notebook

October 10, 2024

Profesor: Ing. Martín Ignacio Errazquín - merrazquin@fi.uba.ar

Alumnos: Ing. Pablo Martin Gomez Verdini - gomezpablo86@gmail.com Ing. Diego Paciotti Iacchelli - diegopaciotti@gmail.com Ing. Joaquin Gonzalez - joagonzalez@gmail.com

Repositorio Github https://github.com/FIUBA-CEIA-18Co2024/AMIA-TP3

1 Trabajo Práctico Final: Linear/Quadratic Discriminant Analysis (LDA/QDA)

1.1 Implementación base

1.1.1 Utils y dependencias

En esta sección se encuentra el codebase sobre el cual se implementan las clases y funciones necesarias para resolver las consignas del trabajo. Hay implementaciones que están dadas y se usaran *as a service* y otras que serán implementadas especialmente (como LDA).

[1]:
```
!pip install pandas
!pip install numpy
!pip install seaborn
!pip install matplotlib
!pip install scikit-learn
!pip install sqlalchemy
!pip install psycopg2==2.9.9
!pip install graphviz
```

```
Requirement already satisfied: pandas in /home/jgonzalez/dev/.virtualenvs/AMIA-
TP3/lib/python3.10/site-packages (2.2.3)
Requirement already satisfied: numpy>=1.22.4 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
pandas) (2.1.1)
Requirement already satisfied: python-dateutil>=2.8.2 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
```

```
pandas) (2024.2)
Requirement already satisfied: six>=1.5 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
python-dateutil>=2.8.2->pandas) (1.16.0)
Requirement already satisfied: numpy in /home/jgonzalez/dev/.virtualenvs/AMIA-
TP3/lib/python3.10/site-packages (2.1.1)
Requirement already satisfied: seaborn in /home/jgonzalez/dev/.virtualenvs/AMIA-
TP3/lib/python3.10/site-packages (0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
seaborn) (2.1.1)
Requirement already satisfied: pandas>=1.2 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
seaborn) (2.2.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
seaborn) (3.9.2)
Requirement already satisfied: contourpy>=1.0.1 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (1.3.0)
Requirement already satisfied: cycler>=0.10 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (4.54.1)
Requirement already satisfied: kiwisolver>=1.3.1 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (1.4.7)
Requirement already satisfied: packaging>=20.0 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (24.1)
Requirement already satisfied: pillow>=8 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (3.1.4)
Requirement already satisfied: python-dateutil>=2.7 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
pandas>=1.2->seaborn) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
pandas>=1.2->seaborn) (2024.2)
Requirement already satisfied: six>=1.5 in
```

```
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.16.0)
Requirement already satisfied: matplotlib in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (3.9.2)
Requirement already satisfied: contourpy>=1.0.1 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
matplotlib) (1.3.0)
Requirement already satisfied: cycler>=0.10 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
matplotlib) (4.54.1)
Requirement already satisfied: kiwisolver>=1.3.1 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
matplotlib) (1.4.7)
Requirement already satisfied: numpy>=1.23 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
matplotlib) (2.1.1)
Requirement already satisfied: packaging>=20.0 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
matplotlib) (24.1)
Requirement already satisfied: pillow>=8 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
matplotlib) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
matplotlib) (3.1.4)
Requirement already satisfied: python-dateutil>=2.7 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
python-dateutil>=2.7->matplotlib) (1.16.0)
Requirement already satisfied: scikit-learn in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (1.5.2)
Requirement already satisfied: numpy>=1.19.5 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
scikit-learn) (2.1.1)
Requirement already satisfied: scipy>=1.6.0 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
scikit-learn) (1.14.1)
Requirement already satisfied: joblib>=1.2.0 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
scikit-learn) (3.5.0)
```

```
Requirement already satisfied: sqlalchemy in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (2.0.35)
Requirement already satisfied: typing-extensions>=4.6.0 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
sqlalchemy) (4.12.2)
Requirement already satisfied: greenlet!=0.4.17 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (from
sqlalchemy) (3.1.1)
Requirement already satisfied: psycopg2==2.9.9 in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (2.9.9)
Requirement already satisfied: graphviz in
/home/jgonzalez/dev/.virtualenvs/AMIA-TP3/lib/python3.10/site-packages (0.20.3)
```

[2]:

```python
# Aqui se importarán todas las librerías utilizadas en el contexto de este␣
 ↪trabajo
import gc
import time
import timeit
import statistics
import tracemalloc
import numpy as np
import pandas as pd
import seaborn as sns
from sqlalchemy import text
from datetime import datetime
import matplotlib.pyplot as plt
from numpy.linalg import det, inv
from sqlalchemy import create_engine
from sklearn.metrics import confusion_matrix
from sklearn.datasets import load_iris, fetch_openml
from sklearn.model_selection import train_test_split

from typing import List, Union, Callable
from sqlalchemy.exc import SQLAlchemyError
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker, close_all_sessions
from sqlalchemy import Column, Integer, String, Float, DateTime


# Random Seeds
RNG_SEED = 6543
RNG_SEED_2 = 4111
RNG_SEED_3 = 2323

# Number of iterations used by timeit to calculate avg and std
repeat=600
```

```python
# Dataset test partition size
TEST_SIZE=0.4

# Set seaborn theme
sns.set_theme()

config = {
    "SQLALCHEMY_DATABASE_PREFIX": "",
    "SQLALCHEMY_DATABASE_URL": "postgresql+psycopg2://ceia:ceia2024@qwerty.com.
 ↪ar:5433/ceia",
    "SQLALCHEMY_DATABASE_ECHO": False,
    "DATABASE_TABLE_METRICS": "amia_lda_qda_v11",
}
```

1.1.2 Base de datos

Con el objetivo de persistir los datos en una base de datos relacional para posterior análisis se crea un esquema con la siguiente clase

[3]:
```python
engine = create_engine(
    config['SQLALCHEMY_DATABASE_URL'],
    # SQLite requires the next arg:
    # connect_args={"check_same_thread": False},
    echo=config['SQLALCHEMY_DATABASE_ECHO'],
    pool_pre_ping=True,
    connect_args={
        "keepalives": 1,
        "keepalives_idle": 30,
        "keepalives_interval": 10,
        "keepalives_count": 5,
    }
)

Base = declarative_base()
SessionLocalFactory = sessionmaker(bind=engine, autoflush=False,␣
 ↪autocommit=False)


class DatabaseService:
    def __init__(self):
        print("Initializing DatabaseService instance")
        self.session_factory = SessionLocalFactory

    def __del__(self):
        print("Closing all connections...")
```

```python
        close_all_sessions()

    @staticmethod
    def init_database():
        print("Initializing database...")
        Base.metadata.create_all(engine)
        close_all_sessions()

    def query_all(self, model, query_filter=None) -> List[Base]:
        with self.session_factory() as session:
            result = session.query(model)
            if query_filter:
                result = result.filter_by(**query_filter)
        return result.all()

    def query_one(self, model, query_filter=None) -> Union[Base, None]:
        with self.session_factory() as session:
            result = session.query(model)
            if query_filter:
                result = result.filter_by(**query_filter)
            one = result.first()
        return one

    def delete(self, model, query_filter) -> int:
        try:
            with self.session_factory() as session:
                with session.begin():
                    print(f'Deleting from DB {query_filter}')
                    r = session.query(model).filter_by(**query_filter).delete()
        except SQLAlchemyError as e:
            print(f'Error deleting from DB. Detail: {e}')
            r = 0
        return r

    def add(self, model_instance: Base) -> None:
        try:
            with self.session_factory() as session:
                with session.begin():
                    print(f"Adding '{model_instance}' into DB.")
                    merged = session.merge(model_instance)
                    session.add(merged)
                    session.commit()
        except SQLAlchemyError as e:
            print(f"Error adding into DB. Detail: {e}")
        else:
            print(f"Add successful.")
```

```
/tmp/ipykernel_211156/3859482305.py:15: MovedIn20Warning: The
``declarative_base()`` function is now available as
sqlalchemy.orm.declarative_base(). (deprecated since: 2.0) (Background on
SQLAlchemy 2.0 at: https://sqlalche.me/e/b8d9)
  Base = declarative_base()
```

[4]:
```python
class Metrics(Base):
    __tablename__ = config["DATABASE_TABLE_METRICS"]

    id = Column(Integer, primary_key=True, autoincrement=True)
    timestamp = Column(DateTime, unique=True)
    model_name = Column(String(200))
    dataset_name = Column(String(200))
    seed = Column(Integer)
    error = Column(Float)
    accuracy = Column(Float)
    memory_allocation = Column(Float)
    execution_time_ms = Column(Float)
    execution_time_dv_ms = Column(Float)
    comments = Column(String(1000))

    def __repr__(self):
        return f"<{self.__class__.__name__} id: '{self.id}' model_name: '{self.
 ↪model_name}'  timestamp: '{self.timestamp}'>"

    def __str__(self):
        return self.__repr__()

DatabaseService.init_database()
```

```
Initializing database…
```

[5]:
```python
# Connection with external database for data analysis
metrics_table=config["DATABASE_TABLE_METRICS"]
query = f"select * from {metrics_table}"
print(engine)

try:
    df = pd.read_sql_query(query, con=engine)
    df.head()
except Exception as e:
    print(f"Table does not exists. Error: {e}")
```

```
Engine(postgresql+psycopg2://ceia:***@qwerty.com.ar:5433/ceia)
```

Se utiliza patrón decorator para poder insertar métricas a cada corrida del modelo

[6]:
```python
def metrics(func):
    def wrapper(*args, **kwargs):
```

```
        print(f'Decorator parameters: {kwargs["model_name"]},␣
↪{kwargs["dataset_name"]}, {kwargs["seed"]}')

        lambda_func = lambda: func(*args, **kwargs)

        # Disable garbage collection for cleaner timing
        gc.disable()

        # Execution time analysis
        execution_times = timeit.repeat(
            stmt=lambda_func,
            number=kwargs["number"],
            repeat=kwargs["repeat"]
        )

        # Enable garbage collection back
        gc.enable()

        execution_times_ms = [time * 1000 for time in execution_times] # to ms
        mean_time = statistics.mean(execution_times_ms)
        std_dev_time = statistics.stdev(execution_times_ms)

        # Memory analysis and results
        tracemalloc.start()
        result = func(*args, **kwargs)
        _, memory_peak = tracemalloc.get_traced_memory()
        memory_peak /= 1024*1024
        tracemalloc.stop()

        # Result always returns predict over dataset_x so acc is calculated␣
↪against dataset_y
        model_accuracy = accuracy(kwargs["dataset_y"], result)

        # Insert data into db
        db = DatabaseService()

        db.add(Metrics(
            timestamp = datetime.now(),
            model_name = kwargs["model_name"],
            dataset_name = kwargs["dataset_name"],
            seed = kwargs["seed"],
            error = float(f'{1-model_accuracy:4f}'),
            accuracy = float(f'{model_accuracy:4f}'),
            memory_allocation = memory_peak,
            execution_time_ms = mean_time,
            execution_time_dv_ms = std_dev_time,
            comments = "",
```

```
        ))

        return result
    return wrapper
```

[7]:
```
# @metrics
def dispatcher(
    perdict_method: Callable,
    dataset_x: pd.DataFrame,
    dataset_y: pd.DataFrame,
    model_name: str,
    dataset_name: str,
    seed: str,
    number: int,
    repeat: int,
) -> pd.DataFrame:
    return perdict_method(dataset_x, )
```

Leer datos persistidos de corridas anteriores

[8]:
```
df = pd.read_sql_query(query, con=engine)
```

[9]:
```
df.head()
```

[9]:
```
Empty DataFrame
Columns: [id, timestamp, model_name, dataset_name, seed, error, accuracy,
memory_allocation, execution_time_ms, execution_time_dv_ms, comments]
Index: []
```

1.1.3 Clases base y Modelos

Bayesian Classifier

[10]:
```
class ClassEncoder:
  def fit(self, y):
    self.names = np.unique(y)
    self.name_to_class = {name:idx for idx, name in enumerate(self.names)}
    self.fmt = y.dtype
    # Q1: por que no hace falta definir un class_to_name para el mapeo inverso?

  def _map_reshape(self, f, arr):
    return np.array([f(elem) for elem in arr.flatten()]).reshape(arr.shape)
    # Q2: por que hace falta un reshape?

  def transform(self, y):
    return self._map_reshape(lambda name: self.name_to_class[name], y)

  def fit_transform(self, y):
    self.fit(y)
```

```
    return self.transform(y)

  def detransform(self, y_hat):
    return self._map_reshape(lambda idx: self.names[idx], y_hat)
```

[11]:
```
class BaseBayesianClassifier:
  def __init__(self):
    self.encoder = ClassEncoder()

  def _estimate_a_priori(self, y):
    a_priori = np.bincount(y.flatten().astype(int)) / y.size
    # Q3: para que sirve bincount?
    return np.log(a_priori)

  def _fit_params(self, X, y):
    # estimate all needed parameters for given model
    raise NotImplementedError()

  def _predict_log_conditional(self, x, class_idx):
    # predict the log(P(x|G=class_idx)), the log of the conditional probability␣
↪of x given the class
    # this should depend on the model used
    raise NotImplementedError()

  def fit(self, X, y, a_priori=None):
    # first encode the classes
    y = self.encoder.fit_transform(y)

    # if it's needed, estimate a priori probabilities
    self.log_a_priori = self._estimate_a_priori(y) if a_priori is None else np.
↪log(a_priori)

    # check that a_priori has the correct number of classes
    assert len(self.log_a_priori) == len(self.encoder.names), "A priori␣
↪probabilities do not match number of classes"

    # now that everything else is in place, estimate all needed parameters for␣
↪given model
    self._fit_params(X, y)
    # Q4: por que el _fit_params va al final? no se puede mover a, por ejemplo,␣
↪antes de la priori?

  def predict(self, X):
    # this is actually an individual prediction encased in a for-loop
    m_obs = X.shape[1]
    y_hat = np.empty(m_obs, dtype=self.encoder.fmt)
```

```
    # m_obs = 90 (in iris test for instance)
    for i in range(m_obs):
      encoded_y_hat_i = self._predict_one(X[:,i].reshape(-1,1)) # for each row predict_log_cond.. iter over each class (3) = 90 * 3 = 270
      y_hat[i] = self.encoder.names[encoded_y_hat_i]

    # return prediction as a row vector (matching y)
    return y_hat.reshape(1,-1)

  def _predict_one(self, x):
    # calculate all log posteriori probabilities (actually, +C)
    log_posteriori = [ log_a_priori_i + self._predict_log_conditional(x, idx) for idx, log_a_priori_i
                     in enumerate(self.log_a_priori) ]

    # return the class that has maximum a posteriori probability
    return np.argmax(log_posteriori)
```

_predict_log_conditional: return 0.5*np.log(det(self.tensor_inv_cov)) - 0.5* inner_prod.flatten()

$$\log f_j(x) = -\frac{1}{2}\log|\Sigma_j^{-1}| - \frac{1}{2}(x-\mu_j)^T\Sigma_j^{-1}(x-\mu_j)$$

_predict_one: return 0.5*np.log(det(inv_cov)) -0.5* unbiased_x.T @ inv_cov @ unbiased_x

$$\log f_j(x) = -\frac{1}{2}\log|\Sigma_j^{-1}| - \frac{1}{2}(x-\mu_j)^T\Sigma_j^{-1}(x-\mu_j) + \log(\pi_j)$$

QDA

[12]:
```
class QDA(BaseBayesianClassifier):
  name = "qda"

  def _fit_params(self, X, y):
    # estimate each covariance matrix
    self.inv_covs = [inv(np.cov(X[:,y.flatten()==idx], bias=True))
                      for idx in range(len(self.log_a_priori))]
    # print(len(self.inv_covs))
    # print(self.inv_covs[0].shape)
    # Q5: por que hace falta el flatten y no se puede directamente X[:,y==idx]?
    # Q6: por que se usa bias=True en vez del default bias=False?
    self.means = [X[:,y.flatten()==idx].mean(axis=1, keepdims=True)
                  for idx in range(len(self.log_a_priori))]
    # Q7: que hace axis=1? por que no axis=0?

  def _predict_log_conditional(self, x, class_idx):
    # predict the log(P(x|G=class_idx)), the log of the conditional probability of x given the class
    # this should depend on the model used
    inv_cov = self.inv_covs[class_idx]
```

```
    unbiased_x =  x - self.means[class_idx]
    return 0.5*np.log(det(inv_cov)) -0.5 * unbiased_x.T @ inv_cov @ unbiased_x
```

[13]:
```
"""
qda = QDA()
qda.fit(train_x_iris, train_y_iris)

r = qda.predict(train_x_iris)

print(r)
"""
```

[13]:
```
'\nqda = QDA()\nqda.fit(train_x_iris, train_y_iris)\n\nr =
qda.predict(train_x_iris)\n\nprint(r)\n'
```

