Data mining

Giacomo Fantoni
Telegram: @GiacomoFantoni
Github: https://github.com/giacThePhantom/DataMining

October 15, 2021

Contents

Chapter 1

Introduction

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T as measured by P, improves with experience E.
From that it is understood that machine learning is a form of inductive learning: it generalize from examples to a concept. There is no certainty of correctness.

1.1 Formalization of a machine learning problem

1.1.1 Components of a machine learning problem

The components of a machine learning problem are:

- Task to be addressed by the system.
- Performance measure to evaluate the learned system.
- Training experience to train the learning system.

1.1.2 Designing a machine learning system

The designing of a machine learning system can be described in a process that consist of different phases:

1. Formalize the learning task.
2. Collect data.
3. Extract features.
4. Choose class of learning models.
5. Train model.
6. Evaluate model.

1.1.2.1 Formalize the learning task

To formalize the learning task means to define the task that should be addressed by learning system. This type of task, or learning problem, is often composed of a number of related tasks, sub-problems or side-tasks. It is also needed an appropriate performance measure for evaluating the learned system.

1.1.2.2 Collect data

To collect the data means to collect a set of training example in machine readable format. Data collection is often the most cumbersome part of the process, implying manual intervention especially in labelling examples for supervised learning. Recent approaches to the problem of data labelling try to make use of the availability of unlabelled data (semi-supervised learning it tries to learn using both supervised and unsupervised examples).

1.1.2.3 Extract features

A relevant set of features need to be extracted from the data in order to provide inputs to the learning system. Prior knowledge is usually necessary in order to choose the appropriate features for the task in mind. Extracting too few features can miss relevant information preventing the system from learning the task with reasonable performance. Extracting too many feature like can make the learning problem harder and require a number of examples greater than those available for training. Another problem arises when considering noisy features. It is noticeable that there is a need to choose the correct number and type of feature to permit a correct and efficient solution.

1.1.2.4 Choose class of learning models

Every problem has a class of learning model that is able to learn it best. A simple model like a linear classifier is easy to train but insufficient for non linearly separable data. A too complex model can memorize noise in training data failing to generalize to new examples. The algorithm need not to optimize but it needs to generalize, there can be outliers or labelling error. The more complex the model the more it will tend to overfit training noise.

1.1.2.5 Train model

Training a model implies searching though the space of possible models given the chosen model class. Such search typically aims at fitting the available training examples well according to the chosen performance measure. However the learned model should perform well on unseen data (generalization) and not simply memorize training examples (overfitting). Different techniques can be used to improve generalization, usually by trading off model complexity with training set fitting.

1.1.2.6 Evaluate model

The learned model is evaluated according to its ability to generalize to unseen examples. These example are collected in a test set. Evaluation can provide insights into the model weaknesses and suggest directions for refining and modifying it. Evaluation can imply comparing different models or learners in order to decide the best performing one. Statistical significance of observed differences between performance of different models should be assessed with appropriate statistical tests.

1.2 Learning settings

1.2.1 Supervised learning

The learner is provided with a set of inputs/output pairs $(x_i, y_i) \in X \times Y$. The learned model $f : x \to Y$ should map input examples into their output. A domain expert is typically involved in labelling input examples with output examples in the training set.

1.2.1.1 Tasks

1.2.1.1.1 Classification

- Binary: assign an example to one of two possible classes often a positive and a negative one.
- Multiclass: assign an example to one of $n > 2$ possible classes.
- Multilabel: assign an example to a subset $m \leq n$ of the possible classes.

1.2.1.1.2 Regression Assign a real value to an example.

1.2.1.1.3 Ordinal regression or ranking Order a set of examples according to their relative importance or quality with respect to the class.

1.2.2 Unsupervised learning

The learner is provided a set of input examples $x_i \in X$ with no labelling information. The learner models training examples, for examples clustering them together into clusters according to their similarity.

