Aprendizaje Automático
Trabajo 3:
Programación

Rafael Vázquez Conejo

Mayo 2020

Índice

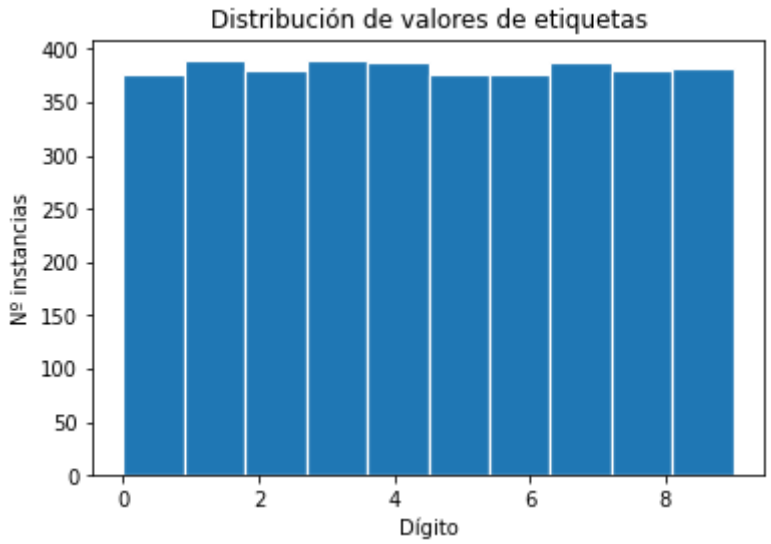
1. Clasificación (Optical Recognition of Handwritten Digits Data Set)

1.1. Comprender el problema a resolver.

Comenzamos estudiando el problema de clasificación. Este problema se realiza sobre una base de datos de dígitos manuscritos, compuesta por 5620 imágenes que representan dígitos del 0 al 9, siendo cada instancia representada por 64 atributos.

Tras presentar los datos, el objetivo de este problema (f) consiste en predecir a qué dígito decimal (0 al 9) corresponde un dígito escrito a mano. Las etiquetas (Y) son dígitos binarios, con valores comprendidos en el mismo rango que dígitos posibles, es decir de [0,9].
Los datos (X) corresponden a vectores de 64 enteros, tomando valores comprendidos entre 0 y 16, que corresponden a los valores obtenidos tras aplicar una convolución sobre los píxeles de la imagen.

Un problema que se podría presentar es que el número de ejemplos de las clases presentes en el conjunto de entrenamiento no fuera equivalentes. Por suerte este no es el caso y no tenemos una clase con pocas instancias.



Podemos observar lo comentado, los valores de las etiquetas están equilibrados. Esta idea es esencial, pues es un punto muy importante a la hora de determinar el método de validación.

Finalmente comentar que se trata de un problema de aprendizaje supervisado, ya que conocemos las clases a las que pertenecen las instancias antes de su clasificación.

1.2. Selección de las clase/s de funciones a usar. Identificar cuáles y porqué.

Las funciones utilizadas son las funciones lineales que utilizo en cada modelo. Dichos modelos son modelos lineales, tal y como indica el guión de la práctica.

1.3. Fijar conjuntos de training y test que sean coherentes.

Partimos de dos archivos, en cuanto a división de los datos de entrenamiento (.tra) y prueba (.tes), una división inicial que nos supone una gran ventaja, ya que no es necesario definir una función para su división.

1.4. Preprocesado los datos: codificación, normalización, proyección, etc.

Esta es una fase muy importante, ya que en ella partimos de la idea de obtener un conjunto de datos de mejor calidad para extraer el mejor conocimiento.

Para ello trataremos de reducir de forma eficiente el tamaño del conjunto de datos, no siempre cuanto más variables mejor será nuestro modelo, ya que puede darse el caso de que encontremos variables erróneas o irrelevantes, lo que produciría un aprendizaje incorrecto.

También se intentará reducir la complejidad del conjunto de datos, normalizando los datos, eliminando los datos de baja variabilidad o duplicados…

- MaxAbsScaler(): he utilizado esta función de *sklearn* para escalar cada característica por su máximo valor absoluto. Esta función escala y traduce cada característica individualmente de modo que el valor absoluto máximo de cada característica en el conjunto de entrenamiento será de 1. No desplaza o centra los datos, y por lo tanto no se destruye ninguna dispersión
- Normalize(): para normalizar aplico esta función, que escala las muestras individuales para tener una norma de unidad. También es una función de *sklearn.*

Primero asignaremos un escalador y lo aplicaremos al conjunto de características, luego normalizaremos los datos del conjunto de datos. Relizaremos este proceso tanto en el conjunto de training como en el conjunto test.

