Sofia Paola Medina Chica

Proyecto final - Clasificador de Vinos

Introducción

El vino es una de las bebidas que guarda una mayor cantidad de sabores debido a que es una de las mas naturales, su proceso de obtención proviene de la fermentación de las uvas, el cual se lleva a cabo de forma natural. Esta bebida a tomado una gran fama a nivel mundial por su dulce sabor, y por ser un gran acompañante en las comidas, la que la posiciona como una de las bebidas alcohólicas mas sofisticadas internacionalmente.

Todo esto ha llevado a generar una necesidad de calificar y clasificar a vinos según ciertas de sus características, lo que genero una fuerte demanda de catadores de vino, dado que, dependiendo de la calificación que obtenga un vino, dependerá su valor comercial. No obstante, contratar a un buen catador de vino se ha convertido en algo muy costoso.

Este proyecto, pretende clasificar un vino de 1 a 10, en donde 10 es la mejor calidad de vino, sin la necesidad de contratar a un catador. Este clasificador se desarrolló a partir de unos datos ya previamente clasificados por un catador, donde se tienen en cuenta las siguientes 11 variables: acidez fija, acidez volátil, ácido cítrico, azúcar residual, cloruros, dióxido de azufre libre, dióxido de azufre total, densidad, pH, sulfatos y alcohol.

Implementación y Resultados

Se realizará la implementación de un problema supervisado de clasificación, para el data set encontrado se revisó que no hubiera ningún dato vacío para poder así llevar a cabo el problema. El data set de la calidad de vinos cuenta con 1599 datos tomados, los cuales se dividieron en el 75% para entrenar y el 25% para validar. A partir de estos datos, se escalizarón las etiquetas con StandardScaler.

Representación y reducción dimensional

Para la reducción dimensional por PCA, primero se determino el peso de cada una de las características con el código que se puede ver a continuación:

```
n=11
pca = decomposition.PCA(n_components=n,whiten=True)

pca.fit(X)
X = pca.transform(X)

print("Pesos de PCA:",pca.explained_variance_ratio_)
```

```
Pesos de PCA: [0.28168526 0.17507198 0.14105725 0.1102609  0.08714744 0.0599412
 0.05308397 0.03852238 0.03124977 0.01654991 0.00542992]
```

Con estos pesos fue posible determinar que se lograba 97.8% de la varianza explicada con 9 de las características, por lo que se volvió a realizar PCA para 9 componentes para realizar la reducción dimensional, los pesos quedaron de la siguiente forma:

```
n=9
pca = decomposition.PCA(n_components=n,whiten=True)

pca.fit(X)
X_pca = pca.transform(X)

print("Pesos de PCA:",pca.explained_variance_ratio_)
```

```
Pesos de PCA: [0.28168526 0.17507198 0.14105725 0.1102609  0.08714744 0.0599412
 0.05308397 0.03852238 0.03124977]
```

Clasificadores

Para la métrica de evaluación de los clasificadores se tomaron el accuracy, F1-score y el MCC.

- **Support Vector Machine**
  En este clasificador se decidió iterar el tipo de Kernel entre 'linear', 'poly', 'rbf' y 'sigmoid'. En donde se obtuvieron los siguientes resultados para cada caso:
  - Accuracy
    `[0.615, 0.6075, 0.6225, 0.635]`
  - F1-score
    `[0.615, 0.6075, 0.6225, 0.635]`
  - MCC
    `[0.34097532338839537, 0.3403853233580038, 0.3933993648489817, 0.3897627510659096]`

- **Naive Bayes**
  Este clasificador no tiene parámetros que iterar, por lo que solo se obtuvo un resultado:
  - Accuracy
    `0.5575`
  - F1-score
    `0.5575`
  - MCC
    `0.28939124228150137`

- **Red Neuronal**
  En este clasificador se decidió iterar el tipo de función de activación de las neuronas, que en este caso son: 'identity', 'logistic', 'tanh' y 'relu'. Además de esto se itero el tamaño de las subcapas de la red en valores de 10 en 10, en donde la primera y la tercera tenían el doble de la segunda. (ej: si la segunda era 10, la tercera y la primera tenían 20), por lo que para poder mostrar más los resultados se mostrara los que se obtuvieron para una misma función de activación, pero diferente cantidad de neuronas. En donde se obtuvieron los siguientes resultados para cada caso:
  - Accuracy
    - *Identity*
      `0.64, 0.6475, 0.64, 0.6325, 0.6375`
    - *Logistic*
      `0.6, 0.6175, 0.6025, 0.6075, 0.5925`
    - *Tanh*
      `0.6025, 0.605, 0.62, 0.6225, 0.5925`