[14]:
```
class TensorizedQDA(QDA):

    def _fit_params(self, X, y):
        # ask plain QDA to fit params
        super()._fit_params(X,y)

        # stack onto new dimension
        self.tensor_inv_cov = np.stack(self.inv_covs)
        self.tensor_means = np.stack(self.means)

    def _predict_log_conditionals(self,x):

        # print(x.shape)
        # print(self.tensor_inv_cov.shape)

        unbiased_x = x - self.tensor_means
        inner_prod = unbiased_x.transpose(0,2,1) @ self.tensor_inv_cov @␣
↪unbiased_x # 1 x 1

        return 0.5*np.log(det(self.tensor_inv_cov)) - 0.5 * inner_prod.flatten()

    def _predict_one(self, x):
        # return the class that has maximum a posteriori probability
        return np.argmax(self.log_a_priori + self._predict_log_conditionals(x))
```

[15]:
```
"""
qda = TensorizedQDA()
qda.fit(train_x_iris, train_y_iris)

r = qda.predict(train_x_iris)

print(r)
```

```
"""
```

[15]: '\nqda = TensorizedQDA()\nqda.fit(train_x_iris, train_y_iris)\n\nr = qda.predict(train_x_iris)\n\nprint(r)\n'

[16]:
```python
class FasterQDA(TensorizedQDA):
    def __init__(self, ultra_faster: bool = True):
        super().__init__()
        self.ultra_faster = ultra_faster
        self.n_x_n_matrix = []

    def _predict_log_conditional(self, X):
        # Calcular las probabilidades para todas las observaciones sin bucles
        #n = X.shape[0]
        log_probs = []

        # print(X.shape)

        for class_idx in range(len(self.means)):
            mean = self.means[class_idx]
            inv_cov = self.inv_covs[class_idx]

            # print(f'inv cov: {inv_cov.shape}')

            unbiased_X = X - mean
            # Evitamos la matriz n x n
            # diag(A @ B) = sum(A * B^T) usando broadcasting
            if self.ultra_faster:
                # 90*4 elements = 360 therefore 360/8100 = 4.4%  and 100 - 4.4␣
↪= 95.6% of memory savings
                diag_elements = np.sum((unbiased_X.T @ inv_cov) * unbiased_X.T,␣
↪axis=1)
                #                      90 x 4 @ 4 x 4 = (90 x 4) * (90 x 4)
            else:
                # 90*90 elements = 8100
                self.n_x_n_matrix = unbiased_X.T @ inv_cov @ unbiased_X
                diag_elements = np.diagonal(self.n_x_n_matrix)

            log_prob = -0.5 * diag_elements + self.log_a_priori[class_idx]
            log_probs.append(log_prob)

        return np.array(log_probs).T

    def get_n_x_n_matrix(self):
        return self.n_x_n_matrix

    def predict(self, X):
```

```
        log_probs = self._predict_log_conditional(X)
        return self.encoder.names[np.argmax(log_probs, axis=1)]
```

[17]:
```
"""
qda = FasterQDA(ultra_faster=False)
qda.fit(train_x_iris, train_y_iris)

r = qda.predict(train_x_iris)

print(r)
# Imprimimos la Matrix de 90x90
print(qda.get_n_x_n_matrix())
"""
```

[17]:
```
'\nqda = FasterQDA(ultra_faster=False)\nqda.fit(train_x_iris, train_y_iris)\n\nr
= qda.predict(train_x_iris)\n\nprint(r)\n# Imprimimos la Matrix de
90x90\nprint(qda.get_n_x_n_matrix())\n'
```

LDA

[18]:
```
class LDA(BaseBayesianClassifier):
    name = "lda"

    def _fit_params(self, X, y):
        n_features, n_samples = X.shape
        n_classes = len(self.encoder.names)
        # Calcular las medias para cada clase
        self.means = [X[:, y.flatten() == idx].mean(axis=1, keepdims=True)
                      for idx in range(n_classes)]

        # Inicializar la matriz de covarianza común
        #self.shared_cov = np.zeros((n_features, n_features))
        shared_cov = np.zeros((n_features, n_features))

        # Calcular la matriz de covarianza común
        for idx in range(n_classes):
            class_samples = X[:, y.flatten() == idx]
            cov = np.cov(class_samples, bias=True)
            shared_cov += cov * class_samples.shape[1]
        # Normalizar la matriz de covarianza común
        shared_cov /= n_samples  # Divide por el número total de muestras
        # Invertir la matriz de covarianza
        self.inv_cov = inv(shared_cov)

    def _predict_log_conditional(self, x, class_idx):
        # Predecir log(P(x|G=class_idx)), el logaritmo de la probabilidad␣
↪condicional de x dada la clase
        unbiased_x = x - 0.5 * self.means[class_idx]
```

```python
        #log_p_conditional = (0.5 * np.log(det(self.inv_cov)) - 0.5 *␣
↪(unbiased_x.T @ self.inv_cov @ unbiased_x))
        log_p_conditional = self.means[class_idx].T @ self.inv_cov @ unbiased_x
        return log_p_conditional
```

[19]:
```python
class TensorizedLDA(LDA):

    def _fit_params(self, X, y):
        # ask plain QDA to fit params
        super()._fit_params(X,y)

        # stack onto new dimension
        #self.tensor_inv_cov = np.stack(self.inv_covs)
        self.tensor_means = np.stack(self.means)

    def _predict_log_conditionals(self,x):
        #unbiased_x = x - self.tensor_means
        unbiased_x = x - 0.5 * self.tensor_means
        inv_cov_unbiased_x = self.inv_cov @ unbiased_x
        #print(self.tensor_means.transpose(0,2,1).shape)
        #print(inv_cov_unbiased_x.shape)
        #inner_prod = unbiased_x.transpose(0,2,1) @ self.inv_cov @ unbiased_x
        z = self.tensor_means.transpose(0,2,1) @ inv_cov_unbiased_x
        #print(z.flatten())
        #print(type(z))
        return z.flatten()

    def _predict_one(self, x):
        # return the class that has maximum a posteriori probability
        return np.argmax(self.log_a_priori + self._predict_log_conditionals(x))
```

[20]:
```python
class FasterLDA(TensorizedLDA):
    def __init__(self, ultra_faster: bool = True):
        super().__init__()
        self.ultra_faster = ultra_faster

    def _predict_log_conditional(self, x):
        # Calcular las probabilidades para todas las observaciones sin bucles
        log_probs = []

        for class_idx in range(len(self.means)):
            mean = self.means[class_idx]

            unbiased_X = x - mean
            # Evitamos la matriz n x n
            if self.ultra_faster:
                # diag(A @ B) = sum(A * B^T) usando broadcasting
```

```
                diag_elements = np.sum(unbiased_X.T @ self.inv_cov * unbiased_X.
↪T, axis=1)
            else:
                diag_elements = np.diagonal(unbiased_X.T @ self.inv_cov @␣
↪unbiased_X)

            log_prob = -0.5 * diag_elements + self.log_a_priori[class_idx]
            log_probs.append(log_prob)

        return np.array(log_probs).T

    def predict(self, X):
        log_probs = self._predict_log_conditional(X)
        return self.encoder.names[np.argmax(log_probs, axis=1)]
```

1.1.4 Preparación: Dataset loaders

Se agrega bool flag para plot de distribución de dataset para etapa de exploración.

[21]:
```
def get_iris_dataset(plot: bool = False):
  data = load_iris()
  print(data.feature_names)
  X_full = data.data
  y_full = np.array([data.target_names[y] for y in data.target.reshape(-1,1)])

  if plot:
      _, ax = plt.subplots()
      scatter = ax.scatter(data.data[:, 0], data.data[:, 1], c=data.target)
      ax.set(xlabel=data.feature_names[0], ylabel=data.feature_names[1])
      _ = ax.legend( scatter.legend_elements()[0], data.target_names,␣
↪loc="lower right", title="Classes")

  return X_full, y_full

def get_penguins(plot: bool = False):
    # get data
    df, tgt = fetch_openml(name="penguins", return_X_y=True, as_frame=True,␣
↪parser='auto')

    # Agregar la columna de la especie (target) al DataFrame
    aux = df.copy()
    aux['species'] = tgt

    if plot:
        # Crear un scatter plot
        plt.figure(figsize=(10, 6))
```

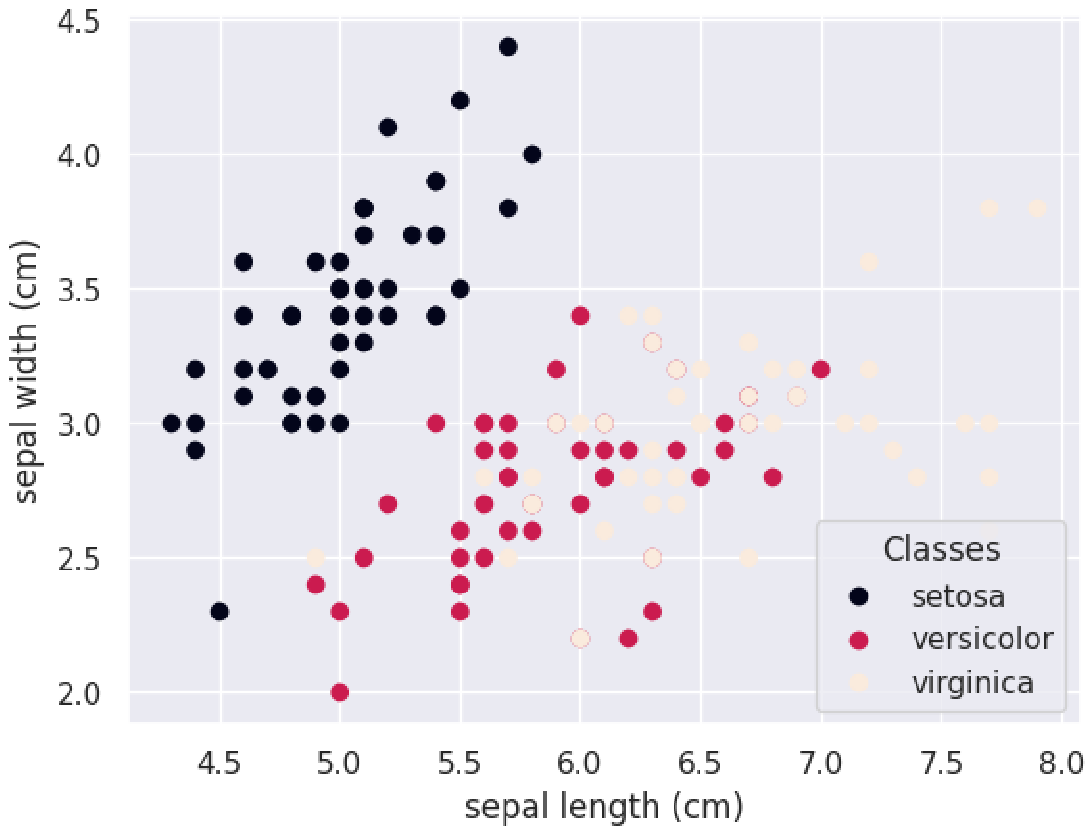
```
        sns.scatterplot(data=aux, x='culmen_length_mm', y='flipper_length_mm',␣
↪hue='species', style='species', s=80)
        plt.title('Relación entre longitud del culmen y longitud de la aleta␣
↪por especie')
        plt.xlabel('Longitud del culmen (mm)')
        plt.ylabel('Longitud de la aleta (mm)')
        plt.legend(title='Especie')
        plt.show()

    # drop non-numeric columns
    df.drop(columns=["island","sex"], inplace=True)

    # drop rows with missing values
    mask = df.isna().sum(axis=1) == 0
    df = df[mask]
    tgt = tgt[mask]
    print(df.keys())
    return df.values, tgt.to_numpy().reshape(-1,1)

def plot_classes_penguins():
    # Cargar dataset de penguins desde seaborn
    penguins = sns.load_dataset("penguins")

    # Verificar si hay datos nulos
    penguins.dropna(subset=['species'], inplace=True)

    # Graficar la distribución de clases
    plt.figure(figsize=(6, 4))
    sns.countplot(data=penguins, x='species')
    plt.title('Distribución de Clases en el Dataset de Penguins')
    plt.xlabel('Especie')
    plt.ylabel('Cantidad')
    plt.show()

def plot_classes_iris():
    # Load Iris dataset
    iris = load_iris()
    data = pd.DataFrame(iris.data, columns=iris.feature_names)
    data['species'] = iris.target
    print(f'Size of dataset: {len(data)}')

    # Map target integers to species names
    species_mapping = dict(zip(range(3), iris.target_names))
    data['species'] = data['species'].map(species_mapping)

    # Plot the class distribution
    plt.figure(figsize=(6, 4))
```

```python
    sns.countplot(data=data, x='species')
    plt.title('Distribución de clases en dataset Iris')
    plt.xlabel('Especies')
    plt.ylabel('Cantidad')
    plt.show()

def plot_confusion_matrix(cm):
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.title('Confusion Matrix')
    plt.show()
```

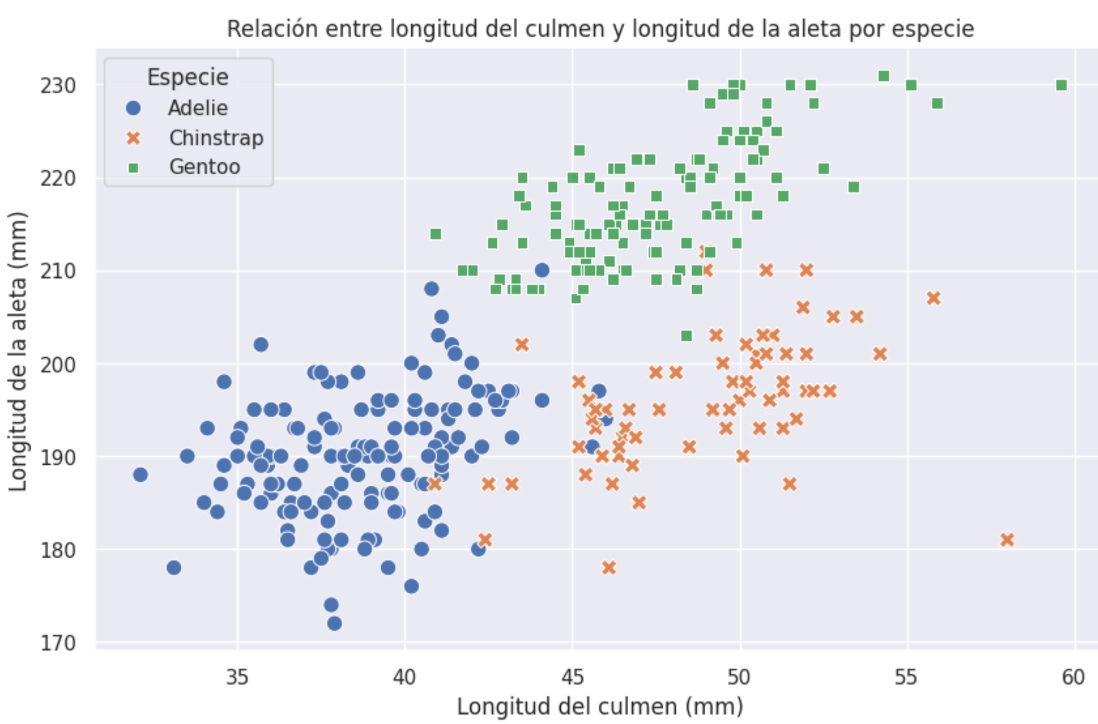
[22]:

```python
# showing for iris
X_full_iris, y_full_iris = get_iris_dataset(plot=True)
X_full_penguin, y_full_penguin = get_penguins(plot=True)

print(f"Shapes for iris dataset - X: {X_full_iris.shape}, Y:{y_full_iris.
 ↪shape}")
print(f"Shapes for iris penguins - X: {X_full_iris.shape}, Y:{y_full_penguin.
 ↪shape}")
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width
(cm)']
```