Estas dos modificaciones serán muy importante, pues este conjunto tiene muchas características y muchas de ellas con valores que varían poco o constantes. De esta forma reduciremos considerablemente el número de características a evaluar.

1.5. Fijar la métrica de error a usar. Discutir su idoneidad para el problema.

He desarrollado dos tipos distintos de métricas, la matriz de confusión y el informe de clasificación.

- La matriz de confusión: nos muestra el comportamiento de nuestro modelo con respecto a las diferentes clases de nuestro conjunto de datos. La represento por medio de una tabla en la que las filas representan las etiquetas reales del conjunto, y las columnas representan el número de predicciones de cada clase.

  Esta nos permite observar cuantos valores han sido bien clasificados, que serán los que se encuentran en la diagonal de la matriz, mientras que el resto de valores será el número de datos que se han clasificados respecto a la clase real.

- El informe de clasificación: nos muestra de nuevo una tabla con distintos parámetros sobre cómo ha clasificado para cada uno de los números.

  Al de la tabla se presentan los 3 valores más relevantes: micro_avg, que representa la media del total de verdaderos positivos, falsos negativos y falsos positivos; macro_avg, representa la media no ponderada de todas las etiquetas; y weighted_avg, representa la media ponderada de todas las etiquetas.

-

1.6. Discutir la técnica de ajuste elegida.

Respecto al modelo lineal he decidido utilizar la técnica de Gradiente Descendente Estocástico, principalmente porque su funcionamiento e idea me quedaron muy claro tras la primera práctica.

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

Realizaremos modificaciones para que el ajuste sea mejor.

Para eliminar las características no relevantes del conjunto de datos utilizaremos una regularización “l2” ó “Ridge”. Es el método de regularización más utilizado para los problemas que no poseen una única solución, y esta añade una penalización equivalente al cuadrado de la magnitud de los coeficientes. Este regularización reduce los coeficientes a un valor cercano a 0, a diferencia de la regularización “Lasso”, en la que algunos coeficientes si pueden ser reducidos a 0.

1.8. Identificar los modelos a usar.

Se han utilizado tres tipos de modelos, siendo estos Perceptron, Regresión Logística y Gradiente Descendente Estocástico (SGD).

Partimos de una serie de parámetros iguales que reciben los tres modelos, estos valores se modificarán más adelante para una mejor valoración de resultados. Estos parámetros son:

- Max_iter → Número máximo de iteraciones, por defecto establezco 2000.

- Epsilon → Umbral en el que se detendrá la ejecución, por defecto el valor es 10^{-4}
- Penalty → Corresponde a la regularización, cómo ya hemos mencionado en el apartado anterior la utilizada será “Ridge” para los tres casos, tras ello con el que elija mejor utilizaré también la “Lasso” para comparar.

A continuación presentaremos los tres modelos.

1. Gradiente Descendente Estocástico (SGD) → el objetivo de este algoritmo es minimizar funciones. Este comienza en un punto y va descendiendo por la pendiente más pronunciada. Este no actúa sobre la muestra completa sino que toma una parte de ella (minibatch).

   La función creada SGDClassifier(penality, max_iter, lr, epsilon), posee los parámetros ya mencionados junto con “lr”, learning rate, que por defecto tendrá el valor 0,01. Para imlementarlo utilzo linear_model de “sklearn”.

2. Regresión Logística → este algoritmo utiliza el gradiente descendente para hacer la clasificación, pero con una serie de diferencias. La función que implemento es RegresionLogistica(penalty, max_iter), e implementa el algoritmo de regresión logística gracias a “sklearn”, mediante LogisticRegression.

3. Perceptron → este algoritmo calcula el hiperplano que divide una muestra clasificada de forma binaria. La función que utilizo y creo es Perceptron(penalty, max_iter, epsilon), parámetros que ya hemos comentado. De nuevo utilizo “Perceptron” de “linear_model” de “sklearn”.

