Machine Learning: Classification/Supervised Learning

Overview

A large number of data analytical procedures are used for the purposes of *prediction*. The general form of a *prediction problem* is as follows:

> *Given a set of points* $X = \{\bar{x_1}, \ldots, \bar{x_n}\}$ *and the values some function* f *takes on these points:* $Y = \{y_1 \ldots, y_n\}$, *where* $(\forall i \in [1, \ldots, n])(y_i = f(\bar{x_i}))$, *predict the values of* $f()$ *on any input* $\bar{x}$.

Uusually, the prediction is to be made in a way that *minimizes* some objective function $error$:

$$\hat{f} = argmin_{\hat{f'}} error(f, \hat{f'})$$

The $error()$ can be computed differently depending on circumstances. The space of possible predictions $\{\hat{f'()}\}$ may be limited in some ways as well.

There are a number of special cases for the general *prediction problem*. We identify some of them below:

1. **Regression problems.** The range of $f(\bar{x})$ is the set of real numbers[1].
2. **Classification problems.** Also known as *supervised learning problems.* The range of $f(\bar{x})$ is a finite nominal or ordinal (categorical) set of values, known as *classes* or *categories*.
3. **Clustering problems** otherwise known as *unsupervised learning problems*. The range of $f(\bar{x})$ is an **unknown** finite nominal or ordinal set of values. The observed data consists only of the $\{\bar{x_1}, \ldots \bar{x_n}\}$ set of data points, without

[1] Or simply an infinite, or very large finite set of numbers.

values of $f(\bar{x_i})$ provided. In these problems, it is assumed that $f(\bar{x_i}) = f(\bar{x_j})$ if $d(\bar{x_i}, \bar{x_j})$ is sufficiently small, for some distance function $d()$. A *cluster* is an identified collection of data points $\{\bar{z_1}, \ldots, \bar{z_k}\} \subseteq X$ which all take the same value of $f()$.

Classification Problem. Definitions

Data. Consider a set $A = \{A_1, \ldots, A_n\}$ of attributes, and an additional categorical attribute C, which we call a **class attribute** or **category attribute**.

$dom(C) = \{c_1, \ldots, c_k\}$. We call each value c_i a **class label** or a **category label.**

The **learning dataset** is a relational table D.

Two formats:

1. **Training (data)set**. D has schema $(A_1, \ldots, A_n, C)$, i.e.,

   for each element of the dataset we are given its class label.

2. **Test (data)set**. D has schema $(A_1, \ldots, A_n)$, i.e.,

   the class labels of the records in D are not known.

Classification Problem. Given a (training) dataset D, construct a **classification/prediction function** that correctly predicts the class label for every record in D.

Classification function = prediction function = classification model = classifier.

Supervised learning because training set contains class labels. Thus we can compare (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

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

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

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!)

k-Nearest Neighbors Classification (kNN)

k-Nearest Neighbors Classifiers are among some of the simplest classification techniques. They are, **however** surprisingly rather accurate and robust and produce good results for a wide range of datasets.

kNN classifiers follow the prinicplpe of **lazy evaluation** and do not construct the data model until a question is asked.

The principle of **lazy evaluation** is to *postpone any data analysis until an actual question has been asked.*

In case of supervised learning, **lazy evaluation** means **not building a classifier** in advance of reading data from the test data set.

k-Nearest Neighbors Classification algorithm (kNN). kNN is a simple, but surprisingly robust **lazy evaluation** algorithm. The idea behind kNN is as follows:

- The input of the algorithm is a training set $D_{training}$, an instance d that needs to be classified and an integer $k > 1$.
- The algorithm computes the *distance* between d and every item $d' \in D$.
- The algorithm selects k **most similar** or **closest** to d records from D: $d_1, \ldots, d_k$, $d_i \in D$.
- The algorithm assigns to d the class of the plurality of items from the list $d_1, \ldots, d_k$.

Distance/similarity measures. The distance (or similarity) between two records can be measured in a number of different ways.

Note: Similarity measures increase as the similarity between two objects increases. **Distance measures** decrease as the similarity between two objects increases.