1.2.2.1 Tasks

1.2.2.1.1 Dimensionality reduction Reduce dimensionality of the data maintaining as much information as possible.

1.2.2.1.2 Clustering Cluster data into homogeneous groups according to their similarity.

1.2.2.1.3 Novelty detection Detect novel examples which differ from the distribution of a certain set of data.

1.2.3 Semi-supervised learning

The learner is provided with a set of input output pairs $(x_i, y_i) \in X \times Y$. A typically much bigger additional set of unlabelled examples $x_i \in X$ is also provided. Like in supervised learning the learned model $f : X \to Y$ should map input examples into their output. Unlabelled data can be exploited to improve performance, by forcing the model to produce similar outputs for similar inputs, or by allowing to learn a better representation of examples.

1.2.4 Reinforcement learning

The learner is provided a set of possible states S and for each state a set of possible actions A moving it to a next state. In performing action a from state s the learner is provided an immediate reward $r(s, a)$. The task is to learn a policy allowing to choose for each state s the action a maximizing the overall reward. The learner has to deal with problems of delayed reward coming from future moves and trade-off between exploitation and exploration. Typical application include moving policies for robots and sequential scheduling problems in general.

1.3 Probabilistic reasoning

Probabilistic reasoning is the reasoning in presence of uncertainty. It evaluates the effect of a certain piece of evidence on other related variables. It estimates probabilities and relations between variables from a set of informations. They depends on variables and their relations.

1.4 Choice of learning algorithms

1.4.1 Based on information available

- Full knowledge of probability distributions of data: Bayesian decision theory.
- Form of probabilities known, parameters unknown: parameter estimation from training data.
- Form of probabilities unknown, training examples available: discriminative methods: do not model input data, learn a function predicting the desired output given the input.
- Form of probabilities unknown, training examples unavailable: unsupervised methods, cluster examples by similarity.

Chapter 2

Decision trees learning

2.1 Introduction

Decision trees tend to be interpretable, it is easy to see the reason for a certain decision. They represent a disjunction of conjunctions of constraints over attribute values. Each path from the root to a leaf is a conjunction of the constraints specified in the nodes along it: they can be seen as a disjunctive normal formula. In this way every class can be written as a DNF. The leaf contains the label to be assigned to instances reaching it. The disjunction of all paths is the logical formala expressed by the tree.

2.1.1 Appropriate problems for decision trees

The class of problems that can be solved by decision trees are:

- Binary or multiclass classification with an extension to regression (with a linear regression as the leaf).
- Instances represented as attribute-value pairs.
- Different explanations for the concept are possible (disjunction).
- Some instances have missing attributes, its dealing done with probabilistic models.
- There is need for an interpretable explanation of the output. In fact the main reason for using decision tress is interpretability.

2.2 Learning decision trees

2.2.1 Greedy top-down strategy

For each node, starting from the root with full training set:

1. Choose best attribute to be evaluated.
2. Add a child for each attribute value.
3. Split node training set into children according to value of chosen attribute.

4. Stop splitting a node if it contains examples from a single class or there are no more attributes to test.

It is also known as the divide et impera approach.

2.2.2 Choosing the best attribute

A measure to choose the attribute is entropy. It measures the amount of information contained in a collection of instances S which can take a number c of possible values:

$$H(s) = -\sum_{i=1}^{c} p_i \log_2 p_i$$

Where p_i is the fraction of S taking value i. In our case instances are training examples and values are class labels. The entropy of a set of labelled examples measures its label's inhomogeneity.

2.2.2.1 Information gain

Expected reduction in entropy obtained by partitioning a set S according to the value of a certain attribute A.

$$IG(S, A) = H(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} H(S_V)$$

Where $Values(A)$ is the set of possible values taken by A and S_v is the subset of S taking value v at attribute A. The second term represents the sum of entropies of subsets of examples obtained partitioning over A values, weighted by their respective sizes. An attribute with high information gain tends to produce homogeneous groups in terms of labels, thus favouring their classification. The idea is to use the greedy strategy in which at each choice it is chosen the attribute with the maximum information gain.

