Clasificación Vinos

Objetivo

El objetivo de este documento es registrar los analisis y los procesos realizados para clasificar Vinos, basado en el dataset que se encuentra en https://archive.ics.uci.edu/dataset/109/wine

Los datos en CSV pueden ser accedidos directamente desde:
Datos (CSV): https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data

Dado que se trata de predecir una categoría en un listado de 3 opciones, usaremos algoritmos de Clasificación donde el objetivo es clasificar los distintos elementos de un conjunto de datos en varias categorías (3). agrupando los datos en función de su similitud. Como los conjuntos de datos tienen características comunes, es más fácil predecir su comportamiento.

Para el proceso se ha creado el libro en Databricks de la siguiente manera

Libro: Pre Analisis Wine Classification
Descripción: Este notebook consigna el codigo realizado para Exploratory Data Analysis y prueba de diferentes algoritmos de clasificación.
URL:
https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/1365877123506783/3295404153434753/8948153789405631/latest.html

Paso 1: Análisis Exploratorio EDA

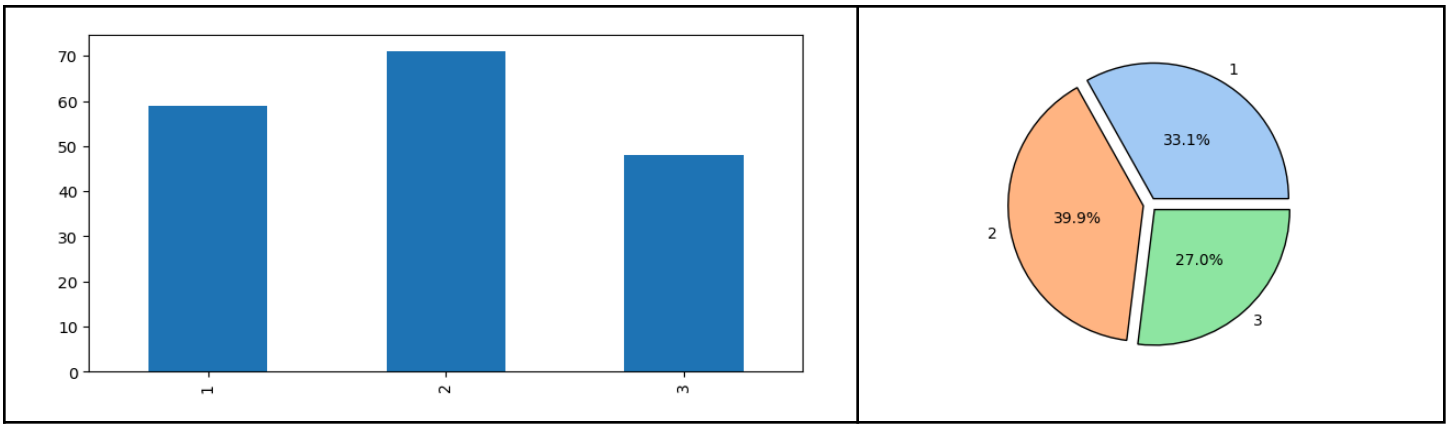
El dataset contiene:

```
Cantidad de filas: 178
Cantidad de columnas: 14
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   wine_class                    178 non-null    int64
 1   alcohol                       178 non-null    float64
 2   malic_acid                    178 non-null    float64
 3   ash                           178 non-null    float64
 4   alcalinity_of_ash             178 non-null    float64
 5   magnesium                     178 non-null    int64
 6   total_phenols                 178 non-null    float64
 7   flavanoids                    178 non-null    float64
 8   nonflavanoid_phenols          178 non-null    float64
 9   proanthocyanins               178 non-null    float64
 10  color_intensity               178 non-null    float64
 11  hue                           178 non-null    float64
 12  od280_od315_of_diluted_wines  178 non-null    float64
 13  proline                       178 non-null    int64
```

Dónde **wine_class** es la variable de categoría (target) que viene en el dataset. Se decide nombrar con ese nombre para tenerla como referencia.

La distribución de la variable basada en la categoría del dataset viene de la siguiente manera

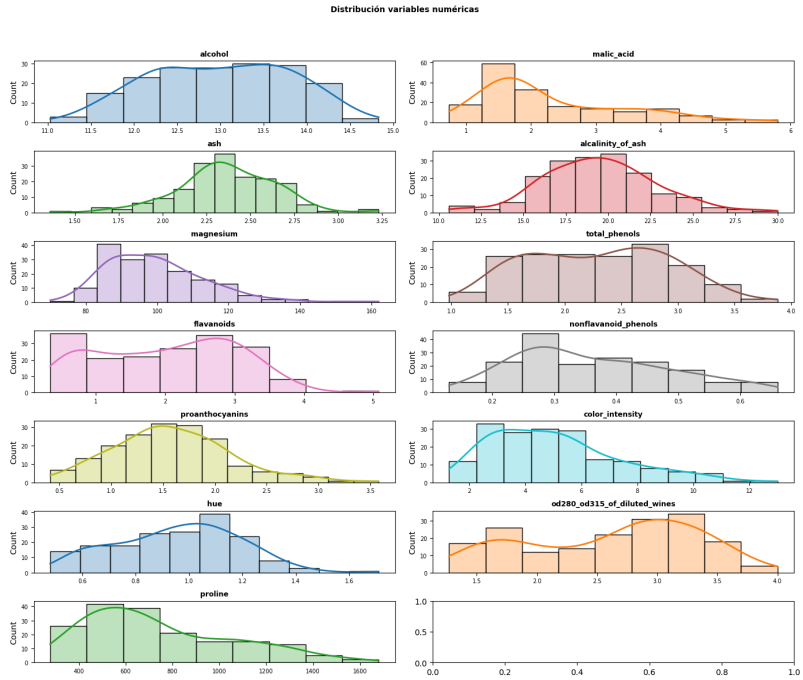
	wine_class
wine_class	
1	59
2	71
3	48



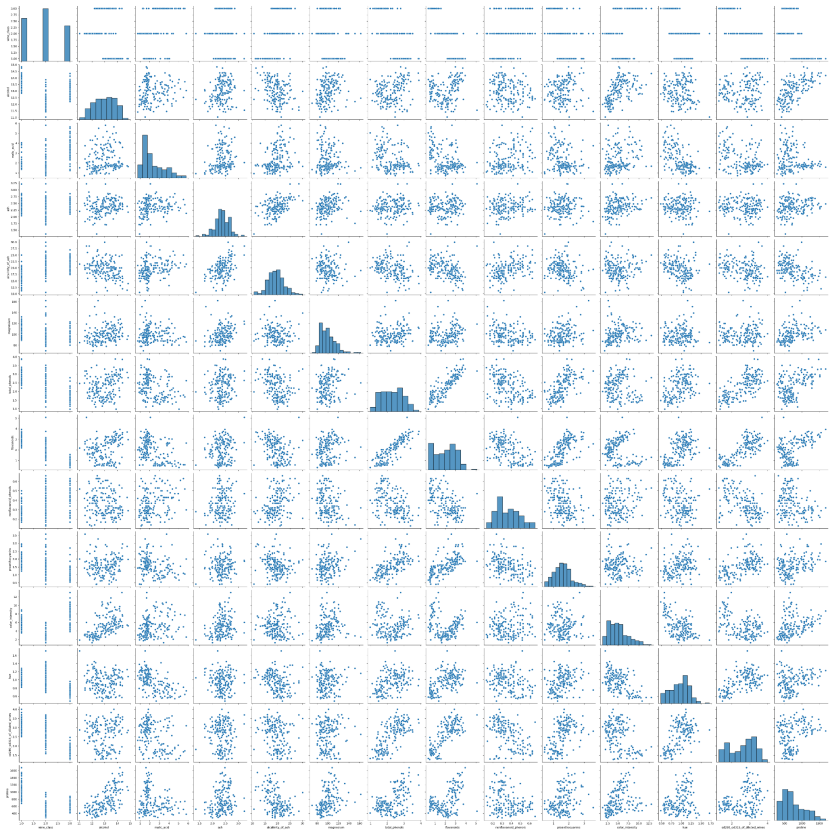
Evaluar Valores faltantes por columna:
Se crea funcion para validar columna por columna su % de afectación en el dataset
El dataset no contiene valores nulos o faltantes.

```
wine_class :  0.0%
alcohol :  0.0%
malic_acid :  0.0%
ash :  0.0%
alcalinity_of_ash :  0.0%
magnesium :  0.0%
total_phenols :  0.0%
flavanoids :  0.0%
nonflavanoid_phenols :  0.0%
proanthocyanins :  0.0%
color_intensity :  0.0%
hue :  0.0%
od280_od315_of_diluted_wines :  0.0%
proline :  0.0%
```

Con el objetivo de conocer la distribución de los valores por cada variable, se crea una grafica por cada variable que muestre la distribución de estas, usando histograma por cada una.

