Máster en Big Data Analytics

Curso 2017-2018

# Introducción al Aprendizaje Automático

*Práctica 8*

*Support Vector Machines*

Jon Ander Gómez Adrián

jon@dsic.upv.es

Departament de Sistemes Informàtics i Computació

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

## Índice



## 1. Motivación

Las *Support Vector Machines* (SVM) son una técnica que básicamente se utiliza para tareas de clasificación, aunque también se utiliza en problemas de

regressión. En la presente práctica sólo vamos a aplicar las SVM a problemas de clasificación.

Así como en algunas de las técnicas vistas anteriormente se ha ofrecido una implementación, no eficiente pero útil didácticamente, para las SVM no será así porque dependen de muchos parámetros de ajuste y su implementación no es nada fácil, de hecho, desde el propio toolkit *Scikit-Learn* se utiliza la implementación en C/C++ *libSVM*, que está disponible para su descarga en el siguiente enlace y forma parte del repositorio de Linux.

`http://www.csie.ntu.edu.tw/~cjlin/libsvm/`

Lo importante de esta práctica es comprender el funcionamiento de las SVM aplicadas a distintos *datasets* y entender el efecto de cada uno de los hiperparámetros que pueden variarse. La principal tarea al utilizar SVM suele ser ajustar apropiadamente la combinación de los hiperparámetros, siendo uno de ellos el tipo de función kernel a utilizar.

Cuadro 1: *Funciones kernel típicas que se utilizan en las SVM*

| | | |
|---|---|---|
| Lineal (linear) | $\langle x, x' \rangle$ | |
| Polinomial (poly) | $(\gamma \langle x, x' \rangle + r)^d$ | $d$ es el grado y $r \geq 0$ |
| Radial Basis Functions (rbf) | $e^{-\gamma \lvert x-x' \rvert^2} = e^{-\frac{\lvert x-x' \rvert^2}{2\sigma^2}}$ | $\gamma > 0$ |
| Sigmoid | $tanh(\gamma \langle x, x' \rangle + r)$ | $\gamma > 0$ y $r \geq 0$ |

La Tabla 1 muestra las funciones kernel utilizadas habitualmente, en ellas se aprecian los parámetros ajustables: grado $d$, $\gamma$ y $r$. $r$ es el término independiente de un polinomio, para el kernel polinómico si es distinto de cero provoca que la expansión incluya los términos de grado menor a $d$. Pero también puede verse como cierto desplazamiento u offset, como es en el caso del kernel de tipo sigmoide.

# 2. Objetivos

## 2.1. Conceptuales

- Descubrir que ciertas técnicas, como son las SVM, tienen tiempos de ejecución en la fase de aprendizaje que son considerablemente largos para corpus medianamente grandes.

  Esto debe llevar a tomar las decisiones correctas según el problema a resolver, pues debe saberse si compensa obtener mejoras relativas con respecto a otras técnicas que requieren de mucho menos cómputo.

  Las técnicas de *Machine Learning* que consiguen buenos resultados y que son aplicables a muchos problemas, porque mejoran significativamente los resultados de otras técnicas, suelen tener la contrapartida del tiempo necesario para su entrenamiento.

- Entender que las SVM son una técnica *memory-based* porque basa su predicción en parte de las muestras de aprendizaje. No las guarda todas, sólo las que se consideran **vectores de soporte**.

  La parte difícil y costosa del proceso de aprendizaje es determinar qué muestras formarán parte del conjunto de **vectores de soporte** y qué valor tomará el coeficiente $a_n$ asociado a cada una de ellas.

## 2.2. En el manejo de utilidades

- Saber preparar baterías de pruebas para dejar procesos en marcha o en paralelo que combinen los parámetros ajustables y encuentren la configuración más idónea para un problema concreto.

  Para probar distintas combinaciones se utilizan listas de Python para expandirlas en bucles de tipo 'for' y por cada combinación crear un clasificador, entrenarlo y comprobar su precisión.

