Praktikum Maschinelles Lernen und Datenanalyse

Sommersemester 2007

Fachgebiet Maschinelles Lernen
Institut für Softwaretechnik und
theoretische Informatik
Fakultät IV, Technische Universität Berlin
Prof. Dr. Klaus-Robert Müller
Email: krm@cs.tu-berlin.de

Übungsblatt 4: Grundlagen Klassifikation

Abgabeschluss: Montag, der 18.06.2007 um 10:00 Uhr.

Für dieses Aufgabenblatt sind sowohl Code als auch eine schriftliche Ausarbeitung abzugeben. Sendet Euren Code an `mikio@cs.tu-berlin.de` **und** `buenau@cs.tu-berlin.de` **mit Subject** "`ML-Praktikum Abgabe` *`Name`*". Gebt Eure Ausarbeitungen in unserem Sekretariat (FR 6-9) bei Frau Gerdes (Raum FR 6052) ab.

Bitte beachtet weiter die Coding-Richtlinien von Blatt 1

Aufgaben

Aufgabe 1: Implementation

Crossvalidation (5 Punkte)

Implementiere Kreuzvalidierung als allgemeine Funktion, die für unterschiedliche Methoden und Gütekriterien verwendet werden kann:

```
[ C, opt_param ] = cv(X, y, classifier_handle, { param_name, value_range, ... },
                      nfolds, nrepetitions, loss_function, opt_rule)
```

Die Argumente haben die folgende Bedeutung.

1. `X` ist die $(d \times n)$-Matrix der Daten.
2. `y` ist ein $(1 \times n)$-Vektor, welcher die labels oder regression targets zu jedem Datenpunkt enthaelt.
3. `classifier_handle` ist der Handle des zu verwendenden Classifiers. Dabei entspricht ein Classifier einer Funktion

   ```
   C = classifier(X, y, param1, param2, ...)
   ```

   welche den Klassikator auf den gegebenen Trainingsdaten `X`, `y` und Parametern traininiert und sämtliche zur Anwendung notwendige Information in der Struktur `C` zurueckliefert.

   Insbesondere enthält `C` das feld `applyfunc`, die einen Handle auf die Auswertungsfunktion `y = feval(C.applyfunc, C, X)` enthält, welche den Klassifikator `C` auf die Daten `X` anwendet und die labels (oder regression targets) `y` berechnet. (Hinweis: Der Functionhandle kann auch auf eine lokale Funktion zeigen. Implementiere die Auswertungsfunktion also als lokale Funktion immerhalb des Classifiers.)
4. Eine Liste bestehend aus Parameternamen (`param_name`) und Wertebereichen (`value_range`). Die Kreuzvalidierung wird für alle so gegebenen Kombinationen von Parameterwerten durchgeführt. Die Reihenfolge der Parameter in dieser Liste muss ihrer Reihenfolge in der `train_<NAME>` entsprechen. Siehe Matlab Hilfe zu `varargin` fuer die Verarbeitung von variablen Argumentlisten.
5. `nfolds` ist die Anzahl der Partitionen (m im Skript). Dieser Parameter ist optional mit dem Standardwert 10.
6. `nrepetitions` ist die Anzahl der Wiederholungen (r im Skript). Dieser Parameter ist optional mit dem Standardwert 5.

7. `loss_function` ist der Namen der zu verwendenden Loss-Function. Diese hat die Signatur `l = loss_function(X, y_true, y_pred)` wobei `X` die Daten sind, `y_true` die echten Labels und `y_pred` die berechneten. Schreibe eine Loss-Function mit den Namen `zero_one_loss`, die den Klassifikationsfehler zwischen 0 und 1 zurückliefert. Dieser Parameter ist optional mit `'zero_one_loss'` als Standardwert.

8. `opt_rule` ist der Name des Kriteriums nach dem der beste Parameter ausgewählt wird. Mögliche Werte sind

   - `'min_mean'`: minimaler mittlerer Loss
   - `'min_mean_std'`: minimaler mittlere Loss plus Standardabweichung
   - `'min_median'` minimaler median Loss

   Der Standardwert dieses optionalen Parameters ist `'min_mean'`.

Das Ergebnis von `cv` ist

1. `C`, der Klassifikator trainiert mit den besten Parametern.

2. `opt_param`, eine Struktur welche die optimalen Parameter enthält. Für jeden Parameter gibt es ein gleichnamiges Attribut (siehe `param_name`), das seinen Wert enthaelt.

