Temas Evaluación Parcial

# Pandas: Visualización

Manipular y analizar datos a través de índices (filas y columnas).

## Estructuras de datos

* Series -> 1 dimensión
* DataFrame -> 2 dimensiones (tablas)
* Panel -> 3 dimensiones (cubos)

### Series

Matriz unidimensional homogénea:

s = pd.Series([‘Banana’, ‘Manzana’, ‘Naranja’], dtype = ‘string’)

0 Banana

1 Manzana

2 Naranja

s = pd.Series({‘Banana’: 4, ‘Manzana’: 3, ‘Naranja’: 2})

Banana 4

Manzana 3

Naranja 2

* size: cantidad de elementos
* index: lista de nombres de filas
* dtype: tipo de datos de los elementos

### DataFrame

Cada columna es un objeto de tipo Series. Todos los datos de una columna son del mismo tipo, y las filas son registros que pueden contener datos de distintos tipos.

df = pd.DataFrame([

[‘Juan’, ‘Pérez’, 25, ‘Ingeniero’],

[‘Luis’, ‘Gómez’, 30, ‘Abogado’]],

columns = [‘nombre’, ‘apellido’, ‘edad’, ‘profesion’])

df = pd.DataFrame({

‘nombre’: [‘Juan’, ‘Luis’],

‘apellido’: [‘Pérez’, ‘Gómez’],

‘edad’: [25, 30],

‘profesion’: [‘Ingeniero’, ‘Abogado’]})

* info: número de filas y columnas, índices, tipo de las columnas y memoria usada
* shape: tupla con el número de filas y columnas
* size: número de elementos
* columns: lista con los nombres de las columnas
* index: lista con los nombres de las filas
* types: serie con los tipos de datos de las columnas
* head(n): primeras n filas
* tail(n): últimas n filas
* copy: copia del dataframe
* rename(columns = columnas, inplace = True): columnas es un diccionario que tiene como clave los nombres de las columnas a renombrar y como valor el nombre que se reemplazará
* isna: cantidad de datos nulos
* count: cantidad de datos no nulos por columnas
* crosstab(c1, c2): tabla de dos entradas que cuenta ocurrencias
* groupby(columnas): se agrupan los valores por los nombres de las columnas en la lista de columnas
* describe: datos numéricos (cantidad, media, mínimo, máximo, cuartil 1, cuartil 2, cuartil 3)
* drop: elimina por columnas o índices
* replace: reemplaza un valor por otro

#### Acceso mediante Posiciones

* df.iloc[i, j]: elementos que se encuentra en la fila i y la columna j
* df.iloc[filas, columnas]: dataframe con los elementos de las filas de la lista de filas y de las columnas de la lista de columnas
* df.iloc[i]: serie con los elementos de la fila i

#### Acceso mediante Nombres

* df.loc[fila, columna]: elementos que se encuentra en la fila con nombre fila y la columna con nombre columna
* df.loc[filas, columnas]: dataframe con los elementos que se encuentran en las filas con los nombres de la lista filas y las columnas con los nombres de la lista columnas
* df[columna]: serie con los elementos de la columna de nombre columna
* df.columna: serie con los elementos de la columna de nombre columna (funciona cuando el nombre de la columna no tiene espacios en blanco)

#### Creación desde un archivo de datos

df = pd.read\_csv(‘./dataset/archivo.csv’)

#### Guardado a un archivo de datos

df.to\_csv(‘./dataset/archivo\_out.csv’, index = False)

# 

# Gráficos

## Gráfico de barras

Gráfico de frecuencias para variables cualitativas.

### Pandas

ds[col].value\_counts().plot.bar()

### Matplotlib

ds\_plot = ds.groupby([x\_col]).count()

x = ds\_plot.index.tolist()

y = ds\_polot[y\_col].tolist()

plt.locator\_params(‘x’, nbins=x\_groups)

plt.bar(x, y)

plt.show()

### Seaborn

ds\_plot = ds.groupby([x\_col]).count()

x = ds\_plot.index.tolist()

y = ds\_plot[y\_col].tolist()

sns.barplot(x=x, y=y, data=ds\_plot)

## Histogramas

Gráfico de frecuencias para variables cuantitativas. Separamos en intervalos la variable que estamos analizando y hacemos una barra con la altura de la frecuencia que obtenemos.