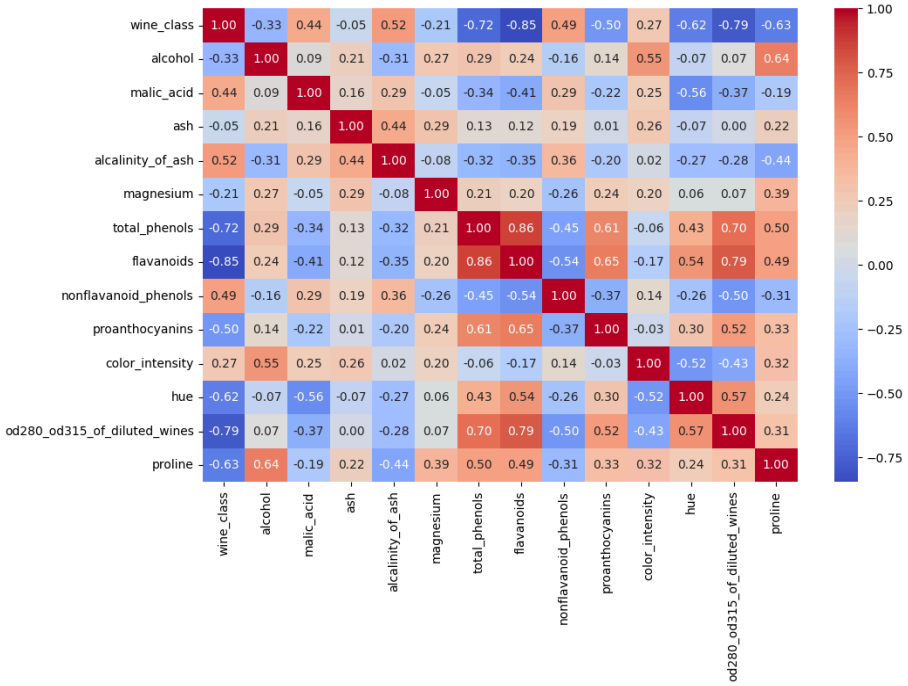


Graficamos luego mediante un diagrama de pares por cada variable, el objetivo es conocer la distribucion de los valores por cada una de estas, respecto a otra variable



Crear Matriz de Correlación

Con este paso, buscamos conocer la relación entre las variables y cual de las variables tiene mas relacion con nuestra variable target, wine_class



Podemos ver que la relación más fuerte con la variable wine_class son **total_phenols**, **alcalinity_of_ash** y **nonflavanoid_phenols**.
Sin embargo, la relacion mas fuerte entre variables se da entre **total_phenols** y **od280_od315_of_diluted_wines** y **od280_od315_of_diluted_wines** y **flavanoids**.

Paso 2. Prueba de Algoritmos

Con el objetivo de saber que modelo implementar, se realiza la prueba de varios algoritmos de Clasificación, teniendo presente que es un proceso de Multiclase.
Seleccionamos clasificación multiclase sobre clasificación multi etiqueta, dado que el objetivo es asignar instancias a una de varias clases o categorías predefinidas, **donde cada instancia pertenece exactamente a una clase**. Mientras que la clasificación multi etiqueta es una tarea donde cada instancia se puede asociar con **múltiples etiquetas simultáneamente**, lo que permite la asignación de múltiples etiquetas binarias a la instancia.
Dado que se trata de una sola clase a una instancia, evaluamos el proceso como clasificación multiclase.

Aunque lo ideal para algoritmos de clasificación es tener una gran cantidad de datos, por ahora entrenamos los datos que nos da el dataset.

La siguiente es la lista de algoritmos de clasificación que probaremos

- Logistic Regression
- Nearest Neighbors
- Linear SVM
- RBF SVM
- Gaussian Process
- Decision Tree
- Random Forest
- Neural Net
- Naive Bayes

El proceso se prueba de la siguiente manera
Se prueban con set de entrenamiento y pruebas de la siguiente manera:
Test_size toma los siguientes valores: 0.10, 0.15, 0.20, 0.25, 0.30
Donde

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=test_size)
```

Se itera sobre la cantidad de test_size que puede tomar, esto con el fin de valorar el accuracy de acuerdo al tamaño del set de entrenamiento y test.

Luego se itera por cada algoritmo y se guarda la información que permite mostrar de manera clara cómo son los valores de accuracy de entrenamiento, accuracy, precision y recall de los datos de test.

Ejemplo de la salida:

```
====================================================================================================
Test Size: 0.1
====================================================================================================
+--------------------+-----------+----------------+---------------+----------------+-------------+
| Modelo             | Test Size | Accuracy Train | Accuracy Test | Precision Test | Recall Test |
+--------------------+-----------+----------------+---------------+----------------+-------------+
| Logistic Regression |       0.1 |            1.0 |           1.0 |            1.0 |         1.0 |
| Nearest Neighbors   |       0.1 |            1.0 |           1.0 |            1.0 |         1.0 |
| Linear SVM          |       0.1 |            1.0 |           1.0 |            1.0 |         1.0 |
| RBF SVM             |       0.1 |       0.555556 |      0.555556 |       0.185185 |    0.333333 |
| Gaussian Process    |       0.1 |            1.0 |           1.0 |            1.0 |         1.0 |
| Decision Tree       |       0.1 |       0.888889 |      0.888889 |       0.944444 |    0.833333 |
| Random Forest       |       0.1 |            1.0 |           1.0 |            1.0 |         1.0 |
| Neural Net          |       0.1 |            1.0 |           1.0 |            1.0 |         1.0 |
| Naive Bayes         |       0.1 |            1.0 |           1.0 |            1.0 |         1.0 |
+--------------------+-----------+----------------+---------------+----------------+-------------+
```

Test Size: 0.2

Modelo	Test Size	Accuracy Train	Accuracy Test	Precision Test	Recall Test
Logistic Regression	0.2	0.944444	0.944444	0.939394	0.958333
Nearest Neighbors	0.2	0.916667	0.916667	0.914141	0.9375
Linear SVM	0.2	0.944444	0.944444	0.944444	0.958333
RBF SVM	0.2	0.444444	0.444444	0.148148	0.333333
Gaussian Process	0.2	0.944444	0.944444	0.948864	0.945833
Decision Tree	0.2	0.944444	0.944444	0.944444	0.958333
Random Forest	0.2	0.972222	0.972222	0.969697	0.979167
Neural Net	0.2	0.972222	0.972222	0.969697	0.979167
Naive Bayes	0.2	0.972222	0.972222	0.969697	0.979167

Test Size: 0.3

Modelo	Test Size	Accuracy Train	Accuracy Test	Precision Test	Recall Test
Logistic Regression	0.3	1.0	1.0	1.0	1.0
Nearest Neighbors	0.3	0.925926	0.925926	0.936508	0.939394
Linear SVM	0.3	1.0	1.0	1.0	1.0
RBF SVM	0.3	0.407407	0.407407	0.135802	0.333333
Gaussian Process	0.3	0.944444	0.944444	0.953704	0.933333
Decision Tree	0.3	0.907407	0.907407	0.910234	0.917172
Random Forest	0.3	0.962963	0.962963	0.96633	0.962626
Neural Net	0.3	1.0	1.0	1.0	1.0
Naive Bayes	0.3	0.962963	0.962963	0.964015	0.965241

Notas:
Es posible notar como el accuracy varía con el tamaño del set de entrenamiento.
Podemos notar que se nos da accuracy del 100%, aunque puede parecer un ideal, y dado el set de datos tan pequeño, optamos por trabajar con el 80% para entrenamiento y el 20% para el test.

Nota: Cuando se obtiene una precisión del 100 %, lo más probable es que se trate de una forma de sobreajuste (overfitting), y eso es, en última instancia es un error.
Para evitar esto, se pueden usar técnicas como Cross Validation, sin embargo por temas de ser el ejercicio práctico, veremos eso en otro apartado.

Para más detalle seguir la implementación del libro.

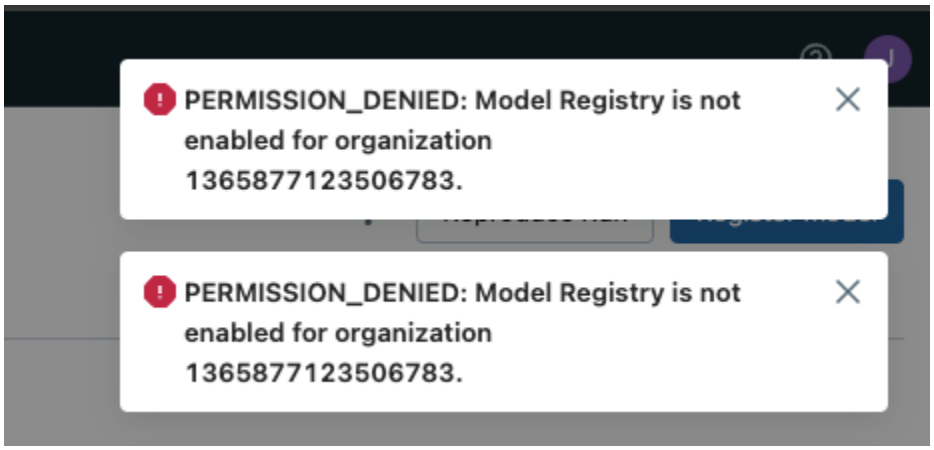
Paso 3. Entrenamientos de Modelo

Para el proceso, se han creado 4 notebooks con el objetivo de probar diferentes algoritmos.

El tamaño del dataset se define en 80% de entrenamiento y 20% para testing.

Basado en el proceso anterior, y teniendo presente que para problemas de Clasificación Multiclase, se seleccionan los siguientes algoritmos para entrenamiento y despliegue del dataset

NOTA: Debido a la versión community de Databricks, los modelos no se pueden registrar para despliegue via Web o usando línea de código más personalizado,



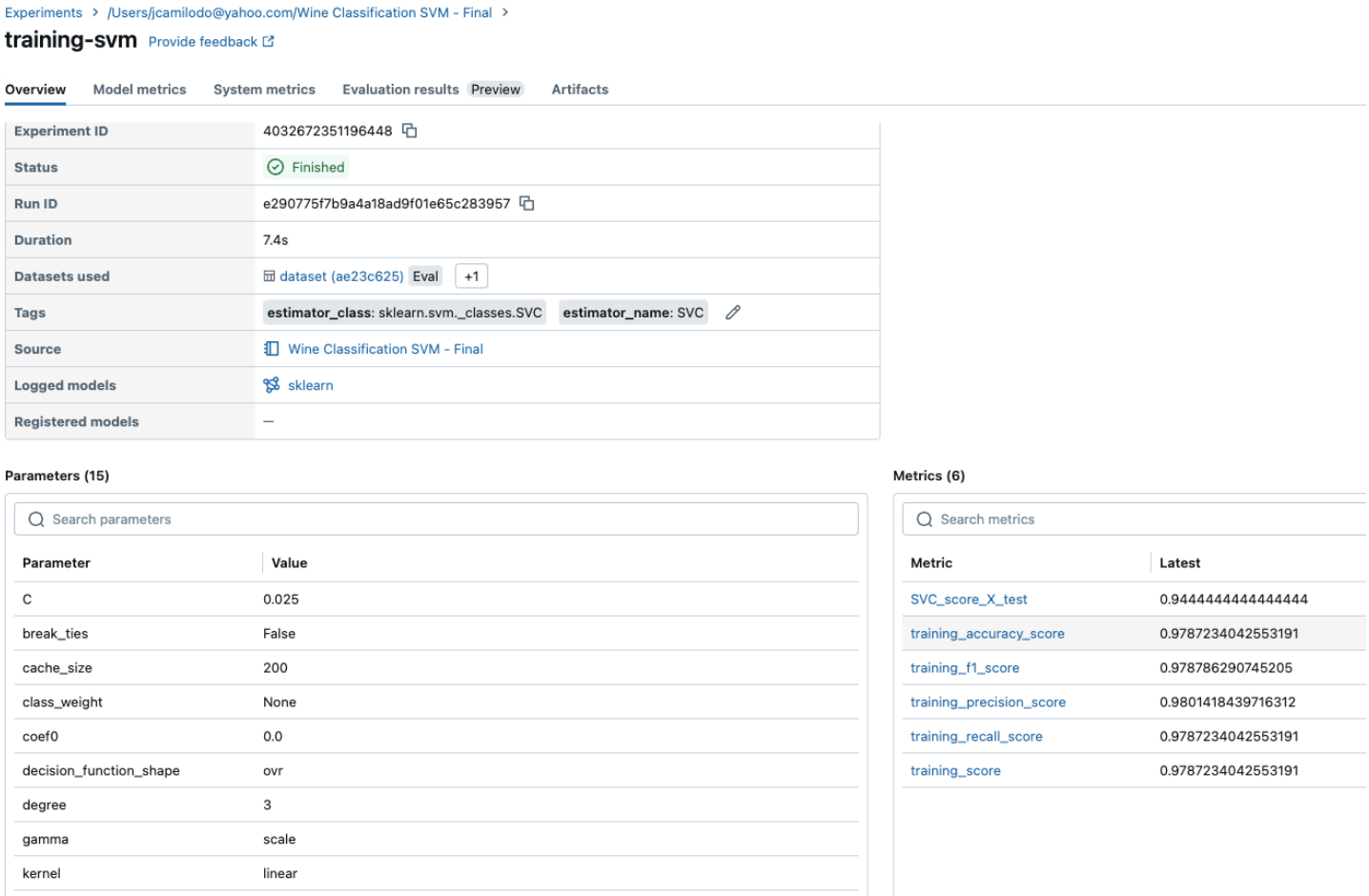
Se puede acceder via Experiment al modelo y ejecutarlo. Por este motivo, se crea un notebook para pruebas el cual puede accederse usando este link.

Support Vector Machine

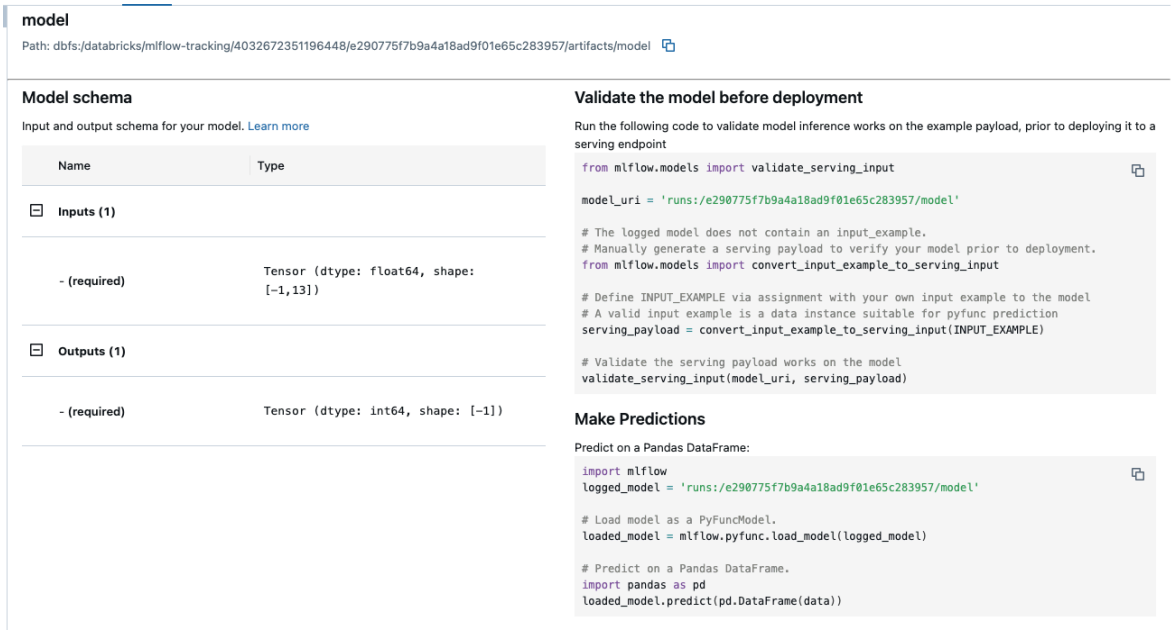
Se define el flujo dentro de MLFlow
Se accede al notebook usando el siguiente link.

