Machine Learning: Classification/Supervised Learning

Machine Learning: Regression Case.

Dataset. Consider a collection of features $\mathbf{X} = \{X_1, \ldots, X_n\}$, Consider also an additional feature Y, such that $dom(Y) = \{c_1, \ldots, c_k\}$. We call Y the class variable.

Let $\mathbf{D} = \{\mathbf{x'_1}, \ldots, \mathbf{x'_m}\}$ be a collection of *data points*, such that $(\forall j \in 1 \ldots m)(\mathbf{x'_j} \in dom(\mathbf{X} \times dom(Y))$. We write $\mathbf{D}$ as

$$\mathbf{D} = \begin{pmatrix} X_1 & X_2 & \ldots & X_n & \\ \hline x_{11} & x_{12} & \ldots & x_{1n} & y_1 \\ x_{11} & x_{12} & \ldots & x_{1n} & y_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{11} & x_{12} & \ldots & x_{1n} & y_m \end{pmatrix}$$

We also write $\mathbf{x_i} = (x_{i1}, \ldots, x_{in})$.

Machine Learning Question. We treat $\mathbf{X}$ as the *independent* or *observed variables* and Y as the *dependent* or *target variable*.

Our goal can be expressed as such:

> Given the dataset $\mathbf{D}$ of data points, find a relationship between the vectors $\mathbf{x_i}$ of observed data and the values y_i of the dependent variable.

Classification. Representing a relationship between a collection of numeric and categorical independent variables and a dependent categorical variable is known as the classification problem.

Supervised learning because D, called *training set* contains class labels. Thus we can "supervise" predictions of our classifier.

Classification is usually performed in two steps:

1. **Step 1. Model fit.** On this step, the incoming training set is analyzed and a classification function is built to fit the training set.
2. **Step 2. Classification (Prediction).** This is the operation of actually producing a prediction $class(\bar{x})$ upon receiving a data point $\bar{x}$ as input. This step uses the function built during the **Model fit** step.

Classification Methodology

Logistic Regression. Extension of regression to the case of binary (categorical) target variable.

Perceptron. A simple linear or non-linear function that bisects the n-dimensional feature space.

Neural Netowoks. Graphical models that construct a ”separation function” based on the training set data by ”chaining” multiple perceptrons.

Naïve Bayes. Estimation of probability that a record belongs to each class.

Support Vector Machines (SVMs). Linear models for two-class classifiers.

Association Rules. Infer association rules with class label on the right side.

Decision Trees. Build a tree-like classifier. (**key advantage:** human-readable!)

Nearest Neighbor classifiers. Lazy evaluation classifiers that predict class of an input point based on its proximity to points with known category labels.

Linear Discriminant Analysis. Classification by finding a low-dimension hyperplane (e.g., a line) projection of all points onto which gives the best separation.

Simple Ensemble methods. Running multiple independent classifiers and using majority/plurality prediction.

Bagging. Bagging = Bootstrap aggregation is a resampling technique used to construct many classifiers (of the same basic type) on bootstrapped versions of the training sets. The class of a given data point is predicted as majority/plurality class for all constructed individual predictors. (Random Forests is a bagging extension of Decision Trees classifiers).

Boosting. Boosting is an ensemble technique where after a classifier is built for a given training set, the misclassified data points are given higher weight in the training set, and a new classifier is built to account for that. The method constructs a sequence of predictors, each of which is trying to correct for the errors of the previous one. (Adaboost is the classical example of a boosting classifier).

Perceptron

Many classification methods are *naturally defined* for the case when there are only *two categories* in the set of category labels. Such situations are usually called binary classification.

One of the simplest *binary classifiers* is perceptron.

Definition. Let $X = \{\bar{x_1}, \ldots, \bar{x_n}\}$ be a set of data points, where each point $\bar{x_i} = (a_1, \ldots, a_d)$ is a vector of length d. Let $C = \{+1, -1\}$ is the set of category labels, and let $Y = \{y_1, \ldots, y_n\}$, $y_i \in C$ be the category labels $y_i = class(\bar{x_i})$.

