Parte I
Conceptos

Parte II
Modelos

1. Modelos Lineales

Si disponemos de atributos numéricos y buscamos predecir un atributo también numérico, entonces es natural explorar los modelos lineales. Estos modelos asumen que el atributo a predecir $y^{(k)}$ responde a la forma $w_0 + \sum_i w_i x_i^{(k)}$, es decir a un hiperplano. Como se espera la presencia de errores dentro de las mediciones o simplemente se busca aproximar la forma "real" de y se minimiza alguna métrica de distancia entre $y^{(k)}$ y $y^{(k)'} = w_o + \sum_i w_i x_i$ como pueden ser según [5]:

- Minima Suma de Cuadrados

$$\min \sum_i (y^{(i)} - y^{(i)'})^2$$

- Minimos Valores Absolutos

$$\min \sum_i |y^{(i)} - y^{(i)'}|$$

- M, L y S Estimadores
- Minima Suma Podada de Cuadrados (Least Trimmed Squares)

$$\min \sum_j (y^{(j)} - y^{(j)'})^2, \quad \text{donde} \quad \{y^{(j)} - y^{(j)'}\} \subset \{y^{(i)} - y^{(i)'}\}$$

- Minima Media de Cuadrados

$$\min \text{med}_i (y^{(i)} - y^{(i)'})^2$$

La mayor característica de los métodos lineales es, precisamente, su linealidad. Considerando que es raro encontrar un fenómeno que se comporte linealmente, la linealidad es una desventaja. Sin embargo, esto también permite que el modelo presente un menor sesgo y se verá mas adelante que esto se suele aprovechar.

Segun [6], tenemos que LinearRegression() utiliza la minimización de la suma de los cuadrados sin embargo dispone de un método de selección de atributos basado en el Criterio de Información de Akaike (ver 6) que simplifica el modelo resultante y está activado por defecto.

2. Ejemplos

```
// Cargamos los datos de alguna manera
Instances dataset = ...
// Preparamos una instancia de prueba
Instance testInstance = ...

// Creamos una instancia del modelo
Classifier model = new SimpleLinearRegression();
// Entrenamos con los datos de 'dataset'
model.buildClassifier(dataset);

// Podemos imprimir los detalles del modelo
System.out.println(model);
// Tambien podemos predecir una instancia
double prediction = model.classifyInstance(testInstance);
```

También podemos utilizar LinearRegression(), MultilayerPerceptron()[1] y con algunas modificaciones Logistic().

3. Árboles

Una de las maneras usuales de atacar problemas es mediante la estrategia "divide y conquistarás", los arboles, con su estructura recursiva, represetan fielmente esta idea. Existen multiud de variedades de árboles utilizados en minería de datos, sin embargo la idea predominante es dividir el conjunto de muestras en dos o mas subconjuntos utilizando un criterio de decisión, usualmente el valor de un atributo, y repetir esto hasta que se alcance la máxima complejidad aceptable o repetir hasta que se alcanze una predicción lo suficientemente buena.

En el caso de atributos nominales, la división es trivial y única. Para los atributos numéricos, se dispone de multitud de métodos listados en [4] y [1]. En WEKA disponemos de weka.filters.unsupervised.attribute.NumericToNominal que simplemente crea una biyección entre cada valor numérico y una clase, weka.filters.unsupervised.attribute.Discretize divide el atributo en clases con la misma anchura (equal-width binning) con opciones de optimización o en clases con una misma frecuencia (equal-frecuency binning) y finalmente weka.filters.supervised.attribute.Discretize el cual utiliza el método MDL de Fayyad e Irani, ver [3], que es el método por defecto utilizado en los modelos.

3.1. J48()

El modelo J48() es el mas conocido entre los árboles presentes en WEKA, este modelo es la versión libre del algoritmo C4.5. En este modelo se construye un árbol de forma voráz seleccionando el atributo que "mejor" divide a las muestras. El criterio de selección está basado en la ganancia de información midiendo la diferencia de entropía entre el conjunto de datos original y el mismo conjunto de datos divido por el atributo.

$$\text{Entropía} = H = \sum_i -p_i \log_2(p_i)$$

Veamos este ejemplo. Las muestras estan originalmente clasificados en $(0,0,0,0,1,1,1)$ y luego de la división segun un atributo A se llega right $= (0,0,0,1)$ y left $=$

[1] Si activamos la opción -G (GUI) podremos modificar la topología de la red.

$(0, 1, 1)$. Medimos la entropía de las muestras antes de la división.

$$H_0 = -\frac{4}{7}\log\frac{4}{7} - \frac{3}{7}\log\frac{3}{7} \approx 0{,}985$$

Medimos la entropía en cada subconjunto.

$$H_{right} = -\frac{3}{4}\log\frac{3}{4} - \frac{1}{4}\log\frac{1}{4} \approx 0{,}811$$

$$H_{left} = -\frac{1}{3}\log\frac{1}{3} - \frac{2}{3}\log\frac{2}{3} \approx 0{,}918$$

Obtenemos la entropía al final de la división haciendo una suma ponderada según el tamaño relativo de los subconjuntos.

$$H_1 = \frac{4}{7}H_{right} + \frac{3}{7}H_{left} = \frac{6}{7} \approx 0{,}857$$

Finalmente la Ganancia de Información, en inglés Information Gain, es la diferencia de entropías.

$$IG = H_0 - H_1 \approx 0{,}128$$

Luego de calcular la IG para cada atributo, se selecciona el mayor valor. Y se aplica recursivamente sobre los hijos. El J48() aplica otros procedimientos para simplificar aún mas el árbol resultante.