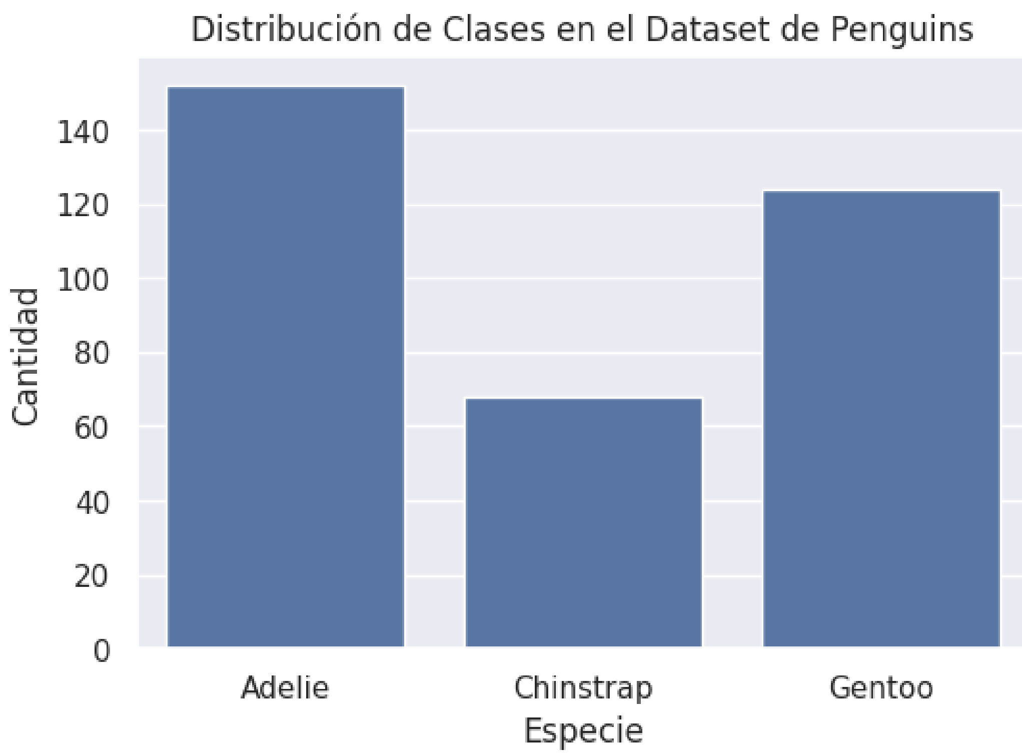
sepal width (cm)
sepal length (cm)
Classes
setosa
versicolor
virginica



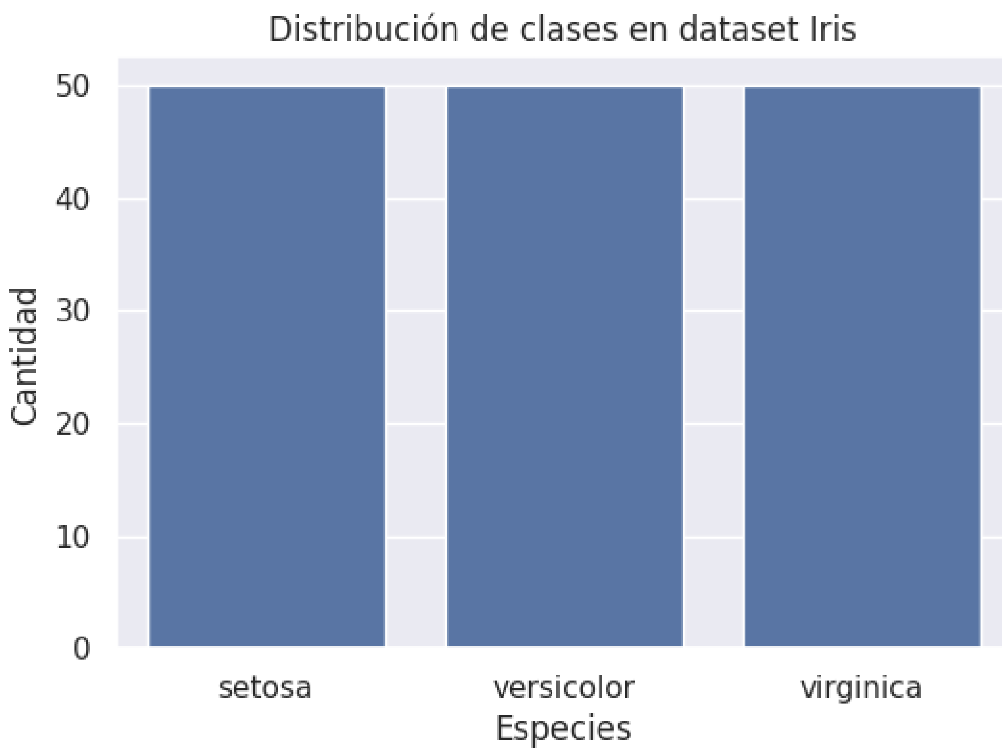
Relación entre longitud del culmen y longitud de la aleta por especie
Especie
Adelie
Chinstrap
Gentoo
Longitud de la aleta (mm)
Longitud del culmen (mm)

```
Index(['culmen_length_mm', 'culmen_depth_mm', 'flipper_length_mm',
       'body_mass_g'],
      dtype='object')
Shapes for iris dataset - X: (150, 4), Y:(150, 1)
Shapes for iris penguins - X: (150, 4), Y:(342, 1)
```

[23]:
```
plot_classes_penguins()
plot_classes_iris()
```




```
Size of dataset: 150
```



1.1.5 Preparación: Dataset split

[24]:
```
def split_transpose(X, y, test_sz, random_state):
    # split
    X_train, X_test, y_train, y_test = train_test_split(X, y,␣
 ↪test_size=test_sz, random_state=random_state, stratify=y)

    # transpose so observations are column vectors
    return X_train.T, y_train.T, X_test.T, y_test.T

def accuracy(y_true, y_pred):
  return (y_true == y_pred).mean()
```

[25]:
```
train_x_iris, train_y_iris, test_x_iris, test_y_iris =␣
 ↪split_transpose(X_full_iris, y_full_iris, TEST_SIZE, RNG_SEED)
train_x_penguin, train_y_penguin, test_x_penguin, test_y_penguin =␣
 ↪split_transpose(X_full_penguin, y_full_penguin, TEST_SIZE, RNG_SEED)

print("IRIS DATASET")
display(train_x_iris.shape, train_y_iris.shape, test_x_iris.shape, test_y_iris.
 ↪shape)
print("PENGUIN DATASET")
```

```
display(train_x_penguin.shape, train_y_penguin.shape, test_x_penguin.shape,␣
↪test_y_penguin.shape)
```

```
IRIS DATASET

(4, 90)

(1, 90)

(4, 60)

(1, 60)

PENGUIN DATASET

(4, 205)

(1, 205)

(4, 137)

(1, 137)
```

1.2 Consigna 1: Implementación base

Entrenar un modelo QDA sobre el dataset *iris* utilizando las distribuciones *a priori* a continuación ¿Se observan diferencias?¿Por qué cree? *Pista: comparar con las distribuciones del dataset completo,* ***sin splitear***.

1.2.1 1.1.1 Uniforme (cada clase tiene probabilidad 1/3)

[26]:
```
# without a priori distributions
print(f'### WITHOUT A PRIORI ###')
qda = QDA()
qda.fit(train_x_iris, train_y_iris)
train_acc = accuracy(train_y_iris, qda.predict(train_x_iris))
test_acc = accuracy(test_y_iris, qda.predict(test_x_iris))
print(f"Train (apparent) error is {1-train_acc:.4f} while test error is␣
↪{1-test_acc:.4f}")

# Model: QDA - Dataset: IRIS - SPLIT: Train
model_name="qda"
dataset_name="iris_train"
seed=RNG_SEED
number=1

silence = dispatcher(perdict_method=qda.predict,
           dataset_x=train_x_iris,
           dataset_y=train_y_iris,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
```

```
            number=number,
            repeat=repeat
        )

# Model: QDA - Dataset: IRIS - SPLIT: Test
dataset_name="iris_test"

silence = dispatcher(perdict_method=qda.predict,
            dataset_x=test_x_iris,
            dataset_y=test_y_iris,
            model_name=model_name,
            dataset_name=dataset_name,
            seed=seed,
            number=number,
            repeat=repeat
        )

# a priori distributions
print(f'### A PRIORI [1/3,1/3,1/3] ###')
qda.fit(train_x_iris, train_y_iris, a_priori= np.array([1/3, 1/3, 1/3]))
train_acc = accuracy(train_y_iris, qda.predict(train_x_iris))
test_acc = accuracy(test_y_iris, qda.predict(test_x_iris))
print(f"Train (apparent) error is {1-train_acc:.4f} while test error is␣
 ↪{1-test_acc:.4f}")

# Model: QDA - Dataset: IRIS - SPLIT: Train - A_PRIORI
model_name="qda_a_priori_1_3_1_3_1_3"
dataset_name="iris_train"

silence = dispatcher(perdict_method=qda.predict,
            dataset_x=train_x_iris,
            dataset_y=train_y_iris,
            model_name=model_name,
            dataset_name=dataset_name,
            seed=seed,
            number=number,
            repeat=repeat
        )

# Model: QDA - Dataset: IRIS - SPLIT: Test - A_PRIORI
dataset_name="iris_test"

silence = dispatcher(perdict_method=qda.predict,
            dataset_x=test_x_iris,
            dataset_y=test_y_iris,
            model_name=model_name,
            dataset_name=dataset_name,
```

```
            seed=seed,
            number=number,
            repeat=repeat
        )
```

```
### WITHOUT A PRIORI ###
Train (apparent) error is 0.0333 while test error is 0.0000
Decorator parameters: qda, iris_train, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda'  timestamp: '2024-10-10
23:16:14.630317'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: qda, iris_test, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda'  timestamp: '2024-10-10
23:16:18.312860'>' into DB.
Add successful.
Closing all connections…
### A PRIORI [1/3,1/3,1/3] ###
Train (apparent) error is 0.0333 while test error is 0.0000
Decorator parameters: qda_a_priori_1_3_1_3_1_3, iris_train, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_a_priori_1_3_1_3_1_3'  timestamp:
'2024-10-10 23:16:23.330258'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: qda_a_priori_1_3_1_3_1_3, iris_test, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_a_priori_1_3_1_3_1_3'  timestamp:
'2024-10-10 23:16:26.256218'>' into DB.
Add successful.
Closing all connections…
```

1.2.2 1.1.2 Una clase con probabilidad 0.9, las demás 0.05 (probar las 3 combinaciones)

[27]:
```
# a priori distributions
print(f'### A PRIORI [0.9, 0.05, 0.05] ###')
qda.fit(train_x_iris, train_y_iris, a_priori= np.array([0.9, 0.05, 0.05]))
train_acc = accuracy(train_y_iris, qda.predict(train_x_iris))
test_acc = accuracy(test_y_iris, qda.predict(test_x_iris))
print(f"Train (apparent) error is {1-train_acc:.4f} while test error is␣
 ↪{1-test_acc:.4f}")

# Model: QDA - Dataset: IRIS - SPLIT: Train - A PRIORI: [0.9, 0.05, 0.05]
model_name="qda_a_priori_09_05_05"
```

```
dataset_name="iris_train"
seed=RNG_SEED
number=1

silence = dispatcher(perdict_method=qda.predict,
           dataset_x=train_x_iris,
           dataset_y=train_y_iris,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )

# Model: QDA - Dataset: IRIS - SPLIT: Test - A PRIORI: [0.9, 0.05, 0.05]
dataset_name="iris_test"

silence = dispatcher(perdict_method=qda.predict,
           dataset_x=test_x_iris,
           dataset_y=test_y_iris,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )


print(f'### A PRIORI [0.05, 0.9, 0.05] ###')
qda.fit(train_x_iris, train_y_iris, a_priori= np.array([0.05, 0.9, 0.05]))
train_acc = accuracy(train_y_iris, qda.predict(train_x_iris))
test_acc = accuracy(test_y_iris, qda.predict(test_x_iris))
print(f"Train (apparent) error is {1-train_acc:.4f} while test error is␣
 ↪{1-test_acc:.4f}")

# Model: QDA - Dataset: IRIS - SPLIT: Train - A PRIORI: [0.05, 0.9, 0.05]
model_name="qda_a_priori_05_09_05"
dataset_name="iris_train"

silence = dispatcher(perdict_method=qda.predict,
           dataset_x=train_x_iris,
           dataset_y=train_y_iris,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
```

```
        )

# Model: QDA - Dataset: IRIS - SPLIT: Test - A PRIORI: [0.05, 0.9, 0.05]
dataset_name="iris_test"

silence = dispatcher(perdict_method=qda.predict,
           dataset_x=test_x_iris,
           dataset_y=test_y_iris,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )

print(f'### A PRIORI [0.05, 0.05, 0.9] ###')
qda.fit(train_x_iris, train_y_iris, a_priori= np.array([0.05, 0.05, 0.9]))
train_acc = accuracy(train_y_iris, qda.predict(train_x_iris))
test_acc = accuracy(test_y_iris, qda.predict(test_x_iris))
print(f"Train (apparent) error is {1-train_acc:.4f} while test error is␣
 ↪{1-test_acc:.4f}")

# Model: QDA - Dataset: IRIS - SPLIT: Train - A PRIORI: [0.05, 0.05, 0.9]
model_name="qda_a_priori_05_05_09"
dataset_name="iris_train"

silence = dispatcher(perdict_method=qda.predict,
           dataset_x=train_x_iris,
           dataset_y=train_y_iris,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )

# Model: QDA - Dataset: IRIS - SPLIT: Test - A PRIORI: [0.05, 0.05, 0.9]
dataset_name="iris_test"

silence = dispatcher(perdict_method=qda.predict,
           dataset_x=test_x_iris,
           dataset_y=test_y_iris,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
```

```
        )
```

```
### A PRIORI [0.9, 0.05, 0.05] ###
Train (apparent) error is 0.0333 while test error is 0.0000
Decorator parameters: qda_a_priori_09_05_05, iris_train, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_a_priori_09_05_05'  timestamp:
'2024-10-10 23:16:30.320972'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: qda_a_priori_09_05_05, iris_test, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_a_priori_09_05_05'  timestamp:
'2024-10-10 23:16:33.303339'>' into DB.
Add successful.
Closing all connections…
### A PRIORI [0.05, 0.9, 0.05] ###
Train (apparent) error is 0.0444 while test error is 0.0833
Decorator parameters: qda_a_priori_05_09_05, iris_train, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_a_priori_05_09_05'  timestamp:
'2024-10-10 23:16:36.500713'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: qda_a_priori_05_09_05, iris_test, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_a_priori_05_09_05'  timestamp:
'2024-10-10 23:16:38.742939'>' into DB.
Add successful.
Closing all connections…
### A PRIORI [0.05, 0.05, 0.9] ###
Train (apparent) error is 0.0444 while test error is 0.0333
Decorator parameters: qda_a_priori_05_05_09, iris_train, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_a_priori_05_05_09'  timestamp:
'2024-10-10 23:16:41.640919'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: qda_a_priori_05_05_09, iris_test, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_a_priori_05_05_09'  timestamp:
'2024-10-10 23:16:43.849477'>' into DB.
Add successful.
Closing all connections…
```

[28]:
```
def _estimate_a_priori(y, full: bool = False):
    a_priori = np.bincount(y.flatten().astype(int)) / y.size
```

```
    print(f"Train split dataset: {a_priori}" if not full else f"Full dataset:␣
↪{a_priori}")
    print(f"Train split dataset: {np.log(a_priori)}" if not full else f"Full␣
↪dataset: {np.log(a_priori)}")
    return np.log(a_priori)

a_priori = None
encoder = ClassEncoder()
y_full = encoder.fit_transform(y_full_iris)
y_train_split = encoder.fit_transform(train_y_iris)

log_a_priori_full = _estimate_a_priori(y_full, full=True) if a_priori is None␣
↪else np.log(a_priori)
log_a_priori_train = _estimate_a_priori(y_train_split) if a_priori is None else␣
↪np.log(a_priori)

a_priori = [1/3, 1/3, 1/3]
log_a_priori = _estimate_a_priori(y_train_split) if a_priori is None else np.
↪log(a_priori)

print(f"A priori [1/3, 1/3, 1/3]: {log_a_priori}")
```

```
Full dataset: [0.33333333 0.33333333 0.33333333]
Full dataset: [-1.09861229 -1.09861229 -1.09861229]
Train split dataset: [0.33333333 0.33333333 0.33333333]
Train split dataset: [-1.09861229 -1.09861229 -1.09861229]
A priori [1/3, 1/3, 1/3]: [-1.09861229 -1.09861229 -1.09861229]
```

Da distinto accuracy por que la distribución en las particiones de test y train de las clases no se mantiene igual que en el dataset full.

1.2.3 1.2: Repetir el punto anterior para el dataset *penguin*.