A perceptron is binary linear classifier that consists of

1. a linear function
$$f(\bar{x}) = \sum_{j=1}^{d} w_j \cdot a_j$$
for some vector $\mathbf{w} = (w_1, \ldots, w_d)$ of *weights*,

2. a threshold value θ, and

3. a decision procedure:
$$class(\bar{x}) = \begin{cases} +1 & \text{if } f(\bar{x}) > \theta; \\ -1 & \text{if } f(\bar{x}) < \theta; \end{cases}$$

Intuition. The perceptron function $f(\bar{x}) = \mathbf{w} \cdot \bar{x}$ defines a $d-1$ dimensional hyperplane through the d-dimensional feature space. Points on the *positive* side of f are classified into the *positive class* (the $+1$ class). Points on the negative side of f are classified to the negative class (the -1 class).

Notes.

- For a perceptron to correctly classify the data, the data must be *linearly separable*. A dataset is called *linearly separable* if there exists a hyperplane through its feature space that separates the points in one category from the points in another category.

- If there are multiple hyperplanes that linearly separate the data, the perceptron will converge to *one of them.* The error function for the perceptron is essentially
$$Error(f(\bar{x})) = \sum_{i=1}^{n} |f(\bar{x_i}) - y_i|,$$
i.e. the number of incorrectly classified data points. Therefore $Error(f) = 0$ for **any** hyperplane f that linearly separates the dataset, and the perceptron does not differentiate between such hyperplanes.

Training Perceptron

We first present the perceptron training algorithm for $\theta = 0$.

1. Set $\mathbf{w} = (0, \ldots, 0)$.

2. Pick $\eta > 0$, the *learning rate* of the perceptron.

3. For each training example $(\bar{x}, y), \bar{x} \in X$ do:

   (a) $y' = \mathbf{w} \cdot \bar{x}$

(b) if y' and y have the same sign, do nothing.

(c) if y' and y have different signs:

$$w := w + \eta \cdot y \cdot \bar{x}$$

To train the perceptron with an arbitrary value of θ:

- replace the vector $\mathbf{w} = (w_1, \dots w_d)$ with the vector $\mathbf{w}' = (w_1, \dots, w_d, \theta)$.
- replace every vector $\bar{x} \in X$, where $x = (a_1, \dots, a_d)$ with the vector $\bar{x'} = (a_1, \dots, a_d, -1)$.
- Train the perceptron using the algorithm above on the weights $\mathbf{w}'$ and feature vectors $X' = \{\bar{x'_1}, \dots, \bar{x'_n}\}$.

Note: If you squint at it, the training process for the perceptron classifier should remind you of something. Hint: where else have you seen the *learning rate* parameter?

Indeed, this algorithm is a special case of *gradient descent/gradient ascent*.

When to stop

The training can stop if:

- All $\bar{x} \in X$ have been correctly classified (i.e., when classification error = 0).
- Failing that, perceptron training can be stopped in one of the following ways:
  - After *M* iterations for some number $M > n$.
  - After the following detection error:

$$Error' = \frac{1}{2} \sum_{i=1}^{n} |\mathbf{w} \cdot \bar{x_i} \cdot (sign(\mathbf{w} \cdot \bar{x_i}) - y_i)|$$

  stops decreasing.

  (Note: $Error'$ computes the sum of distances from the separating hyperplane of all points that are misclassified. We need the $\frac{1}{2}$ normalizing factor because $|sign(\mathbf{w} \cdot \bar{x_i}) - y_i| = 2$ when the perceptron misclassifies a data point.)

References

[1] Jure Leskovec, Anand Rajaraman, Jeffrey D. Ullman, *Mining of Massive Datasets*, 2nd Edition, Cambridge University Press, 2014.

[2] Mohammed J. Zaki, Wagner Meira Jr., *Data Mining and Analysis: Fundamental Concepts and Algorithms*, Cambridge University Press, 2014.