Análisis y reporte sobre desempeño de modelo

KNN (K Nearest Neighbors)

Andrés Piñones Besnier - A01570150

En este análisis daremos un repaso de los pasos seguidos para implementar un algoritmo de Machine Learning haciendo uso de un framework, en este caso será el de KNN implementado en una de mis entregas.

Una vez teniendo resultados del desempeño buscaremos encontrar mejoras por medio de pruebas y modificaciones en caso de ser necesario.

Para este ejemplo utilizaremos el dataset de Palmer Archipelago (Antarctica) penguin data que documenta los datos de algunas características físicas y lugar de origen de 3 especies diferentes de pinguinos.

El objetivo es encontrar un modelo que clasifique dichas 3 especies para un input test set.

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

Análisis exploratorio de datos

Comenzamos con el análisis del dataset para entender los datos.

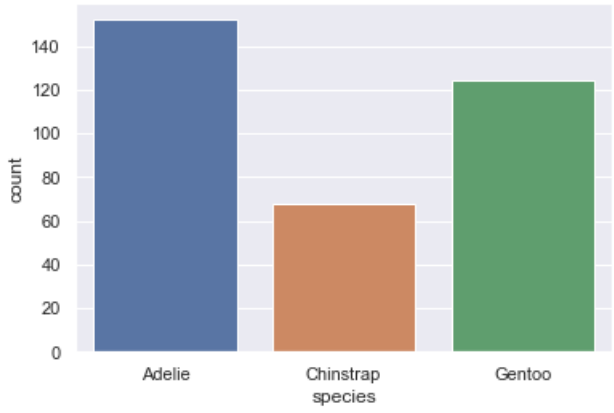
In [ ]:
```
#leemos el dataset y lo asignamos a un pandas df
df = pd.read_csv('penguins_size.csv')
df.head()
```

Out[ ]:

	species	island	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	MALE
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	FEMALE
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	FEMALE
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	FEMALE

