Reducción de Dimensionalidad en Scikit-Learn

Alfonso Tobar Arancibia

Data Scientist

01-06-2021

github.com/Rcubes/clases-ml/Slides

La maldición de la Dimensionalidad

El proceso de Modelamiento suele ser un proceso tedioso de mucho costo computacional. En estricto rigor un modelo entre más data para aprender tenga mucho mejor desempeño tendrá. El problema de eso es que eso conlleva un costo.

> Es necesario reducir el tamaño de mi procesamiento afectando el modelo lo menos posible.

Soluciones:

- **Factor Analysis**
- **PCA y amigos**
- **Variable Selection**

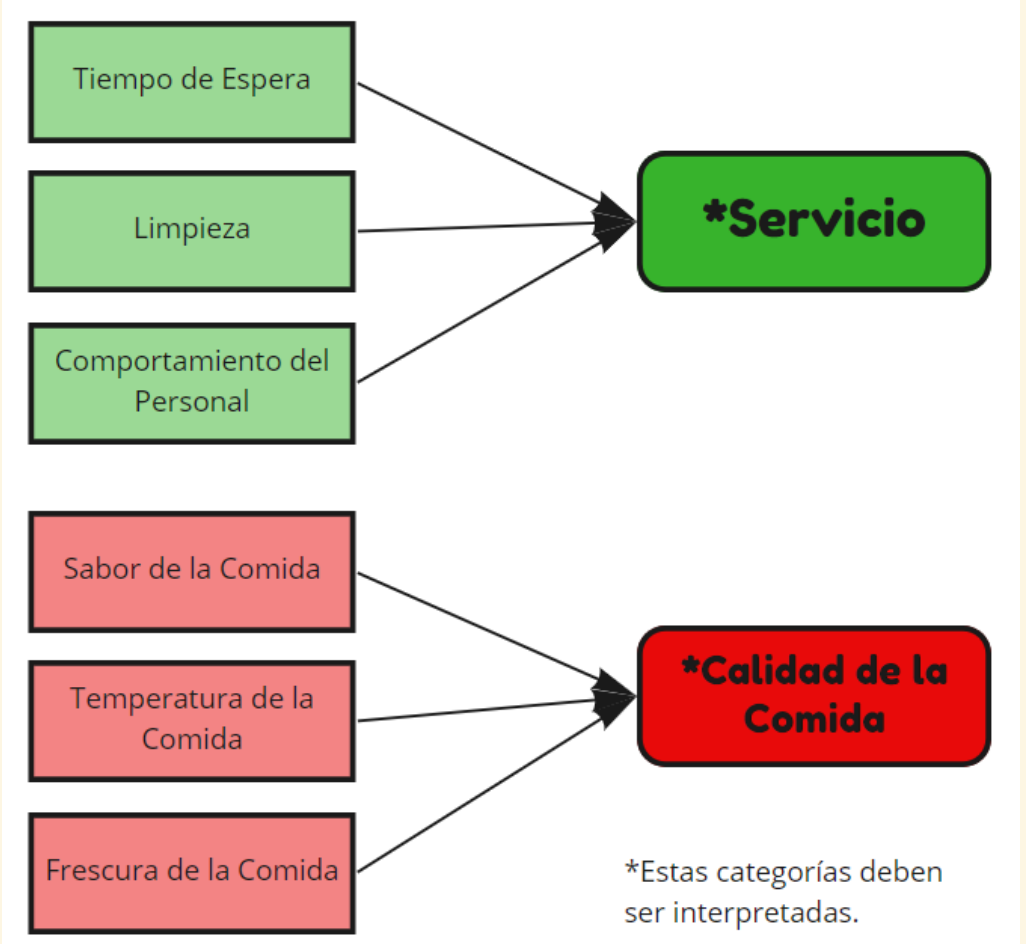
Análisis Factorial (Factores Latentes)

Una primera solución es el Análisis Factorial. Este análisis se basa en la hipótesis de que descomponiendo la matriz de atributos en factores latentes que agrupan/resumen las variables originales. Está fuertemente orientado a la interpretabilidad.

