Taller 3: máquinas de soporte vectorial

Integrantes: Luis Frontuso, Miguel Zúñiga

Este documento tiene como objetivo describir el desarrollo de la implementación de las máquinas de soporte vectorial a partir de la arquitectura[1] propuesta en el curso. También se presenta el uso del modelo para identificar casos de cáncer de mama a partir del conjunto de datos "Breast Cancer Wisconsin (Diagnostic)" y, por último, se comparan los resultados obtenidos con los diferentes optimizadores, técnicas de validación y regularización.

Para alcanzar los objetivos anteriormente descritos, se realizó un diagrama UML de la arquitectura actual y se incluyeron cambios en métodos, atributos y clases. Estos cambios pueden agruparse en correcciones y desarrollos. El diagrama se encuentra en el archivo «PUJ_ML.drawio», y el programa para visualizar el diagrama se puede obtener en el siguiente repositorio. También se desarrollaron y ajustaron componentes que no están en el diagrama de clases, pues no representan clases específicas, por ejemplo, los siguientes scripts: «main.py», «Helpers.py» y cada uno de los archivos «init.py».

Correcciones:

- **Método _regularization de la clase base de Model:** Originalmente, este método retornaba únicamente la derivada de la regularización *Elastic Net*. Por esta razón, se agregó un parámetro *derivate* que, cuando es verdadero, retorna el valor de la derivada, y cuando es falso, retorna el valor de la función original. Adicionalmente, se añadió el parámetro *lambda*[2] a la función y un control que garantiza que la suma de L1 y L2 sea igual a 0 o 1. Esto se hizo con el objetivo de obtener tanto el valor de la función de regularización para la función de costo como su derivada para el gradiente.
    - **Función de regularización:**

$$R(\Theta) = \lambda \left[\gamma \sum_{j=0}^{n} |\theta_j| + (1-\gamma) \sum_{j=0}^{n} \theta_j^2 \right]$$

[1] feed_forward_almost_complete

[2] Este cambio implicó modificar el archivo main.py para obtener el valor de lambda directamente del usuario, así como ajustar todas las clases de optimización que hacen uso de los métodos cost_gradient o cost, con el fin de incluir dicho parámetro en sus llamados.

$$\frac{\partial R(\theta_j)}{\partial \theta_j} = \begin{cases} \left[2\alpha \sum \theta_j - (1-\gamma)\right] \; ; \; \theta_j < 0 \\ \left[2\alpha \sum \theta_j + (1-\gamma)\right] \; ; \; \theta_j > 0 \end{cases}$$

- **Función SplitDataForBinaryLabeling de Helpers.py:** Esta función está pensada para clases binarias que toman valores de 0 o 1. Sin embargo, la función de pérdida de bisagra (*hinge loss*) está diseñada para valores de 1 o -1. Por lo tanto, se ajustó para que funcione con estos nuevos valores.
- **Función Confussion de Helpers.py:** En este caso, se adaptó la función no solo para que funcionara con los valores de clase 1 y -1, sino también para que pudiera leer modelos del tipo *SGDClassifier* de la librería *scikit-learn*.

Desarrollos:

Clase SVM: Para la implementación de esta clase, se tomó como referencia la clase Logistic; sin embargo, métodos como cost_gradient y cost fueron implementados completamente según las definiciones vistas en clase.

- **Método _evaluate:** La implementación de este método es muy similar a la del mismo método de la clase Logistic, debido que su principal diferencia se encuentra en que el sesgo se resta.

$$\boldsymbol{W}^T\boldsymbol{X} - b \; en \; lugar \; de \; \boldsymbol{W}^T\boldsymbol{X} + b$$

- **Método cost:** La implementación de este método se basó principalmente en la función de costo vista en clase y se generalizó para para que hiciera uso de la regularización *Elastic Net,* de la siguiente manera:

  - Función de perdida de bisagra (*hinge loss*):

$$l_i(\vec{w}, b) = \begin{cases} 0 & ; \; y_i(\overrightarrow{w_i} \cdot \overrightarrow{x_i} - b) \geq 1 \\ 1 - y_i(\overrightarrow{w_i} \cdot \overrightarrow{x_i} - b) \; ; otherwise \end{cases}$$

  - Para la función de costo, suponga que n representa el número de pesos y están representados por theta mayúscula de la siguiente manera:

$$\Theta = \; [b, w_1, w_2, \dots, w_n] = [\theta_0, \theta_1, \theta_2, \dots, \theta_n]$$

Por lo tanto, se tiene que:

$$J(\vec{w},b)=\begin{cases}R(\Theta) & ;\ y_i(\overrightarrow{w_i}\cdot\overrightarrow{x_i}-b)\geq 1\\ \frac{1}{m}\sum_{i=1}^{m} l_i(\vec{w},b)+R(\Theta) & ;otherwise\end{cases}$$