1.2.1: Inferencia y analisis de datos

[29]:
```
qda = QDA()
qda.fit(train_x_penguin, train_y_penguin)

print('### DATASET PENGUIN ###')
train_acc = accuracy(train_y_penguin, qda.predict(train_x_penguin))
test_acc = accuracy(test_y_penguin, qda.predict(test_x_penguin))
print(f"Train (apparent) error is {1-train_acc:.4f} while test error is␣
↪{1-test_acc:.4f}")

# Model: QDA - Dataset: PENGUIN - SPLIT: Train - A PRIORI: None
model_name="qda"
dataset_name="penguin_train"
seed=RNG_SEED
number=1
```

```python
silence = dispatcher(perdict_method=qda.predict,
           dataset_x=train_x_penguin,
           dataset_y=train_y_penguin,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )

# Model: QDA - Dataset: PENGUIN - SPLIT: Test - A PRIORI: None
dataset_name="penguin_test"

silence = dispatcher(perdict_method=qda.predict,
           dataset_x=test_x_penguin,
           dataset_y=test_y_penguin,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )

# a priori distributions
print(f'### A PRIORI [1/3, 1/3, 1/3] ###')
qda.fit(train_x_penguin, train_y_penguin, a_priori= np.array([1/3, 1/3, 1/3]))
train_acc = accuracy(train_y_penguin, qda.predict(train_x_penguin))
test_acc = accuracy(test_y_penguin, qda.predict(test_x_penguin))
print(f"Train (apparent) error is {1-train_acc:.4f} while test error is␣
 ↪{1-test_acc:.4f}")

# Model: QDA - Dataset: PENGUIN - SPLIT: Train - A PRIORI: [1/3, 1/3, 1/3]
model_name="qda_a_priori_1_3_1_3_1_3"
dataset_name="penguin_train"

silence = dispatcher(perdict_method=qda.predict,
           dataset_x=train_x_penguin,
           dataset_y=train_y_penguin,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )

# Model: QDA - Dataset: PENGUIN - SPLIT: Test - A PRIORI: [1/3, 1/3, 1/3]
```

```
dataset_name="penguin_test"

silence = dispatcher(perdict_method=qda.predict,
           dataset_x=test_x_penguin,
           dataset_y=test_y_penguin,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )


print(f'### A PRIORI [0.9, 0.05, 0.05] ###')
qda.fit(train_x_penguin, train_y_penguin, a_priori= np.array([0.9, 0.05, 0.05]))
train_acc = accuracy(train_y_penguin, qda.predict(train_x_penguin))
test_acc = accuracy(test_y_penguin, qda.predict(test_x_penguin))
print(f"Train (apparent) error is {1-train_acc:.4f} while test error is␣
 ↪{1-test_acc:.4f}")

# Model: QDA - Dataset: PENGUIN - SPLIT: Train - A PRIORI: [0.9, 0.05, 0.05]
model_name="qda_a_priori_09_05_05"
dataset_name="penguin_train"


silence = dispatcher(perdict_method=qda.predict,
           dataset_x=train_x_penguin,
           dataset_y=train_y_penguin,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )

# Model: QDA - Dataset: PENGUIN - SPLIT: Test - A PRIORI: [0.9, 0.05, 0.05]
dataset_name="penguin_test"

silence = dispatcher(perdict_method=qda.predict,
           dataset_x=test_x_penguin,
           dataset_y=test_y_penguin,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )
```

```python
print(f'### A PRIORI [0.05, 0.9, 0.05] ###')
qda.fit(train_x_penguin, train_y_penguin, a_priori= np.array([0.05, 0.9, 0.05]))
train_acc = accuracy(train_y_penguin, qda.predict(train_x_penguin))
test_acc = accuracy(test_y_penguin, qda.predict(test_x_penguin))
print(f"Train (apparent) error is {1-train_acc:.4f} while test error is␣
 ↪{1-test_acc:.4f}")

# Model: QDA - Dataset: PENGUIN - SPLIT: Train - A PRIORI: [0.05, 0.9, 0.05]
model_name="qda_a_priori_05_09_05"
dataset_name="penguin_train"


silence = dispatcher(perdict_method=qda.predict,
           dataset_x=train_x_penguin,
           dataset_y=train_y_penguin,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )

# Model: QDA - Dataset: PENGUIN - SPLIT: Test - A PRIORI: [0.05, 0.9, 0.05]
dataset_name="penguin_test"

silence = dispatcher(perdict_method=qda.predict,
           dataset_x=test_x_penguin,
           dataset_y=test_y_penguin,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )


print(f'### A PRIORI [0.05, 0.05, 0.9] ###')
qda.fit(train_x_penguin, train_y_penguin, a_priori= np.array([0.05, 0.05, 0.9]))
train_acc = accuracy(train_y_penguin, qda.predict(train_x_penguin))
test_acc = accuracy(test_y_penguin, qda.predict(test_x_penguin))
print(f"Train (apparent) error is {1-train_acc:.4f} while test error is␣
 ↪{1-test_acc:.4f}")

# Model: QDA - Dataset: PENGUIN - SPLIT: Train - A PRIORI: [0.05, 0.05, 0.9]
model_name="qda_a_priori_05_05_09"
```

```python
dataset_name="penguin_train"


silence = dispatcher(perdict_method=qda.predict,
           dataset_x=train_x_penguin,
           dataset_y=train_y_penguin,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )

# Model: QDA - Dataset: PENGUIN - SPLIT: Test - A PRIORI: [0.05, 0.05, 0.9]
dataset_name="penguin_test"

silence = dispatcher(perdict_method=qda.predict,
           dataset_x=test_x_penguin,
           dataset_y=test_y_penguin,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )
```

```
### DATASET PENGUIN ###
Train (apparent) error is 0.0098 while test error is 0.0219
Decorator parameters: qda, penguin_train, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda'  timestamp: '2024-10-10
23:16:49.046225'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: qda, penguin_test, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda'  timestamp: '2024-10-10
23:16:52.906735'>' into DB.
Add successful.
Closing all connections…
### A PRIORI [1/3, 1/3, 1/3] ###
Train (apparent) error is 0.0049 while test error is 0.0146
Decorator parameters: qda_a_priori_1_3_1_3_1_3, penguin_train, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_a_priori_1_3_1_3_1_3'  timestamp:
'2024-10-10 23:16:58.143739'>' into DB.
Add successful.
Closing all connections…
```

```
Decorator parameters: qda_a_priori_1_3_1_3_1_3, penguin_test, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_a_priori_1_3_1_3_1_3'  timestamp:
'2024-10-10 23:17:02.096856'>' into DB.
Add successful.
Closing all connections…
### A PRIORI [0.9, 0.05, 0.05] ###
Train (apparent) error is 0.0146 while test error is 0.0292
Decorator parameters: qda_a_priori_09_05_05, penguin_train, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_a_priori_09_05_05'  timestamp:
'2024-10-10 23:17:14.653657'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: qda_a_priori_09_05_05, penguin_test, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_a_priori_09_05_05'  timestamp:
'2024-10-10 23:17:22.594510'>' into DB.
Add successful.
Closing all connections…
### A PRIORI [0.05, 0.9, 0.05] ###
Train (apparent) error is 0.0098 while test error is 0.0365
Decorator parameters: qda_a_priori_05_09_05, penguin_train, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_a_priori_05_09_05'  timestamp:
'2024-10-10 23:17:36.150892'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: qda_a_priori_05_09_05, penguin_test, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_a_priori_05_09_05'  timestamp:
'2024-10-10 23:17:44.854826'>' into DB.
Add successful.
Closing all connections…
### A PRIORI [0.05, 0.05, 0.9] ###
Train (apparent) error is 0.0049 while test error is 0.0146
Decorator parameters: qda_a_priori_05_05_09, penguin_train, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_a_priori_05_05_09'  timestamp:
'2024-10-10 23:17:56.257096'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: qda_a_priori_05_05_09, penguin_test, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_a_priori_05_05_09'  timestamp:
'2024-10-10 23:18:01.063395'>' into DB.
Add successful.
Closing all connections…
```

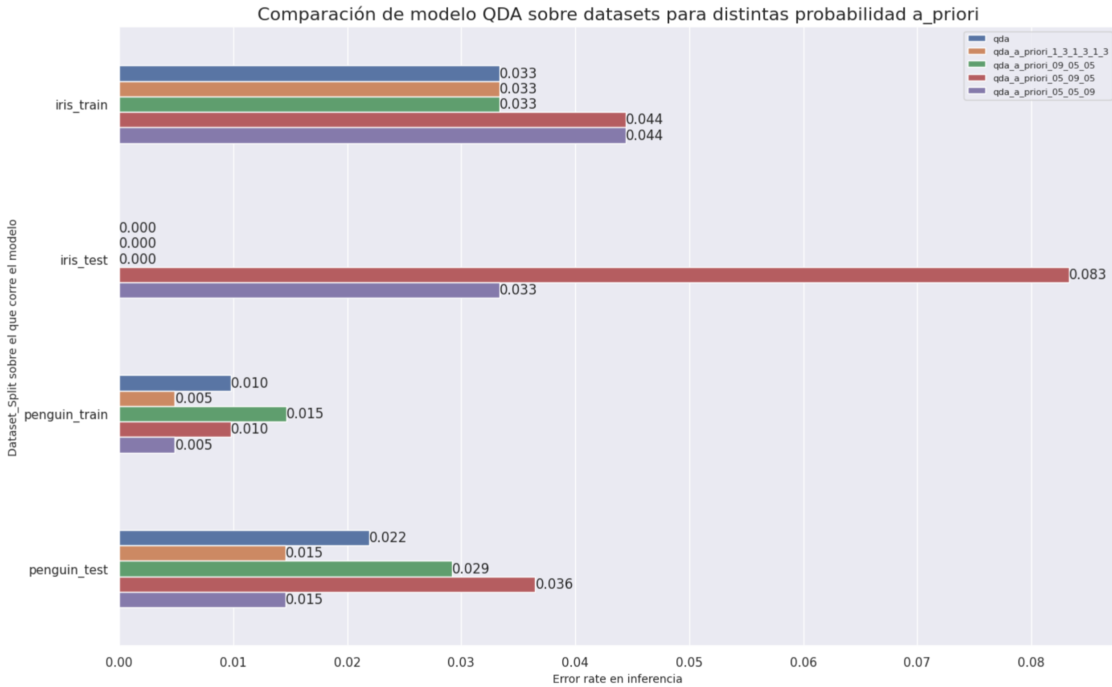
[30]:
```
query = text(f'SELECT * FROM {metrics_table} WHERE model_name LIKE :pattern or␣
↪model_name=:pattern2 and seed=:pattern3')
df = pd.read_sql_query(query, con=engine, params={'pattern': '%qda_a_priori%',␣
↪'pattern2': 'qda', 'pattern3': '6543'})
```

[31]:
```
df.head()
```

[31]:
```
   id                  timestamp                model_name dataset_name  seed  \
0   1 2024-10-10 23:16:14.630317                       qda   iris_train  6543
1   2 2024-10-10 23:16:18.312860                       qda    iris_test  6543
2   3 2024-10-10 23:16:23.330258  qda_a_priori_1_3_1_3_1_3   iris_train  6543
3   4 2024-10-10 23:16:26.256218  qda_a_priori_1_3_1_3_1_3    iris_test  6543
4   5 2024-10-10 23:16:30.320972     qda_a_priori_09_05_05   iris_train  6543

      error  accuracy  memory_allocation  execution_time_ms  \
0  0.033333  0.966667           0.005965           6.411409
1  0.000000  1.000000           0.004926           4.485744
2  0.033333  0.966667           0.006018           6.710519
3  0.000000  1.000000           0.004926           3.270032
4  0.033333  0.966667           0.005965           5.131223

   execution_time_dv_ms comments
0              0.894431
1              0.731992
2              0.951744
3              0.444752
4              0.650266
```

[32]:
```
sns.set(rc={'figure.figsize':(16,10)})
ax = sns.barplot(
    x='error',
    y='dataset_name',
    data=df,
    hue='model_name',
    errorbar=None,
    width=.5,
    #capsize=.2,
    #hue_order=df_sorted['model_name'].unique()
)#, palette="vlag")
for container in ax.containers:
    ax.bar_label(container, fmt='%.3f')
plt.title('Comparación de modelo QDA sobre datasets para distintas probabilidad␣
↪a_priori', fontsize=16)
plt.ylabel('Dataset_Split sobre el que corre el modelo', fontsize = 10)
plt.xlabel('Error rate en inferencia', fontsize = 10)
plt.legend(fontsize=8)
#plt.savefig('img/mem_allocation_algs.png', dpi='figure', bbox_inches='tight')
```

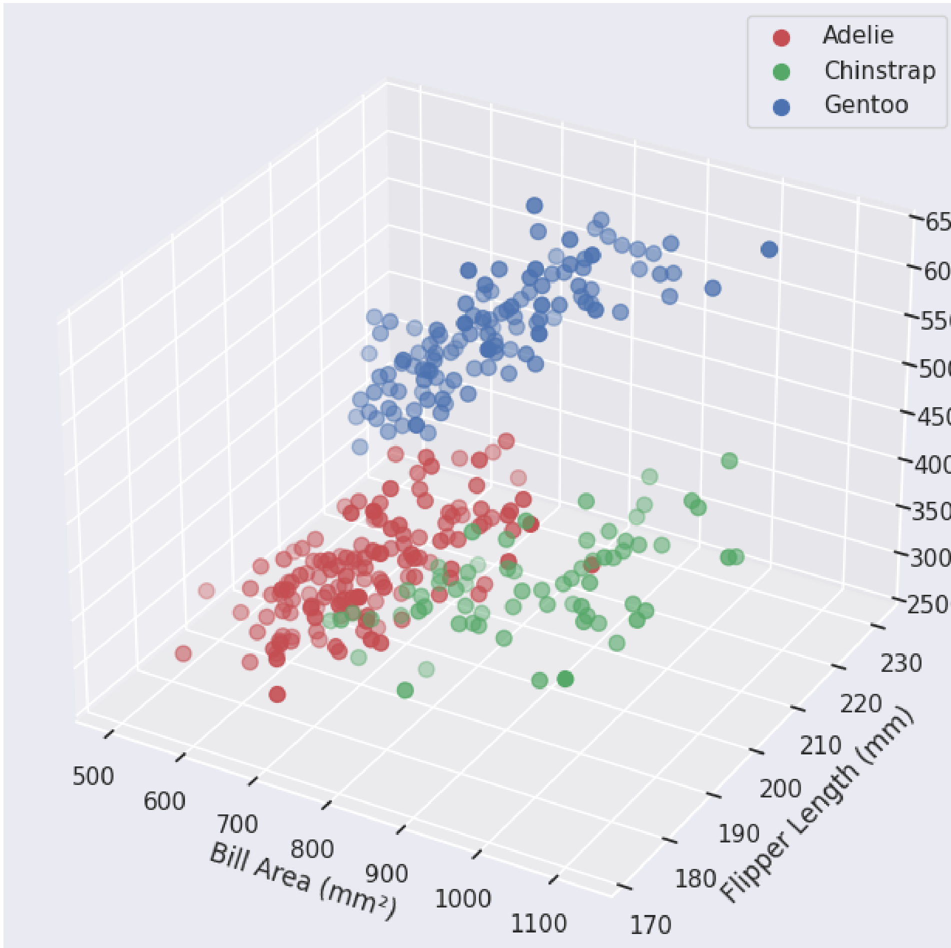
[32]: <matplotlib.legend.Legend at 0x7c56369df430>



[33]:
```python
def plot_penguins_3d_area_distribution():
    # Cargar el dataset de penguins de seaborn
    penguins = sns.load_dataset('penguins')

    # Eliminar filas con valores NaN
    penguins = penguins.dropna()

    # Calcular el área del pico y el área de la aleta
    pico_area = penguins['bill_length_mm'] * penguins['bill_depth_mm']   #␣
↪longitud del pico * profundidad del pico
    aleta_area = penguins['flipper_length_mm']  # longitud de la aleta (como␣
↪proxy de área)
    body_mass = penguins['body_mass_g']  # masa corporal en gramos

    # Crear el gráfico 3D
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='3d')

    # Dibujar el gráfico 3D con diferentes especies
    species_unique = penguins['species'].unique()
    colors = ['r', 'g', 'b']  # colores para las tres especies
```