```python
# mlflow.sklearn.autolog() requires mlflow 1.11.0 or above.
mlflow.sklearn.autolog()
# With autolog() enabled, all model parameters, a model score, and the fitted model are automatically logged.
with mlflow.start_run(run_name="training-svm"):
    wine_dataframe = read_data()
    wine_dataframe['wine_class'] = wine_dataframe.wine_class.replace({1: 0, 2: 1, 3: 2})
    X = wine_dataframe.iloc[: , 1:].values
    y = wine_dataframe.iloc[: ,0].values
    X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2)

    sc=StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.fit_transform(X_test)

    run_id = mlflow.active_run().info.run_id

    model = SVC(kernel="linear", C=0.025)
    model.fit(X_train, y_train)
    score = model.score(X_test, y_test)
    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test,y_pred)
```

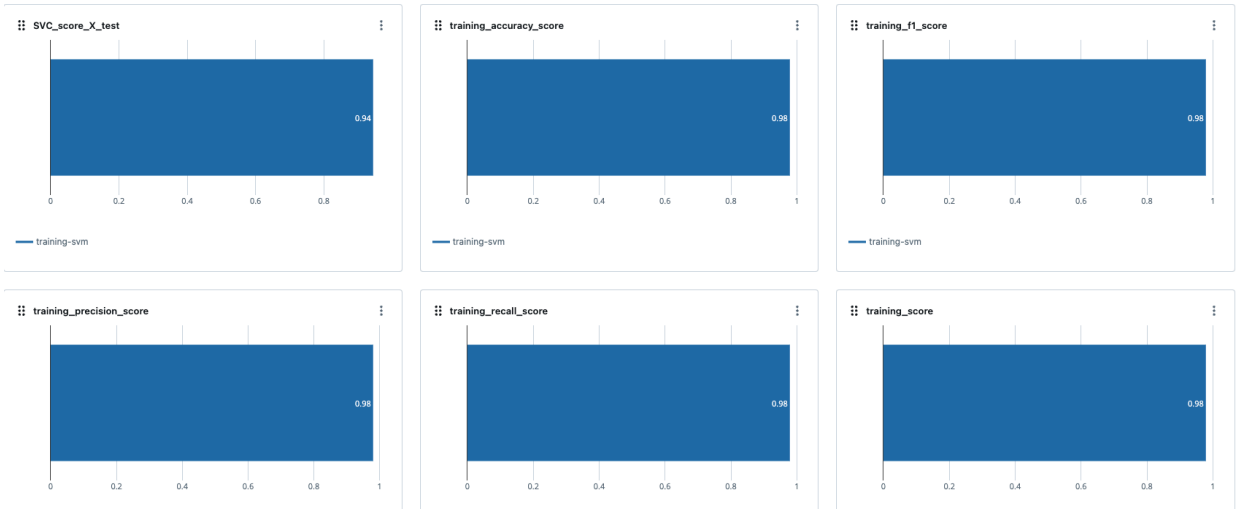
Podemos ver los resultados en el deploy del modelo



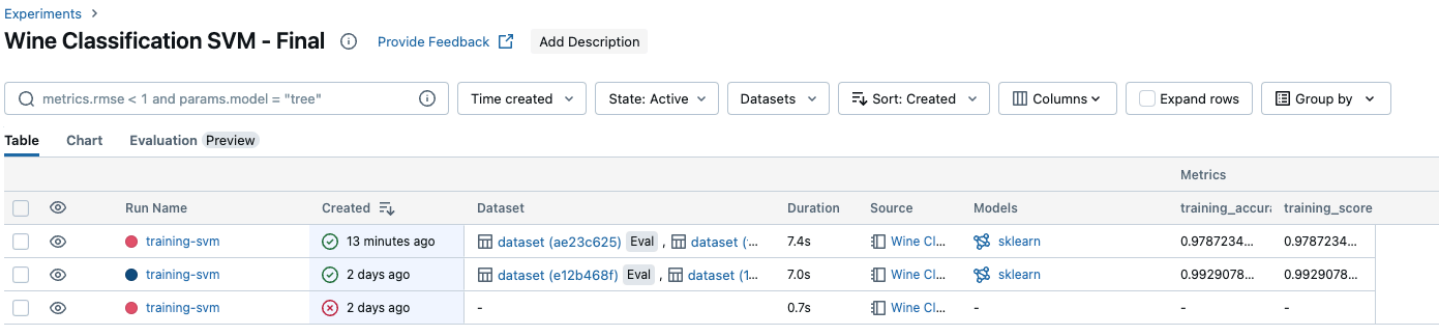
Podemos ver detalles para usar el modelo en la página de Artifacts.



Es posible ver las métricas del entrenamiento en la pestaña Model Metrics

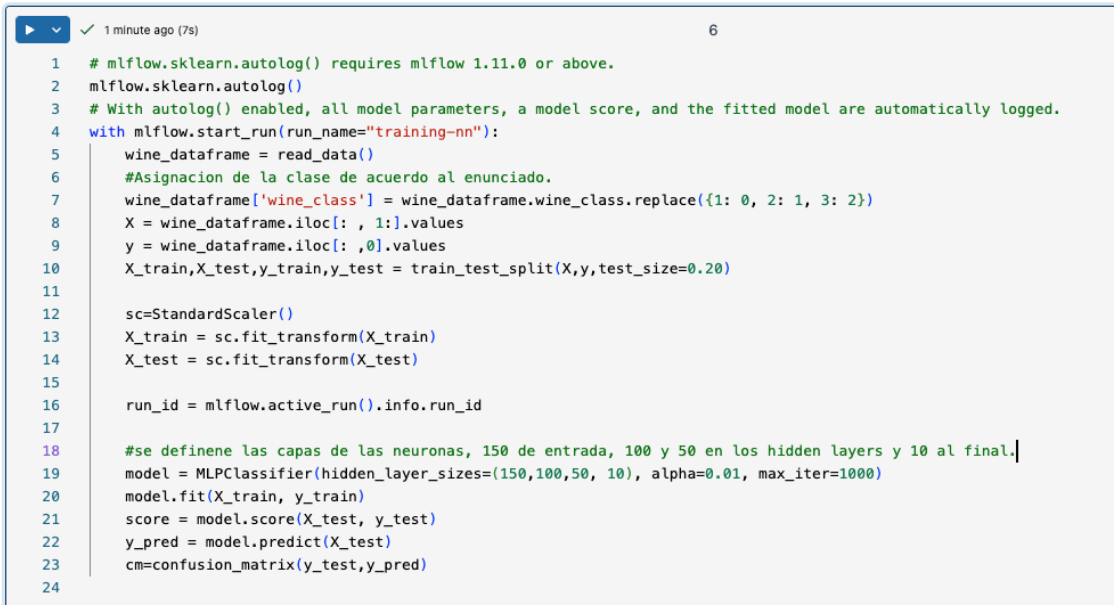


Podemos observar los experimentos ejecutados, accediendo por el menú Experiments,
Es posible acceder usando el link.



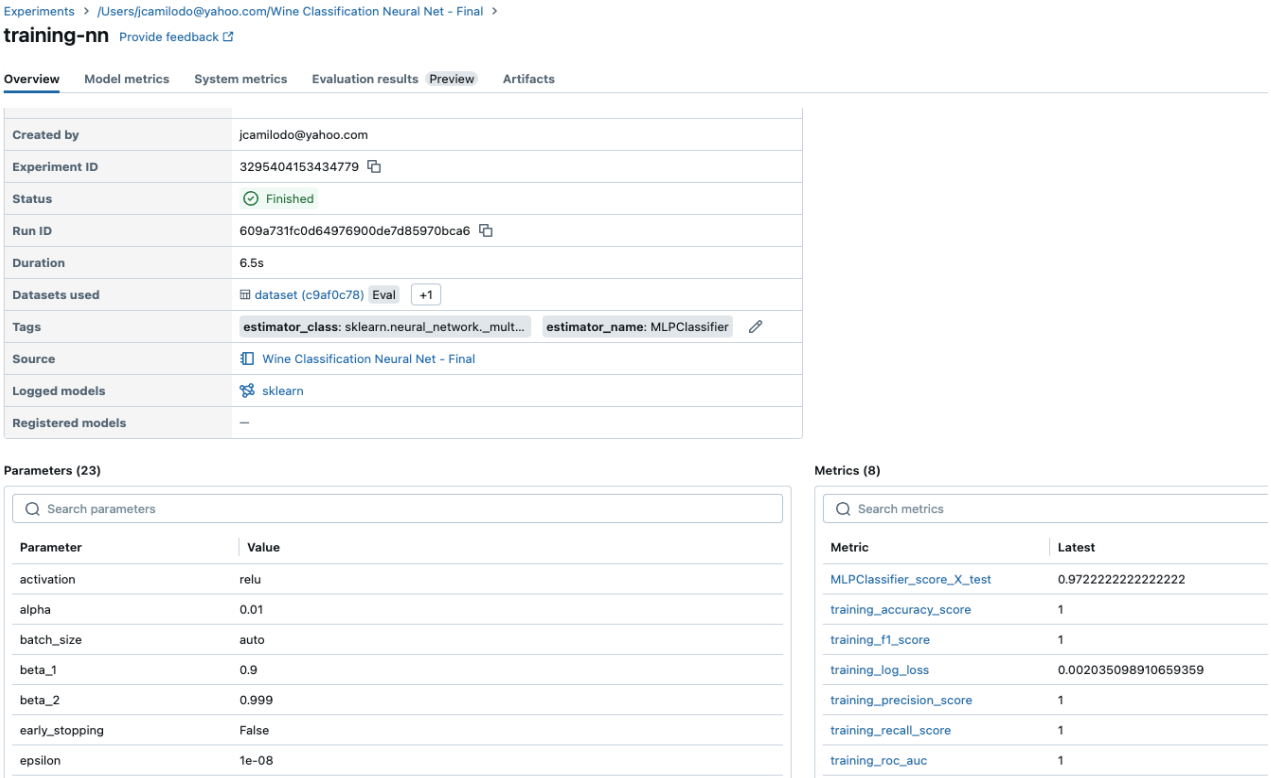
Neural Net (MLPClassifier)

Se define el flujo dentro de MLFlow
Se accede al notebook usando el siguiente link.



