1. Palabras clave

Regresión lineal, clasificación, algoritmos genéticos, python.

2. Objetivo

Clasificar los datos numéricos para “Weka”, utilizando uno de los atributos como descriptor de clases.

3. Descripción

El modelo de regresión lineal (RL) se representa con la siguiente ecuación:

$$y = b_0 + \sum_{i=1}^{n} b_i x_i + \varepsilon$$

dónde y – variable dependiente;
$\{x_i\}$ – variables independientes;
$\{b_i\}$ – coeficientes del modelo;
ε – error.

Supongamos que sabemos los valores de la variable dependiente $\left\{ y_j^{\,ideal} \right\}$ y los valores correspondientes de las variables independientes $\left\{ \{x_i\}^{j,\,k} \right\}$ (k es el número del ejemplo que corresponde a y_j).

Nos interesa encontrar los coeficientes $\{b_i\}$ tales que la suma de los errores cuadráticos

$$\sum_{j=1}^{N} \left(y_j^{ideal} - y_j^{regr} \,(= \varepsilon) \right)^2$$

sea mínima (N es el número de distintas clases).

Existen varios métodos de búsqueda de los extremos de una función.
Escogí los algoritmos genéticos (AG) para resolver el problema porque a diferencia de los métodos numéricos clásicos, los AG encuentran (si cumplen ciertas condiciones) el mínimo/máximo global.

Los AG habitualmente usan dos operadores genéticos: “crossover” y mutación. Inspirado por el paquete “global optimisation” de “Matlab”, agregue el tercer operador “élite”, el cual transfiere los mejores de los mejores a la siguiente generación sin cambios.

4. Recursos técnicos

Python: módulos “random” y “operator”.

5. Implementación

linregr_weka_numeric.py: read_and_minumize_linreger

1. archivo de "Weka" .arff;
2. atributo para considerar como descriptor de clase;
3. [Opcional] atributos a remover.

El programa lee el archivo proveído (read_weka_data.py), extrayendo los atributos y los datos. Después de haber removido los atributos (3), se agrupan los datos según el atributo (2) y se guardan en un diccionario:

$$\{\text{clase (2)} : \underline{\text{atributos}}\} \Leftrightarrow \left\{ y_j^{ideal} \rightarrow \left\{ \{x_i\}^{j,k} \right\} \right\},$$

dónde atributos no incluyen (2) y (3).

El diccionario se utiliza para construir la función de error cuadrática (linregression.py):

$$error\_func : \{coef\} \rightarrow \Re_{\geq 0}$$

$$\forall \left(y^{ideal},\ \{\{x_i\}\} \right) \in Dict \Rightarrow$$
$$\forall \{x_i\} \in \{\{x_i\}\} \Rightarrow$$
$$y^{regr} = coef_0 + \sum_{i=1}^{n} coef_i\, x_i$$
$$error = \left(y^{ideal} - y^{regr} \right)^2$$
$$\sum \{error\}$$

La función de error se utiliza como función fitness para el AG.

linregression.py: linregr_classifier

1. descriptores de classes;
2. coeficientes de RL.

Utiliza los descriptores y coeficientes para construir una función clasificador:

$$classify : \{atributos\} \rightarrow class$$

$$y^{repr} = RL(coeficientes,\ atributos)$$
$$class = Min_{\,c \in \{class\}} \left\{ \left| c - y^{repr} \right| \right\}$$

---

ga.py: minimize_until_stall (nuevo)

1. función fitness;
2. precisión distinguindo falta de progreso;
3. máximo de faltas de progreso enseguida;
4. opciones de AG;

Genera nuevas generaciones hasta que la diferencia entre los promedios de fitness de élites cambia a un valor menor de (2) (3) veces enseguida.

ga.py: minimize_to_target

1. función fitness;
2. objetivo de fitness;
3. opciones de AG;
4. [Opcional] número de reintentos.

Genera nuevas generaciones hasta que encuentra al menos un ejemplar con fitness menor a (2). Si (4) está especificado, va reintentar correr el AG de nuevo las veces especificadas.

Opciones de AG:

- Population: el tamaño de población.
- Genes: el número de genes en un cromosoma (ejemplar).
- Max_Generations: el número máximo de generaciones a producir. El AG para después de haber producido número dado de generaciones. (default: None)
- N_Elite: el número de ejemplares para considerar élites. (default: Population / 10)
- Crossover_Fraction: la fracción de la generación que se utiliza para el crossover (menos el número de élites). (default: 0.5)
- Crossover: la función de crossover. (default: xover_simple_between_best)
- Crossover_Mutate_Chance: el chance de mutación de un niño de crossover. (default: 0.5)
- Crossover_Mutate_Preserve: preservación de los niños de crossover en caso de mutación (se guardan ambas versiones). (default: True)
- Mutate: la función de mutación. (default: mutate_simple_random)
- Gene_Mutation_Chance: la probabilidad de mutación de un gen. (default: 0.2)
- Print_Info_Each: imprimir la información sobre el progreso de AG cada x generaciones. (default: 10)
- Mutate_Stdev: la desviación estándar para la generación de mutación. (default: 1)
- Mutate_Shrink_Stdev: indica que Mutate_Stdev debe disminuirse a (Mutate_Stdev * Mutate_Shrink_Stdev) al llegar a Max_Generations.

ga.py: next_generation

1. opciones de AG.

Utiliza las opciones proveídas para construir una función, la cual genera la siguiente población dado la previa.

En conjunto las funciones:

1. ordenan los padres según su fitness;
2. escogen los mejores ejemplares como “élite”;
   (su cantidad se configura en opciones)
3. escogen una fracción de los mejores para crossover;
   (la fracción se configura en opciones)
4. hacen copias de los élites con mutaciones;
5. mutan a una fracción de los niños de crossover; si está escogida la opción de preservación de los niños, los mutantes son copias;
   (la fracción y la preservación se configuran en opciones)
6. calculan el número del resto de padres que se puede utilizar (tamaño de población - número de niños ya generados);
7. mutan la parte del resto, establecida en (6);
8. vuelve la nueva generación, compuesta de los élites (2), los élites mutantes (4), los niños de crossover con los mutantes (y con preservados si aplica) (3, 5), el resto de mutantes (7).