### Pandas

ds[col].hist(bins=b)

### Matplotlib

plt.hist(ds[col], bins=b)

plt.show()

### Seaborn

sns.histplot(data=ds[col])

## Gráficos de Densidad

Representación de la distribución de una variable (histograma suavizado).

### Pandas

ds[col].plot.kde()

### Seaborn

sns.histplot(data=ds[col], kde=True)

sns.kdeplot(data=ds[col]

## Boxplots

### Pandas

ds[col].plot.box()

### Seaborn

sns.boxplot(y=col, data=ds)

## Gráficos de a pares

### Pandas

from pandas.plotting import scatter\_matrix

scatter\_matrix(ds[ds.columns.tolist().remove(‘col’)])

### Seaborn

sns.pairplot(data=ds[columns], hue=’col’)

## Dispersogramas

### Pandas

ds.plot.scatter(c=col)

### Seaborn

sns.relplot(data=ds)

## Gráfico de Violín

Muestra la distribución de una variable cuantitativa condicionada a distintos valores de una variable categórica de forma que las distribuciones pueden ser comparadas de forma visual rápidamente.

### Seaborn

sns.violin(data=ds)

## Heatmap

tabla = pd.crosstab(ds[col1], ds[col2])

sns.heatmap(tabla, annot=True)

# Métodos de Regresión

* **Población**: conjunto de elementos con características en común que se quiere estudiar. Se las estudia a través de una muestra cuidadosamente seleccionada.
* **Muestra**: subconjunto representativo de la población, de menor tamaño.
* **Media Poblacional**: medida de tendencia central.
* **Desviación Estándar Poblacional**: índice numérico de la dispersión de un conjunto de datos.
* **Media Muestral**: media de la muestra.

## Covarianza y Correlación

Medidas de asociación lineal entre dos variables.

* Covarianza: indica la dirección/sentido de la asociación lineal.
* Correlación: mide tanto la fuerza como el sentido de la asociación.

Estimamos el coeficiente de correlación con Pearson.

np.cov(x, y)

np.corrcoef(x, y)

ds.cov()

ds.corr()

En un heatmap:

sns.heatmap(mat\_corr, annot=True)

En un pairplot:

sns.pairplot(ds)

## Correlación != Causalidad

Que dos variables tengan un alto índice de correlación no significa que una cause a la otra. Las correlaciones pueden suceder por otros motivos, como por ejemplo el azar.

## Modelo Lineal Simple

Un **modelo de regresión lineal** es un método de aprendizaje supervisado. Buscamos predecir un valor en un rango continuo para ciertos valores de entrada.

Es un modelo para el vínculo de dos variables, se dice simple ya que vincula sólo una variable predictora con la variable de respuesta.

Resumiendo, el modelo consiste en predecir una respuesta numérica Y en base a una única variable predictora X suponiendo una relación lineal. Vamos a estimar los parámetros B\_0 y B\_1.

modelo\_lineal\_simple = LinearRegression()

x\_r = x.reshape(-1,1)

y\_r = y.reshape(-1,1)

modelo\_lineal\_simple.fit(x\_r, y\_r)

B1\_e = modelo\_lineal\_simple.coef\_[0][0]

B0\_e = modelo\_lineal\_simple.intercept\_[0] #ordenada al origen

# y = {B1\_e} \* X + {B0\_e}

Hay un método directo para predecir en los modelos:

modelo\_lineal\_simple.predict(var)

### Métricas

#### Coeficiente de determinación R^2

Cuánto explica mi modelo del fenómeno que estoy estudiando, cuánto capta -el modelo- de la variabilidad de los datos.

Es la proporción de variabilidad explicada por el modelo y su valor está entre 0 y 1 (0: el modelo no se ajusta nada; 1: el modelo se ajusta perfecto a cómo varían los datos).