Parte III
Evaluacion

Parte IV
Selección de Modelos

4. Principio de Parsimonia

Tambien llamado Navaja de Ockham o, en inglés, Ockham's Razor. Es un principio según el cual "entre varias explicaciones equivalentes, se prefiere la mas simple".

Cabe destacar que fueron muchas las críticas contra este principio por "ser imprudente". La *anti navaja* de A. Einstein: "Simple, pero no más simple.".

5. Maximum Log-Likelihood

En español, maximización de verosimilitud. Dado un modelo con parametros θ y un conjunto de muestras X independientes e identicamente distribuidos, la verosimilitud de los parametros dado el conjunto de muestras $\mathcal{L}(\theta|X)$ es igual a la probabilidad de obtener las muestras dado los parametros $P(X|\theta)$. Es decir, cual es la "probabilidad" de que los parametros θ "expliquen" las muestras X.

Luego tenemos que si maximizamos $\mathcal{L}(\theta|X)$ tendremos un conjunto de parametros $\hat{\theta}$ tal que sean o sean lo suficientemente cercanos a los parametros "reales"[2]. Entonces tendremos que $\mathcal{L}(\theta|X) = P(X|\theta) = \prod_i P(x_i|\theta)$, sin embargo maximizar esto en la práctica es dificil por ello se busca maximizar el logaritmo de el, $\text{máx}_\theta \mathcal{L}(\theta|X) = \text{máx}_\theta \log \mathcal{L}(\theta|X) = \log \mathcal{L}(\hat{\theta}|X) = \sum_i P(x_i|\theta)$.

6. Akaike Information Criterion

El AIC se define como $AIC = 2K - 2\ln \mathcal{L}(\hat{\theta}|X)$, donde K es la cantidad de parametros que se utilizaron en el modelo mas uno y $\mathcal{L}(\hat{\theta}|X)$ es el maximo de la función de verosimilitud. Este criterio indica que el modelo es "mejor" si posee un **menor** valor y es un criterio de comparación entre modelos **bajo los mismos datos**, es decir que el valor en si mismo no es significativo sino si el valor es menor o no al AIC de otro modelo con el cual comparamos. La interpretación es la siguiente: el término $2K$ es la penalización, por tener signo positivo y estamos buscando un menor valor, por la cantidad de parámetros que el modelo posee ya que a mayor cantidad de parámetros **mayor varianza** y mayor posibilidad de "overfitting", por otra parte el término $-2\ln \mathcal{L}(\hat{\theta}|X)$ favorece a una mejor calificación, por el tener un signo negativo, y está relacionado a verosimilitud del modelo para esos parámetros y esas muestras y también la un **menor sesgo**[2].

Parte V
Importación y Exportación

7. Exportación

Para exportar un modelo hacemos uso de weka.core.SerializationHelper() disponible desde la versión 3.5.5. Destacamos que el método write también trabaja con flujos y que es necesario capturar las excepciones que arroja.

```
// Disponemos del modelo a exportar
Classifier model = ...

// Utilizamos el metodo .write
String outputFilename = ...
SerializationHelper.write(outputFilename, model);
```

8. Importación

Para importar un modelo utilizamos weka.core.SerializationHelper() disponible desde la versión 3.5.5. Destacamos que el método read también trabaja con flujos y que es necesario capturar las excepciones que arroja.

```
Classifier model = (Classifier) SerializationHelper.read(
    inputFilename);
```

[2]Suponiendo que el modelo "real" sea el mismo que el utilizado.

9. Trabajado en MATLAB

Para trabajar con las herramientas que provee WEKA utilizamos javaaddpath para agregar la ubicacion de la biblioteca[3].

Referencias

[1] James Dougherty, Ron Kohavi, Mehran Sahami y col. «Supervised and unsupervised discretization of continuous features». En: *Machine learning: proceedings of the twelfth international conference*. Vol. 12. 1995, págs. 194-202.

[2] Shuhua Hu. *Estimation error*. 2007.

[3] Keki B Irani. «Multi-interval discretization of continuous-valued attributes for classification learning». En: (1993).

[4] Sotiris Kotsiantis y Dimitris Kanellopoulos. «Discretization techniques: A recent survey». En: *GESTS International Transactions on Computer Science and Engineering* 32.1 (2006), págs. 47-58.

[5] David Ruppert y Raymond J Carroll. «Trimmed least squares estimation in the linear model». En: *Journal of the American Statistical Association* 75.372 (1980), págs. 828-838.

[6] I.H. Witten, E. Frank y M.A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2011. ISBN: 9780080890364. URL: `https://books.google.com.py/books?id=bDtLM8CODsQC`.

[3] El archivo que usualmente llamado "weka.jar".