Die Funktion soll auf der Kommandozeile ueber den Fortschritt berichten und dabei auch eine Schätzung für die verbleibende Laufzeit angeben (siehe `tic` und `toc`)

Kernel Ridge Regression (15 Punkte)

Implementiere Kernel Ridge Regression

```
C = krr(X, y, kernel, kernelparameter, regularization)
```

Die folgenden Kerne (mit ihren dazugehörigen Parametern) sollen implementiert werden:

Name	Kern	Parameter
`linear`	$k(x,z) = \langle x, z\rangle$	(keiner)
`polynomial`	$k(x,z) = (\langle x, z\rangle + 1)^p$	Grad $p \in \{1, 2, 3, \ldots\}$
`gaussian`	$k(x,z) = \exp(-\|x-z\|^2/2dw^2)$	Kernbreite w

Hier ist d die Dimension der X.

Der Parameter `regularization` ist die Regularisierungskonstante c in $\hat{\alpha} = (K + cI)^{-1}y$. Falls `regularization` null ist, führe Leave-One-Out Crossvalidierung über c effizient durch. Verwende als Kandidaten für c die Eigenwerte der Kernmatrix K.

Verwende die "noisy sinc function" (siehe Homepage) zum Testen der Funktion.

Decision Trees (10 Punkte)

Implementiere Decision Trees fuer Klassifikation in der im Skript angegebenen Variante als Funktion

```
C = train_classTree(X, y, lambda, min_points, max_depth)
```

wobei `X` die $(d \times n)$-Matrix der Trainingsdaten und `y` der $(1 \times n)$ Vektor der dazugehoerigen labels ist. Die Parameter der Methode haben die folgende Bedeutung.

- `lambda` ist der Regularisierungsparameter (siehe λ im Skript), der den trade-off zwischen der Anzahl an Partitionen und impurity reguliert. Der Standardwert dieses optionalen Parameters ist 0.

- `min_points` ist die untere Schranke fuer die Anzahl an Trainingsdaten in einer Partition. Der Standardwert dieses optionalen Parameters ist `1`.

- `max_depth` ist die maximale Tiefe des Entscheidungsbaumes. Der Standardwert dieses optionalen Parameters ist `Inf`.

Der Rueckgabewert `C` ist eine Struktur, die alle Informationen zum trainierten Klassifikator enthaelt wobei das Attribut `C.applyfunc` ein function handle auf die Anwendungsfunktion ist, welche auch in `train_class.m` definiert ist.

Schreibe ausserdem eine Funktion

```
plot_classTree(C, X, y)
```

welche einen gegebenen Entscheidungsbaum `C` fuer zweidimensionalen Daten `X` und labels `y` visualisiert. Dazu sollen die Daten `X` in einem zweidimensionalen Koordinatensystem geplottet werden, wobei in Abhaengigkeit des labels ein anderes Symbol gewaehlt wird. Die Partition des $\mathbb{R}^2$ durch `C` soll durch Striche bzw. Kaesten visualisiert werden (siehe Beispiel im Skript). Denke Dir ausserdem eine geeignete Visualisierung fuer das Klassifikationsergebnis in jeder Partition aus.

Aufgabe 2: Anwendung

ROC Curves (5 Punkte)

Schreibe eine Funktion

```
roc_curve(n)
```

welche fuer das eindimensionale binaere Klassifikationsproblem mit Gaussschen Klassen und identischen class priors,

$$\begin{aligned} p(x|y=-1) &\sim \mathcal{N}(\mu=0,\sigma^2=1) \\ p(x|y=+1) &\sim \mathcal{N}(\mu=2,\sigma^2=1) \\ p(y=-1) &= 0.5 \\ p(y=+1) &= 0.5 \end{aligned}$$

die ROC-Curve (siehe Skript) des linearen Klassifikators f_{x_0} gegeben durch

$$f_{x_0}(x) = \begin{cases} -1 & : \quad x \leq x_0 \\ +1 & : \quad x > x_0 \end{cases}$$

plottet. Die Funktion `roc_curve` soll zwei subplots in einer figure erstellen:

1. Die analytisch berechnete ROC-Curve. Verwende dazu die gegebenen Dichten.
2. Die ROC-Curve ermittelt durch Simulation mit `n` Datenpunkten. Generiere Dir dazu zufaellige Daten aus den gegebenen Verteilungen.

Versehe beide Plots mit Titel und Achsenbeschriftungen.

Klassifikation mit Modellselektion (25 Punkte)

Auf unserer Homepage findest Du verschiedene Datensätze für Kernel-Ridge-Regression und Decision Trees. Wende die Algorithmen jeweils auf die Datensätze an und bestimme per Crossvalidiation den besten Classifier. Gib jeweils den besten Klassifier (d.h. die Struktur `C`) als `.mat`-file ab. Plotte die resultierenden ROC-Kurven fuer Kernel-Ridge-Regression, indem Du den bias-term verschiebst.

Weiterhin findest Du auf unserer Homepage zu einigen Trainingsdatensaetzen auch Testdaten ohne labels. Gib zu jedem dieser Datensaetze Deine berechneten Labels als Vektor `y` ab.