In [ ]:
```
sns.countplot(x=df['species'])
```

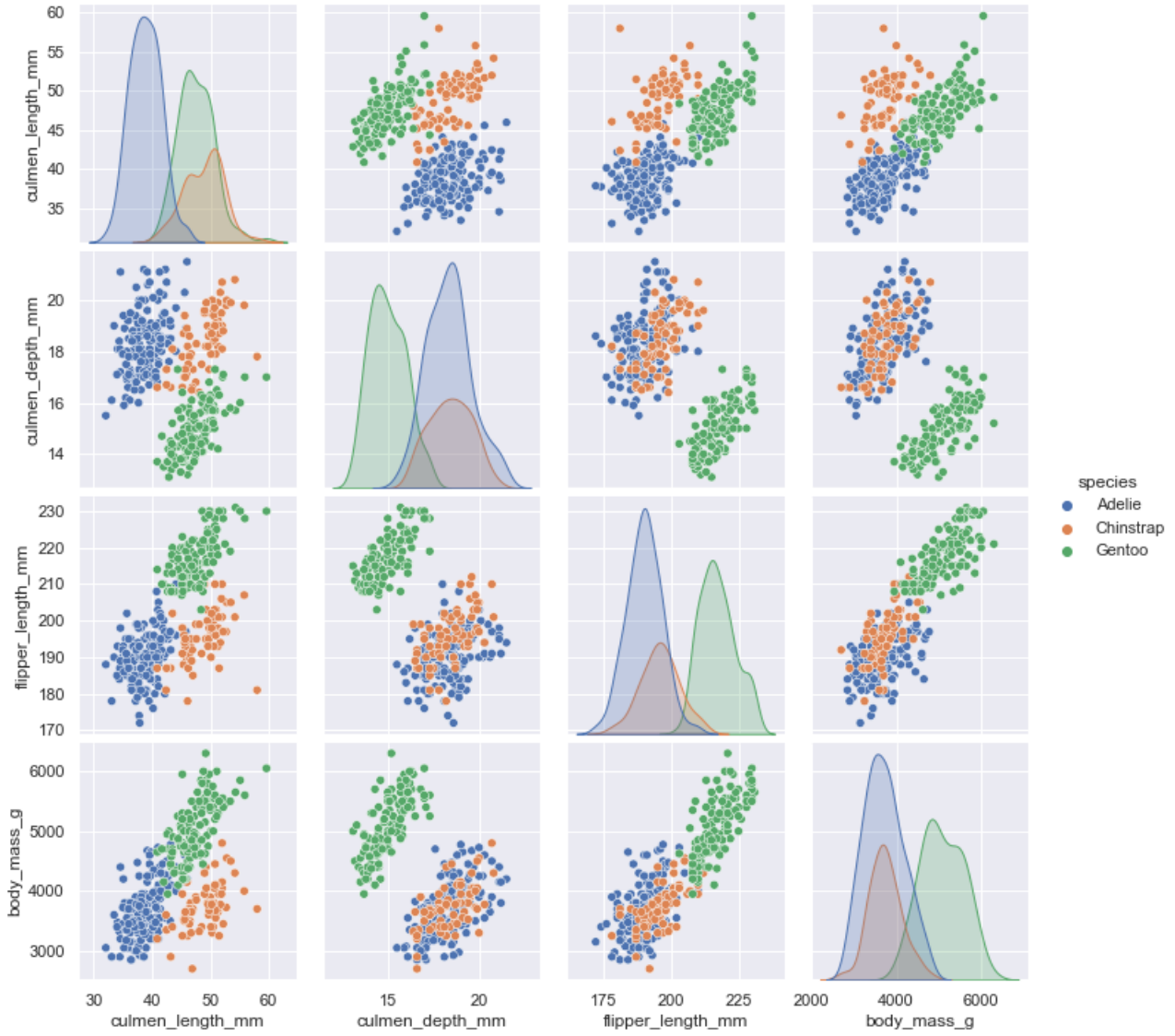
Out[ ]:
```
<AxesSubplot:xlabel='species', ylabel='count'>
```



Como se puede observar por el gráfico anterior no tenemos el mismo numero de samples para cada clase, esto en

ocasiones puede tener efectos en los resultados o un cierto sesgo. Existen dos métodos para balancear los samples, el up-sampling minority y el down-sampling majority. En este caso como son pocos datos así lo dejaremos.

In [ ]:
```
sns.pairplot(df,hue='species');
```



Aqui se puede apreciar visualmente lo definidas que están las especies en las diversas variables, unas más evidentes que otras pero en general se pueden ver los 'clusters' donde se concentran las diferentes clases. Esto puede indicar que hay diferencias algo marcadas que pueden facilitar la clasificación.

In [ ]:
```
df.loc[:,df.columns != 'species'].describe()
```

Out[ ]:

	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g
count	342.000000	342.000000	342.000000	342.000000
mean	43.921930	17.151170	200.915205	4201.754386
std	5.459584	1.974793	14.061714	801.954536
min	32.100000	13.100000	172.000000	2700.000000
25%	39.225000	15.600000	190.000000	3550.000000
50%	44.450000	17.300000	197.000000	4050.000000
75%	48.500000	18.700000	213.000000	4750.000000
max	59.600000	21.500000	231.000000	6300.000000

In [ ]:
```
df.loc[:,df.columns != 'species'].var()
```

```
/var/folders/rw/kl6lbv6s10g1zlg813xt_w_40000gn/T/ipykernel_20441/3639205615.py:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only valid columns before calling the reduction.
  df.loc[:,df.columns != 'species'].var()
```

Out[ ]:
```
culmen_length_mm         29.807054
culmen_depth_mm           3.899808
flipper_length_mm       197.731792
body_mass_g          643131.077327
dtype: float64
```

Variables categóricas

In [ ]:
```
df.species.value_counts()
```

Out[ ]:
```
Adelie       152
Gentoo       124
Chinstrap     68
Name: species, dtype: int64
```