La implmentación en Python es simple:

```
from factor_analyzer import FactorAnalyzer
fa = FactorAnalyzer(n_factors = 2)
fa.fit(X)
fa.transform(X)
```

pero existen algunas condiciones que deben verificarse antes:



Análisis Factorial (Factores Latentes)

Condiciones

- **Test de Esfericidad de Bartlett**: Corresponde a un test que mide las siguientes Hipótesis:
  - H_0: La matriz de correlaciones es igual a la matriz Identidad
  - H_1 La matriz de corelaciones no es igual a la matriz Identidad

```
chi_square_value,p_value=calculate_bartlett_sphericity(X)
p_value # Chequear que p-value sea menor que 0.05
```

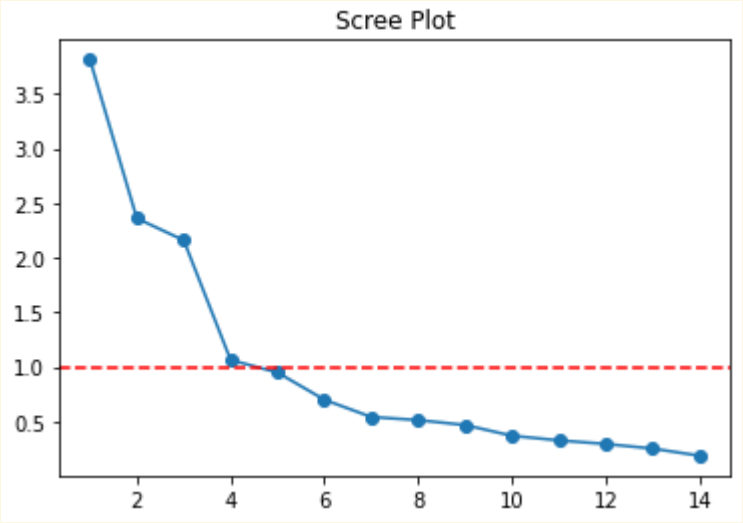
- **Criterio de Kaiser-Meyer-Olkin**: Representa el grado en el que cada variable observada es predicha sin error, mediante el uso del resto del dataset. KMO debe ser mayor a 0.6.

```
from factor_analyzer.factor_analyzer import calculate_kmo
kmo_all,kmo_model=calculate_kmo(X)
print(kmo_model) # corresponde al valor total del modelo
print(kmo_all) # corresponde al kmo por variable
```

Análisis Factorial (Factores Latentes)

Factores Latentes Óptimos

Para encontrar el número óptimo se realiza un análisis de los valores propios del problema de descomposición. Se suelen utilizar los valores propios mayores a 1 ya que eso representa la cantidad de variables a las que puede explicar su varianza.



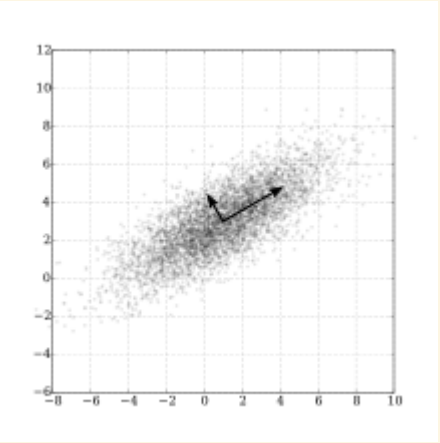
Principal Component Analysis

Quizás es el procedimiento más conocido y más poderoso en cuanto a Reducción de Dimensionalidad. El PCA tiene como objetivo principal descomponer un set de variables en componentes ortogonales que expliquen la mayor cantidad de varianza.

> NOTA: El resultado de este análisis es un nuevo dataset, en que las columnas resultantes corresponden a las componentes principales.

Pro: Es un Transformer no supervisado.

Contra: Las componentes principales resultantes no son interpretables.



