ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE TELECOMUNICACIÓN DE LA UGR

DEPARTAMENTO DE DECSAI

PROYECTO FINAL DE APRENDIZAJE AUTOMÁTICO:

HUMAN ACTIVITY RECOGNITION USING SMARTPHONE

Este proyecto ha sido realizado por David Gil Bautista y Santiago Vidal Martínez

---

Índice

1. Comprender el problema a resolver

Para el proyecto hemos decidido afrontar el problema de Reconocimiento de la Actividad Humana por el uso de teléfonos inteligentes (**Human Activity Recognition Using Smartphones**) porque solicitamos dos opciones que no aparecían en la lista y se nos recomendó utilizar uno de dicha lista. Como no podíamos mandar la nueva propuesta a la plataforma docente de DEC-SAI, lo indicamos en esta memoria.

Dicho dataset cuenta con los datos obtenidos por el acelerómetro y giroscopio que son los sensores de un smartphone (Samsung Galaxy II) mientras que distintas personas llevan el dispositivo en la cintura. Los datos recogidos han sido aportados por un grupo de 30 personas con edades comprendidas entre los 19 y 48 años, y cada persona ejecuta 6 posibles acciones: (caminar, subir escaleras, bajar escaleras, sentarse, levantarse, tumbarse)[1]

Hemos visto que hay varios articulos en los que se estudia el problema, y más concretamente, uno de Davide Anguita[2], en el que usa SVM (**Support Vector Machine**) para clasificar los datos.

Dado que no podemos aportar los datos, incluiremos aquí el enlace a la base de datos de nuestro problema:

`https://archive.ics.uci.edu/ml/machine-learning-databases/00240/`

Para el correcto funcionamiento del código, los cuatro conjuntos deben estar dentro de un directorio llamado "datos".

2. Preprocesado de datos

Se nos ofrece un set de datos de 10299 ejemplos con 561 atributos.

Para cada instancia se ofrece la aceleración del acelerómetro, la velocidad angular del giroscopio, un vector con 561 características que es producto de operaciones con los datos con los sensores medidos en periodos temporales de 2.56 segundos a 50 Hz, un identificador de la persona que está realizando

[1] Aquí ponemos un ejemplo de una persona realizando las 6 acciones: `https://www.youtube.com/watch?v=XOEN9W05_4A`.

[2] Human Activity Recognition on Smartphones using a Multiclass Hardware-Friendly Support Vector Machine.

la acción y la acción en sí.

Para todo esto nos encontramos con un conjunto de datos que ya ha sido preprocesado para reducir el ruido.

Por un lado, el conjunto está normalizado en el intervalo [-1,1] y además tiene categorizadas las etiquetas desde el valor 1 hasta el 6 en los que cada uno de estos valores representa una acción.

Para la representación de la curva ROC, hemos tenido que binarizar las categorías, sin embargo, para el ajuste no hemos utilizado las etiquetas binarizadas.

3. Selección de clases de funciones a usar

La clase de funciones nos permite junto a los datos del conjunto de train ajustar nuestro modelo. Para esto hemos utilizado las siguientes clases de funciones:

- Linear loss:
$$h(x) = sign(w^T x)$$
- Sigmoid - Probabilistic loss:
$$h(x) = \theta(w^T x)\theta(s) = \frac{e^s}{1+e^s}$$
- Hinge loss:
$$\ell(y) = max(0, 1 - t \cdot y)$$
- Squared Hinge loss:
$$\ell(y) = (max(0, 1 - t \cdot y))^2$$
- Gini impurity:
$$H(X_m) = \sum_k p_{mk}(1 - p_{mk})$$
- Exponential loss:
$$E = \sum_{i=1}^{N} e^{-y_i C_m(x_i}$$

4. Definición de los conjuntos de *training*, *test* y validación

Encontramos que el conjunto viene dividido en dos conjuntos, en los que el *training* posee el 70 % de los datos y el *test* el 30 %.

Para la validación usaremos tanto *bagging* como validación cruzada, y en esta última usaremos un conjunto de validación del 20 %.