In [ ]:
```
df.sex.value_counts()
```

Out[ ]:
```
MALE      168
FEMALE    166
Name: sex, dtype: int64
```

In [ ]:
```
df.island.value_counts()
```

Out[ ]:
```
Biscoe       168
Dream        124
Torgersen     52
Name: island, dtype: int64
```

Limpieza

Revisamos valores nulos

In [ ]:
```
dataset = pd.concat([df], sort=False)
pd.DataFrame({'No. NaN': dataset.isna().sum(), '%': dataset.isna().sum() / len(dataset)})
```

Out[ ]:

	No. NaN	%
species	0	0.000000
island	0	0.000000
culmen_length_mm	2	0.005814
culmen_depth_mm	2	0.005814
flipper_length_mm	2	0.005814
body_mass_g	2	0.005814
sex	10	0.029070

Son pocos datos nulos por lo que podemos optar por imputación de la media en donde faltan datos.

In [ ]:
```
#para cada columna donde faltan valores le ponemos la media de dicha columna
df['sex'].fillna(df['sex'].mode()[0],inplace=True)
col_with_null = ['culmen_length_mm', 'culmen_depth_mm','flipper_length_mm', 'body_mass_g']
for i in col_with_null:
    df[i].fillna(df[i].mean(),inplace=True)
```

Para poder utilizar nuestras variables categóricas es necesario volverlas dummies

In [ ]:
```
#encodeamos la variable objetivo (la especie)
df['species']=df['species'].map({'Adelie':0,'Gentoo':1,'Chinstrap':2})

#hacemos dummies de las variables categoricas
dummies = pd.get_dummies(df[['island','sex']],drop_first=True)
```

Escalado de variables

Como el algoritmo de KNN calcula distancias para la clasificacion y las variables tienen escalas muy distintas es necesario escalarlas.

In [ ]:
```
#tenemos que escalar las variables, removemos las categoricas pues estas no se escalan.
df_scalable = df.drop(['island','sex'],axis=1)
#variable objetivo
target = df_scalable.species

df_feat= df_scalable.drop('species',axis=1)
```

Usaremos el standard scaler de sklearn y con esto terminamos el preprocesamiento y limpieza de datos.

In [ ]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(df_feat)
df_scaled = scaler.transform(df_feat)
df_scaled = pd.DataFrame(df_scaled,columns=df_feat.columns[:4])
df_preprocessed = pd.concat([df_scaled,dummies,target],axis=1)
df_preprocessed.head()
```

Out[ ]:

	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g	island_Dream	island_Torgersen	sex_MALE	s
0	-8.870812e-01	7.877425e-01	-1.422488	-0.565789	0	1	1	
1	-8.134940e-01	1.265563e-01	-1.065352	-0.503168	0	1	0	
2	-6.663195e-01	4.317192e-01	-0.422507	-1.192003	0	1	0	
3	-1.307172e-15	1.806927e-15	0.000000	0.000000	0	1	1	
4	-1.328605e+00	1.092905e+00	-0.565361	-0.941517	0	1	0	

Modelo

In [ ]:
```
#Asignamos nuestros datos para definir la variable objetivo y las predictoras (dependiente e independie
#declaramos como X todas nuestras variables predictoras
X = df_preprocessed.drop(['species'], axis = 1)
# con esto tomamos solo a la variable 'species' como variable dependiente que declaramos antes como tar
Y = target
```

Split de entrenamiento y prueba

In [ ]:
```
from sklearn.model_selection import train_test_split

# hacemos el split de entrenamiento y prueba pues es un algoritmo supervisado y removemos
# en este caso usaremos un split de 75% entrenamiento y 25% prueba.
X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size=0.25, random_state=10 )
```

K Nearest Neighbors

Como el objetivo es clasificar los pinguinos por especie utilizamos KNN como modelo de clasificación de aprendizaje supervisado pues tenemos las clases definidas en el entrenamiento.

Entrenamiento

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier

#creamos el modelo y le damos un numero 1 de vecinos para ponerlo a prueba de manera default
knn = KNeighborsClassifier(n_neighbors=1)

#entrenamos el modelo con el set de entrenamiento correspondiente
knn.fit(X_train,y_train)
```

Out[ ]:
```
▾        KNeighborsClassifier
KNeighborsClassifier(n_neighbors=1)
```

Validación

Realizamos 5 predicciones para validar el aprendizaje y evaluamos con una matriz de confusión y un score de precisión.