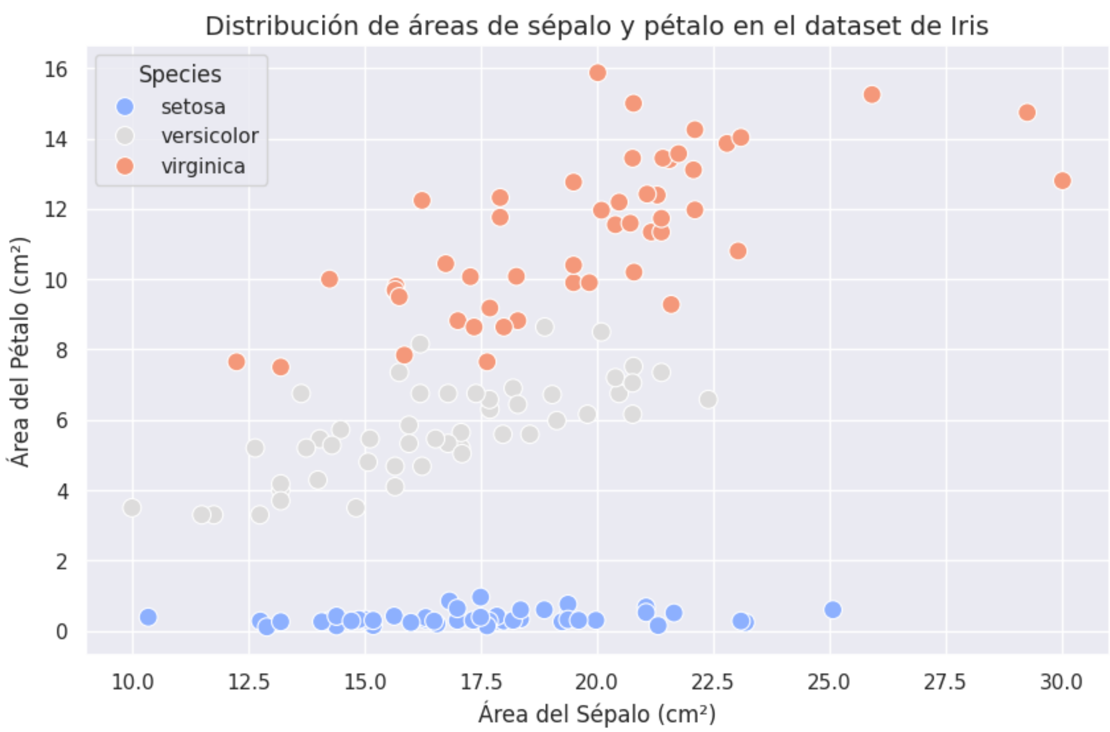
```
    for idx, species in enumerate(species_unique):
        species_data = penguins[penguins['species'] == species]
        ax.scatter(
            species_data['bill_length_mm'] * species_data['bill_depth_mm'],  #␣
 ↪Área del pico
            species_data['flipper_length_mm'],  # Longitud de la aleta
            species_data['body_mass_g'],  # Masa corporal
            color=colors[idx], label=species, s=50
        )

    # Etiquetas de los ejes
    ax.set_xlabel('Bill Area (mm²)')
    ax.set_ylabel('Flipper Length (mm)')
    ax.set_zlabel('Body Mass (g)')

    # Título y leyenda
    ax.set_title('Distribución 3D de los pingüinos por área de pico, aleta y␣
 ↪masa corporal', fontsize=14)
    ax.legend()

    # Mostrar el gráfico
    plt.show()

# Llamar a la función para graficar
plot_penguins_3d_area_distribution()
```

Distribución 3D de los pingüinos por área de pico, aleta y masa corporal



[34]:
```
def plot_iris_area_distribution():
    # Cargar el dataset de Iris
    iris = load_iris()
    X = iris.data
    y = iris.target
    feature_names = iris.feature_names

    # Calcular el área del sépalo y el área del pétalo
    sepalo_area = X[:, 0] * X[:, 1]  # longitud del sépalo * ancho del sépalo
    petalo_area = X[:, 2] * X[:, 3]  # longitud del pétalo * ancho del pétalo

    # Crear un DataFrame con las áreas y los labels
    iris_df = pd.DataFrame({
        'Sepal Area': sepalo_area,
        'Petal Area': petalo_area,
```

```
        'Species': [iris.target_names[label] for label in y]
    })

    # Crear el gráfico de dispersión usando seaborn
    plt.figure(figsize=(10, 6))
    sns.scatterplot(data=iris_df, x='Sepal Area', y='Petal Area',␣
 ↪hue='Species', palette='coolwarm', s=100)

    # Títulos y etiquetas
    plt.title('Distribución de áreas de sépalo y pétalo en el dataset de Iris',␣
 ↪fontsize=14)
    plt.xlabel('Área del Sépalo (cm²)', fontsize=12)
    plt.ylabel('Área del Pétalo (cm²)', fontsize=12)

    # Mostrar la gráfica
    plt.show()

# Llamar a la función para graficar
plot_iris_area_distribution()
```



1.2.2: Conclusiones y explicaciones **NOTA**: Comenzamos observando que la funcion de split de dataset no conservaba la distribución de clases original. Para no agregar ruido al análisis respecto de las probabilidades a priori y su efecto en la inferencia, es que se toma la decisión de agregar el parámetro **stratify** para conservar la relación de muestras al realizar el split. Todos los resultados de este trabajo, de aqui en mas, deberán evaluarse tomando esto en consideración.

DATASET IRIS

Como se puede ver en el diagrama de distribución de áreas sépalo/pétalo (se tomaron áreas para poder contener la información de las 4 features en un plot 2D) la clase setosa, al estar mas apartada (media más distanciada) de las otras clases es esperable que sea menos propensa a error de clasificación. Las otras clases, al tener medias más próximas y mayor varianza, se espera que tengan mayor error.

Como vemos en el gráfico de performance de QDA con distintas probabilidades a priori, vemos que QDA y QDA(1/3,1/3,1/3) tienen la misma performance sobre IRIS ya que originalmente es la distribución de clases del dataset (Ver sección Preparación: Dataset Laoaders).

Lo mismo sucede con QDA(0.9,0.05,0.05), en donde se influencia con la probabilidad a priori la clase setosa, que al ser facilmente clasificable respecto de las otra como se puede observa en figura anterior, no hay cambios en la performance. Esto no sucede cuando la probabilidad a priori de 0.9 cae en las clases versicolor o virginica, ya que se aumenta la probabilidad de elegir en el discriminante una clase mas solapada con otras y mas propensa a error de clasificación.

DATASET PENGUIN

Sea el vector de probabilidades a prori [Adelie, Chinstrap, Gentoo] donde, como se observa en el plot 3D que las primeras dos especies compartes el plano de (Bill Area/Flipper Lenght) en donde muestran superposición, se puede decir que serán clases mas propensas a errores de clasificación a diferencia de Gentoo, que tiene una media más separada.

Además, como se puede ver en el plot de frecuencia por clase (Ver sección Preparación: Dataset Loaders) de este dataset, la probabilidad a priori más alta es para la clase Adelie, justo una de las clases con mayor solapamiento y propensa a error.

En el plot de performance del modelo, puede verse que QDA(1/3,1/3,1/3) tiene menor error que QDA estándar. Esto se explica ya que se igualan las probabilidades a priori, quitandole peso a Adelie, la cual potencialmente puede aportar fallos de clasificación. Asi mismo, aumentar el peso de Gentoo no produce error alguno (al estar más separada en el espacio de features Body Mass), manteniendo una distribución pareja entre Adelie y Chinstrap.

Se observa que en el conjunto de test, la tasa de error de clasificación entre Adelie y Chinstrap se invierten con respecto a **train**. Entendemos que esto puede explicarse si hay mas muestras en test cercanas a la frontera de decision entre esas dos clases.

1.2.4 1.3: Implementar el modelo LDA, entrenarlo y testearlo contra los mismos sets que QDA (no múltiples prioris) ¿Se observan diferencias? ¿Podría decirse que alguno de los dos es notoriamente mejor que el otro?

- Se calcula
- $\hat{\mu}_j = \bar{x}_j$ el promedio de los x de la clase j
- $\hat{\pi}_j = f_{R_j} = \frac{n_j}{n}$ la frecuencia relativa de la clase j en la muestra (dada por la clase BaseBayesianClassifier)
- $\hat{\Sigma} = \frac{1}{n} \sum_{j=1}^{k} n_j \cdot s_j^2$ el promedio ponderado (por frecs. relativas) de las matrices de covarianzas de todas las clases. *Observar que se utiliza el estimador de MV y no el insesgado*

- En np.cov(class_samples, rowvar=False, bias=True), el argumento bias=True asegura que se está utilizando el estimador de máxima verosimilitud. Este estimador divide por n_j (en lugar de $n_j - 1$), que corresponde a la fórmula de máxima verosimilitud y no es insesgado

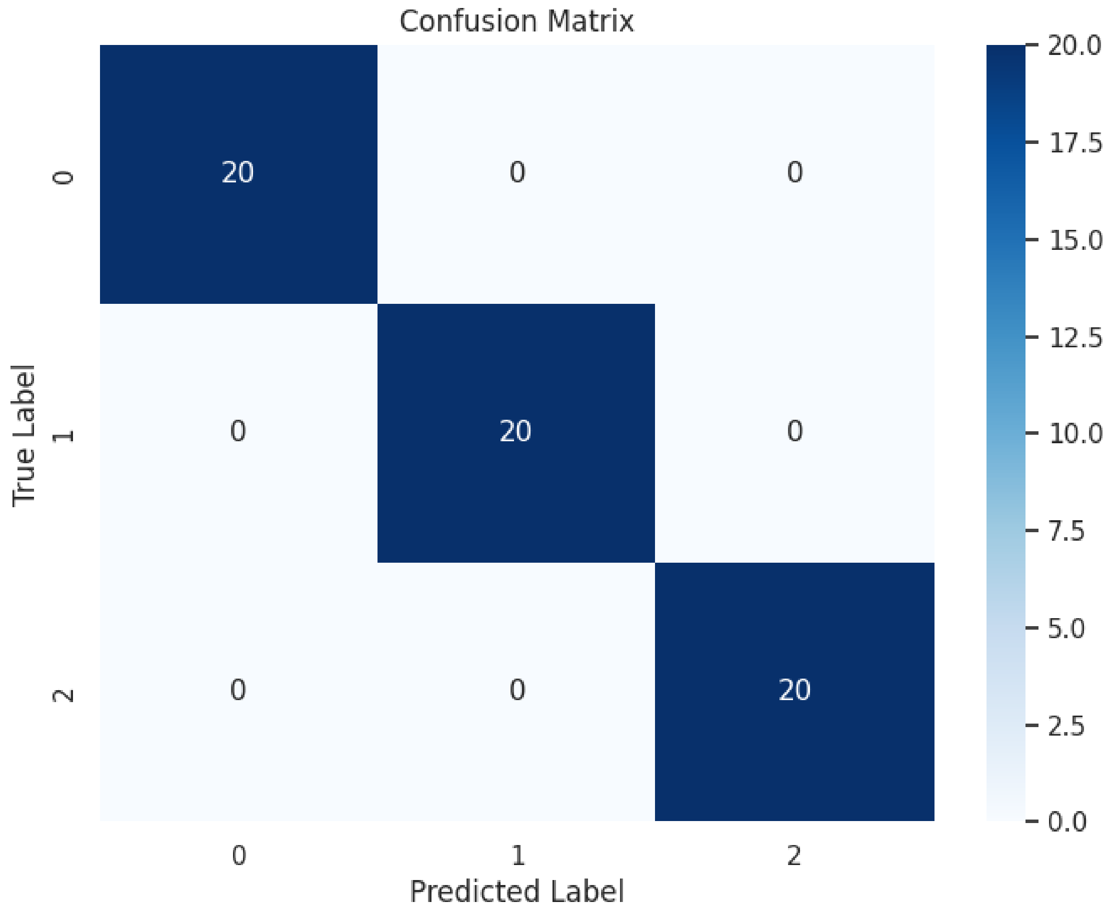
[35]:
```
# Se instancia el modelo LDA implementado en la sección Clases Base - LDA
lda = LDA()
```

[36]:
```
# IRIS DATASET
print("### IRIS DATASET ###")
lda.fit(train_x_iris, train_y_iris)
train_acc = accuracy(train_y_iris, lda.predict(train_x_iris))
test_acc = accuracy(test_y_iris, lda.predict(test_x_iris))
print(f"Train (apparent) error is {1-train_acc:.4f} while test error is␣
 ↪{1-test_acc:.4f}")

# Model: LDA - Dataset: IRIS - SPLIT: Train - A PRIORI: None
model_name="lda"
dataset_name="iris_train"
seed=RNG_SEED
number=1

silence = dispatcher(perdict_method=lda.predict,
           dataset_x=train_x_iris,
           dataset_y=train_y_iris,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )

# Model: LDA - Dataset: IRIS - SPLIT: Test - A PRIORI: None
dataset_name="iris_test"

silence = dispatcher(perdict_method=lda.predict,
           dataset_x=test_x_iris,
           dataset_y=test_y_iris,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )

# CONFUSION MATRIX
test_y_flatten = test_y_iris.flatten()
```

```python
test_x_flatten = lda.predict(test_x_iris).flatten()

df_compare = pd.DataFrame({'True Label': test_y_flatten, 'Predicted Label':␣
↪test_x_flatten})
display(df_compare[df_compare['True Label'] != df_compare['Predicted Label']])

# Calcular la matriz de confusión
cm = confusion_matrix(test_y_flatten, test_x_flatten)
plot_confusion_matrix(cm=cm)
```

```
### IRIS DATASET ###
Train (apparent) error is 0.0333 while test error is 0.0000
Decorator parameters: lda, iris_train, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'lda'  timestamp: '2024-10-10
23:18:04.661040'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: lda, iris_test, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'lda'  timestamp: '2024-10-10
23:18:06.315344'>' into DB.
Add successful.
Closing all connections…
```

```
Empty DataFrame
Columns: [True Label, Predicted Label]
Index: []
```



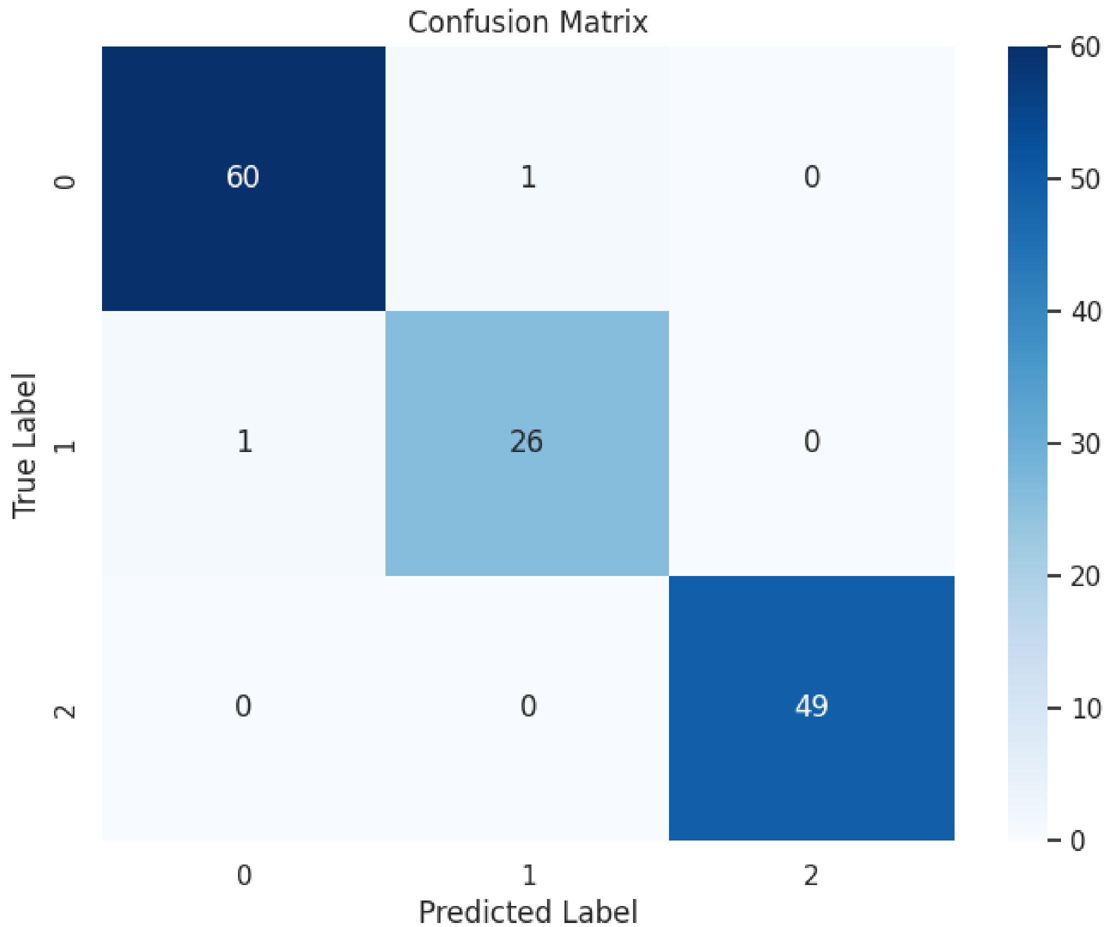
[37]:
```python
# PENGUIN DATASET
print("### PENGUIN DATASET ###")
lda.fit(train_x_penguin, train_y_penguin)
train_acc = accuracy(train_y_penguin, lda.predict(train_x_penguin))
test_acc = accuracy(test_y_penguin, lda.predict(test_x_penguin))
print(f"Train (apparent) error is {1-train_acc:.4f} while test error is␣
 ↪{1-test_acc:.4f}")


# Model: LDA - Dataset: PENGUIN - SPLIT: Train - A PRIORI: None
model_name="lda"
dataset_name="penguin_train"
seed=RNG_SEED
number=1

silence = dispatcher(perdict_method=lda.predict,
           dataset_x=train_x_penguin,
           dataset_y=train_y_penguin,
```

```
            model_name=model_name,
            dataset_name=dataset_name,
            seed=seed,
            number=number,
            repeat=repeat
        )

# Model: LDA - Dataset: IRIS - SPLIT: Test - A PRIORI: None
dataset_name="penguin_test"

silence = dispatcher(perdict_method=lda.predict,
            dataset_x=test_x_penguin,
            dataset_y=test_y_penguin,
            model_name=model_name,
            dataset_name=dataset_name,
            seed=seed,
            number=number,
            repeat=repeat
        )

# CONFUSION MATRIX
test_y_flatten = test_y_penguin.flatten()
test_x_flatten = lda.predict(test_x_penguin).flatten()

df_compare = pd.DataFrame({'True Label': test_y_flatten, 'Predicted Label':␣
 ↪test_x_flatten})
display(df_compare[df_compare['True Label'] != df_compare['Predicted Label']])

# Calcular la matriz de confusión
cm = confusion_matrix(test_y_flatten, test_x_flatten)
plot_confusion_matrix(cm=cm)
```

```
### PENGUIN DATASET ###
Train (apparent) error is 0.0098 while test error is 0.0146
Decorator parameters: lda, penguin_train, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'lda'  timestamp: '2024-10-10
23:18:09.795230'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: lda, penguin_test, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'lda'  timestamp: '2024-10-10
23:18:12.352630'>' into DB.
Add successful.
Closing all connections…
```

```
   True Label Predicted Label
```

```
10  Chinstrap         Adelie
38     Adelie      Chinstrap
```