2.3 Issues in decision tree learning

2.3.1 Overfitting avoidance

Requiring that each leaf has only examples of a certain class can lead to very complex trees. A complex tree can easily overfit the training set, incorporating random regularities not representative of the full distribution, or noise in the data. It is possible to accept impure leaves, assigning them the label of the majority of their training examples. Two possible strategies to prune a decision tree are:

- Pre-pruning; decide whether to stop splitting a node even if it contains training examples with different labels.
- Post-pruning: learn a full tree and then prune it.

2.3.2 Post-pruning

In post-pruning you expand the tree and then you prune it. It is introduced the validation set.

1. For each node in the tree evaluate the performance on the validation set when removing the subtree rooted at it.

2. If all node removals worsen performance, stop.
3. Choose the node whose removal has the best performance inprovement.
4. Replace the subtree rooted at it with a leaf.
5. Assign to the leaf the majority label of all examples in the subtree.
6. Return to 1.

2.3.3 Dealing with continues valued attributes

Continuous valued attributes need to be discretized in order to be used in internal node tests. Discretization threshold can be chosen in order to maximize the attribute quality criterion.

1. Examples are sorted according to their continuous attribute values.
2. For each pair of successive examples having different labels, a candidate threshold is placed as the average of the two attribute values.
3. For each candidate threshold the infogain achieved splitting examples according to it is computed.
4. The threshold producing the higher infogain is used to discretize the attribute.

2.3.4 Alternative attribute test measures

The information gain criterion tends to prefer attributes with a large number of possible values. As an extreme the unique ID of each examples is an attribute perfectly splitting the data into singletons, but it will be no use on new examples. A measure of such spread is the entropy of the dataset with respect to the attribute value instead of the class value:

$$H_A(S) = -\sum_{v \in Values(A)} \frac{|S_V|}{|S|} \log_2 \frac{|S_V|}{|S|}$$

The gain ration measures downweights the information gain by such attribute value entropy:

$$IGR(S, A) = \frac{IG(S, A)}{H_A(S)}$$

2.3.5 Handling attributes with missing values

Assume example x with class $c(x)$ has missing value for attribute A. When attribute A is to be tested at node n:

- Simple solution: Assign to x the most common attribute values among examples in n or the most common of examples in n with class $c(x)$.
- Complex solution: Propagate x to each of the children of n with a fractional value equal to the proportion of examples with the corresponding attribute value.

The complex solution implies that at test time, for each candidate class, all fractions of the test example which reached a leaf with that class are summed, and the example is assigned the class with highest overall value.

2.4 Random forest

Random forests are an ensemble of decision trees. An ensemble is a method by which predictions are given by a set of predictors and is taken the prediction most represented. They improve stability and accuracy of the predictions. Random forests are effective and one of the methods of choice in case of tabular data.

2.4.1 Training

To train a random forest:

1. Given a training set of N examples, sample N examples with replacement.
2. Train a decision tree on the sample, selecting at each node m features at random among which to choose the best one.
3. Repeat the first two step M times in order to generate a forest of M trees.

2.4.2 Testing

To test a random forest:

1. Test the example with each tree in the forest.
2. Return the majority class among the predictions.

Chapter 3

K-nearest neighbours

3.1 Introduction

The K-nearest neighbours is an algorithm that, given a training set represented as a vector of features, gives the label for a new sample as label of the majority of the K nearest sample in the training set.

3.2 Measuring the instance between instances

3.2.1 Metric or distance definition

Given a set $\mathcal{X}$ a function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$ is a metric for $\mathcal{X}$ if for any $x, y, z \in \mathcal{X}$ the following properties are satisfied:

- Reflexivity $d(x, y) = 0 \Leftrightarrow x = y$.
- Symmetry $d(x, y) = d(y, x)$.
- Triangle inequality $d(x, y) + d(y, z) \geq d(x, z)$.