Finalmente, reemplazando la función de regularización que definimos anteriormente obtenemos que:

$$J(\vec{w},b)=\begin{cases}\lambda\left[\gamma\sum_{j=0}^{n}\left|\theta_j\right|+(1-\gamma)\sum_{j=0}^{n}\theta_j^2\right] & ;\ y_i(\overrightarrow{w_i}\cdot\overrightarrow{x_i}-b)\geq 1\\ \frac{1}{m}\sum_{i=1}^{m}1-y_i(\overrightarrow{w_i}\cdot\overrightarrow{x_i}-b)+\lambda\left[\gamma\sum_{j=0}^{n}\left|\theta_j\right|+(1-\gamma)\sum_{j=0}^{n}\theta_j^2\right] & ;otherwise\end{cases}$$

- **Método cost_gradient:** Este método implementa el vector de derivadas de la función de costo con respecto a cada peso y al sesgo. Partiendo de la Función de perdida de bisagra (*hinge loss*) y al derivar se obtiene:

$$\frac{\partial l_i(\vec{w},b)}{\partial\vec{w}}=\begin{cases}0 & ;\ y_i(\overrightarrow{w_i}\cdot\overrightarrow{x_i}-b)\geq 1\\ -y_i\cdot\overrightarrow{x_i} & ;otherwise\end{cases}$$

$$\frac{\partial l_i(\vec{w},b)}{\partial b}=\begin{cases}0 & ;\ y_i(\overrightarrow{w_i}\cdot\overrightarrow{x_i}-b)\geq 1\\ y_i & ;otherwise\end{cases}$$

Ahora a partir de la función de costo, se espera que su derivada sea:

$$\frac{\partial J(\vec{w},b)}{\partial(\vec{w},b)}=\begin{cases}\frac{1}{m}\frac{\partial l_i(\vec{w},b)}{\partial b}+\frac{\partial R(\theta_0)}{\partial\theta_0} & ;b=\theta_0\\ \frac{1}{m}\frac{\partial l(\vec{w}_j,b)}{\partial\vec{w}_j}+\frac{\partial R(\theta_j)}{\partial\theta_j} & ;otherwise\end{cases}$$

Validación de la implementación

Para la validación se entrenaron dos modelos, con una división de los datos entrenamiento-validación de 70-30. Se entrenó un *SGDClassifier* de la librería scikit-learn y un SVM con la librería construida en clase. Para que los resultados fueran comparables, ambos modelos se entrenaron con los mismos parámetros. En el caso de *SGDClassifier*, se utilizaron los siguientes parámetros:

- loss='hinge'
- penalty='elasticnet'
- alpha=2
- l1_ratio=0.5
- max_iter=200

mientras que para el modelo SVM se utilizaron parámetros[3] equivalentes:

- L1=0.5
- L2=0.5
- L=2
- Optimizar= 'Adam'
- Epochs=200

En la siguiente tabla se observan los resultados de cada modelo y su derivada. Aunque las métricas calculadas en el conjunto de entrenamiento no son exactamente iguales, tampoco presentan diferencias significativas. Esta variación se debe a que *SGDClassifier* utiliza un gradiente descendente estocástico (*Stochastic Gradient Descent*), lo que puede impedir la convergencia exacta de las soluciones. Sin embargo, dado que los resultados son cercanos, se puede asumir que se alcanzó una implementación aceptable del modelo SVM.

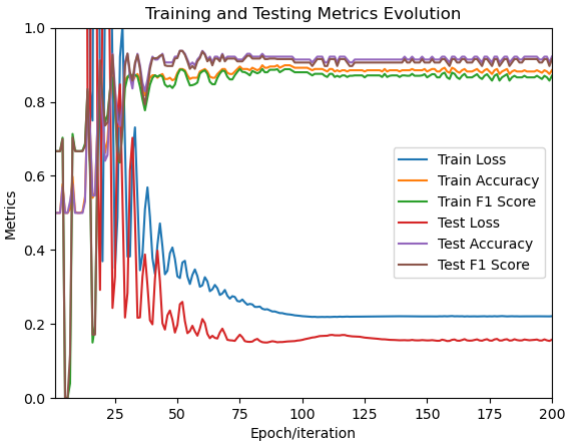
SCORE	SVM	SGDCLASSIFIER	DRIFT
SENSIBILITY	79.7297%	95.2703%	15.5405%
SPECIFICITY	99.3243%	83.1081%	-16.2162%
ACCURACY	89.5270%	89.1892%	-0.3378%
F1	88.3895%	89.8089%	1.4194%

[3] bash: python3 main.py data/data.csv -test 0.3 -d , -L1 0.5 -L2 0.5 -L 2 -o Adam -e 200

Detección del cáncer de mama

Un caso de uso de las máquinas de soporte vectorial es la detección de cáncer de mama a partir de características extraídas de una imagen digitalizada de una aspiración con aguja fina (PAAF) de una masa mamaria.