In [ ]:
```
from sklearn.metrics import confusion_matrix,accuracy_score

y_pred = knn.predict(X_train)

print('Training set matrix and score:')
print(confusion_matrix(y_train,y_pred))
print(accuracy_score(y_train,y_pred))
```

```
Training set matrix and score:
[[116   0   0]
 [  0  94   0]
 [  0   0  48]]
1.0
```

Como se puede ver que el entrenamiento demuestra que tuvo aprendizaje para clasificar con precisión de 100% todos los datos. Esto puede indicar overfitting, pero realizaremos pruebas ya con el set de pruebas para verificar el modelo.

In [ ]:
```
y_pred = knn.predict(X_test)

print('Testing set matrix and score:')
print(confusion_matrix(y_test,y_pred))
print(accuracy_score(y_test,y_pred))
```

```
Testing set matrix and score:
[[35  0  1]
 [ 0 30  0]
 [ 0  0 20]]
0.9883720930232558
```

Como se puede ver el modelo tuvo una muy buena predicción con precisión de 98% solamente clasificó erroneamente uno de los pinguinos.

Debido a que el entrenamiento nos devuelve un 100% y la predicción es menor se puede decir que hay algo de overffiting pero como es un muy buen resultado podemos decir que es un buen fit el modelo.

A pesar de esto hagamos pruebas modificando hiperparametros del proceso para ver si es posible mejorar o ver que efecto tienen estos.

Pruebas y mejora

Hay diferentes factores que influyen o pueden tener impacto en el resultado de nuestro modelo, uno de ellos puede ser la cantidad de datos de entrenamiento en el split, el random_state de la función igualmente, los hiperparametros de la propia función y en este caso la propia K del KNN. Hagamos un poco de experimentación para ver como mejora o empeora el modelo.

In [ ]:
```
#modificamos el porcentaje del split en lugar de 75% entrenamiento lo reducimos a 70%
X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size=0.30)

knn = KNeighborsClassifier(n_neighbors=1)

knn.fit(X_train,y_train)
#realizamos pruebas

#entrenamiento y validación
y_pred = knn.predict(X_train)

print('Training set matrix and score:')
print(confusion_matrix(y_train,y_pred))
print(accuracy_score(y_train,y_pred))
```

```
#pruebas
y_pred = knn.predict(X_test)

print('Testing set matrix and score:')
print(confusion_matrix(y_test,y_pred))
print(accuracy_score(y_test,y_pred))
```

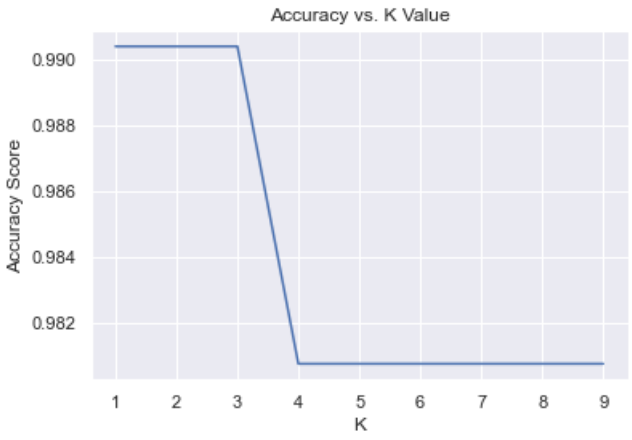
```
Training set matrix and score:
[[103   0   0]
 [  0  87   0]
 [  0   0  50]]
1.0
Testing set matrix and score:
[[48  0  1]
 [ 0 37  0]
 [ 0  0 18]]
0.9903846153846154
```

Como se puede observar tuvo una mejoría en la precisión, se puede decir que esto afectó indirectamente el modelo pues se entrenó con menos datos ahora reduciendo algo de overfitting. Pero en general como se dijo anteriormente es un buen fit.