3.2.2 Euclidean distance

The euclidean distance in $\mathbb{R}^n$ is:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

3.3 Algorithms

3.3.1 Classification

: `Knn-Classification()`

foreach *test examples* x **do**
  foreach *training examples* (x_i, y_i) **do**
    compute distance $d(x, x_i)$
  select k-nearest neighbours of x
  %`return class of` x `as majority class among neighbours`
  return $\arg\max_{x_y} \sum_{i=1}^{k} \delta(y, y_i)$

3.3.2 Regression

: `Knn-Regression()`

foreach *test examples* x **do**
  foreach *training examples* (x_i, y_i) **do**
    compute distance $d(x, x_i)$
  select k-nearest neighbours of x
  %`return the average output value among neighbours`
  return $\frac{1}{k} \sum_{i=1}^{k} y_i$

3.4 Characteristics

- Instance-based learning: the model used for prediction is calibrated for the test example to be processed.
- Lazy learning: the computation is mostly deferred to the classification phase.
- Local learner: assumes prediction should mainly influenced by nearby instances.
- Uniform feature weighting: all feature are uniformly weighted in computing dinstances.

3.5 Distance weighted k-nearest neighbour

The distance weighted k-nearest neighbour is a variant of the classic k-nearest neighbour in which the distance is weighted. The weight of a point is calculated as:

$$w_i = \frac{1}{d(x, x_i)}$$

The class is decided for classification according to the formula:

$$\arg\max_{x_y} \sum_{i=1}^{k} w_i \delta(y, y_i)$$

For regression the formula is instead:

$$\frac{\sum_{i=1}^{k} w_i y_i}{\sum_{i=1}^{k} w_i}$$

Chapter 4

Linear algebra

4.1 Vector space

A set $\mathcal{X}$ is called a vector space over $\mathbb{R}$ if addition and scalar multiplication are defined and satisfy for all $x, y, z \in \mathcal{X}$ and $\lambda, \mu \in \mathcal{R}$:

- Addition:
  - Association: $x + (y + z) = (x + y) + z$.
  - Commutation: $x + y = y + x$.
  - There is an identity element: $\exists 0 \in \mathcal{X} : x + 0 = x$.
  - There is an inverse element: $\forall x \in \mathcal{X} \exists x' \in \mathcal{X} : x + x' = 0$.
- Scalar multiplication:
  - Is distributive over elements: $\lambda(x + y) = \lambda x + \lambda y$.
  - Is distributive over scalars: $(\lambda + \mu)x = \lambda x + \mu x$.
  - Is associative over scalars: $\lambda(\mu x) = (\lambda\mu)x$.
  - There is an identity element: $\exists 1 \in \mathbb{R} : 1x = x$.

4.1.1 Properties and operations

4.1.1.1 Subspace

A subspace is any non-empty subset of $\mathcal{X}$ being itself a vector space.

4.1.1.2 Linear combination

Given $\lambda_i \in \mathbb{R} \wedge x_i \in \mathcal{X}$, a linear combination is:

$$\sum_{i=1}^{n} \lambda_i x_i$$

4.1.1.3 Span

The span of vectors $x_1, \ldots, x_n$ is defined as the set of their linear combination:

$$\{\sum_{i=1}^{n} \lambda_i x_i, \lambda_i \in \mathbb{R}\}$$

4.1.1.4 Linear independence

A set of vector x_i is linearly independent if none of them can be written as a linear combination of the others.

4.1.2 Basis

A set of vectors x_i is a basis for $\mathcal{X}$ if any element in $\mathcal{X}$ can be uniquely written as a linear combination of vectors x_j. The vectors x_j need to be linearly independent. All bases of $\mathcal{X}$ have the same number of elements, called the dimension of the vector space.

