n binary classifiers trained to detect one class each (e.g. 10 binary digit detectors); output the class with the highest score
  2. *one-vs-one* (*OvO*): $\frac{N(N-1)}{2}$ binary class-vs-class classifiers (e.g. 45 binary digit-vs-digit classifiers); output class that wins most



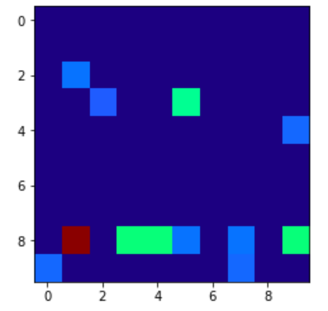
Multi-class classification

Error analysis

Confusion matrix:

```
array([[36,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 36,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  1, 34,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  1, 34,  0,  2,  0,  0,  0,  0],
       [ 0,  0,  0,  0, 35,  0,  0,  0,  0,  1],
       [ 0,  0,  0,  0,  0, 37,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0, 36,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0, 36,  0,  0],
       [ 0,  4,  0,  2,  2,  1,  0,  1, 23,  2],
       [ 1,  0,  0,  0,  0,  0,  0,  1,  0, 34]])
```

Confusions heatmap:



Practical 2: Classification

Your task

- two datasets: iris flower dataset (150 samples, 3 classes, 4 features), and hand-written digits dataset ($\approx 1.8K$ samples, 10 classes, 64 features)
- learn about binary and multi-class classification in practice
- investigate whether data is linearly separable and what to do when it is not
- apply 3 classifiers discussed in this lecture
- focus on evaluation of the classifiers
- one dataset is used to illustrate the ML techniques; your task is to implement all the above steps for the other one