[38]:
```
query = text(f'SELECT * FROM {metrics_table} WHERE model_name=:pattern or␣
↪model_name=:pattern3 and seed=:pattern2')
df = pd.read_sql_query(query, con=engine, params={'pattern': 'lda', 'pattern2':␣
↪'6543', 'pattern3': 'qda'})
```

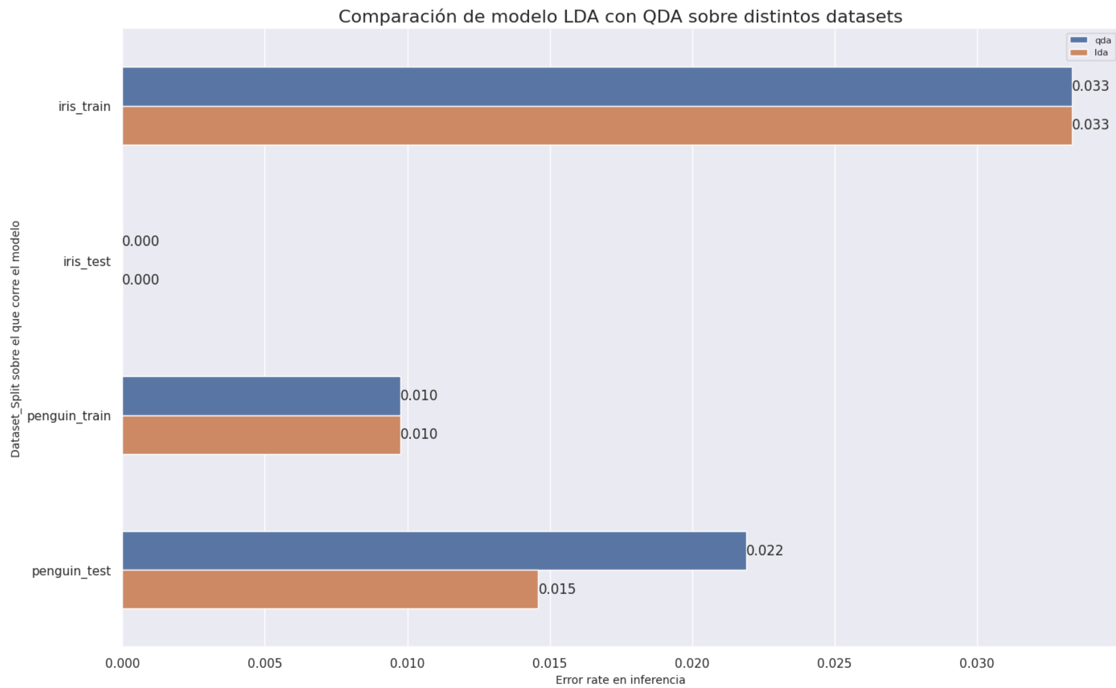
[39]:
```
df.head()
```

[39]:
```
   id                  timestamp model_name  dataset_name  seed     error  \
0   1 2024-10-10 23:16:14.630317        qda    iris_train  6543  0.033333
1   2 2024-10-10 23:16:18.312860        qda     iris_test  6543  0.000000
2  11 2024-10-10 23:16:49.046225        qda penguin_train  6543  0.009756
3  12 2024-10-10 23:16:52.906735        qda  penguin_test  6543  0.021898
4  21 2024-10-10 23:18:04.661040        lda    iris_train  6543  0.033333

   accuracy  memory_allocation  execution_time_ms  execution_time_dv_ms  \
```

```
0  0.966667            0.005965            6.411409              0.894431
1  1.000000            0.004926            4.485744              0.731992
2  0.990244            0.004096            6.968637              0.751504
3  0.978102            0.003682            4.810729              0.516173
4  0.966667            0.005867            1.709647              0.178541

  comments
0
1
2
3
4
```

[40]:
```
sns.set(rc={'figure.figsize':(16,10)})
ax = sns.barplot(
    x='error',
    y='dataset_name',
    data=df,
    hue='model_name',
    errorbar=None,
    width=.5,
    #capsize=.2,
    #hue_order=df_sorted['model_name'].unique()
)#, palette="vlag")
for container in ax.containers:
    ax.bar_label(container, fmt='%.3f')
plt.title('Comparación de modelo LDA con QDA sobre distintos datasets',␣
 ↪fontsize=16)
plt.ylabel('Dataset_Split sobre el que corre el modelo', fontsize = 10)
plt.xlabel('Error rate en inferencia', fontsize = 10)
plt.legend(fontsize=8)
#plt.savefig('img/mem_allocation_algs.png', dpi='figure', bbox_inches='tight')
```

[40]:
```
<matplotlib.legend.Legend at 0x7c55d67266b0>
```



Para la predicción de clases dada una nueva observación no se notan diferencias significativas. Entendemos que esto puede estar asociado a que los datasets tienen pocos datos (100x)[Ver cita ISL] y a que las medias de las clases estan lo suficientemente distanciadas como para que un plano pueda ser un buen elemento de separación para clasificar como se observa en el plot 3D de Penguins y 2D Iris (ver sección 1.2.1).

An Introduction to Statistical Learning, With Applications in Python - Page 153

```
But there is a trade-off: if LDA's assumption that
the K classes share a common covariance matrix is badly off, then LDA
can suffer from high bias. Roughly speaking, LDA tends to be a better bet
than QDA if there are relatively few training observations and so reducing
variance is crucial. In contrast, QDA is recommended if the training set is
very large, so that the variance of the classifier is not a major concern, or if
the assumption of a common covariance matrix for the K classes is clearly
untenable
```

1.2.5 1.4: Utilizar otros 2 (dos) valores de random seed para obtener distintos splits de train y test, y repetir la comparación del punto anterior ¿Las conclusiones previas se mantienen?

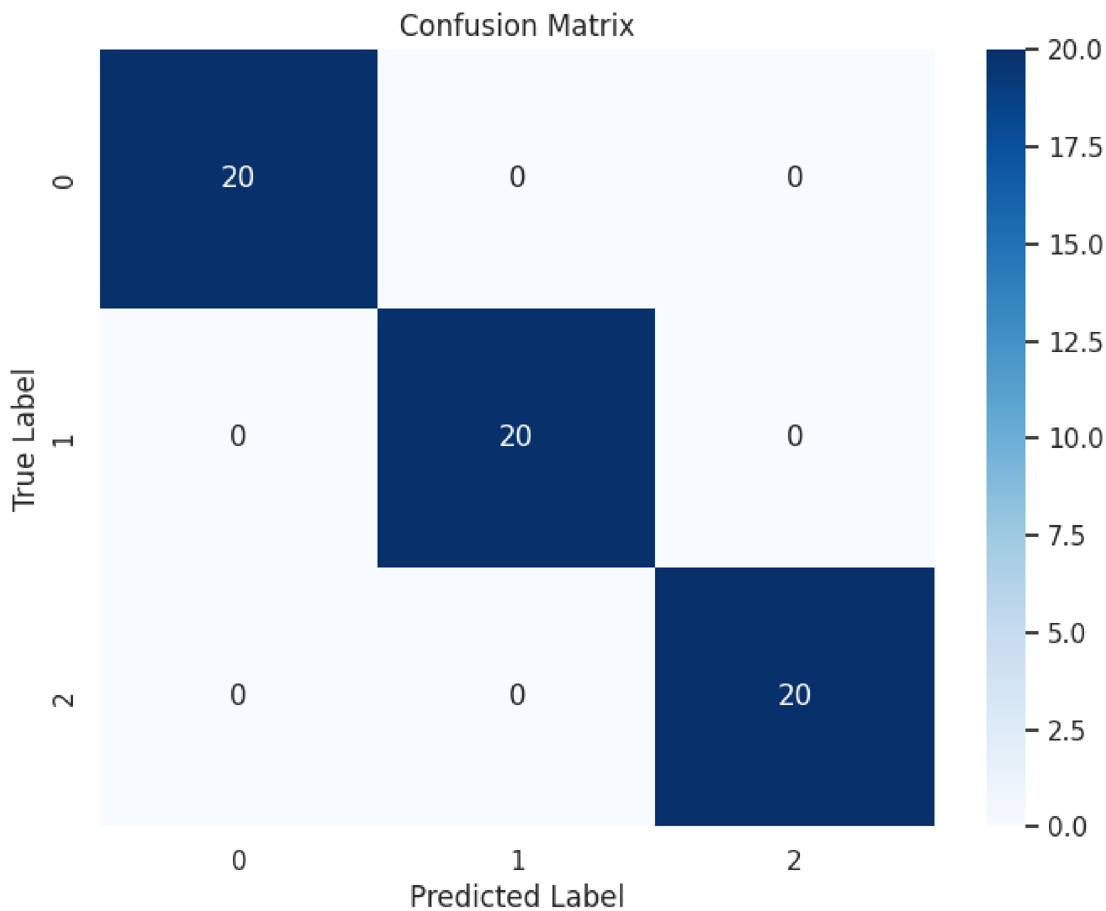
[41]:
```
seed_list = [RNG_SEED_2, RNG_SEED_3]
models_list = [qda, lda]

for seed_item in seed_list:
    for model_item in models_list:
```

```
    print(f"Running for {model_item.name} and seed {seed_item}")

    train_x_iris, train_y_iris, test_x_iris, test_y_iris =␣
↪split_transpose(X_full_iris, y_full_iris, TEST_SIZE, seed_item)
    train_x_penguin, train_y_penguin, test_x_penguin, test_y_penguin =␣
↪split_transpose(X_full_penguin, y_full_penguin, TEST_SIZE, seed_item)

    print("### IRIS DATASET ###")
    model_item.fit(train_x_iris, train_y_iris)
    train_acc = accuracy(train_y_iris, model_item.predict(train_x_iris))
    test_acc = accuracy(test_y_iris, model_item.predict(test_x_iris))
    print(f"Train (apparent) error is {1-train_acc:.4f} while test error is␣
↪{1-test_acc:.4f}")

    model_name=model_item.name
    dataset_name="iris_train"
    seed=seed_item
    number=1

    silence = dispatcher(perdict_method=model_item.predict,
              dataset_x=train_x_iris,
              dataset_y=train_y_iris,
              model_name=model_name,
              dataset_name=dataset_name,
              seed=seed,
              number=number,
              repeat=repeat
            )

    dataset_name="iris_test"

    silence = dispatcher(perdict_method=model_item.predict,
              dataset_x=test_x_iris,
              dataset_y=test_y_iris,
              model_name=model_name,
              dataset_name=dataset_name,
              seed=seed,
              number=number,
              repeat=repeat
            )

    # CONFUSION MATRIX
    test_y_flatten = test_y_iris.flatten()
    test_x_flatten = model_item.predict(test_x_iris).flatten()

    df_compare = pd.DataFrame({'True Label': test_y_flatten, 'Predicted␣
↪Label': test_x_flatten})
```

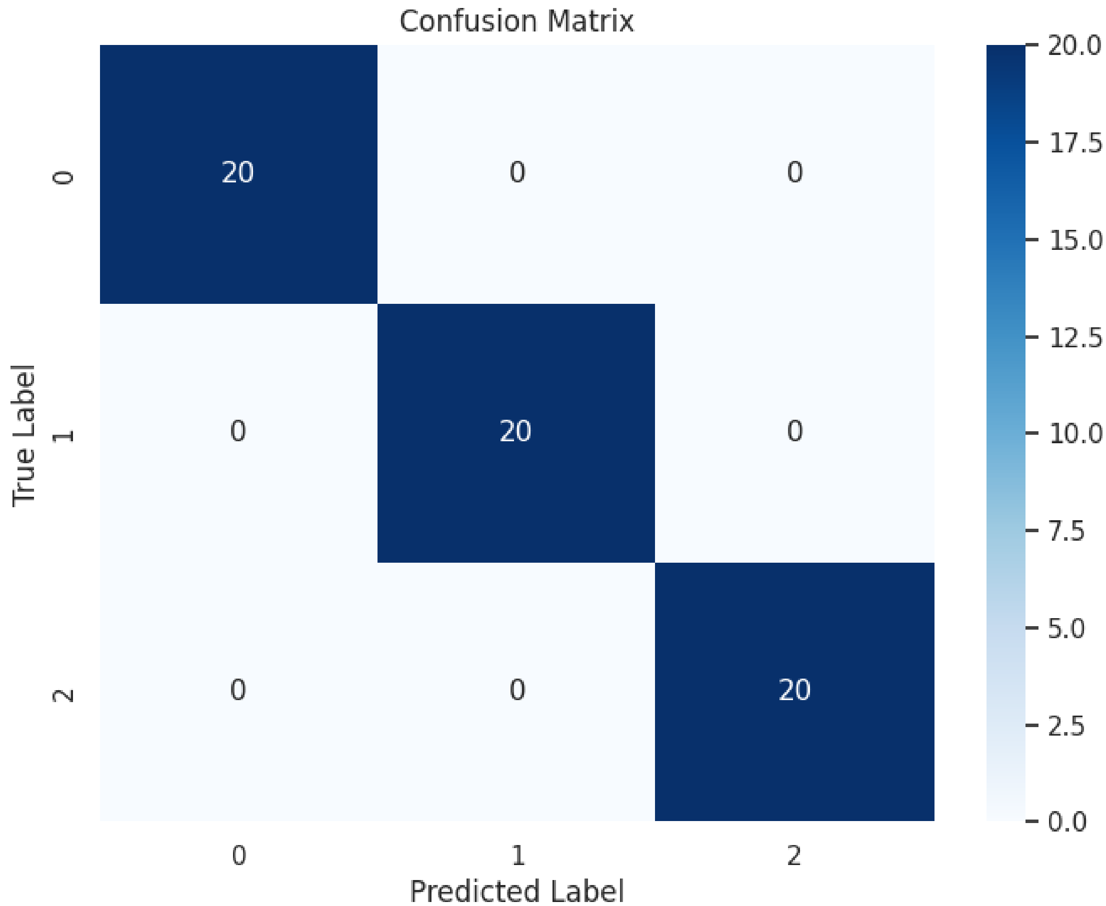
```
        display(df_compare[df_compare['True Label'] != df_compare['Predicted␣
↪Label']])

        # Calcular la matriz de confusión
        cm = confusion_matrix(test_y_flatten, test_x_flatten)
        plot_confusion_matrix(cm=cm)
```

```
Running for qda and seed 4111
### IRIS DATASET ###
Train (apparent) error is 0.0333 while test error is 0.0000
Decorator parameters: qda, iris_train, 4111
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda'  timestamp: '2024-10-10
23:18:16.391133'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: qda, iris_test, 4111
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda'  timestamp: '2024-10-10
23:18:18.542148'>' into DB.
Add successful.
Closing all connections…
```

```
Empty DataFrame
Columns: [True Label, Predicted Label]
Index: []
```

