Universidad de Buenos Aires
Facultad De Ingeniería
Año 2021 - 2do Cuatrimestre

Organización de Datos (75.06)

TRABAJO PRÁCTICO Nº2
TEMA:
FECHA DE ENTREGA: 8/12/2021

INTEGRANTES:

DE LUCA ANDREA, Felipe	- #88888
FOPPIANO, Elián	- #88888

Índice

1. Preprocesamientos

Nombre del procesamiento	Descripción	Nombre de la función
Común	Drop de *llovieron_hamburguesas_hoy* Reemplazar *si* y *no* por 1 y 0 en variable target Drop de muestras con missings en variable target Drop de muestras con datos inválidos Corrección de typos en nombres de features Si el parámetro **fecha_to_int** es True, convierte *dia* en un entero con formato AAAAMMDD	**common**()
Día a mes	Convierte *dia* en el mes correspondiente a la fecha	**dia_a_mes**()
Viento trigonométrico	Convierte las features de dirección del viento en pares de features $\langle \sin\theta, \cos\theta \rangle$, donde θ es el ángulo de dicha dirección	**viento_trigonometrico**()
Barrios a comunas	Realiza un hashing de los valores de la feature **barrio**, donde el nombre de cada barrio se reemplaza por el nombre de su comuna (*Comuna 1*, *Comuna 2*, etc)	**barrios_a_comunas**()
Estandarización	Agrega un **StandardScaler()** a la **Pipeline**, que transforma los datos tal que su distribución tenga esperanza 0 y varianza 1 (deben ser todas las features numéricas)	**standarizer**()
Imputador simple	Agrega un **SimpleImputer()** a la **Pipeline**, que imputa los valores faltantes con el promedio	**simple_imputer**()
Imputador iterativo	Agrega un **IterativeImputer()** a la **Pipeline**, que imputa los valores faltantes a partir de regresiones iterativas	**iterative_imputer**()
Drop categóricas	Dropea las features categóricas del dataset (Si se ejecutó el preprocesamiento *Viento trigonometrico*, solo dropea la feature **barrio**)	**drop_categoricas**()
Hashing trick	Transforma una feature categórica en varias columnas con valores 0 o 1 a partir de un vector obtenido con una función de hash sobre la variable	**hashing_trick**()
Drop poco importantes	Dado una lista de scores y un threshold, dropea las features cuyos scores (importancia de feature) sea menor al threshold	**drop_poco_importantes**()
Feature selection	Dado un modelo y un porcentaje de features a mantener, selecciona las features que maximicen el score del modelo dado mediante forward selection	**feature_selection**()

Cuadro 1.1: Preprocesamientos

2. Modelos

#	Nombre del modelo	Preprocesamientos	AUC ROC	Accuracy	Precision	Recall	F1 score
1	NeuralNetwork	Común (**fecha_to_int=True**) Viento trigonométrico Hashing Trick (feature **barrio**) Imputador iterativo Estandarización	0.895	0.860	0.774	0.529	0.629
2	CascadingClassifier	Común (**fecha_to_int=True**) Viento trigonométrico Hashing Trick (feature **barrio**) Imputador iterativo Estandarización	0.895	0.862	0.778	0.538	0.636
3	AdaBoostClassifier	Común (**fecha_to_int=True**) Barrios a comunas Viento trigonométrico Imputador simple Estandarización	0.894	0.861	0.806	0.498	0.615
4	RandomForestClassifier	Común (**fecha_to_int=False**) Viento trigonométrico Día a mes Drop categóricas Estandarización Imputador iterativo	0.889	0.859	0.764	0.538	0.631
5	SVM	Común (**fecha_to_int=True**) Viento trigonométrico Estandarización Imputador iterativo Hashing trick (feature **barrio**)	0.882	0.857	0.794	0.491	0.607
6	KNN	Común (**fecha_to_int=True**) Viento trigonométrico Estandarización Imputador iterativo Hashing trick (feature **barrio**)	0.873	0.838	0.805	0.365	0.503
7	DecisionTreeClassifier	Común (**fecha_to_int=True**) Viento trigonométrico Barrios a comunas Día a mes Imputador iterativo	0.870	0.849	0.763	0.470	0.582
8	NB (Naive Bayes)	Común (**fecha_to_int=True**) Viento trigonométrico Drop categóricas Imputador iterativo Feature selection (**n = 0.6**)	0.850	0.832	0.644	0.565	0.602
9	Perceptrón	Común (**fecha_to_int=True**) Viento trigonométrico Barrios a comunas Drop poco importantes (todas las features que tuvieron coeficientes 0 en un perceptrón previo) Imputador iterativo Estandarización	0.833	0.820	0.770	0.282	0.413

Cuadro 2.1: Métricas de los modelos

3. Conclusión

De todos los modelos entrenados, los dos que resultaron ser mejores fueron la red neuronal y el ensamble en cascada. En ambos casos su AUC-ROC fue de 0.895, lo cual es bastante aceptable. Si tuvieramos que elegir uno, nos decantaríamos por la red neuronal, ya que es el modelo más sencillo de los dos: el cascading ya de por sí tiene como uno de los modelos que utiliza a la misma red.

La baseline que hicimos a partir del análisis de los datos tenía un accuracy de alrededor de 83 %. Podemos ver que esta métrica fue ampliamente superada por casi todos los modelos (exceptuando el perceptrón y Naive Bayes), por lo que los resultados fueron bastante positivos.

Algo a notar es que todos los modelos tienen un Recall bastante bajo. El Recall es una métrica que responde a la pregunta: de las instancias positivas, ¿qué porcentaje fueron clasificadas correctamente?. Debido a que el dataset utilizado estaba bastante desbalanceado, todos los modelos tendieron a aprender a responder por la negativa. Esto provocó que haya una gran cantidad de falsos negativos, lo que es la causa de que esta métrica haya dado tan baja.

Si tuvieramos que elegir el modelo con la menor cantidad de falsos positivos, eligiríamos el que tiene el Precision más alto, que responde a la pregunta: de los detectadaos como positivos, ¿qué porcentaje realmente lo eran?. Entonces, a Precision más alto, menor cantidad de falsos positivos. En este caso, el modelo con Precision más alto resultó ser el *AdaBoostClassifier*. Podemos notar que esa ganancia no fue gratuita: perdió mucho en Recall.

Si necesitaramos una lista de los potenciales días que vayan a llover, es decir, minimizar los falsos negativos, volveríamos al problema del Recall. El modelo con mejor Recall en este caso es Naive Bayes. Nuevamente, pierde mucho en otras métricas: es el segundo peor modelo en nuestra lista ordenada por AUC-ROC. De todas formas, los modelos encontrados tienen todos un Recall exageradamente bajo, y si realmente necesitaramos tener pocos falsos negativos, probablemente intentaramos con modelos nuevos o incluso pensamos que sería mejor un modelo que responda siempre que 'si' antes que los que pudimos entrenar.

Podemos también considerar que a la hora de entrenar los modelos, se optimizó para maximizar la métrica de AUC-ROC. Si nos encontraramos en los 2 casos hipotéticos descritos en los párrafos anteriores, probablemente hubieramos elegido otros hiperpárametros en los modelos a entrenar. Por ejemplo, en el notebook del árbol de decisión, si elegíamos el parámetro de *class_weight* = "*balanced*", daba bastante mejor Recall que en el que terminó quedando en la tabla anterior.