5. Discutir la necesidad de regularización y en su caso la función usada para ello

Para entrar un poco en contexto, vamos a explicar qué es la regularización. Este concepto tiene como finalidad solucionar problemas con el sobreajuste o que no se ajuste lo suficiente. Sabiendo esto, siempre que sea posible deberíamos de utilizarlo a la hora de obtener un modelo. En nuestro caso, hemos optado por utilizar Lasso, Ridge y Elastic-Net:

- Ridge: Realiza una aproximación a cero de los coeficientes de los predictores y no excluye ninguno de estos.
- Lasso: Realiza también una aproximación a cero pero excluye predictores.
- Elastic-net: Es una combinación lineal de Ridge y Lasso.

6. Definir los modelos a usar y estimar sus parámetros e hyperparámetros.

Los modelos que hemos decidido utilizar son tanto modelos lineales como modelos no lineales. Hemos realizado esto con la finalidad de ver la diferencia de clasificación entre ambos modelos. Para calcular dichos modelos, hemos utilizado los siguientes algoritmos:

- **SGD**:
  - Regularización: Lasso, Ridge y Elastic-net
  - λ : 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1
  - Máximo de iteraciones: 10

    - Pérdida: Log y Perceptrón

- **SVM**:
    - Regularización: Lasso y Ridge
    - C : 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1
    - Máximo de iteraciones: 10
    - Pérdida: Hinge y Squared Hinge

- **Random Forest**:
    - Número de árboles: 250
    - Profundidad máxima : 25
    - Máximo de nodos hoja: 20

    Hemos aplicado esta profundidad y este máximo de nodos hoja para realizar poda en nuestros árboles. Y esto nos va a favorecer a ganar estabilidad y que no presente mucha incertidumbre. Esto quiere decir que evitaremos el sobreajuste.

- **Boosting**:
    - Número de estimadores: 10
    - learning-rate: 10^{-3}, 10^{-2}, 10^{-1}, 1

- **Redes neuronales**:
    - Capas: 9
    - λ : 10^{-4},10^{-3},10^{-2},10^{-1}
    - learning rate inicial: 10^{-3}, 10^{-2}, 10^{-1}

Para la selección de los mejores hyperparámetros hemos utilizado el método grid_search con validación cruzada usando como score el número de aciertos y como porcentaje del conjunto de validación del 20 %.

Los algoritmos que hemos implementado calculan la estimación de los parámetros.

7. Selección y ajuste modelo final.

Tras la experimentación realizada con los diferentes algoritmos y los errores que hemos obtenido, hemos visto que con todos los algoritmos obtenemos modelos que nos ofrecen un buen E_{in}, aunque con algunos sacrificamos un poco el tiempo.

Finalmente hemos escogido el algoritmo de Support Vector Machine ya que nos ofrece sin duda alguna el mejor resultado en un tiempo más que razonable. Además de haber probado con otros algoritmos más complejos, pero hemos obtenido peores resultados tanto en error como en tiempo de ejecución. Por último, cabe destacar que los autores utilizaron este algoritmo para probar que se obtenía un modelo que ofrecía buenos resultados.

8. Discutir la idoneidad de la métrica usada en el ajuste

La métrica que hemos utilizado para el ajuste es la tasa de clasificación (accuracy). Esta métrica es la más idónea ya que estamos tratando un problema de clasificación y por esto creemos que es de vital importancia conocer los errores de clasificación que hemos obtenido.

Como añadido a este problema hemos realizado la matriz de confusión junto con la curva ROC correspondiente en nuestro modelo para conocer la correlación que hay entre los falsos positivos (en el eje x) y los verdaderos positivos (eje y).

9. Estimación del error E_{out} del modelo lo más ajustada posible.

El error de salida, nos lo proporciona el modelo de SVM con los datos de test y los parámetros calculados con grid-search y cross-validation, que indicaremos a continuación:

```
Best classifier:  LinearSVC(C=0.1, class_weight=None, dual=False,

fit_intercept=True, intercept_scaling=1, loss='squared_hinge',

max_iter=10, multi_class='ovr', penalty='l2', random_state=None,
```

```
tol=0.0001, verbose=0)

Eout 0.0414
```

Utilizando el mejor clasificador que hemos obtenido, nos da un error de clasificación en el conjunto de test de un 4.14 %.