1.9. Estimación de hiperparámetros y selección del mejor modelo.

Los hiperparámetros de cada función han sido mencionados y comentados en el apartado anterior.

Respecto a la selección del mejor modelo, utilizaremos las métricas definidas anteriormente en cada uno de los modelos:

Informe de casillas de SGD

	precision	recall	f1-score	support
0	1.00	0.98	0.99	177
1	0.91	0.91	0.91	182
2	0.97	0.98	0.97	177
3	0.97	0.94	0.95	183
4	0.96	0.97	0.96	181
5	0.91	0.98	0.94	182
6	0.99	0.98	0.98	181
7	0.97	0.93	0.95	179
8	0.89	0.87	0.88	174
9	0.89	0.90	0.90	180
accuracy			0.94	1796
macro avg	0.94	0.94	0.94	1796
weighted avg	0.94	0.94	0.94	1796

Matriz de Confusión de SGD

```
[[174   0   0   0   1   2   0   0   0   0]
 [  0 165   4   0   0   0   0   0   5   8]
 [  0   1 173   1   0   0   0   1   1   0]
 [  0   0   1 172   0   3   0   3   1   3]
 [  0   2   0   0 175   0   0   1   3   0]
 [  0   0   0   0   0 179   1   0   0   2]
 [  0   1   0   0   2   0 177   0   1   0]
 [  0   0   0   0   1   7   0 166   2   3]
 [  0  10   0   3   0   4   1   0 152   4]
 [  0   3   1   2   4   2   0   0   6 162]]
```

Informe de casillas de Regresión Logística

	precision	recall	f1-score	support
0	1.00	0.98	0.99	177
1	0.91	0.93	0.92	182
2	0.97	0.97	0.97	177
3	0.99	0.92	0.95	183
4	0.95	0.97	0.96	181
5	0.91	0.98	0.94	182
6	0.99	0.98	0.99	181
7	0.97	0.91	0.94	179
8	0.90	0.90	0.90	174
9	0.88	0.94	0.91	180
accuracy			0.95	1796
macro avg	0.95	0.95	0.95	1796
weighted avg	0.95	0.95	0.95	1796

Matriz de Confusión de Regresión Logística

```
[[173   0   0   0   1   3   0   0   0   0]
 [  0 169   0   0   0   1   0   0   5   7]
 [  0   3 172   0   0   0   0   1   1   0]
 [  0   0   4 168   0   3   0   3   3   2]
 [  0   1   0   0 176   0   0   1   3   0]
 [  0   0   1   0   0 178   1   0   0   2]
 [  0   1   0   0   2   0 177   0   1   0]
 [  0   0   0   0   2   5   0 163   2   7]
 [  0   9   0   0   0   3   0   0 156   6]
 [  0   2   0   1   4   2   0   0   2 169]]
```

Informe de casillas de Perceptron

	precision	recall	f1-score	support
0	0.98	0.98	0.98	177
1	0.57	0.96	0.71	182
2	0.99	0.79	0.88	177
3	0.99	0.80	0.88	183
4	0.96	0.86	0.91	181
5	0.97	0.90	0.93	182
6	0.99	0.80	0.88	181
7	0.96	0.89	0.92	179
8	0.73	0.83	0.78	174
9	0.85	0.87	0.86	180
accuracy			0.87	1796
macro avg	0.90	0.87	0.87	1796
weighted avg	0.90	0.87	0.87	1796

Matriz de Confusión de Perceptron

```
[[173   0   0   0   1   0   0   0   3   0]
 [  0 175   0   0   0   0   0   0   0   7]
 [  0  28 140   1   0   0   0   1   7   0]
 [  2   6   0 146   0   1   0   5  15   8]
 [  0  24   0   0 156   0   0   0   1   0]
 [  1   9   1   0   0 163   2   0   5   1]
 [  0  26   0   0   2   0 145   0   8   0]
 [  0   3   0   0   3   4   0 159   1   9]
 [  0  28   0   0   0   0   0   0 144   2]
 [  0  10   0   0   1   0   0   0  13 156]]
```

Podemos observar que hemos obtenido buenos resultados con todos los modelos, todos con una media de aciertos superior al 90%. Pero entre todos los que mejor han clasificado son primero la Regresión Logística con un 95% de media y SGD con un 94% de media. Entre estos dos que poseen unos resultados de similar calidad me quedaría con SGD cómo mejor modelo, ya que este requiere un menor tiempo que la Regresión Logística.