Optimizadores: con este objetivo en mente, se evaluará la convergencia de los optimizadores utilizando los parámetros mencionados anteriormente, excepto por el optimizador. En las siguientes imágenes, a la derecha se observan los resultados del gradiente descendente, un método de optimización que no muestra una convergencia clara hacia una solución y en contraste, el método de optimización Adam, mostrado a la izquierda, presenta una convergencia clara hacia un costo bajo, así como altos valores de precisión y F1, tanto en el conjunto de entrenamiento como en el de validación.



Grupos: En este apartado se evalúan las técnicas de validación como Leave-One-Out, K-Fold Cross Validation y el ajuste tradicional del modelo con todo el conjunto de entrenamiento. Para ello, se utilizó el optimizador Adam con los mismos parámetros empleados durante la validación de la implementación, y se calcularon la sensibilidad (sensitivity), especificidad (specificity), precisión (accuracy) y F1 en el conjunto de validación. Los resultados se presentan en la siguiente tabla, donde se observa que, cuando el modelo se entrena con LOO o K-Fold, no logra aprender adecuadamente debido a que la distribución de clases en la validación es de 50% positivas y 50% negativas. Esto ocasiona que la especificidad parezca tener un “buen” desempeño; sin embargo, el modelo está clasificando todas las observaciones como negativas.

Score	*LOO*	*Kfold5*	*MCE*
sensibility	0.0000%	0.0000%	79.0541%
specificity	100.0000%	100.0000%	98.6486%
accuracy	50.0000%	50.0000%	88.8514%
F1	0.0000%	0.0000%	87.6404%

Regularización: En este punto se ponen a prueba las técnicas de regularización disponibles en la implementación, mediante el entrenamiento del modelo con el optimizador Adam y la técnica de validación MCE, para cada uno de los experimentos descritos en la siguiente tabla:

TRIAL	**L**	**L1**	**L2**
T0	0	0%	0%
T1 LASSO	1	100%	0%
T2 RIDGE	1	0%	100%
T3 ELASTIC NET	1	50%	50%

De los resultados obtenidos se observa que el experimento en el que no se aplicó ninguna regularización es el que presenta mayor sobreajuste: en promedio, las métricas evaluadas en entrenamiento están seis puntos porcentuales por encima de las de validación. También se observa que el experimento T1, con menor sobreajuste, coincide con el mejor desempeño en las métricas evaluadas. Esto sugiere que una técnica más agresiva de regularización puede ofrecer mejores resultados.

TRIAL	**TEST**				**DRIFT**				**AVERAGE**
	Sensibility	**Specificity**	**Accuracy**	**F1**	**Sensibility**	**Specificity**	**Accuracy**	**F1**	
T0	76.56%	98.44%	87.50%	86%	11.28%	0.89%	6.08%	7.23%	**6.37%**
T1	82.81%	100.00%	91.41%	91%	-1.06%	-1.35%	-1.20%	-1.30%	**-1.23%**
T2	78.13%	98.44%	88.28%	87%	2.28%	0.89%	1.58%	1.85%	**1.65%**
T3	71.88%	100.00%	85.94%	84%	9.21%	-0.68%	4.27%	5.58%	**4.59%**

A partir del experimento con mejor desempeño (T1), se proponen diferentes valores de lambda (L) para evaluar si es posible mejorar los resultados obtenidos anteriormente. En ese orden de ideas, los experimentos propuestos son los siguientes:

TRIAL	L
T1.1 LASSO	0.25
T1.2 LASSO	0.5
T1.3 LASSO	0.75
T1.4 LASSO	1.5
T1.5 LASSO	2

De este grupo de experimentos se encontró que el T1.5 LASSO mejoró todas las métricas en el conjunto de entrenamiento y, además, presentó una diferencia promedio entre entrenamiento y validación de todas las métricas de -28 puntos básicos.

TRIAL	TEST				DRIFT				AVERAGE
	Sensibility	**Specificity**	**Accuracy**	**F1**	**Sensibility**	**Specificity**	**Accuracy**	**F1**	
T1.1 LASSO	87.50%	96.88%	92.19%	92%	-3.72%	2.45%	-0.63%	-0.96%	**-0.72%**
T1.2 LASSO	82.81%	98.44%	90.63%	90%	0.30%	0.89%	0.59%	0.61%	**0.60%**
T1.3 LASSO	89.06%	98.44%	93.75%	93%	-7.31%	0.89%	-3.21%	-3.81%	**-3.36%**
T1.4 LASSO	84.38%	100.00%	92.19%	92%	-5.32%	-1.35%	-3.34%	-3.88%	**-3.47%**
T1.5 LASSO	84.38%	100.00%	92.19%	92%	0.76%	-1.35%	-0.30%	-0.22%	**-0.28%**