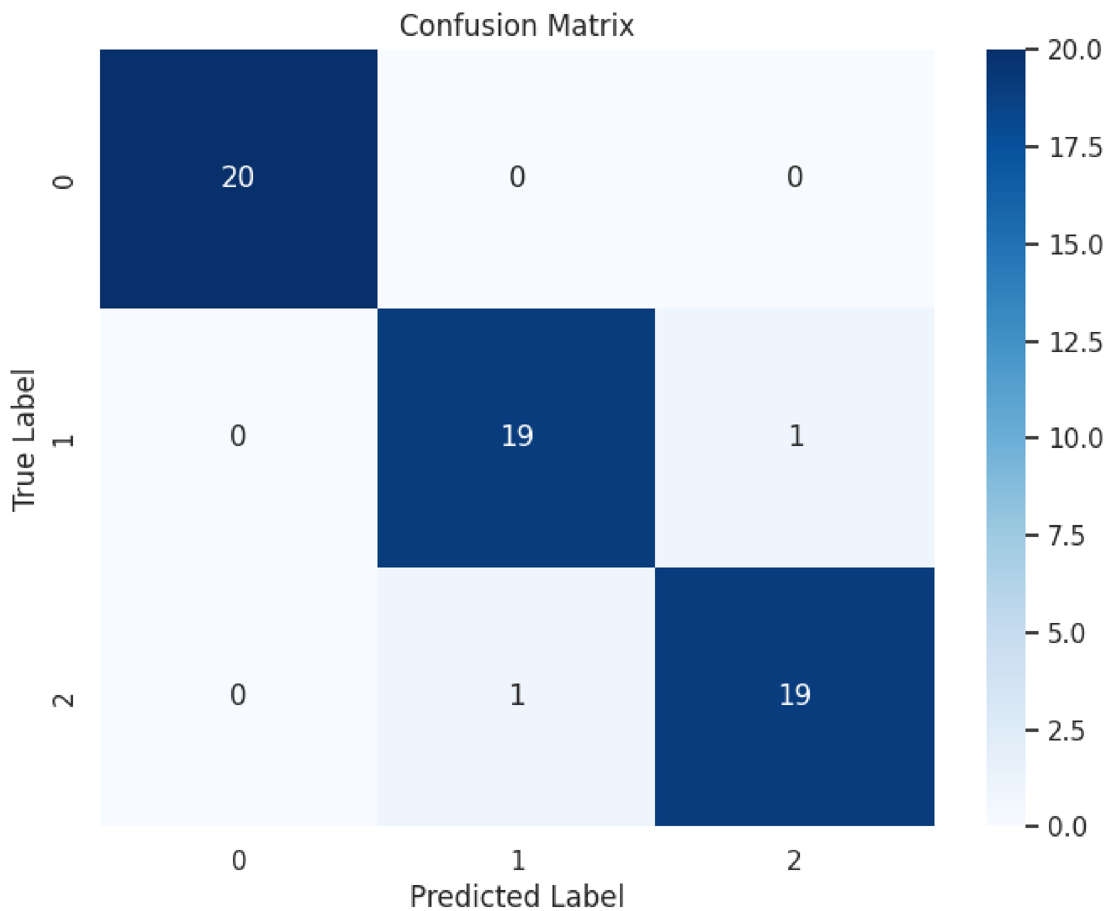



```
Running for lda and seed 4111
### IRIS DATASET ###
Train (apparent) error is 0.0333 while test error is 0.0000
Decorator parameters: lda, iris_train, 4111
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'lda'  timestamp: '2024-10-10
23:18:20.721214'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: lda, iris_test, 4111
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'lda'  timestamp: '2024-10-10
23:18:22.444195'>' into DB.
Add successful.
Closing all connections…

Empty DataFrame
Columns: [True Label, Predicted Label]
Index: []
```

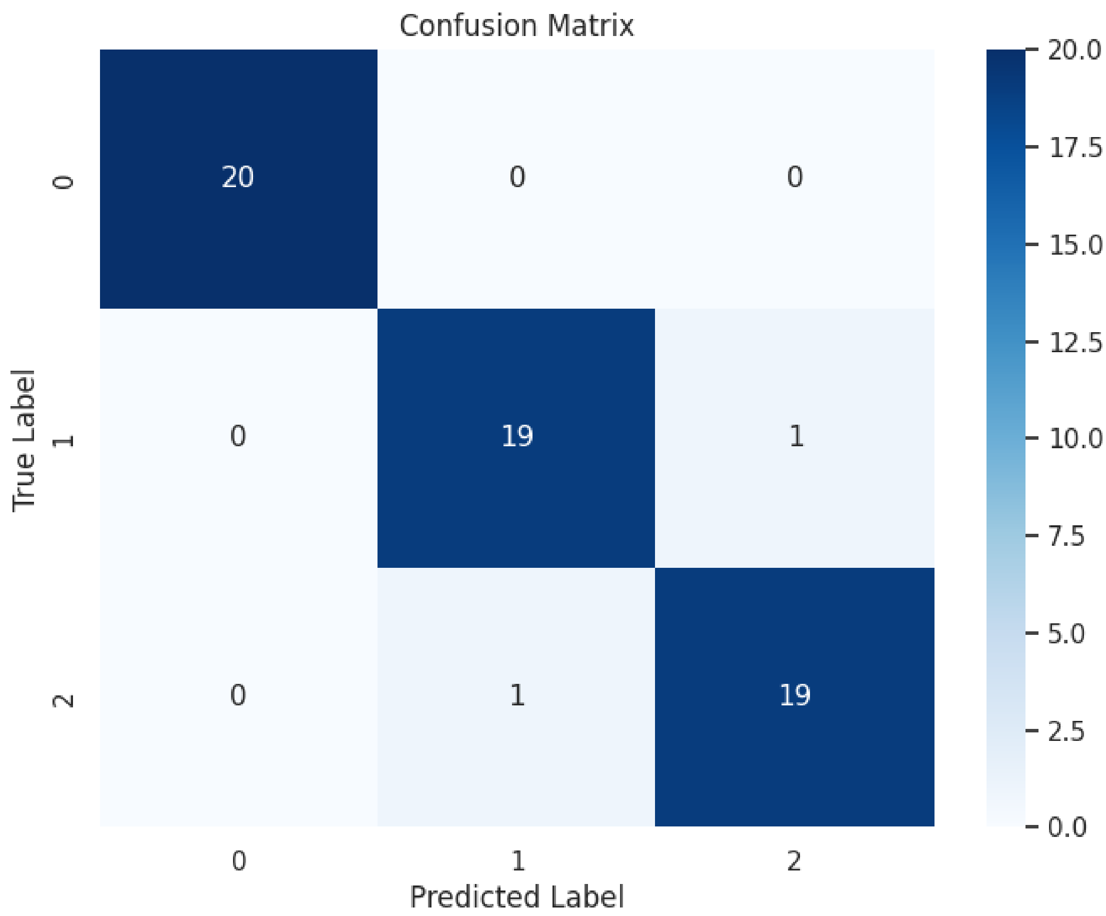



```
Running for qda and seed 2323
### IRIS DATASET ###
Train (apparent) error is 0.0111 while test error is 0.0333
Decorator parameters: qda, iris_train, 2323
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda'  timestamp: '2024-10-10
23:18:25.505667'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: qda, iris_test, 2323
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda'  timestamp: '2024-10-10
23:18:27.959766'>' into DB.
Add successful.
Closing all connections…
```

```
    True Label Predicted Label
31   virginica      versicolor
50  versicolor       virginica
```




```
Running for lda and seed 2323
### IRIS DATASET ###
Train (apparent) error is 0.0111 while test error is 0.0333
Decorator parameters: lda, iris_train, 2323
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'lda'  timestamp: '2024-10-10
23:18:31.300585'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: lda, iris_test, 2323
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'lda'  timestamp: '2024-10-10
23:18:33.568272'>' into DB.
Add successful.
Closing all connections…

    True Label Predicted Label
31   virginica      versicolor
50  versicolor       virginica
```



[42]:

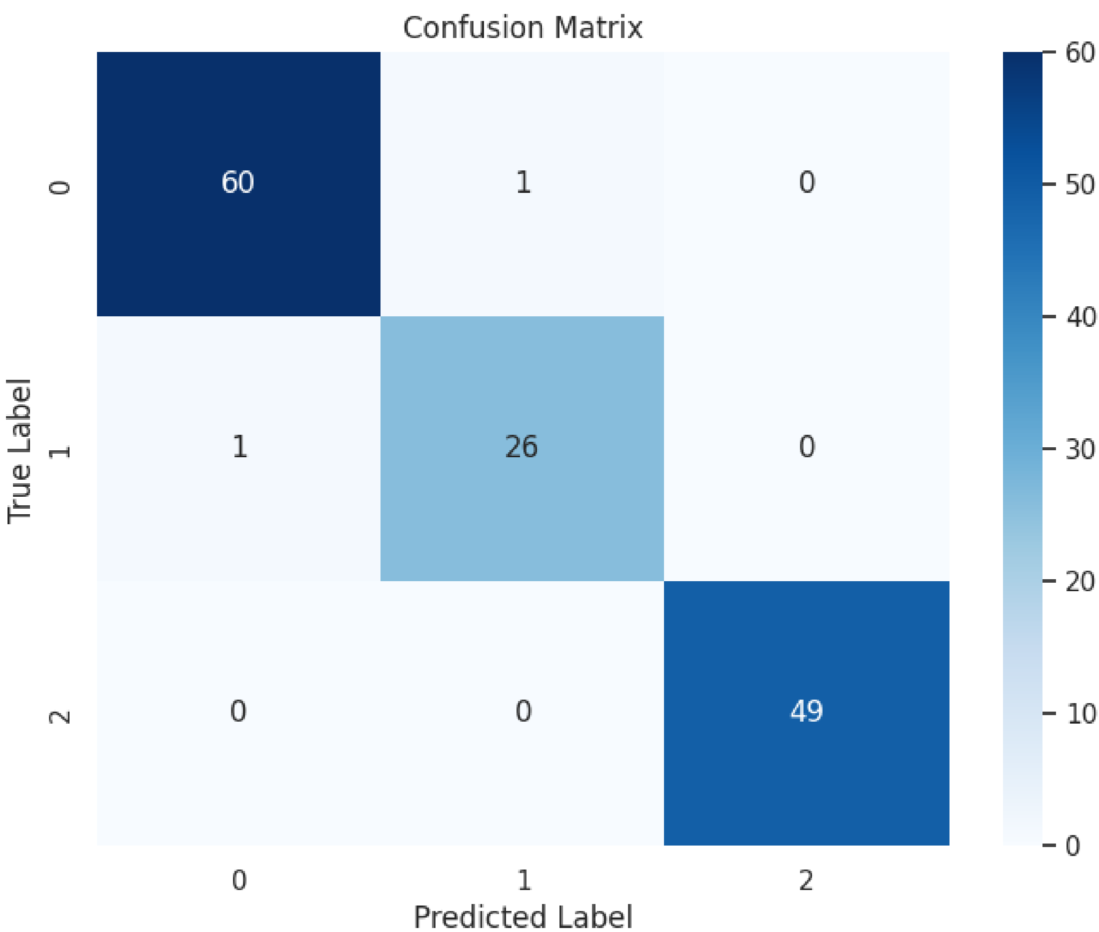
```
seed_list = [RNG_SEED_2, RNG_SEED_3]
models_list = [qda, lda]

for seed_item in seed_list:
    for model_item in models_list:
        print(f"Running for {model_item.name} and seed {seed_item}")
        train_x_iris, train_y_iris, test_x_iris, test_y_iris =␣
↪split_transpose(X_full_iris, y_full_iris, TEST_SIZE, seed_item)
        train_x_penguin, train_y_penguin, test_x_penguin, test_y_penguin =␣
↪split_transpose(X_full_penguin, y_full_penguin, TEST_SIZE, seed_item)

        print("### PENGUIN DATASET ###")
        model_item.fit(train_x_penguin, train_y_penguin)
        train_acc = accuracy(train_y_penguin, model_item.
↪predict(train_x_penguin))
        test_acc = accuracy(test_y_penguin, model_item.predict(test_x_penguin))
        print(f"Train (apparent) error is {1-train_acc:.4f} while test error is␣
↪{1-test_acc:.4f}")
```

```
        model_name=model_item.name
        dataset_name="penguin_train"
        seed=seed_item
        number=1

        silence = dispatcher(perdict_method=model_item.predict,
                    dataset_x=train_x_penguin,
                    dataset_y=train_y_penguin,
                    model_name=model_name,
                    dataset_name=dataset_name,
                    seed=seed,
                    number=number,
                    repeat=repeat
                )

        dataset_name="penguin_test"

        silence = dispatcher(perdict_method=model_item.predict,
                    dataset_x=test_x_penguin,
                    dataset_y=test_y_penguin,
                    model_name=model_name,
                    dataset_name=dataset_name,
                    seed=seed,
                    number=number,
                    repeat=repeat
                )

        # CONFUSION MATRIX
        test_y_flatten = test_y_penguin.flatten()
        test_x_flatten = model_item.predict(test_x_penguin).flatten()

        df_compare = pd.DataFrame({'True Label': test_y_flatten, 'Predicted␣
↪Label': test_x_flatten})
        display(df_compare[df_compare['True Label'] != df_compare['Predicted␣
↪Label']])

        # Calcular la matriz de confusión
        cm = confusion_matrix(test_y_flatten, test_x_flatten)
        plot_confusion_matrix(cm=cm)
```

```
Running for qda and seed 4111
### PENGUIN DATASET ###
Train (apparent) error is 0.0098 while test error is 0.0146
Decorator parameters: qda, penguin_train, 4111
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda'  timestamp: '2024-10-10
```

```
23:18:42.832296'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: qda, penguin_test, 4111
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda'  timestamp: '2024-10-10
23:18:48.706142'>' into DB.
Add successful.
Closing all connections…
```

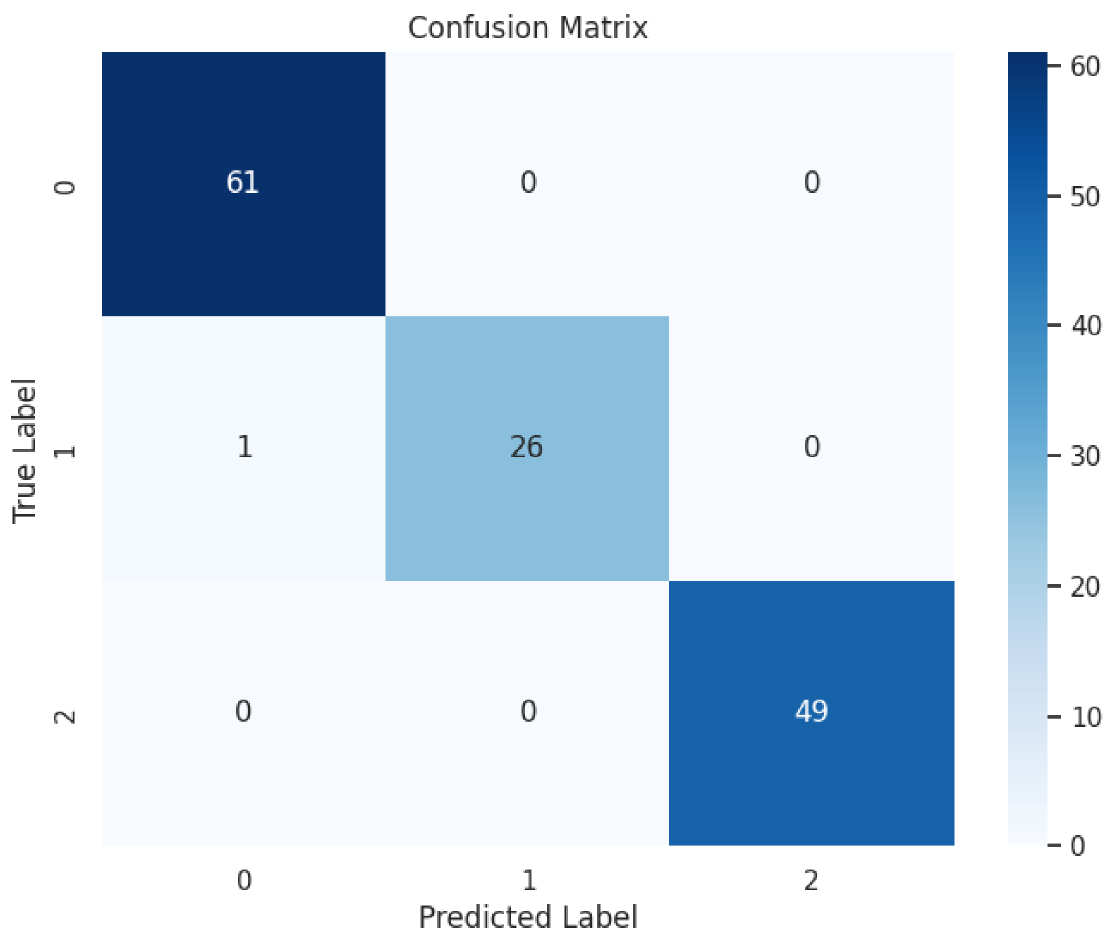
```
   True Label Predicted Label
55     Adelie       Chinstrap
89  Chinstrap          Adelie
```




```
Running for lda and seed 4111
### PENGUIN DATASET ###
Train (apparent) error is 0.0098 while test error is 0.0073
Decorator parameters: lda, penguin_train, 4111
Initializing DatabaseService instance
```

```
Adding '<Metrics id: 'None' model_name: 'lda'  timestamp: '2024-10-10
23:18:54.040331'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: lda, penguin_test, 4111
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'lda'  timestamp: '2024-10-10
23:18:57.707806'>' into DB.
Add successful.
Closing all connections…
```