En la matriz de confusión Cada fila corresponde a cada dígito (0-9) con las columnas lo mismo (0-9), es decir, si nos fijamos en la matriz de confusión de SGD, en la fila 9 y la columna 1, podemos ver que el dígito 9 se ha clasificado 3 veces como dígito 1. Esto nos permite analizar en mayor profundidad la clasificación realizada. De nuevo podemos ver que en Perceptron es donde más errores se cometen.

1.10. Estimación por validación cruzada del error E out del modelo. Compárela con E test , ¿que conclusiones obtiene?

Para ello calcularé:

$E_{out} \leq E_{test} + \sqrt{\frac{1}{2N} \ln \frac{2H}{\delta}}$ Partiendo de que $E_{test} = 1 - precisión(0.94)$, siendo N = 1797, H = 1 por estar en test, y con un $\delta = 0.05$, obtenemos:

$$E_{out} \leq 0.09204$$

En conclusión un escalado y una normalización junto con el modelo SGD han sido suficientes para obtener unos buenos resultados, tanto en el conjunto de entrenamiento como en el test. Existe suficiente separabilidad en el problema como para lograr un buen desempeño.

11. Justificación

El modelo que tomaría como mejor es el SGD, ya que aunque la Regresión Logística consigue resultados mínimamente mejores los tiempos de ejecución son más reducidos en el SGD. De nuevo si se tratara sobre este mismo problema tomaría Regresión Logística, puesto que los resultados son mejores, pero si fuera el caso de un problema de mayor dimensiones, la decisión sería SGD, cuyos tiempos serían mucho más inferiores.

Respecto a la calidad del modelo, creo que esta es correcta, un modelo que consigue un 94% de aciertos me parece más que bueno. Es cierto que si nos encontramos ante una situación en la que necesitamos un 100% de precisión no lo utilizaría, casos como problemas médicos no serían posibles de resolver por este modelo, pues un cambio en un número podría poner en riesgo la vida de un paciente.

2. Regresión

2.1. Comprender el problema a resolver.

Este conjunto de datos contiene mucha información y datos sobre delitos y comunidades, como el porcentaje de la población considerada urbana, y el ingreso familiar medio, e involucra a la policía, como el número per cápita de policías y el porcentaje de oficiales asignados a las unidades de drogas.

La idea de este conjunto es predecir el número total de crímenes violentos por cada 100.000 habitantes. Para ellos se parte de un total de 1994 instancias y 128 atributos. Siendo la columna denominada ViolentCrimesPerPop nuestra Y.

2.2. Selección de las clase/s de funciones a usar. Identificar cuáles y porqué.

Las funciones utilizadas son las funciones lineales que utilizo en cada modelo. Dichos modelos son modelos lineales, tal y como indica el guión de la práctica. Estas funciones serán empleadas en nuestro problema junto con el modelo “Ridge” de regularización que utilizaré en uno de los modelos.

Todos los datos numéricos se normalizaron en el rango decimal 0.00-1.00 utilizando un método de agrupación de intervalos iguales sin supervisión.
Los atributos conservan su distribución y sesgo (de ahí que, por ejemplo, el atributo de población tenga un valor medio de 0.06 porque la mayoría de las comunidades son pequeñas).

2.3. Fijar conjuntos de training y test que sean coherentes.

En este problema a diferencia del anterior solamente se nos proporciona un fichero con todos los datos juntos, por ello tendremos que realizar una división en entrenamiento y test. La división la realizaremos gracias a la función “train_test_split” de “sklearn", la cual se encarga de dividir el conjunto de datos de manera que el conjunto de entrenamiento contiene el 80% del dataset y el conjunto de prueba el 20% que resta. Este tamaño se especifica en los parámetros de la función, indicando “test_size=0.2”.

Antes de realizar esta separación hemos tenido que realizar una serie de modificaciones sobre nuestro conjunto de datos que trataremos en el siguiente punto.

2.4. Preprocesado los datos: codificación, normalización, proyección, etc.

Como ya hemos mencionado esta vez partimos de un sólo archivo con el conjunto de datos, archivo que leemos con la función “read_csv” de pandas, función que nos permite añadirle la cabecera a las

columnas del conjunto. El nombre de estas cabeceras lo obtenemos del archivo “communities.name”, en el que nos viene información sobre este dataset.