```
1 minute ago (7s)                                                    6

# mlflow.sklearn.autolog() requires mlflow 1.11.0 or above.
mlflow.sklearn.autolog()
# With autolog() enabled, all model parameters, a model score, and the fitted model are automatically logged.
with mlflow.start_run(run_name="training-nn"):
    wine_dataframe = read_data()
    #Asignacion de la clase de acuerdo al enunciado.
    wine_dataframe['wine_class'] = wine_dataframe.wine_class.replace({1: 0, 2: 1, 3: 2})
    X = wine_dataframe.iloc[: , 1:].values
    y = wine_dataframe.iloc[: ,0].values
    X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.20)

    sc=StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.fit_transform(X_test)

    run_id = mlflow.active_run().info.run_id

    #se definene las capas de las neuronas, 150 de entrada, 100 y 50 en los hidden layers y 10 al final.
    model = MLPClassifier(hidden_layer_sizes=(150,100,50, 10), alpha=0.01, max_iter=1000)
    model.fit(X_train, y_train)
    score = model.score(X_test, y_test)
    y_pred = model.predict(X_test)
    cm=confusion_matrix(y_test,y_pred)

```

Viendo en detalle la información del flujo



Experiments > /Users/jcamilodo@yahoo.com/Wine Classification Neural Net - Final >

training-nn Provide feedback

Overview | Model metrics | System metrics | Evaluation results Preview | Artifacts

Created by	jcamilodo@yahoo.com
Experiment ID	3295404153434779
Status	Finished
Run ID	609a731fc0d64976900de7d85970bca6
Duration	6.5s
Datasets used	dataset (c9af0c78) Eval +1
Tags	estimator_class: sklearn.neural_network._mult... estimator_name: MLPClassifier
Source	Wine Classification Neural Net - Final
Logged models	sklearn
Registered models	–

Parameters (23)

Search parameters

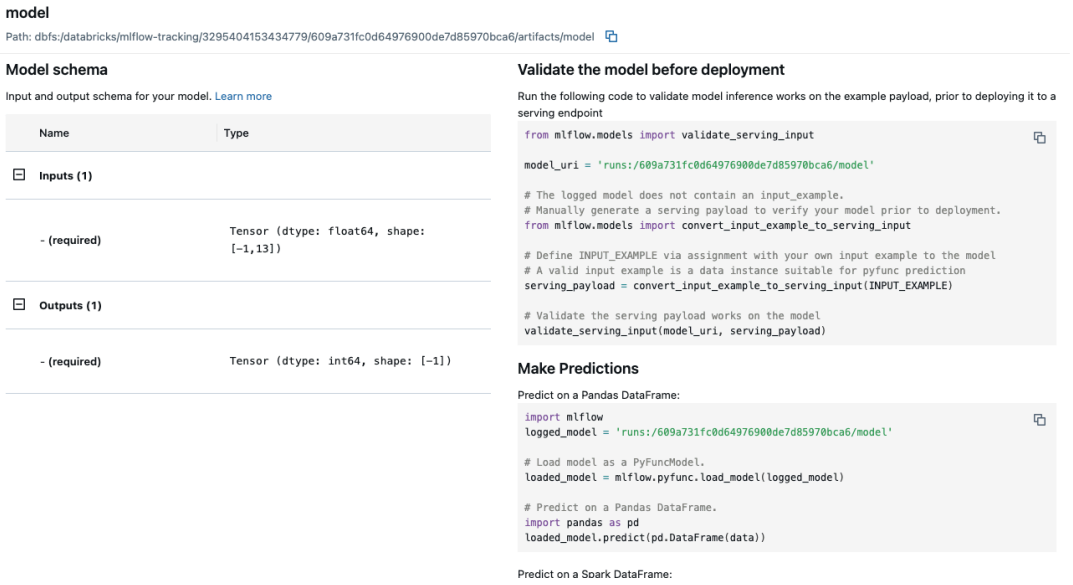
Parameter	Value
activation	relu
alpha	0.01
batch_size	auto
beta_1	0.9
beta_2	0.999
early_stopping	False
epsilon	1e-08

Metrics (8)

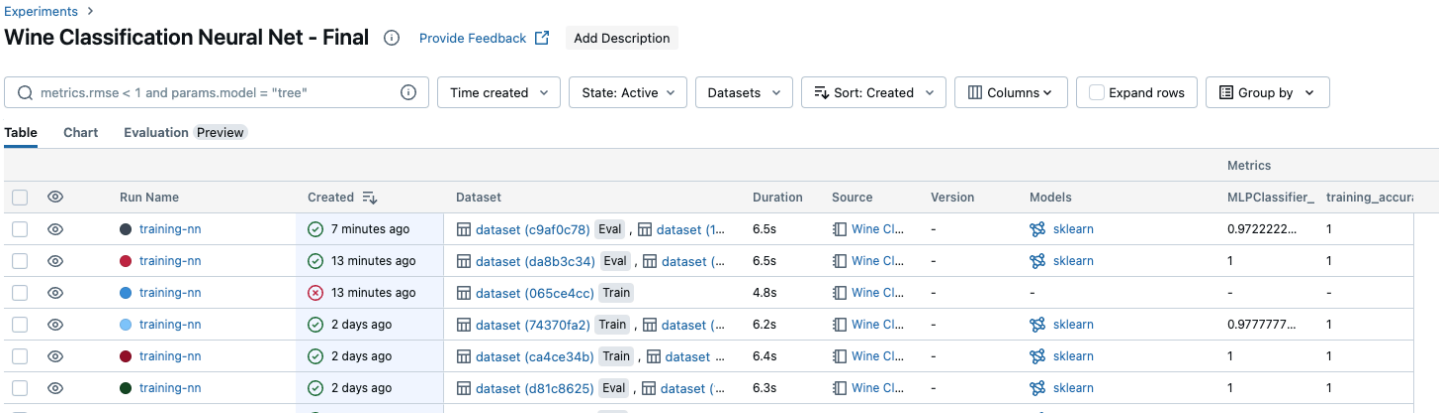
Search metrics

Metric	Latest
MLPClassifier_score_X_test	0.9722222222222222
training_accuracy_score	1
training_f1_score	1
training_log_loss	0.002035098910659359
training_precision_score	1
training_recall_score	1
training_roc_auc	1

Podemos ver la información para invocar el modelo



Puede acceder a la lista de experimentos desde el [link.]



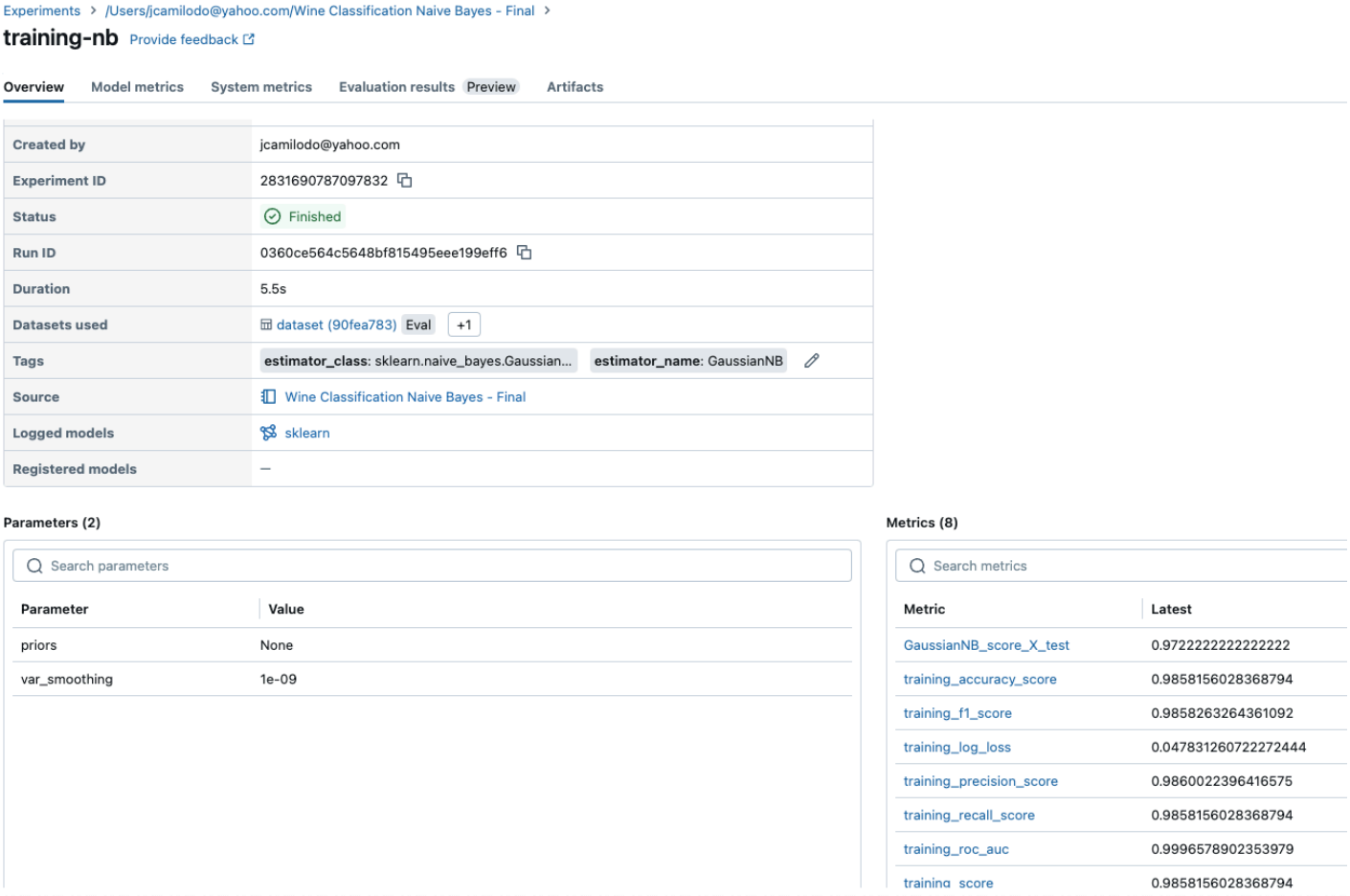
Naive Bayes