Principal Component Analysis

```
from sklearn.decomposition import PCA
pca = PCA(n_components = None)
pca.fit(X)
pca.transform(X)
pca.fit_transform(X)
```

Hiperparámetro:

- **n_components**: Por defecto es `None` que significa utilizar el mismo número de columnas de entrada. Este hiperparámetro puede ser un número para indicar el número de componentes principales a reducir o un valor entre 0 y 1 para indicar las componentes principales que indiquen dicho porcentaje de varianza.
- `pca.explained_variance_ratio_` entrega el porcentaje de varianza que explica cada componente principal.

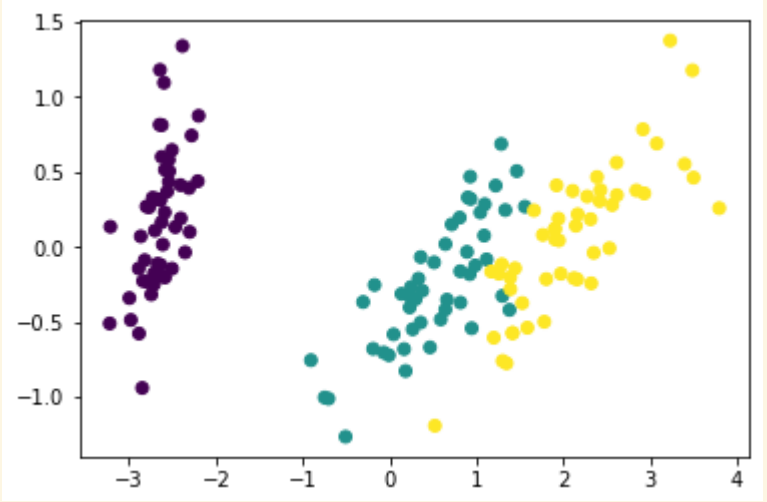
Aplicación en Visualizaciones

Muchas veces los datasets que se manejan tienen tantas dimensiones que no es posible visualizarlos en las dimensiones que como humanos podemos interpretar. Es por esto que reduciendo por ejemplo a 2 dimensiones podemos fácilmente ver cómo se separan datasets famosos como el iris.

```
from sklearn.datasets import load_iris

data = load_iris(as_frame = True)
X = data.data
y = data.target
pca = PCA(n_components = 2)
X_red = pca.fit_transform(X)
y_red = y.astype('category').cat.codes

plt.scatter(X_red[:,0], X_red[:,1], c = y_red)
plt.show()
```



Selección de Variables en `Scikit-Learn`

La API de `scikit-learn` contempla los siguientes Transformers para la selección de variables.

```
# Selecciona las `k` mejores variables donde `k` es un número entero.
SelectKBest(score, k = 10)
# Selecciona las variables con tal de tener un `percentile` % resultante.
SelectPercentile(score, percentile)
# Utiliza un modelo para dejar las variables que sobrepasen un cierto treshold.
SelectFromModel(estimator, threshold = None)
# Utiliza un modelo para eliminar variables de manera recursiva hasta llegar al valor deseado.
RFE(estimator, n_features_to_select = None)
```

Scoring

- **f_regression, f_classif:** Realizan un análisis de varianza (ANOVA) para escoger las variables más significativas.
- **mutual_info_regression, mutual_info_classif:** Realiza un análisis de información Mutua para determinar independencia entre cada uno de los predictores y el target. 0 implica independencia y entre mayor sea el número implica mayor relación de dependencia.
- **chi2 (Sólo Clasificación)**: Este test mide el grado de independencia, pero sólo funciona para variables positivas y en general Booleanos y Conteos.

Selección de Variables en `Scikit-Learn`

> NOTA: Normalmente estos Procedimientos de selección de variable pueden introducir Data Leakage por lo para buscar el k, percentiles óptimo, debieran ser usados dentro de un contexto de Pipeline + GridSearch.

Ejemplo de Uso (Sin GridSearch)

```
from sklearn.feature_selection import f_regression, SelectPercentile

pipe = Pipeline(steps = [
    ('vs', SelectPercentile(f_regression, percentile = 70)),
    ('lr', LinearRegression())
])

pipe.fit(X_train, y_train)
pipe.score(X_test, y_test)
```