Hay un método directo para predecir en los modelos:

metrics.r2\_score(var, var\_estim)

modelo\_lineal\_simple.score(var, var\_estim)

#### MSE (Mean Squared Error)

Medida de qué tan cercana es la recta de regresión a los puntos que representan los datos. Mientras más chico más cerca está nuestro modelo de los datos reales. Es sensible a valores de diferencias grandes.

metrics.mean\_squared\_error(var, var\_estim, squared=True)

#### RMSE (Rood Mean Squared Error)

Tiene las mismas unidades que los valores representados en el eje vertical. Es la distancia de un punto hasta la recta de regresión medida en línea recta. Mide el desvío estándar (cuánto se alejan los valores de la media).

metrics.mean\_squared\_error(var, var\_estim, squared=False)

### Selección del Modelo

El mejor modelo es el que mejor capacidad explicativa tiene: el que mayor R^2 tiene y el que menor error tiene.

### Predicción y Evaluación en datos de testeo

Para evaluar cómo se comporta el modelo ante nuevas observaciones vamos a realizar predicciones sobre casos que no se utilizaron para entrenar: vamos a predecir el precio de venta esperado a partir de observaciones que el modelo nunca vio (este es el nuevo conjunto de test). Para evaluar la performance de nuestros modelos se evalúan las métricas MSE o RMSE luego de realizar las predicciones.

metrics.mean\_squared\_error(y\_true=var, y\_pred=var\_pred, squared=True)

metrics.mean\_squared\_error(y\_true=var, y\_pred=var\_pred, squared=False)

Un modelo que tiene poco error en datos de entrenamiento pero mucho error en datos de test quiere decir que no es bueno: se memorizó todos los datos de entrenamiento y cuando lo sacamos de su zona de “confort” no funciona.

## Modelo Lineal Múltiple

Estimar la relación: y = B\_1\*var\_1 + B\_2\*var\_2 + B\_0 (más de una var predictora).

### Ajuste del Modelo

modelo\_lineal\_multiple = LinearRegression()

x\_r = ds[[columns]]

y\_r = ds[col\_to\_predict].values.reshape(-1,1)

modelo\_lineal\_multiple.fit(x\_r, y\_r)

B1\_e = modelo\_lineal\_simple.coef\_[0][0]

B2\_e = modelo\_lineal\_simple.coef\_[0][1]

B0\_e = modelo\_lineal\_simple.intercept\_[0] #ordenada al origen

# y = {B1\_e} \* var\_1 + {B2\_e} \* var\_2 + {B0\_e}

### Evaluación de las métricas del modelo ajustado

r2 = modelo\_lineal\_multiple.score(x\_r, y\_r)

y\_pred = modelo\_lineal\_multiple.predict(x\_r)

rmse = metris.mean\_squared\_error(y\_true=y\_r, y\_pred=y\_pred, squared=False)

### Predicción y Evaluación en datos de testeo

x\_t = ds\_test[[columns]]

y\_t = ds\_test[y].values.reshape(-1,1)

y\_pred = modelo\_lineal\_multiple.predict(x\_t)

## Regresión Logística

Algoritmo de aprendizaje supervisado en el cual se brindan las categorías de cada una de las observaciones para entrenar. KMeans no es supervisado y determina cómo se tienen que agrupar las observaciones. Regresión Lineal también es supervisado porque le damos un resultado para que se entrene.

### Clasificación Binaria

#### Clasificación Simple en 1 dimensión

Utilizamos regresión lineal y separamos las clases en función del resultado:

clf = LogisticRegression()

clf.fit(x, y)

B0\_e = clf.intercept\_

B1\_e = clf.coef\_[0,0]

Be\_e = clf.coef\_[0,1]

### Entrenamiento y Validación

Dividimos el dataset en 2 conjuntos de datos. Uno lo utilizamos para entrenar el modelo (buscar parámetros) y el otro conjunto lo usamos para evaluar al modelo creado.

#### División Train-Test

x\_train, x\_test, y\_train, y\_test = traint\_test\_split(ds[col], ds[var], test\_size=0.2, random\_state=2)

#### 

#### Validación Cruzada

cv = LogisticRegressionCV(cv=5, scroing=’roc\_auc’)

cv.fit(x\_train, y\_train)

y\_pred = cv.predict(c\_test)

y\_pred\_proba = cs.predict\_proba(x\_test)[:, 1]

### Métricas

metrics.accuracy\_score(cv.var, cv.var\_predict)

metrics.fi\_score(cv.var, cv.var\_predict)

metrics.recall\_score(cv.var, cv.var\_predict)

metrics.precision\_score(cv.var, cv.var\_predict)