Se define el flujo dentro de MLFlow
Se accede al notebook usando el siguiente [link].

Just now (6s) 6

```
# mlflow.sklearn.autolog() requires mlflow 1.11.0 or above.
mlflow.sklearn.autolog()
# With autolog() enabled, all model parameters, a model score, and the fitted model are automatically logged.
with mlflow.start_run(run_name="training-nb"):
    wine_dataframe = read_data()
    wine_dataframe['wine_class'] = wine_dataframe.wine_class.replace({1: 0, 2: 1, 3: 2})
    X = wine_dataframe.iloc[: , 1:].values
    y = wine_dataframe.iloc[: ,0].values
    X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25)

    sc=StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.fit_transform(X_test)

    run_id = mlflow.active_run().info.run_id

    model = GaussianNB()
    model.fit(X_train, y_train)
    score = model.score(X_test, y_test)
    y_pred = model.predict(X_test)
    cm=confusion_matrix(y_test,y_pred)

```

Examinamos el experimento del proceso



Evaluando información para llamar al model

model

Path: dbfs:/databricks/mlflow-tracking/2831690787097832/0360ce564c5648bf815495eee199eff6/artifacts/model

The code snippets below demonstrate how to make predictions using the logged model. You can also register it to the model registry to version control and deploy as a REST endpoint for real time serving.

Model schema

Input and output schema for your model. Learn more

Name	Type
Inputs (1)	
- (required)	Tensor (dtype: float64, shape: [-1,13])
Outputs (1)	
- (required)	Tensor (dtype: int64, shape: [-1])

Validate the model before deployment

Run the following code to validate model inference works on the example payload, prior to deploying it to a serving endpoint

```
from mlflow.models import validate_serving_input

model_uri = 'runs:/0360ce564c5648bf815495eee199eff6/model'

# The logged model does not contain an input_example.
# Manually generate a serving payload to verify your model prior to deployment.
from mlflow.models import convert_input_example_to_serving_input

# Define INPUT_EXAMPLE via assignment with your own input example to the model
# A valid input example is a data instance suitable for pyfunc prediction
serving_payload = convert_input_example_to_serving_input(INPUT_EXAMPLE)

# Validate the serving payload works on the model
validate_serving_input(model_uri, serving_payload)
```

Make Predictions

Predict on a Pandas DataFrame:

```
import mlflow
logged_model = 'runs:/0360ce564c5648bf815495eee199eff6/model'

# Load model as a PyFuncModel.
loaded_model = mlflow.pyfunc.load_model(logged_model)

# Predict on a Pandas DataFrame.
import pandas as pd
loaded_model.predict(pd.DataFrame(data))
```

Se puede acceder a la lista de experimentos en el link.

Experiments >

Wine Classification Naive Bayes - Final Provide Feedback Add Description

metrics.rmse < 1 and params.model = "tree" | Time created | State: Active | Datasets | Sort: Created | Columns | Expand rows | Group by

Table Chart Evaluation Preview

		Run Name	Created	Dataset	Duration	Source	Models	Metrics		
								GaussianNB_s	training_accur	training_score
		training-nb	5 minutes ago	dataset (90fea783) Eval , dataset (...	5.5s	Wine Cl...	sklearn	0.9722222...	0.9858156...	0.9858156...
		training-nb	12 minutes ago	dataset (4cee227f) Train , dataset (...	5.3s	Wine Cl...	sklearn	1	0.9772727...	0.9772727...
		training-nn	13 minutes ago	dataset (62057935) Eval , dataset (...	6.8s	Wine Cl...	sklearn	0.9555555...	0.9848484...	0.9848484...

Nearest Neighbors

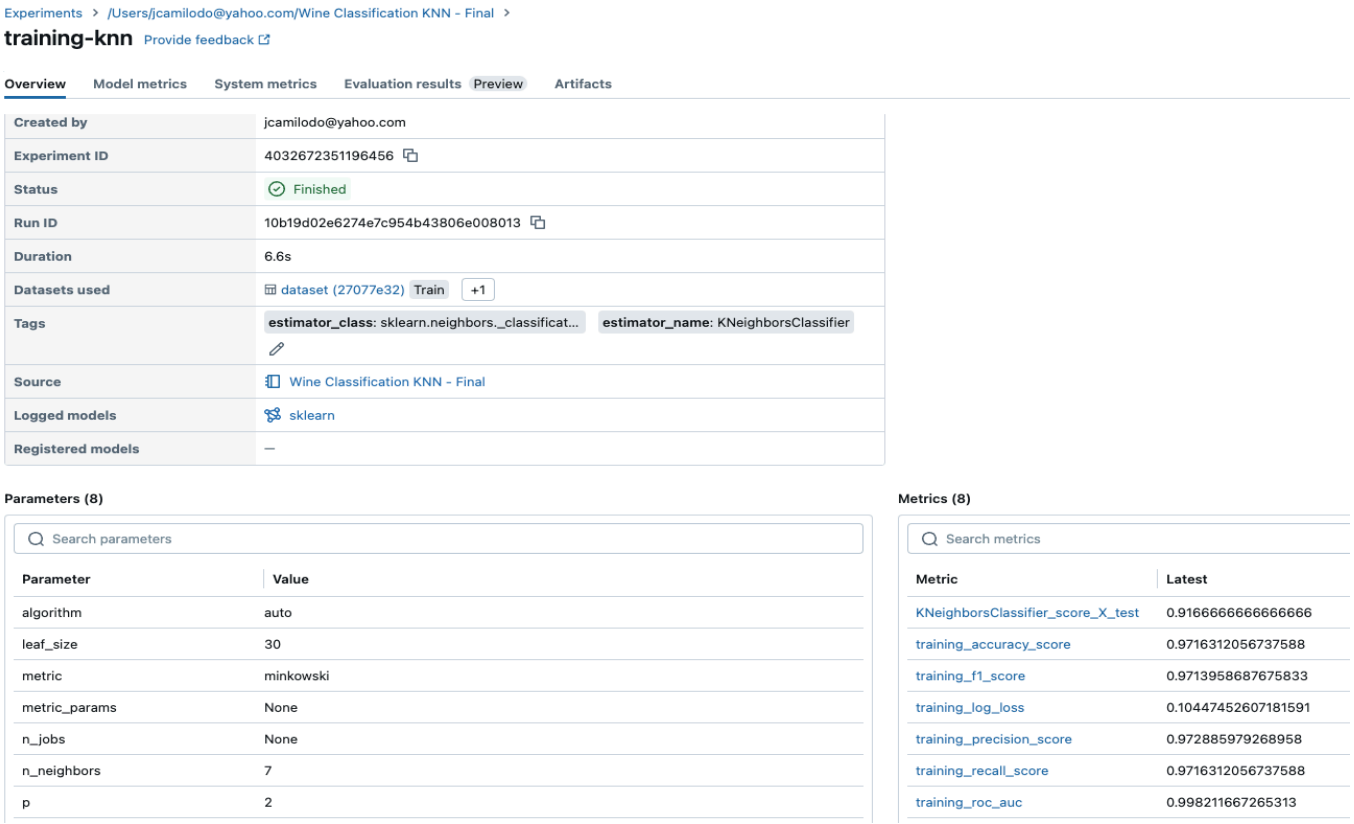
Se define el flujo dentro de MLFlow
Se accede al notebook usando el siguiente link.

4 minutes ago (7s) 6

```
# mlflow.sklearn.autolog() requires mlflow 1.11.0 or above.
mlflow.sklearn.autolog()
# With autolog() enabled, all model parameters, a model score, and the fitted model are automatically logged.
with mlflow.start_run(run_name="training-knn"):
    wine_dataframe = read_data()
    #Asignación de la nueva clase, se reemplazan los valores
    wine_dataframe['wine_class'] = wine_dataframe.wine_class.replace({1: 0, 2: 1, 3: 2})
    X = wine_dataframe.iloc[: , 1:].values
    y = wine_dataframe.iloc[: ,0].values
    X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.20)

    sc=StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.fit_transform(X_test)

    run_id = mlflow.active_run().info.run_id

    model = KNeighborsClassifier(n_neighbors=7)
    model.fit(X_train, y_train)
    score = model.score(X_test, y_test)
    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test,y_pred)

```

El valor para los n_neighbors se seleccionó basado en el Análisis realizado en el libro Pre Analisis Wine Classification.

Examinamos el experimento del proceso



Información para hacer uso del modelo

model

Path: dbfs:/databricks/mlflow-tracking/4032672351196456/10b19d02e6274e7c954b43806e008013/artifacts/model

MLflow Model

The code snippets below demonstrate how to make predictions using the logged model. You can also register it to the model registry to version control and deploy as a REST endpoint for real time serving.

Model schema

Input and output schema for your model. Learn more

Name	Type
Inputs (1)	
- (required)	Tensor (dtype: float64, shape: [-1,13])
Outputs (1)	
- (required)	Tensor (dtype: int64, shape: [-1])

Validate the model before deployment

Run the following code to validate model inference works on the example payload, prior to deploying it to a serving endpoint

```
from mlflow.models import validate_serving_input

model_uri = 'runs:/10b19d02e6274e7c954b43806e008013/model'

# The logged model does not contain an input_example.
# Manually generate a serving payload to verify your model prior to deployment.
from mlflow.models import convert_input_example_to_serving_input

# Define INPUT_EXAMPLE via assignment with your own input example to the model
# A valid input example is a data instance suitable for pyfunc prediction
serving_payload = convert_input_example_to_serving_input(INPUT_EXAMPLE)

# Validate the serving payload works on the model
validate_serving_input(model_uri, serving_payload)
```

Make Predictions

Predict on a Pandas DataFrame:

```
import mlflow
logged_model = 'runs:/10b19d02e6274e7c954b43806e008013/model'

