boruta

January 11, 2023

1 Paquetes a utilizar

[ ]:
```
import sklearn as skl
import pandas as pd
import numpy as np
import xgboost as xgb
from boruta import BorutaPy
```

2 Base de datos

[ ]:
```
df = pd.read_csv("data/wisconsin_breast_cancer_dataset.csv")
df = df.dropna()
df = df.rename(columns={'diagnosis':'Label'})
df['Label'].value_counts()
```

[ ]:
```
df.describe().T
```

[ ]:
```
df.dtypes
```

3 Variable dependiente que debe predecirse y codificación de datos categóricos

[ ]:
```
y = df["Label"].values
Y = skl.preprocessing.LabelEncoder().fit_transform(y)
```

4 Definir x, normalizar valores y definir variables independientes

[ ]:
```
X = df.drop(labels = ["Label", "id"], axis=1)
nombres_de_funciones = np.array(X.columns)
scaler = skl.preprocessing.StandardScaler()
scaler.fit(X)
X = scaler.transform(X)
```

5 Train and test para verificar la precisión después de ajustar el modelo

```
[ ]: X_train, X_test, y_train, y_test = skl.model_selection.train_test_split(X, Y,␣
      ↪test_size=0.25, random_state=42)
```

6 XGBOOST para ser utilizado por Boruta

```
[ ]: modelo = xgb.XGBClassifier()
```

- Crear funciones de sombra: funciones aleatorias y valores aleatorios en columnas
- Entrenar Random Forest / XGBoost y calcular la importancia de la característica a través de la disminución media de la impureza
- Comprobar si las características reales tienen mayor importancia en comparación con las características de sombra
- Repetir esto para cada iteración
- Si la función original funcionó mejor, marcarla como importante.

```
[ ]: # Especificar cómo se determinan las características relevantes utilizando el␣
      ↪algoritmo Boruta
     selector = BorutaPy(modelo, n_estimators='auto', verbose=2, random_state=1)
     # Hallar todas las características importantes
     selector.fit(X_train, y_train)
     # Aplicar el método transform() en el conjunto de datos X para limitarlo a solo␣
      ↪las características seleccionadas
     X_filtered = selector.transform(X_train)  # Utilizar técnicas de selección de␣
      ↪características y obtener el conjunto de datos modificado
```

7 zip nombres de características, rangos y decisiones

```
[ ]: feature_ranks = list(zip(nombres_de_funciones,
                             selector.ranking_,
                             selector.support_))
```

8 Resultados

```
[ ]: for feat in feature_ranks:
         print('Feature: {:<30} Rank: {},  Keep: {}'.format(feat[0], feat[1],␣
      ↪feat[2]))
```

```
[ ]: # Utilizando un conjunto específico de características, ajustar un modelo␣
      ↪XGBoost en los datos de entrenamiento
     xgb_model = xgb.XGBClassifier()
     xgb_model.fit(X_filtered, y_train)
```

```
# Emplear el modelo previamente entrenado para hacer predicciones con el␣
 ↪conjunto de datos de prueba
# Antes de hacer las predicciones, aplicar las mismas técnicas de selección de␣
 ↪características en los datos de prueba para asegurarse de que solo se estén␣
 ↪utilizando las características relevantes
X_test_filtered = selector.transform(X_test)
prediction_xgb = xgb_model.predict(X_test_filtered)
```

8.1 Precisión

[ ]:
```
skl.metrics.accuracy_score(y_test, prediction_xgb)
```