1. **Eucledian distance**. If D has continuous attributes, each $d \in D$ is essentially a point in N-dimensional space (or an N-dimensional vector). Eucledian distance:

$$d(d_1, d_2) = \sqrt{\sum_{i=1}^{n} (d_1[A_i] - d_2[A_i])^2},$$

   works well in this case.

2. **Manhattan distance**. If D has ordinal, but not necessarily continuous attributes, Manhattan distance may work a bit better:

$$d(d_1, d_2) = \sum_{i}^{n} |d_1[A_i] - d_2[A_i]|.$$

3. **Cosine similarity**. Cosine distance between two vectors is the cosince of the angle between them. Cosine similarity ignores the amplitude of the vectors, and measures only the difference in their *direction*:

$$sim(d_1, d_2) = \cos(d_1, d_2) = \frac{d_1 \cdot d_2}{||d_1|| \cdot ||d_2||} = \frac{\sum_{i=1}^{n} d_1[A_i] \cdot d_2[A_i]}{\sqrt{\sum_{i=1}^{n} d_1[A_i]^2} \cdot \sqrt{\sum_{i=1}^{n} d_2[A_i]^2}}.$$

If d_1 and d_2 are *colinear* (have the same direction), $sim(d_1, d_2) = 1$. If d_1 and d_2 are *orthogonal*, $sim(d_1, d_2) = 0$.

Classifier Evaluation

Accuracy Measures

Notation. Let T be a classifier constructed by **any** supervised learning algorithm given a **training set** D.

Let D' be a **test set**, ***drawn from the same data/distribution as*** D.

Let $t \in D'$. As $T(t)$ we denote the **class label** supplied for t by the classifier T.

As $class(t)$ we denote the **actual** class label of t.

As D_{true} we denote the *set of all test cases for which our classifier provides correct prediction*:

$$D_{true} = \{t \in D' | T(t) = class(t)\}$$

As D_{error} we denote the *set of all test cases for which our classifier provides incorrect prediction*:

$$D_{error} = \{t \in D' | T(t) \neq class(t)\}$$

Accuracy. The **accuracy** of the classifier T is:

$$accuracy(T) = \frac{|D_{true}|}{|D'|}.$$

Error rate. The **error rate** of the classifier T is:

$$errorRate(T) = 1 - accuracy(T) = \frac{|D_{error}|}{|D|}.$$

Accuracy Measures for Binary Classification

Binary Classifiers. Many classifiers are **binary**: i.e., the class variable C has only two values. A classifiaction problem with $dom(C) = \{c_1, \ldots c_k\}$, $k > 2$ can be transformed into k classification problems with class variables $C_1, C_2, \ldots, C_k$, such that, $dom(C_i) = \{0, 1\}$. $C_i = 1$ means $C = c_i$.

Classification Errors. Consider a binary classification problem with the class variable C, $dom(C) = \{0, 1\}$, where $C = 1$ is interpreted as *"record belongs to class C"* and $C = 0$ is interpreted as *"record does not belong to class C*.

Let T be a classifier for C. Let D' be a test dataset. Given $t \in D$, we can observe four possibilities:

1. **True Positive:** $T(t) = class(t) = 1$;
2. **True Negative:** $T(t) = class(t) = 0$;
3. **False Positive:** $T(t) = 1; class(t) = 0$;
4. **False Negative:** $T(t) = 0; class(t) = 1$;

There are **two types of errors of classification**:

1. **Type I error:** a.k.a. **error of commission** a.k.a. **false positive**: classifier incorrectly classifies a tuple as belonging to class C.

2. **Type II error:** a.k.a. **error of omission** a.k.a. **false negative**: classifier incorrectly classifies a tuple as NOT belongingto class C.

Notation. Conisder the following notation:

1. D_{TP} : set of all **true positives** in D'; $TP = |D_{TP}|$;

2. D_{TN} : set of all **true negatives** in D'; $TN = |D_{TN}|$;

3. D_{FP} : set of all **false positives** in D'; $FP = |D_{FP}|$;

4. D_{FN} : set of all **false negatives** in D'; $FN = |D_{FN}|$;

Confusion Matrix. The information about the accuracy of a **binary classifier** is usually arranged in a form of **confusion matrix**:

	Classified Positive	Classified Negative
Actual positive	TP	FN
Actual negative	FP	TN

Precision. **Precision** of the classifier is the percentage of the correctly *positively* classified records in the set of all positively classified records:

$$precision(T) = \frac{TP}{TP + FP}.$$

Precision measures *how accurately the classifier selects positive examples*, it reaches 100% when the classifier *admits no false positives*.