Ahora probemos buscando el mejor numero de K para el modelo y veamos si hay un valor que pueda mejorar la predicción.

In [ ]:
```
accuracy = []
for i in range(1,10):
    knn=KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    accuracy.append(accuracy_score(y_test,pred_i))
```

In [ ]:
```
plt.figure()
plt.plot(range(1,10),accuracy)
plt.title('Accuracy vs. K Value')
plt.xlabel('K')
plt.ylabel('Accuracy Score');
```



De acuerdo a este gráfico que generamos a partir de probar diferentes K para buscar una mejor precisión vemos que 1 y 2 nos dan la mejor precisión que es la que ya teníamos anteriormente, si nos fueramos por una k de 4 en adelante empeora la precisión y de ahi en adelante parece que nos da la misma por lo que decidimos mantenerlo en 1.

Conclusiones

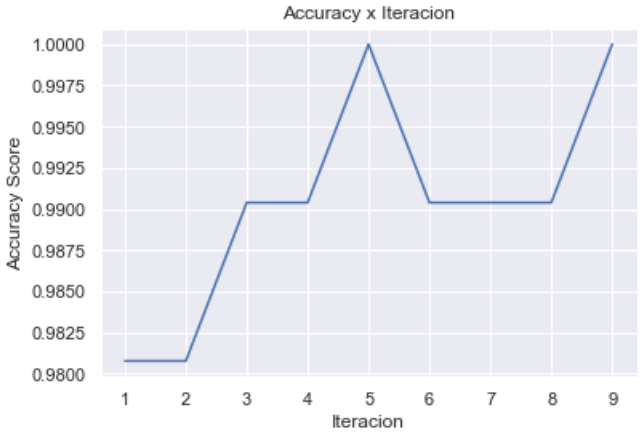
Grado de Bias y Varianza

El modelo de KNN utilizando K = 1 es un modelo de **bajo sesgo** pues ajusta el modelo a solo el (1) vecino más cercano lo que implica que el modelo tendrá mucha similitud con los datos de entrenamiento.

La **varianza es alta** porque optimizar a solo ese unico vecino, implica que la probabilidad de modelar ruido de los datos es alta. Con lo anterior podemos concluir que el modelo depende fuertemente de los datos con los que es entrenado, si se utiliza un dataset random para el entrenamiento, cada iteración el modelo tendrá resultados de precisión diferentes como se mostrará a continuación. No obstante sigue siendo un modelo preciso que tiene buen fit y podemos calcular el accuracy utilizando una media de las iteraciones.

In [ ]:
```
accuracy = []
for i in range(1,10):
    X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size=0.30)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    accuracy.append(accuracy_score(y_test,pred_i))

plt.figure()
plt.plot(range(1,10),accuracy)
plt.title('Accuracy x Iteracion')
plt.xlabel('Iteracion')
plt.ylabel('Accuracy Score');
```



Como se pudo visualizar a lo largo de este análisis, diversos factores tienen impacto en la precisión del modelo, este se verá afectado en primer lugar por la calidad de los datos que se utilizan para su entrenamiento por lo que su limpieza y preprocesamiento son parte fundamental para un buen modelo.

Otro factor es el split pues de este dependerá en ocasiones si existe overfitting o underfitting del modelo por lo que hay que encontrar un buen split para que se entrene el modelo correctamente y pueda realizar en este caso la clasificación con mayor precisión.

Finalmente los hiperparametros, cada modelo tiene estas opciones que son capaces de ajustarse para buscar mejorar la precisión, en este caso la K que no fue necesario pues el modelo ya era muy preciso, pero en otro caso se pueden realizar ajustes y pruebas para determinar el mejor modelo.