Una vez leído el archivo eliminaremos las columnas que nos sean innecesarias en nuestro proceso. Comenzamos con las 5 primeras columnas, ya que estas no nos proporciona información estadística que nos interese. Tras ello tendremos que eliminar aquellas columnas en las que aparezcan valores con el valor “?”, pues esto nos provoca un error a la hora de pasar de “string” a “float”

2.5. Fijar la métrica de error a usar. Discutir su idoneidad para el problema.

Para evaluar cada modelo de regresión utilizaremos los errores MAE y MSE:

- Mean Absolut Error (MAE) → es una medida de errores absolutos $e_i=|y_i-x_i|$ dónde y_i es la predicción y x_i es el valor esperado. MAE calcula el error como el promedio de las diferencias absolutas entre los valores de las etiquetas reales y las etiquetas predecidas.
- Mean Square Error (MSE) → es una medida de errores absolutos, pero en este caso al cuadrado, es decir, $e_i=|y_i-x_i|^2$. MSE mide el error promedio cuadrático de nuestras predicciones, es decir, para cada punto calcula la diferencia entre las predicciones y las etiquetas, y obtiene el promedio de dicha diferencia. Cuando mayor sea este valor, peor será nuestro modelo.

También usaremos el coeficiente de determinación, similar a MSE, que nos permitirá obtener el verdadero error cuadrático de nuestro modelo, puesto que si el valor de salida es muy grande MSE lo sesga, pero el coeficiente de determinación no.

2.6. Discutir la técnica de ajuste elegida.

Respecto al modelo lineal he decidido utilizar la técnica de Gradiente Descendente Estocástico, principalmente porque su funcionamiento e idea me quedaron muy claro tras la primera práctica.

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

Realizaremos modificaciones para que el ajuste sea mejor.

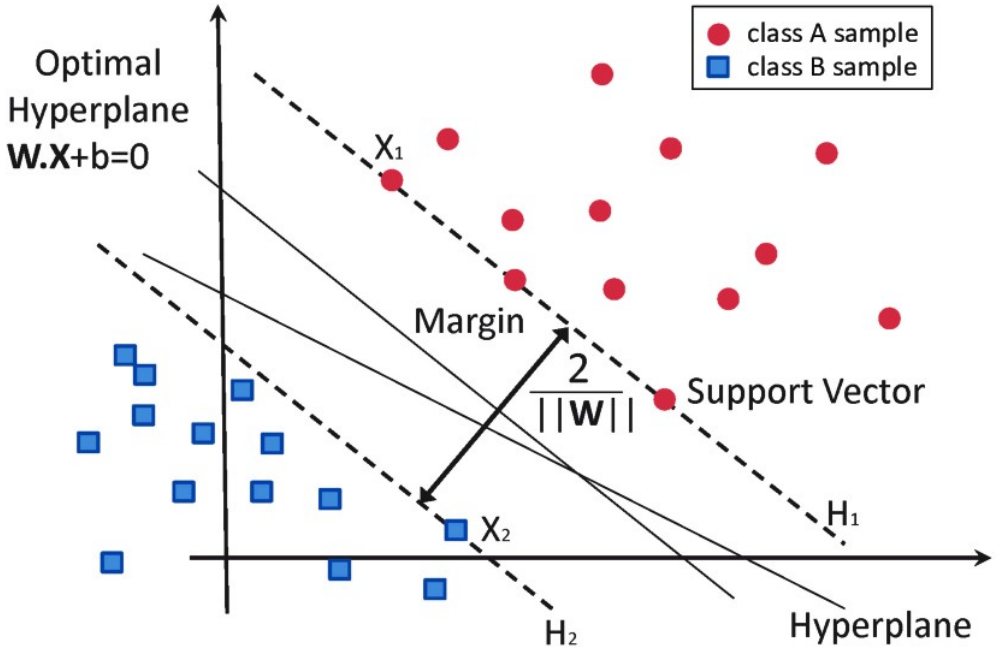
Como ya mencionamos en el problema de clasificación utilizaremos una regularización “l2” ó “Ridge”, el cuál añade una penalización equivalente al cuadrado de la magnitud de los coeficientes. Este regularización reduce los coeficientes a un valor cercano a 0, a diferencia de la regularización “Lasso”, en la que algunos coeficientes si pueden ser reducidos a 0.