# Load model as a PyFuncModel.
loaded_model = mlflow.pyfunc.load_model(logged_model)

# Predict on a Pandas DataFrame.
import pandas as pd
loaded_model.predict(pd.DataFrame(data))
```

Podemos ver el detalle de los experimentos desde el link.

Experiments >

Wine Classification KNN - Final Provide Feedback Add Description

metrics.rmse < 1 and params.model = "tree" | Time created | State: Active | Datasets | Sort: Created | Columns | Expand rows | Group by

Table Chart Evaluation Preview

							Metrics		
Run Name	Created	Dataset	Duration	Source	Models	KNeighborsCla	training_accur	training_score
training-knn	8 minutes ago	dataset (27077e32) Train , dataset ...	6.6s	Wine Cl...	sklearn	0.9166666...	0.97163120...	0.97163120...
training-knn	2 days ago	dataset (029ff717) Eval , dataset (0...	8.1s	Wine Cl...	sklearn	0.9166666...	0.97163120...	0.97163120...
training-knn	2 days ago	dataset (5f288a4d) Train , dataset ...	8.3s	Wine Cl...	sklearn	0.861111111...	0.97163120...	0.97163120...
training-knn	3 days ago	dataset (e04ce445) Train , dataset ...	5.8s	Wine Cl...	sklearn	0.8888888...	0.96	0.96
training-knn	3 days ago	dataset (2f2447ba) Train , dataset (...	6.1s	Wine Cl...	sklearn	0.911111111...	0.9696969...	0.9696969...
training-knn	3 days ago	dataset (0f1eb9fc) Eval , dataset (2...	5.7s	Wine Cl...	sklearn	0.9333333...	0.9848484...	0.9848484...
training	3 days ago	dataset (6de6db5b) Eval , dataset (...	7.1s	Wine Cl...	sklearn	0.9777777...	0.9848484...	0.9848484...

Predicciones de los modelos

Se crea el un libro para usar los modelos anteriormente creados, esto con el objetivo de probar los diferentes algoritmos. No es posible registrar el modelo y usarlo externamente en la versión community.

Puede acceder al libro mediante el siguiente link.

Nota:

El código de los libros puede encontrarlo en el repositorio.

Usar GANs para generar nuevos datos.

Dado la poca cantidad de datos que encontramos en el dataset origina, es posible pensar en emplear datos sintéticos para mejorar el rendimiento de los modelos de machine learning.
Se entregan 2 libros creados en Colab para la generacion de los datos Sinteticos.

Las redes generativas antagónicas, o GAN, por sus siglas en inglés, son un tipo de modelo de inteligencia artificial que puede crear nuevos datos que se parecen a los datos que ha visto antes.
En este caso, queremos crear nuevos datos que se parezcan al conjunto de datos del datasert Wine. Este conjunto de datos es bastante simple. Tiene información sobre las categorias basado en sus caracteristicas en 13 columnas.

Una buena opción para esta tarea sería un tipo especial de GAN llamado GAN condicional, o cGAN para abreviar. La parte "condicional" significa que le damos al modelo información adicional para ayudarlo en su tarea. En este caso, le damos la categoria actualdel vino a la GAN.
Entonces, cuando la cGAN crea nuevos datos, no solo realiza mediciones de la categoria aleatorias; realiza mediciones correspondientes al tipo de categoria que le dijimos que hiciera. Y cuando la cGAN verifica si ha hecho un buen trabajo, también considera el tipo de categoria.

El primero libro, Tabular Generator Tensorflow.ipynb, consiste en generar los datos de una manera mas de proceso creando nuestro propio GAN, significa que se crean dos objetos generator y discriminator.

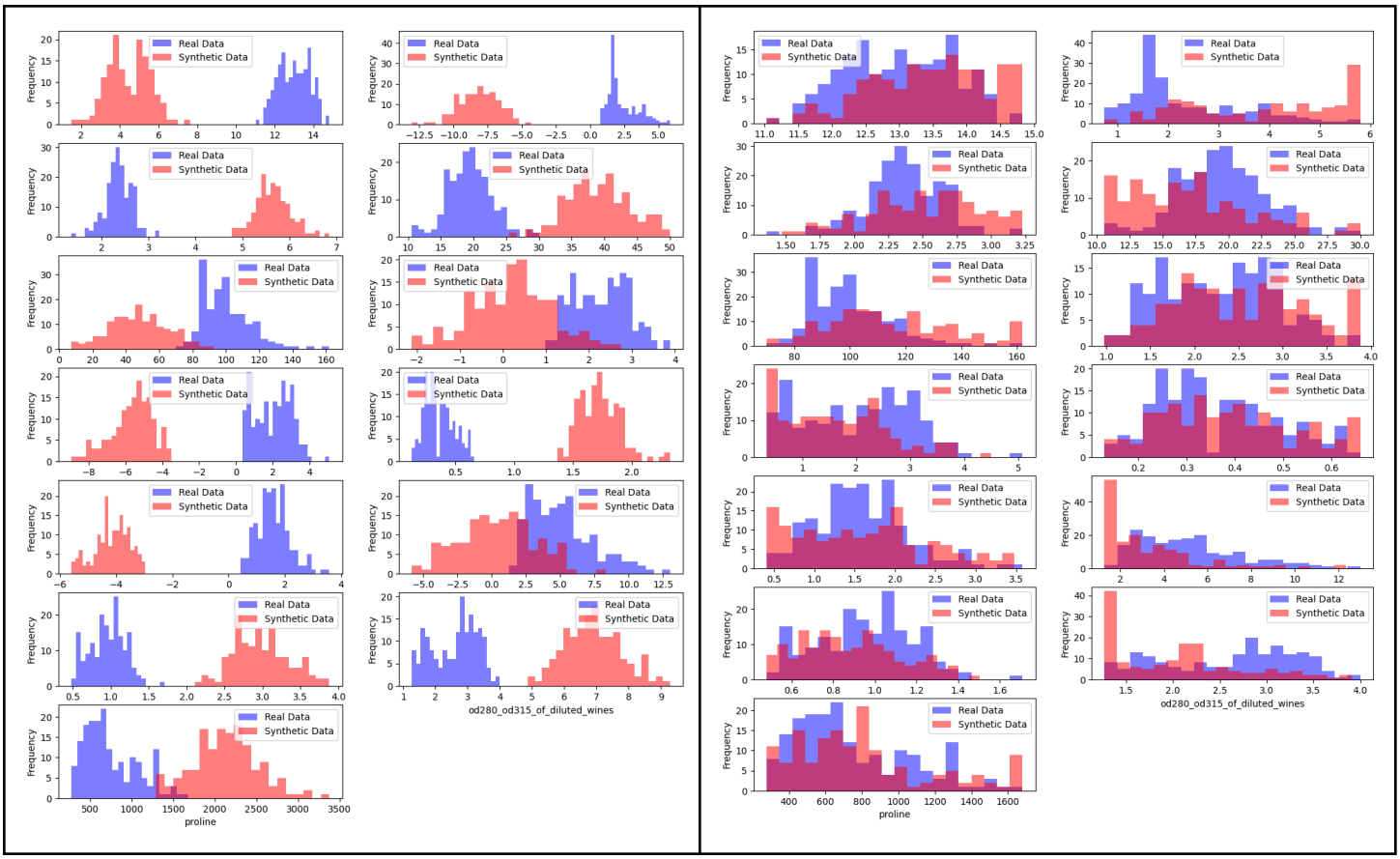
El segundo libro, se usara la libreria Synthetic Data Vault (SDV).
En lugar de crear GAN, se usara Synthetic Data Vault (SDV), que es una biblioteca de generación de datos tabulares sintéticos de código abierto. El libro utiliza el conjunto de datos de clasificación de Wine, entrena un CTGANSynthesizer con los datos reales, genera datos sintéticos y, por último, compara las distribuciones de datos de los conjuntos de datos reales y sintéticos mediante histogramas.

Comparaciones

Generando los datos sinteticos usando un GAN propio, genera datos sinteticos demasiado dispersos de los datos originales, vs la generacion con SDV, que son datos mas parecidos a los datos reales. Esto puede ser porque se deben hacer ajustes de hiperparametros para que llegue a nuestros datos deseados.
Otro punto es el tiempo de entrenamiento. Usando una GAN propia, el tiempo de entrenamiento, puede tomar minutos incluso horas, mientras usando SDV es cuestion de minutos.

Grafica comparativa

GAN propia	SDV


Real Data
Synthetic Data
Frequency
od280_od315_of_diluted_wines
proline