Taller de Aprendizaje Automático

Actividades Taller 3

Instituto de Ingeniería Eléctrica
Facultad de Ingeniería

UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

Montevideo, 2024

Tabla de contenido

Objetivos del Taller 3

- Familiarizarse con los estimadores **Decision Trees**, entender su funcionamiento, y la importancia de los **hiperparámetros de regularización**.
- Trabajar con estimadores más complejos basados en métodos de **ensamble** de árboles como: **Random Forest** y **Gradient Boosting**.
- Comparar los estimadores para un problema en particular de regresión.
- Incorporar funciones de transformación de columnas y de medidas de desempeño *custom*.
- Manipular marcas de tiempo (*timestamps*).

El problema y los datos

- Se quiere predecir la demanda de bicicletas a partir de la combinación de datos históricos sobre demanda y clima.
- Se dispone de un conjunto datos del alquiler de bicicletas por hora durante 2 años.
- **Desafío Kaggle:** El subconjunto de *train* está compuesto por los primeros 19 días de cada mes, mientras que el subconjunto de *test* va desde el día 20 hasta el final del mes.

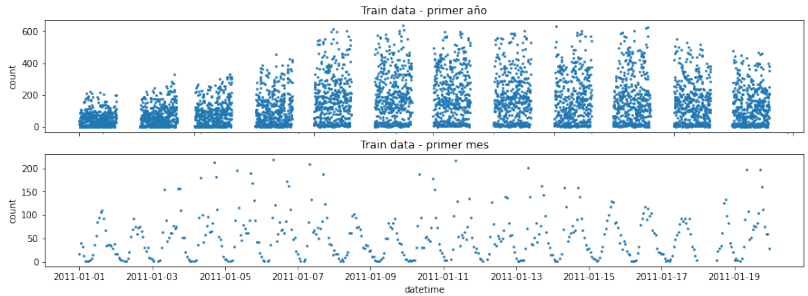

Figure: Variación de la demanda en el conjunto de *Train*.

Preguntas orientadoras para partes 1 y 2

- ¿Cuáles son las características?¿Hay datos faltantes?
- ¿Cómo están codificadas las características?
- ¿Cuáles les parecen más relevantes?
- ¿Cómo varía la demanda en función del tiempo?
- ¿Por qué puede tener sentido usar árboles en vez de un modelo lineal para este problema ?

Características

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   datetime    10886 non-null  object
 1   season      10886 non-null  int64
 2   holiday     10886 non-null  int64
 3   workingday  10886 non-null  int64
 4   weather     10886 non-null  int64
 5   temp        10886 non-null  float64
 6   atemp       10886 non-null  float64
 7   humidity    10886 non-null  int64
 8   windspeed   10886 non-null  float64
 9   casual      10886 non-null  int64
 10  registered  10886 non-null  int64
 11  count       10886 non-null  int64
dtypes: float64(3), int64(8), object(1)
memory usage: 1020.7+ KB
```

No hay datos faltantes

Variación de la demanda en el tiempo

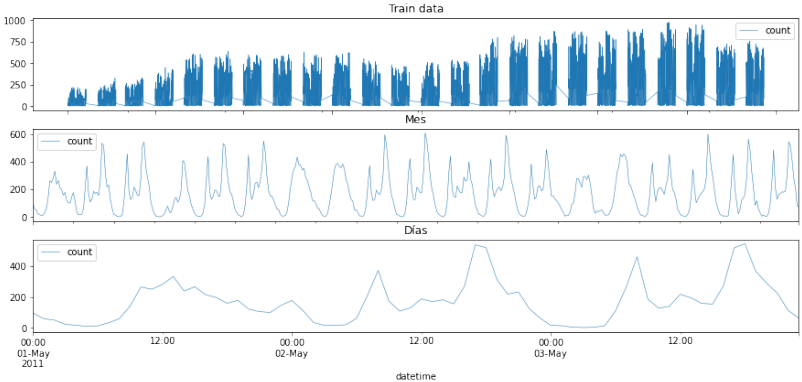

Figure: Características extraídas de datetime

Variación de la demanda en el tiempo

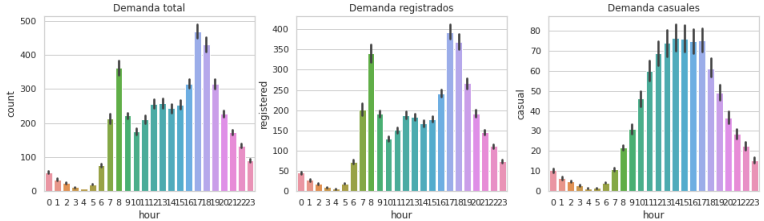

Figure: Variación de la demanda según la hora

Variación de la demanda en el tiempo

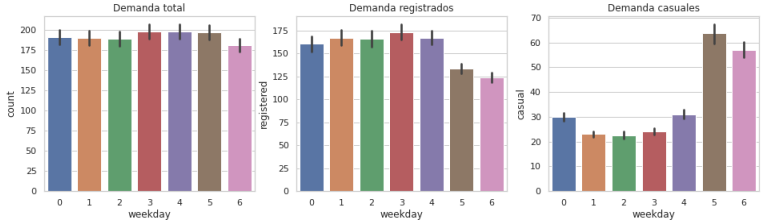

Figure: Variación de la demanda según el dia de la semana

Parte 3

- Medida de desempeño
  Root Mean Squared Logarithmic Error (RMSLE)

$$\sqrt{\frac{1}{n}\sum_{i}^{n}\left(\log(y_i+1)-\log(\hat{y}_i+1)\right)^2}$$

Parte 3

- Medida de desempeño
  Root Mean Squared Logarithmic Error (RMSLE)

$$\sqrt{\frac{1}{n}\sum_{i}^{n}\left(\log(y_i+1)-\log(\hat{y}_i+1)\right)^2}$$

- Preprocesamiento:

```
df_train['hour'] = df_train['datetime'].dt.hour
df_train['weekday'] = df_train['datetime'].dt.weekday

df_train['month'] = df_train['datetime'].dt.month
df_train['year'] = df_train['datetime'].dt.year

y_train = df_train['count']
df_train = df_train.drop([ 'casual', 'registered', 'count','datetime'], axis=1)
```

Parte 4

- Árboles de decisión:
  - Evaluar el desempeño de un árbol con parámetros por defecto utilizando validación cruzada
  - Graficar errores en entrenamiento y test
  - ¿Cómo se puede mejorar el desempeño? ¿Qué hiper-parámetros optimizaría?

Resultado Árbol de Decisión

```
media train 0.008752143092368807
std train 0.002556686785696967
media val 0.5208785175022215
std val 0.10422505761394864
```

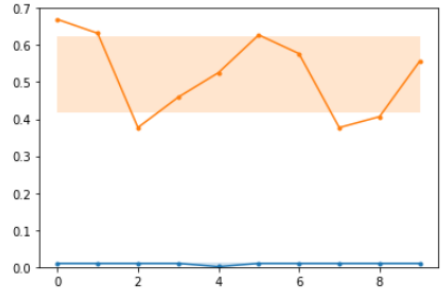

Figure: Decision Tree - 10 Folds.

Partes 4, 5 y 6

- Árboles de decisión:
  - Evaluar el desempeño de un árbol con parámetros por defecto utilizando validación cruzada
  - Graficar errores en entrenamiento y test
  - ¿Cómo se puede mejorar el desempeño? ¿Qué hiper-parámetros optimizaría?
- Random Forest
  - ¿Es esperable una mejora en el desempeño? ¿Por qué?
  - Evaluar el desempeño de Random Forest con parámetros por defecto
  - ¿Qué hiper-parámetros optimizaría en este caso?
- Gradient Boosting
  - ¿Qué es *boosting*? ¿Cuál es la principal diferencia con *bagging*?
  - ¿Qué pasa si el modelo predice un valor negativo?
  - Crear una nueva función de evaluación que fuerce a cero todos los valores negativos de las predicciones y luego calcule el valor de RMSLE
  - Entrene el mejor clasificador posible utilizando *XGBRegressor()*

Parte 7 - Transformación Personalizada

- Construir un *Custom Transform* que extraiga las caracterísicas temporales

Parte 7 - Transformación Personalizada

- Construir un *Custom Transform* que extraiga las caracterísicas temporales
- Ver ejemplo en Cap 2 del libro del curso

```
from sklearn.base import BaseEstimator, TransformerMixin
rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6


class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
        def __init__(self, add_bedrooms_per_room = True): # no *args or **kargs
                self.add_bedrooms_per_room = add_bedrooms_per_room

        def fit(self, X, y=None):
                return self # nothing else to do

        def transform(self, X, y=None):
                rooms_per_household = X[:, rooms_ix] / X[:, households_ix]
                population_per_household = X[:, population_ix] / X[:, households_ix]
                if self.add_bedrooms_per_room:
                        bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
                        return np.c_[X, rooms_per_household, population_per_household,bedrooms_per_room]
                else:
                        return np.c_[X, rooms_per_household, population_per_household]

attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
housing_extra_attribs = attr_adder.transform(housing.values)
```

Opcionales

- Optimizar Random Forest y/o Gradient Boosting y comparar resultados
- Variar la codificación de alguna característica y/o generar alguna nueva
- Utilizar como métrica de ajuste el RMSLE (puede ser útil *TransformedTargetRegressor*)
- Generar dos modelos: uno para los clientes casuales y otro para los registrados
- Hacer una validación cruzada no aleatoria