Recall. **Recall** of the classifier is the percentage of all correctly *positively* classified records in the set of all actual positive records:

$$recall(T) = \frac{TP}{TP + FN}.$$

Recall measures *how successful the classifier is in correctly identifying all positive records.* It reaches 100% when the classifier *admits no false negatives.*

Note: **Precision** and **recall** make sense **only** when combined together.

It is easy to build a classifier with 100% precision: *T(t) = 0* for all $t \in D'$ guarantees that. **But this classifier will have recall of 0**. It is easy to build a classifier with 100% recall: *T(t) = 1* for all $t \in D'$ guarantees that. **But this classifier will have small precision.**

PF. The **PF** measure is defined as:

$$PF(T) = \frac{FP}{FP + TN}.$$

PF measures the *misclassification rate*: the percentage of records **not** in class C that was **incorrectly classified**.

F-measure. The **F-measure** is the harmonic mean of precision and recall:

$$F(T) = \frac{2}{\frac{1}{precision(T)} + \frac{1}{recall(T)}} = \frac{2 \cdot precision(T) \cdot recall(T)}{precicion(T) + recall(T)}.$$

F-measure combines precision and recall into a single number by balancing them against each other.

In some situations, one of the two measures (precision or recall) is more important than the other. **F-measure** can be skewed to favor each. The F_2**-measure** below assumes recall is twice as valuable as precision. The $F_{0.5}$**-measure** below assumes precision is twice as valuable as recall.

$$F_2(T) = \frac{5 \cdot precision(T) \cdot recall(T)}{4 * precision(T) + recall(T)}.$$

$$F_{0.5}(T) = \frac{1.25 \cdot precision(T) \cdot recall(T)}{0.25 * precision(T) + recall(T)}.$$

The formula for F_β, where β represents the relative importance of recall over precision is:

$$F_\beta(T) = \frac{(1 + \beta^2) \cdot precision(T) \cdot recall(T)}{\beta^2 * precision(T) + recall(T)}.$$

Evaluation Techniques

In a typical situation, you are given a **training set** D, and are asked to produce a classifier for it.

If all records from D are used to create a classifier, there will be no way to INDEPENDENTLY test its accuracy.

Holdout set. Divide D into two sets: $D = D_{train} \cup D_{test}$; $D_{train} \cap D_{test} = \emptyset$.

D_{test} is called the **holdout set.**

Create a classifier T using D_{train} as the training set. **Test** T using D_{test}.

Holodout set selection:

- **Random sampling.** Select a fraction x. Randomly sample $x\%$ of records from D, put them in D_{test}.

  Commonly, you use around 90% of D as the training set, reserving the remaining 10% for the holdout set.

- **Time slices.** If D consists of "old" data and "new" data, then, the training set can include all of the "old" data, while the holdout set can include the "new" data. (e.g., in situations where new records appear every day).

Multiple random sampling. This technique is used when D is small.

- Select some number M of repetitions.
- Perform M random samplings of a **holdout set** from D. Run classifier construction on the remaining set D_{train}. Compute the accuracy of the classifier for the current sample.
- Compute the final accuracy as the mean accuracy over all samples.

Multiple random sampling allows us to avoid *flukes* (or, at least, to downgrade their effects).

Cross-Validation. This is a variant of **multiple random sampling** that uses only one random assignment of records, but performs multiple classifications.

- Select n – the number of *slices* of data in D.
- Using *random sampling* split D into n *slices* of equal (or almost equal) size.
- Peform n classification procedures. On step i, use slice D_i as the **holdout set**, while using all other $n - 1$ slices as the **training set**.

Note: Standard cross-validations used in practice are **10-fold**, **5-fold** and **leave-one-out** cross-validations.

References

[1] J.R. Quinlan. *C4.5: Program for Machine Learning*, Morgan Kaufman, 1992.