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

Organización de Datos (75.06)

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

INTEGRANTES:

DE LUCA ANDREA, Felipe	- #105646
FOPPIANO, Elián	- #105836

Índice

1. Introducción

El presente informe es un resumen de todos los resultados obtenidos a lo largo del TP. En total, se entrenaron 9 modelos, de los cuales 3 son ensambles, y uno de ellos fue programado manualmente (**CascadingClassifier**). Todos los modelos fueron entrenados utilizando al menos dos conjuntos de técnicas de preprocessing, de forma que cada preprocessing está conformado por varias funciones del archivo *preprocessing.py*. El análisis exhaustivo de cada modelo se encuentra detallado en sus respectivos *Notebooks*, en los cuales se encuentran los procedimientos, resultados intermedios, *tuning* de hiperparámetros, y todo lo que nos condujo al modelo final.

Por último, para cada modelo final, se calcularon las métricas *ROC-AUC*, *Accuracy*, *Precision*, *Recall*, *F1 Score* y Matriz de confusión, utilizando el set de test-holdout. En base a eso, se evaluaron los distintos modelos entre sí, comparando no solo la métrica a optimizar (en este caso, *ROC-AUC*), sino también su desempeño global, fortalezas, debilidades de cada uno, etc.

2. 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 la feature *dia* en el mes correspondiente a la fecha (numérico)	**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()**
Dummy	Aplica **One Hot Encoding** a una lista de features, dropeando la primera columna y creando una nueva para las instancias con missing	**dummy()**
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) y le aplica el preprocessing **Dummy**	**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()**
MinMax	Agrega un **MinMaxScaler()** a la **Pipeline**, que transforma los datos tal que queden entre 0 y 1 utilizando su mínimo y máximo (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()**
Drop correlacionadas	Dropea features que presentaron una fuerte correlación durante el análisis exploratorio (**temp_max**, **presion_atmosferica_temprano** y **temperatura_temprano**)	**drop_correlacionadas()**
Drop discretas	Dropea las features **nubosidad_temprano** y **nubosidad_tarde**	**drop_discretas()**
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()**

3. Modelos

Todas las métricas presentadas fueron evaluadas respecto al set de test-holdout, que no fue utilizado durante el entrenamiento ni para la selección del preprocesamiento e hiperparámetros de cada modelo.

#	Nombre del modelo	Preprocesamientos	AUC ROC	Accuracy	Precision	Recall	F1 score
1	NeuralNetwork	Común (**fecha_to_int=True**) Viento trigonométrico Dummy (feature **barrio**) Imputador iterativo Estandarización	0.896	0.859	0.782	0.512	0.619
2	CascadingClassifier	Común (**fecha_to_int=True**) Viento trigonométrico Hashing Trick (feature **barrio**) Imputador iterativo Estandarización	0.895	0.859	0.771	0.520	0.621
3	AdaBoostClassifier	Común (**fecha_to_int=True**) Barrios a comunas Viento trigonométrico Imputador simple Estandarización	0.894	0.856	0.795	0.475	0.595
4	RandomForestClassifier	Común (**fecha_to_int=False**) Viento trigonométrico Día a mes Drop categóricas Estandarización Imputador iterativo	0.887	0.854	0.767	0.494	0.601
5	SVM	Común (**fecha_to_int=True**) Viento trigonométrico Estandarización Imputador iterativo Hashing trick (feature **barrio**)	0.886	0.857	0.796	0.480	0.599
6	KNN	Común (**fecha_to_int=True**) Viento trigonométrico Estandarización Imputador iterativo Hashing trick (feature **barrio**)	0.877	0.835	0.779	0.361	0.493
7	DecisionTreeClassifier	Común (**fecha_to_int=True**) Viento trigonométrico Barrios a comunas Día a mes Imputador iterativo	0.870	0.842	0.748	0.439	0.553
8	NB (Naive Bayes)	Común (**fecha_to_int=True**) Viento trigonométrico Drop categóricas Imputador iterativo Feature selection (**n = 0.6**)	0.856	0.829	0.631	0.557	0.592
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	0.830	0.817	0.749	0.265	0.392

4. Conclusión

De todos los modelos entrenados, el que mejor resultó de acuerdo a el área bajo la curva ROC fue la red neuronal, que le dio un valor de de 0.896, lo cual es bastante aceptable. Los ensambles de Cascading y AdaBoost estuvieron bastante cerca, pero levemente por debajo.

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 en general 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 tuviéramos 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 detectados 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 *SVM*, con un valor de 0.796. Podemos ver que notar que esa ganancia no fue gratuita: perdió en Recall.

Si necesitáramos 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 necesitáramos tener pocos falsos negativos, probablemente intentaríamos 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 encontráramos en los 2 casos hipotéticos descritos en los párrafos anteriores, probablemente hubiéramos 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. Para entrenar la red, podríamos haber frenado el progreso de las epochs con una métrica como F-score con $\beta = 2$, que prioriza el Recall dos veces más que la Precision.