# Métodos de Agrupamiento (Clustering)

## K Means

Algoritmo de aprendizaje no supervisado. Agrupa observaciones para un número predefinido de grupos. El centro de cada grupo es la medida aritmética de todas las observaciones pertenecientes a ese grupo. Cada observación está más cerca del centro de su grupo que de cualquier centro de otro grupo.

kmeans = KMeans(n\_clusters=n)

kmeans.fit(x)

y\_kmeans = kmeans.predict(x)

## Métricas de evaluación

### ¿Cómo elegir el tamaño del Cluster?

#### Método de Elbow (Regla del Codo)

Verificar la evolución de la suma de los cuadrados del error para varias cantidades de clusters y verificar cuál es el que brinda el mejor agrupamiento.

sse = []

list\_k = list(range(2, n))

for k in list\_k:

km = KMeans(n\_clusters=k)

km.fit(x)

sse.append(km.inertia\_)

Grafico y me quedo con la cantidad de clusters donde se quiebra la pendiente.

#### Índice de Silhouette

Indica si existen o no estructuras en los datos.

from sklearn.metrics import silhouette\_score

list\_k = list(range(2, n\_clusters+2))

for n\_clusters in list\_k:

clusterer = KMeans(n\_clusters=n\_clusters)

preds = clusterer.fit\_predict(x)

score = silhouette\_score(x, preds)

Graficamos:

from yellowbrick.cluster import SilhouetteVisualizer

model = KMeans(4, random\_state=0)

visualizer = SilhouetteVisualizer(model)

visualizer.fit(x)

plt.show()

Vemos que:

* el ancho de los clusters tienen que ser parejos
* que no se formen colas negativas del índice
* el valor promedio de silhouette sea alto (la clasificación es sólida, está bien hecha)

## Hopkins: Tendencia al Clustering

from pyclustertend import hopkins

hopkins(x, x.shape[0])

Si el número es bajo, el dataset tiene tendencia a clustering.

# Feature Engineering (Transformación de Datos)

## Conversión de variables categóricas

Variables que expresan una cualidad, por lo general no son numéricas (ej: bueno/malo/regular).

### Ordinal Encoder

Transformamos la variable categórica en un número:

oe = OrdinalEncoder(dtype=’int’)

columns\_to\_encode = [columns]

try:

ds[[columns\_encoded]] = oe.fit\_transform(ds[columns\_to\_encode])

except Exception as exc:

print(f’Error: {exc}’)

### Label Encoder

Misma idea esperando una sóla variable ya que se usa para encodear la variable target de un modelo predictivo:

le = LabelEncoder()

ds[column\_label\_encoded] = le.fit\_transform(ds[column].astype(str))

### One Hot Encoding

Crea tantas columnas como clasificaciones existan y pone 1 a la clasificación que pertenezca la observación y 0 al resto.

ohe = OneHotEncoder()

property\_type\_encoded =

ohe.fit\_transform(ds[[column]].astype(str)).todense().astype(int)

property\_type\_encoded = pd.DataFrame(property\_type\_encoded).add\_prefix(column\_)

## Conversión de variables numéricas

### Transformación Min-Max

Buscamos el valor mínimo de la variable y a cada valor que tiene esa columna le restamos el mínimo y lo dividimos entre la resta del máximo y el mínimo. Nos lleva a un valor entre 0 y 1.

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

var\_min\_max = scaler.fit\_transform(ds[var].to\_frame())

El histograma entre la variable original y la nueva col\_min\_max no cambia.

### Transformación Z-Score

Calcula la media de la distribución y lo divide por el desvío:

from sklearn.preprocessing import StandardScaler

standar\_scaler = StandarScaler()

var\_z\_score = standard\_scaler.fit\_transform(ds[var].to\_frame())

### Transformación Decimal Scaling

Busca el valor más grande de la variable y se estandariza dividiendo por 10^cantidad\_digitos\_valor\_mas\_grande.

d = len(str(int(ds[var]max())))

ds[var\_decimal\_scaling] = ds[var]/10\*\*d

### Transformación de Potencias

Se lleva a la variable a una distribución normal, cambia la forma del histograma:

from sklearn.preprocessing import PowerTransformer

power\_scaler = PowerTransformer(metohd=’box-cox’)

var\_power = power\_scaler.fit\_transform(ds[var].to\_frame())

## Discretización

Transformamos una variable continua a una discreta o categórica.