4.2 Matrices

4.2.1 Linear maps

Given two vector spaces $\mathcal{X}$ and $\mathcal{Z}$ a function $f : \mathcal{X} \to \mathcal{Z}$ is a linear map if $\forall x, y \in \mathcal{X} \lambda \in \mathbb{R}$:

- $f(x + y) = f(x) + f(y)$.
- $f(\lambda x) = \lambda f(x)$.

4.2.2 Linear maps as matrices

A linear map between two finite dimensional spaces $\mathcal{X}$ and $\mathcal{Z}$ of dimension n and m can always be written as a matrix. Let $\{x_1, \ldots, x_n\}$ and $\{z_1, \ldots, z_m\}$ be some bases for $\mathcal{X}$ and $\mathcal{Z}$ respectively. For any $x \in \mathcal{X}$:

$$f(x) = f(\sum_{i=1}^{n} \lambda_i x_i) = \sum_{i=1}^{n} \lambda_i f(x_i)$$
$$f(x_i) = \sum_{j=1}^{m} a_{ji}^m a_{ij} z_j$$
$$f(x) = \sum_{i=1}^{n}\sum_{j=1}^{m} \lambda_i a_{ji} z_j = \sum_{j=1}^{m}\big(\sum_{i=1}^{n} \lambda_i a_{ji}\big) z_j = \sum_{j=1}^{m} \mu_j z_j$$

4.2.2.1 Matrix of basis transformation

A matrix can be used to transform the basis is:

$$M \in \mathbb{R}^{m \times n} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \vdots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$

The mapping from basis to basis coefficient is done:

$$M\lambda = \mu$$

4.2.2.2 Matrix changing the coordinates, 2D examples

Let $B = \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\}$ be the standard basis in $\mathbb{R}^2$ and $B' = \left\{ \begin{bmatrix} 3 \\ 1 \end{bmatrix}, \begin{bmatrix} -2 \\ 1 \end{bmatrix} \right\}$ be an alternative basis. The change of coordinate matrix from B' to B is:

$$P = \begin{bmatrix} 3 & -2 \\ 1 & 1 \end{bmatrix}$$

So that:

$$[v]_B = P[v]_{B'} \qquad \wedge \qquad [v]_{B'} = P^{-1}[v]_B$$

For arbitrary B and B' P's columns must be the B' vectors written in terms of the B ones.

4.2.3 Matrix properties

4.2.3.1 Transpose

The transpose matrix is the matrix obtained exchanging the rows with column M^T.

$$(MN)^T = N^T M^T$$

4.2.3.2 Trace

The trace is the sum of the diagonal elements of a matrix:

$$tr(M) = \sum_{i=1}^{n} M_{ii}$$

4.2.3.3 Inverse

The inverse is the matrix which multiplied with the original matrix gives the identity:

$$MM^{-1} = I$$

4.2.3.4 Rank

The rank of an $n \times m$ matrix is the dimension of the space spanned by its columns.

4.2.4 Matrix derivatives

$$\begin{aligned}
\frac{\partial M_X}{\partial x} &= M \\
\frac{\partial y^T M x}{\partial x} &= M^T y \\
\frac{\partial x^T M x}{\partial x} &= (M^T + M)x \\
\frac{\partial x^T M x}{\partial x} &= 2Mx \qquad \text{if } M \text{ is symmetric} \\
\frac{\partial x^T x}{\partial x} &= 2x
\end{aligned}$$

Results are columns vectors. Transposing the matrix gives the row vectors.

4.2.5 Metric structure

4.2.5.1 Norm

A function $||\cdot|| : \mathcal{X} \to \mathbb{R}_0^+$ is a norm $\forall x, y \in \mathcal{X}, \lambda \in \mathbb{R}$:

- $||x + y|| \leq ||x|| + ||y||$
- $||\lambda x|| = |\lambda| ||x||$
- $||x|| > 0 if x \neq 0$