# 3. Tareas

## 3.1. Tarea 1

Ejecutad repetidas veces

```
python svm_sobre_sinteticas.py
```

Estudiad el código para entender cómo se prueban las combinaciones elegidas. En el siguiente trozo de código del programa anterior se preparan las listas:

```
kernels = [ 'rbf', 'linear', 'poly', 'sigmoid' ]
degrees = [ 1, 2, 3, 4, 5 ]
gammas = [ 0.0 ]
values_of_C = [ 1.0e-3, 1.0e-2, ... 1.0e+2, 1.0e+3 ]
coef0 = 1.0
```

En los bucles `for` que siguen se realizan todas las comprobaciones. Nótese cómo si cierto parámetro no es aplicable según el tipo de kernel no se utiliza más que una vez. Estratégicamente se hace uso de la variable temporal `_degrees` que es la lista de grados posibles para el kernel de tipo polinomial y sólo valor 1 para el resto de kernels.

```
for kernel in kernels:
    if kernel == 'poly' :
        _degrees = degrees
    else:
        _degrees = [1]
    for degree in _degrees:
        for gamma in gammas:
            for C in values_of_C:
                classifier = svm.SVC( kernel=kernel, ...
                classifier.fit( X_train, Y_train )
                y_pred = classifier.predict( X_test )
```

## 3.2. Tarea 2

Descargad y ejecutad el código del siguiente enlace.

Este código descargará un *dataset* de caras y utilizará clasificadores basados en SVM para reconocimiento facial o de caras.

`http://scikit-learn.org/stable/auto_examples/applications/`
`/face_recognition.html`

Mientras se ejecuta y después de que acabe la ejecución, repasad el código fijando la atención en los siguientes puntos:

1. Donde realiza el particionado en *training* y *test* mediante la función `train_test_split()`.

2. Donde realiza la transformación mediante PCA para reducir la dimensionalidad de las muestras.

3. Exploración de diferentes valores de algunos de los parámetros ajustables, como són $C$ y $\gamma$. Nótese como *degree* no se utiliza y el kernel es siempre el mismo, de tipo `rbf`.

   El uso de `GridSearchCV()` realiza la exploración de las distintas combinaciones aplicando la técnica de *cross-validation* para sólo con muestras de *training* validar cada modelo y elegir el mejor.

4. El método `classification_report()` muestra los resultados de predicción utilizando distintas métricas estándar y su combinación además de indicar el número de vectores de soporte por cada clase y en total.

   ¿Cuántos elementos de los seleccionados son relevantes?

$$precision = \frac{TP}{TP + FP}$$

   ¿Cuántos elementos relevantes han sido seleccionados?

$$recall = \frac{TP}{TP + FN}$$

   $TP$ = *True Positives*
   $FP$ = *False Positives*
   $FN$ = *False Negatives*
   $TN$ = *True Negatives*

   *F1-score*

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}$$

5. Por último, el método `confusion_matrix()` muestra la matriz de confusión para ver el grado de confusión de las caras de cada persona con respecto a las demás.

## 3.3. Tarea 3

Adaptad el código revisado en la tarea anterior para aplicarlo sobre el *dataset* MNIST.

¿Qué resultados se obtienen?

¿Se puede cambiar el PCA por otro que no sea aleatorio?

¿Cambian los resultados significativamente?

¿Qué efecto tiene utilizar un número de componentes para el PCA distinto a 150, tanto en las caras como en los dígitos?

Provad por ejemplo con: 30, 50, 80, 100, 120 y también con 200.

Como con MNIST tardará mucho, reducid el tamaño del training set y probad para ver qué combinación de $C$ y $\gamma$ da mejores resultados para acotar el rango de posibles valores para estos parámetros y entonces probad con todo el *training set*.

# 4. Conclusiones

Esta práctica, que debe acabarse en casa, muestra cómo trabajar la clase `SVC` de *Scikit Learn* para construir clasificadores tipo SVM, y también enseña como utilizar utilizades disponibles en *Scikit Learn* para presentar los resultados.

También resulta interesante el uso de `train_test_split()` para generar aleatoriamente particiones de un *dataset* en *training set* más *test set*.

Sería interesante, una vez completada la tarea 3, aplicar el `GridSearchCV()` con más valores posibles de los parámetros ajustables con todo el *training set* de los dígitos. Le llevará días, pero si podéis probarlo en vuestros ordenadores de casa podréis sacar conclusiones interesantes y aprender como explorar, casi exhausitivamente, las combinaciones de parámetros que consideréis de interés a tareas de casos reales de uso.

# 5. Bibliografía


[1] S. authors. Deep Learning. `http://www.deeplearning.net`, 2015. [Online: accessed June 2015].

[2] Y. Bengio, I. J. Goodfellow, and A. Courville. Deep learning. Book in preparation for MIT Press, 2015.

[3] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[4] C. M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2006.

[5] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Addison-Wesley, second edition, 2000.

[6] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):2–42, 2006.

[7] G. Hinton. A practical guide to training restricted boltzmann machines, 2010.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, JMLR*, 12:2825–2830, 2011.

[9] Wikipedia. Machine Learning. `http://en.wikipedia.org/wiki/Machine_learning`, 2015. [Online: accessed April 2015].