### Binning

Ordenamos por cuantil: ordenamos todos los valores de la variable, buscamos el valor de la variable que deja tal cantidad de observaciones a izquierda

ds\_discret = ds.var.toframe()

enc = KBinsDiscretizer(n\_bins=n, encode=’ordinal’, strategy=’quantile’)

\_ds = ds\_discret[[var]].reset\_index(drop=True)

enc.fit\_transform(\_ds)

print(f“Límites bins: ”, enc.bin\_edges\_)

## Valores Faltantes

### Eliminar Nans

Si tienen un porcentaje muy alto sobre los datos. Es lo más fácil pero no siempre es lo que más conviene.

### Tratar la variable como una categoría más

Poner un valor constante, por ejemplo: “No existe”.

### Completar utilizando información de esa columna (univariadas)

Completar con la mediana, promedio, moda o constante.

## Análisis de Valores Atípicos

### Análisis Univariado

* Box Plot: analizamos lo que esté arriba de los bigotitos
* Z Score: lo que esté a más de 3 desvíos nos llama la atención
* Z Score Modificado

### Análisis Multivariado

* Mahalanobis: grafico el box plot de distancias y selecciono el umbral de valores atípicos
* Isolation Forest: las observaciones que llegaban a ser hoja mas cerca del nodo raíz, tiene mas chance de ser outlier que los que están más profundos; definimos los valores de los hiperparámetros:
  + contamination: proporción de anomalías en el conjunto de datos
  + max\_samples: número máximo de muestras (para usar todas lo seteamos en ‘auto’)
  + max\_features: número máximo de características para el entrenamiento
  + n\_estimators: número de árboles de aislamiento
* LOF: cantidad de vecinos y qué tanta densidad de vecinos tienen los vecinos. Para definir la cantidad de vecinos hay que ir viendo cuántos me quedarían como outliers con cada corte y ver qué es lo que estoy dejando afuera.

# Árboles de Decisión para Clasificación

Usamos los sets de test.

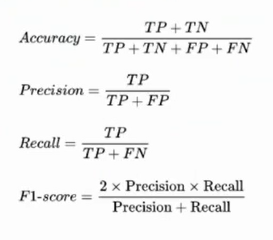
No pueden interpretar datos no numéricos. Las variables categóricas no pueden estar expresados en strings.

Cuando dejamos crecer el árbol en su totalidad no esta bueno porque se va a aprender muy bien los datos de entrenamiento y cuando los llevamos a los datos de tests no performan bien -> overfitting -> para esto podamos. Una técnica de poda es dejarlo crecer y después cortarlo (se implementa con el parámetro alpha).

## Búsqueda/Optimización de Hiperparámetros

Usamos validación cruzada para optimizar hiperparámetros. Buscamos los parámetros sobre el conjunto de entrenamiento (profundidad, poda, cantidad de hojas, etc.: los que tenga el modelo que en este caso es un árbol). Cuando encontramos los mejores entrenamos el modelo y hacemos una evaluación sobre los datos de test. El flujo es: separar, entrenar, ajustar (que dio bien en entrenamiento), probar en test.

Afinamos los parámetros para una métrica en particular. En test medimos esa misma métrica.



### Randomized Search Cross Validation

Para buscar la mejor combinación de hiperparámetros usamos k-fold CV para medir el desempeño de cada combinación. Al terminar nos quedamos con la que mejor desempeño tuvo y entrenamos un único modelo usando todos los datos de train. Hay dos formas:

* Random Search: busca alguna de las combinaciones, de forma rándom
* Grid Search: busca todas las combinaciones que existan combinando los hiperparámetros

Random es mejor que Grid search porque la última tarda mucho.

# Reducción de la dimensionalidad

La idea es dejar en un dataset los features más importantes. Para filtr

# Redes Neuronales

## Descenso por Gradiente

Método para hallar parámetros del modelo buscando que se minimice el error.

## Keras Regresión

epocas = epochs -> cantidad de iteraciones de entrenamiento