2.8. Identificar los modelos a usar.

Los modelos que utilizaré son: Regresión Lineal, SGDRegressor y Support Vector Machine (SVR).

- Regresión Lineal → utiliza mínimos cuadrados ordinarios que realizan la función de predecir los valores. La función definida es RegresionLineal() y no recibe parámetros. Esta utiliza la función “LinearRegression()” de “sklearn”
- Gradiente Descendente Estocástico (SGD) → tiene las bases del algoritmo ya presentado en la clasificación, solo que posee algunas variaciones para su adaptación al problema de regresión. La función que utilizo se denomina SGDRegressor(penalty, max_iter, lr, epsilon), parámetros idénticos al SGD de clasificación. Esta utiliza la función “SGDRegressor()” de “sklearn”
- Support Vector Machine (SVR): este algoritmo no ha sido visto en las prácticas anteriores. Este busca un hiperplano que separe de forma óptima los puntos de una clase de otra, es decir, busca el hiperplano que tenga la máxima distancia (margen) con los puntos que estén más cerca de él.

  Mostramos una imagen para su mejor compresión:



  La función utiliza es LinearSVR(C, max_iter, epsilon), y utiliza la función “LinearSVR()” de “sklearn”

2.9. Estimación de hiperparámetros y selección del mejor modelo.

Respecto a los hiperparámetros, en el caso de SGDRegressor tenemos:

- Max_iter → Número máximo de iteraciones, por defecto establezco 2000.
- Epsilon → Umbral en el que se detendrá la ejecución, por defecto el valor es 10^{-4}
- Penalty → Corresponde a la regularización, cómo ya hemos mencionado en el apartado anterior la utilizada será “Ridge”.

En el caso de SVR no tenemos “penalty”, pero si un nuevo parámetro junto con “max_iter” y “epsilon”, que es el error permitido para este pasillo, es decir, contra más pequeño sea el valor, mayor error va a permitir y más pequeño será el pasillo que divide los datos, este parámetro es “C”.

Veamos ahora los resultados de cada modelo:

```
REGRESION LINEAL
Mean Absolut Error: 0.08948666018359637
Mean Squared Error: 0.017182393976087187

Coefficient of determination :  0.6848138308758629
```

```
GRADIENTE DESCENDENTE ESTOCASTICO
Mean Absolut Error: 0.09369431979136528
Mean Squared Error: 0.019128473910896126

Coefficient of determination :  0.6491158088007424
```

Support Vector Machine (SVR)

```
Mean Absolut Error: 0.0872878753171733
Mean Squared Error: 0.01862888914558716

Coefficient of determination :  0.6582799688444275
```

Como podemos observar el mejor resultado es obtenido por el modelo SGD seguido por SVR.

2.10. Estimación por validación cruzada del error E out del modelo. Compárela con E test , ¿que conclusiones obtiene?

Para ello calcularé:

$$E_{out} \leq E_{test} + \sqrt{\frac{1}{2N} \ln \frac{2H}{\delta}}$$

Partiendo de que $E_{test} = 1 - precisión(0.649116)$, siendo N = 399, porque recordemos que hemos asignado un 20% a test del total (1994)

H = 1 por estar en test, y con un $\delta = 0.05$, obtenemos:

$$E_{out} \leq 0.41887$$

En conclusión hemos obtenido unos resultados no muy acertados fuera de la muestra, las razones serán comentadas en el siguiente punto.

2.11. Justificación

Observando los resultados de cada modelo podemos concluir en que estos representa de forma correcta nuestro conjunto. Como ya hemos visto el mejor resultado lo hemos obtenido con SVR y con el SGD obteniendo unos valores muy cercanos, siendo el caso de SGD el mejor con un porcentaje de acierto 1 – 0.41887 = 0.5811 fuera de la muestra, siendo estos unos resultados de no tan buena calidad.

Una de las causa de estos resultados puede ser por la poca diversidad de los datos dentro del conjunto de datos, ya que al entrenar nuestro datos no obtienen un comportamiento correcto para la estimación de los valores de fuera de la muestra. Quizás una mejora sería aumentar las instancias de nuestros modelos con nuevos datos que no contengan errores ni inconsistencias.