- *Relu*
  0.6275, 0.6125, 0.65, 0.6075, 0.6125

– F1-score
  - *Identity*
    0.64, 0.6475, 0.64, 0.6325, 0.6375
  - *Logistic*
    0.6, 0.6175, 0.6025, 0.6075, 0.5925
  - *Tanh*
    0.6025, 0.605, 0.62, 0.6225, 0.5925
  - *Relu*
    0.6275, 0.6125, 0.65, 0.6075, 0.6125

– MCC
  - *Identity*
    0.40466742981246556, 0.41663348419676743, 0.40252247644808337, 0.3955562556114036, 0.39862736021755646
  - *Logistic*
    0.34627129796577943, 0.37073774769873, 0.3500479342919961, 0.3603112936474103, 0.34382427990531167
  - *Tanh*
    0.3627495669421946, 0.380219615622786, 0.3961222374837941, 0.40762917874057825, 0.35766792097732797
  - *Relu*
    0.4072639214954576, 0.3877956680331222, 0.4392171330102666, 0.37903857490024734, 0.3809271034702023

- **KNN**
  Finalmente, para KNN se itero el peso de la distancia (‘uniform' y 'distance') y la cantidad de vecinos más cercanos de 1 a 9. En donde se obtuvieron los siguientes resultados para cada caso:
  – Accuracy
    - *Uniform*
      0.615, 0.595, 0.61, 0.6075, 0.6125, 0.595, 0.5875, 0.585, 0.5675
    - *Distance*
      0.615, 0.595, 0.61, 0.6075, 0.6125, 0.595, 0.5875, 0.585, 0.5675
  – F1-score
    - *Uniform*
      0.615, 0.595, 0.61, 0.6075, 0.6125, 0.595, 0.5875, 0.585, 0.5675
    - *Distance*
      0.615, 0.595, 0.61, 0.6075, 0.6125, 0.595, 0.5875, 0.585, 0.5675
  – MCC
    - *Uniform*
      0.39527082427517596, 0.34953177506398464, 0.3703577012756389, 0.3698400908412734, 0.37459754162875075, 0.3512622408955923, 0.3372487324666177, 0.3310249449458567, 0.3054690544964874
    - *Distance*
      0.39527082427517596, 0.34953177506398464, 0.3703577012756389, 0.3698400908412734, 0.37459754162875075, 0.3512622408955923, 0.3372487324666177, 0.3310249449458567, 0.3054690544964874

Después de la implementación de todos los modelos se encontró que el mejor modelo es la red neuronal con 30 neuronas en la segunda capa y 60 en la primera y tercera capa, con una función de activación *relu.*

Una vez se tiene el mejor modelo se procede a entrenar el modelo sobre todo el conjunto, para obtener el modelo final.

```
activations = ['identity', 'logistic', 'tanh', 'relu']

p = 3
Modelo = MLPClassifier(hidden_layer_sizes=(60,30,60),alpha=0.01,activation=activations[p],random_state=1,learning_rate='adaptive', max_iter=1000
Modelo.fit(X, y)
```

Conclusiones

El mejor modelo que se obtuvo para resol ver este problema fue una red neuronal con tres capas, era de esperarse que este fuera el modelo que mejor resultados se obtuviera debido a que es uno de los clasificadores más robustos, con este se obtendrá un 65 % de precisión, lo cual no es un valor muy alto, pero sin embargo tiene un buen desempeño. El valor bajo puede deberse a que el data set no se encuentra equilibrado, porque puede haber cientos de datos para una calificación de la calidad de vino de 7 pero hay solo 10 de una calificación de 8 y así.

El modelo en donde se obtuvo una menor precisión fue en Naive Bayes, donde se obtuvo un valor de 55.75%. Esto se debe a que como su nombre lo indica, es un modelo inocente, ya que este no tiene en cuenta que unas variables pueden llegar a tener dependencia de otras y toma la etiquetas como si fueran completamente independientes.

La red neuronal, a pesar de ser el mejor modelo, fue el que más se demoró en entrenar, mientras que los demás modelos entrenaban en cuestión de segundos.