10. Discutir y justificar la calidad del modelo encontrado y las razones por las que considera que dicho modelo es un buen ajuste que representa adecuadamente los datos muestrales.

Para justificar la calidad de nuestro modelo tan solo debemos observar los resultados obtenidos al ajustarlo. Podemos ver que obtenemos un E_{in} del 0.9892 en poco más de un minuto.

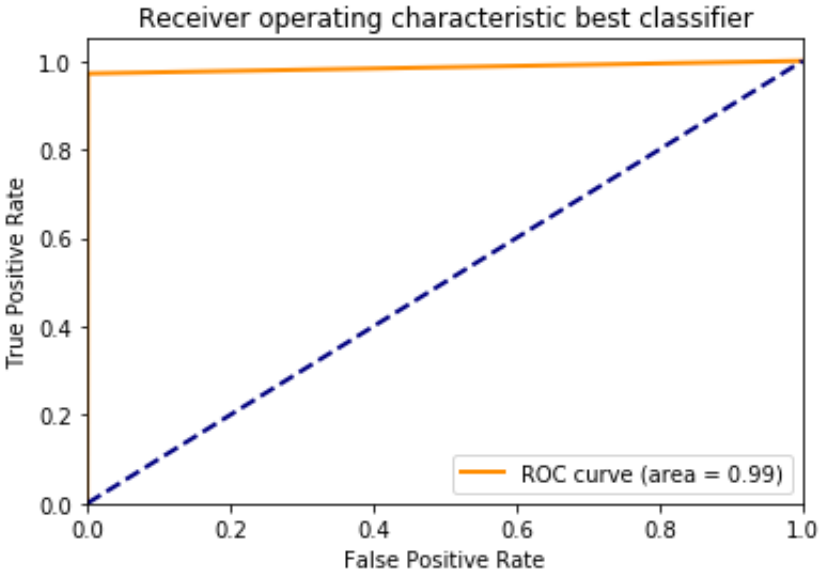
Dado que nos encontramos ante un problema con multiclase y este modelo permite la clasificación de dichos problemas usando la estrategia *One Versus Rest*, ya sabemos que el resultado obtenido será capaz de clasificar correctamente con una alta probabilidad. Para justificar todo esto hacemos también uso de varias métricas como la curva ROC y la matriz de confusión.

CONFUSION MATRIX

$$\begin{pmatrix} 482 & 2 & 12 & 0 & 0 & 0 \\ 46 & 419 & 6 & 0 & 0 & 0 \\ 43 & 52 & 325 & 0 & 0 & 0 \\ 0 & 0 & 0 & 422 & 54 & 1 \\ 0 & 0 & 0 & 82 & 512 & 0 \\ 0 & 0 & 0 & 0 & 0 & 537 \end{pmatrix}$$

Como podemos ver en la matriz, la diagonal principal contiene la mayoría de los datos, lo que supone que la mayoría de los datos han sido clasificados correctamente, sin embargo, encontramos algunas posiciones cercanas a la diagonal que presentan algunos datos, aunque la mayoría está a cero, lo cual para ser un problema multiclase es muy bueno.

También enseñaremos la curva ROC de nuestro mejor modelo:



La diagonal divide el espacio ROC. Los puntos por encima de la diagonal representan buenos resultados de clasificación, puntos por debajo de la línea de los resultados malos.

El mejor método posible de predicción se situaría en un punto en la esquina superior izquierda, o coordenada (0,1) del espacio ROC, representando un 100 % de sensibilidad (ningún falso negativo) y un 100 % también de especificidad (ningún falso positivo).

Como podemos ver, nuestro modelo nos ofrece una curva que se acerca a ese 100 % de sensibilidad y de especificidad. Por lo que podemos decir que nuestro modelo clasifica muy bien los datos.