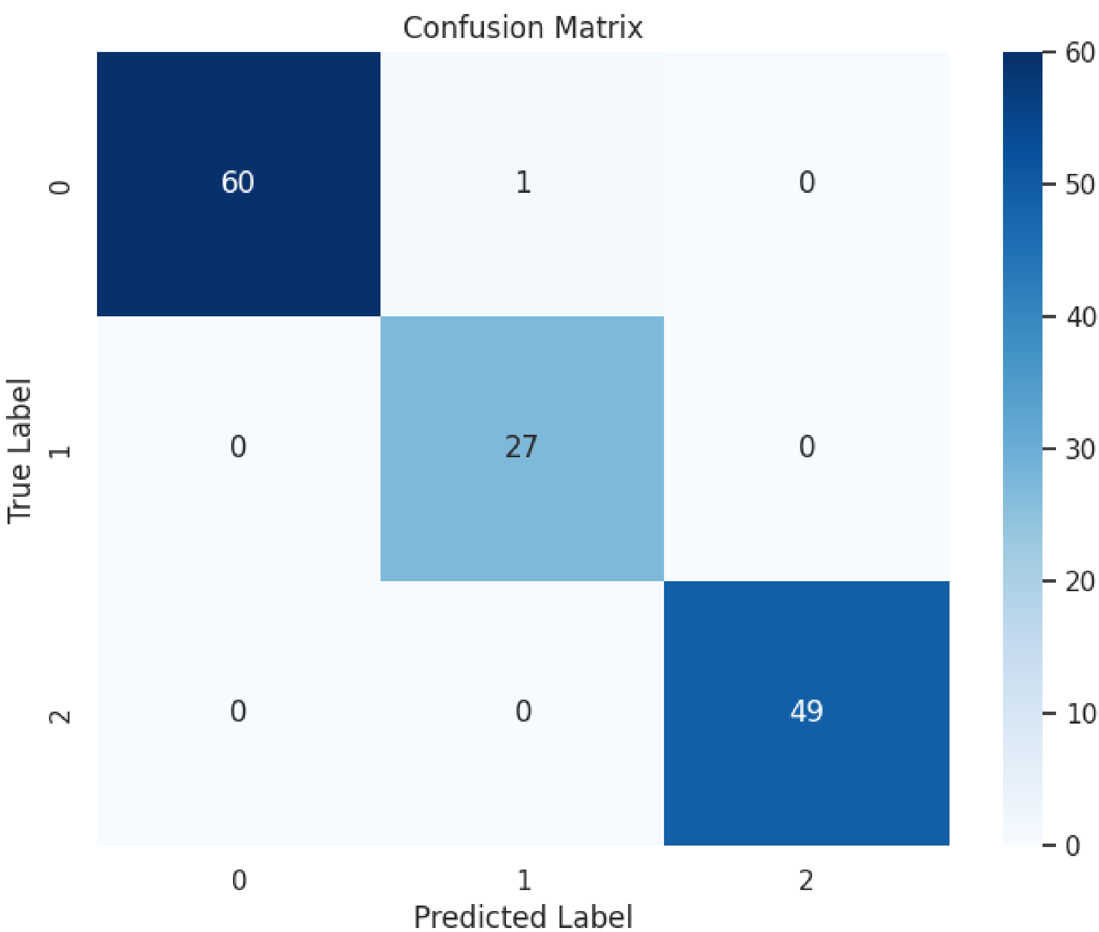
```
   True Label Predicted Label
89  Chinstrap          Adelie
```




```
Running for qda and seed 2323
### PENGUIN DATASET ###
Train (apparent) error is 0.0098 while test error is 0.0073
Decorator parameters: qda, penguin_train, 2323
Initializing DatabaseService instance
```

```
Adding '&lt;Metrics id: 'None' model_name: 'qda'  timestamp: '2024-10-10
23:19:06.155184'&gt;' into DB.
Add successful.
Closing all connections…
Decorator parameters: qda, penguin_test, 2323
Initializing DatabaseService instance
Adding '&lt;Metrics id: 'None' model_name: 'qda'  timestamp: '2024-10-10
23:19:11.867803'&gt;' into DB.
Add successful.
Closing all connections…
```

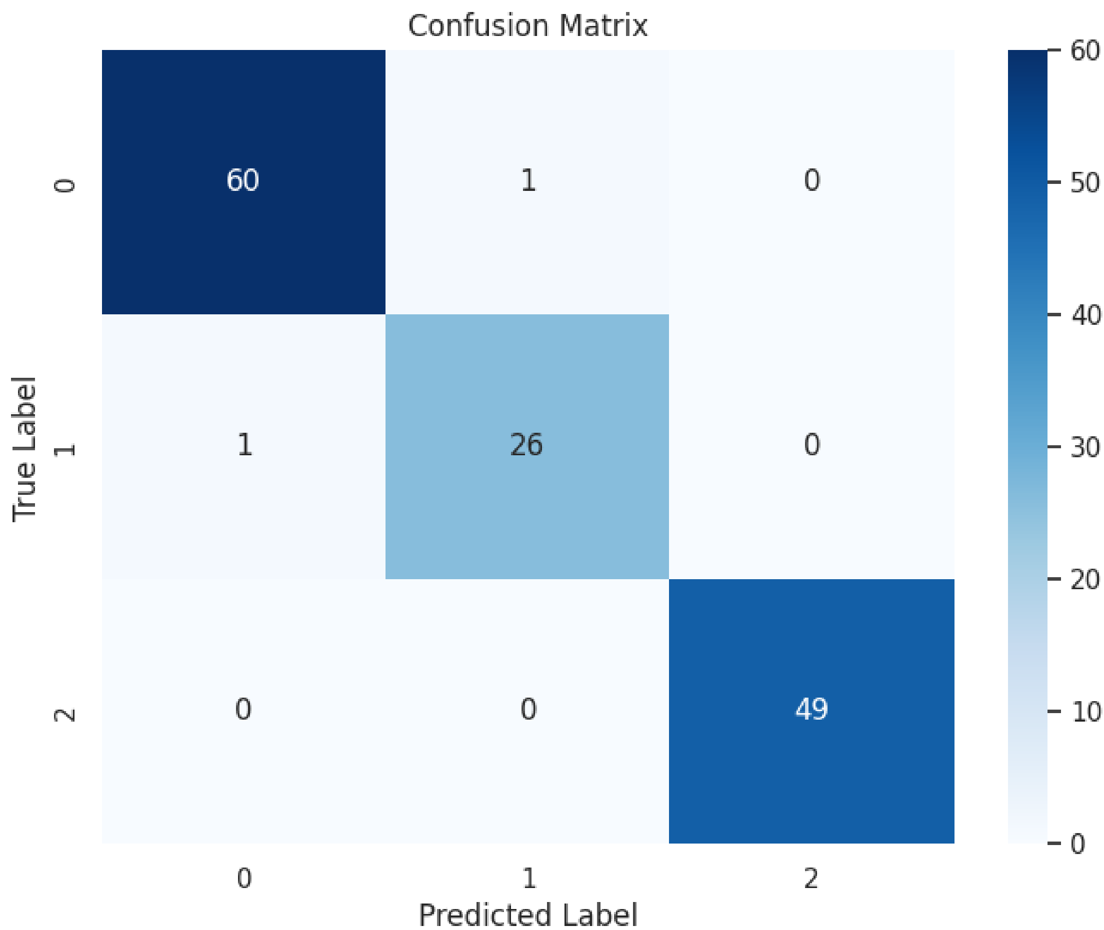
	True Label	Predicted Label
33	Adelie	Chinstrap




```
Running for lda and seed 2323
### PENGUIN DATASET ###
Train (apparent) error is 0.0049 while test error is 0.0146
Decorator parameters: lda, penguin_train, 2323
Initializing DatabaseService instance
```

```
Adding '<Metrics id: 'None' model_name: 'lda'  timestamp: '2024-10-10
23:19:17.174383'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: lda, penguin_test, 2323
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'lda'  timestamp: '2024-10-10
23:19:19.829335'>' into DB.
Add successful.
Closing all connections…
```

```
    True Label Predicted Label
33      Adelie       Chinstrap
121  Chinstrap          Adelie
```



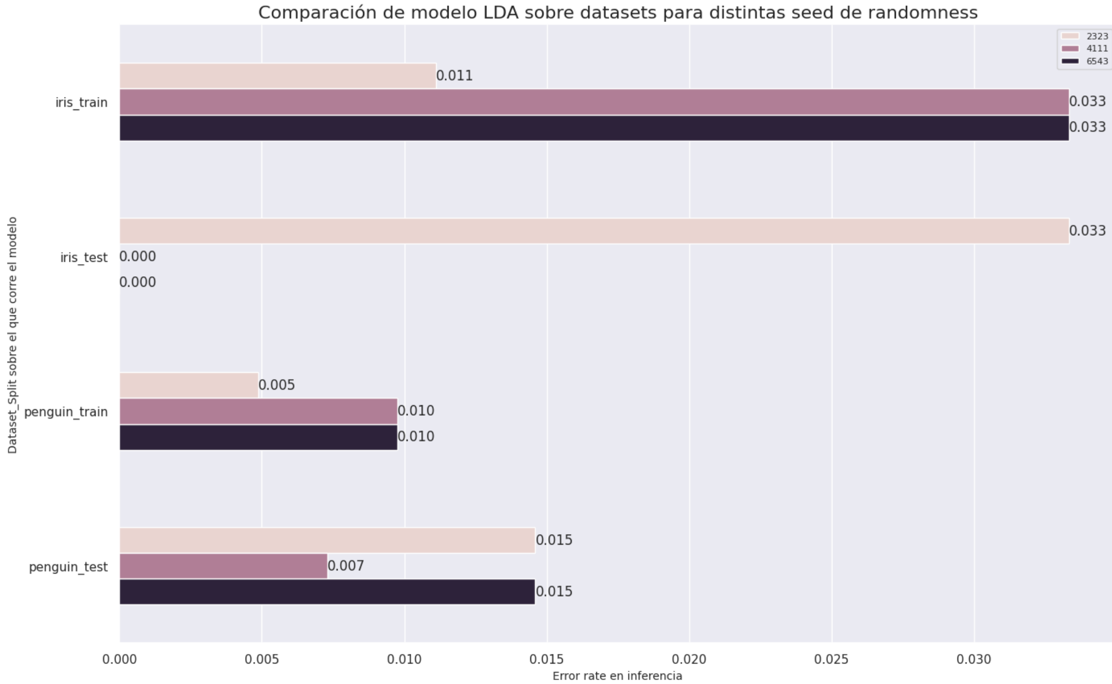
[43]:
```
query = text(f'SELECT * FROM {metrics_table} WHERE model_name=:pattern')
df = pd.read_sql_query(query, con=engine, params={'pattern': 'lda'})
```

[44]:
```
df.head()
```

[44]:
```
   id                  timestamp model_name   dataset_name  seed     error  \
0  21 2024-10-10 23:18:04.661040        lda     iris_train  6543  0.033333
1  22 2024-10-10 23:18:06.315344        lda      iris_test  6543  0.000000
2  23 2024-10-10 23:18:09.795230        lda  penguin_train  6543  0.009756
3  24 2024-10-10 23:18:12.352630        lda   penguin_test  6543  0.014599
4  27 2024-10-10 23:18:20.721214        lda     iris_train  4111  0.033333

   accuracy  memory_allocation  execution_time_ms  execution_time_dv_ms  \
0  0.966667           0.005867           1.709647              0.178541
1  1.000000           0.004723           1.135281              0.068964
2  0.990244           0.003998           3.960398              0.340679
3  0.985401           0.003479           2.646981              0.273339
4  0.966667           0.005989           1.795082              0.227456

  comments
0
1
2
3
4
```

[45]:
```
sns.set(rc={'figure.figsize':(16,10)})
ax = sns.barplot(
    x='error',
    y='dataset_name',
    data=df,
    hue='seed',
    errorbar=None,
    width=.5,
    #capsize=.2,
    #hue_order=df_sorted['model_name'].unique()
)#, palette="vlag")
for container in ax.containers:
    ax.bar_label(container, fmt='%.3f')
plt.title('Comparación de modelo LDA sobre datasets para distintas seed de␣
↪randomness', fontsize=16)
plt.ylabel('Dataset_Split sobre el que corre el modelo', fontsize = 10)
plt.xlabel('Error rate en inferencia', fontsize = 10)
plt.legend(fontsize=8)
#plt.savefig('img/mem_allocation_algs.png', dpi='figure', bbox_inches='tight')
```

[45]:
```
<matplotlib.legend.Legend at 0x7c55d62209a0>
```



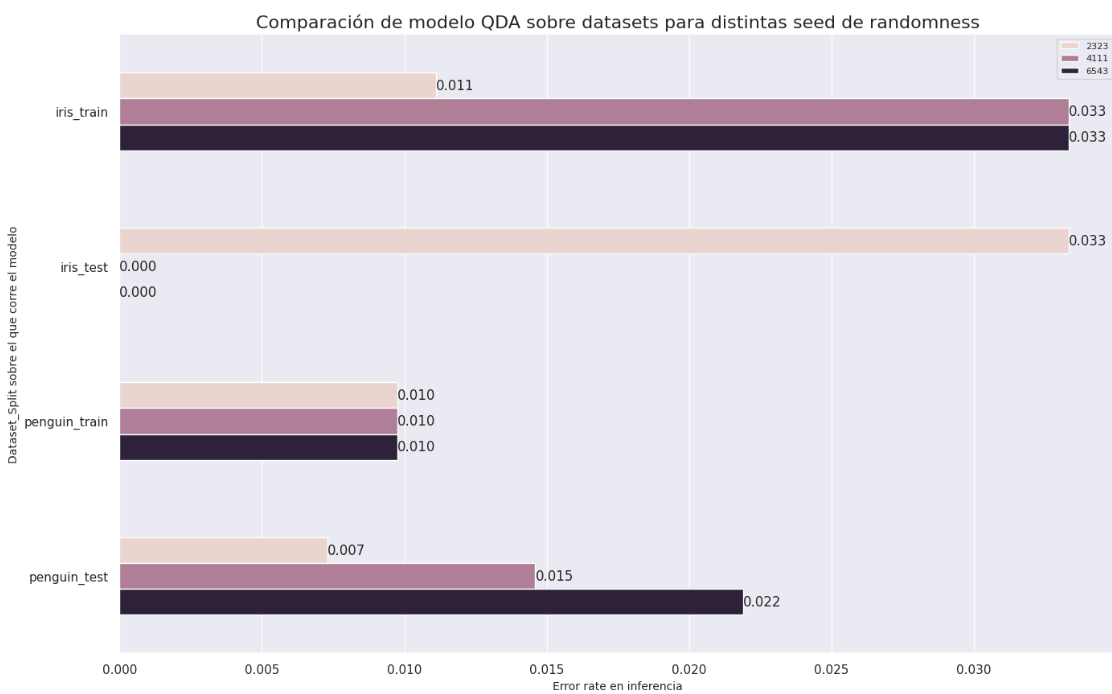
[46]:
```
query = text(f'SELECT * FROM {metrics_table} WHERE model_name=:pattern')
df = pd.read_sql_query(query, con=engine, params={'pattern': 'qda'})
```

[47]:
```
df.head()
```

[47]:
```
   id                  timestamp model_name  dataset_name  seed     error  \
0   1 2024-10-10 23:16:14.630317        qda    iris_train  6543  0.033333
1   2 2024-10-10 23:16:18.312860        qda     iris_test  6543  0.000000
2  11 2024-10-10 23:16:49.046225        qda penguin_train  6543  0.009756
3  12 2024-10-10 23:16:52.906735        qda  penguin_test  6543  0.021898
4  25 2024-10-10 23:18:16.391133        qda    iris_train  4111  0.033333

   accuracy  memory_allocation  execution_time_ms  execution_time_dv_ms  \
0  0.966667           0.005965           6.411409              0.894431
1  1.000000           0.004926           4.485744              0.731992
2  0.990244           0.004096           6.968637              0.751504
3  0.978102           0.003682           4.810729              0.516173
4  0.966667           0.005965           2.933711              0.250066

  comments
0
1
2
3
4
```

[48]:
```
sns.set(rc={'figure.figsize':(16,10)})
ax = sns.barplot(
    x='error',
    y='dataset_name',
    data=df,
    hue='seed',
    errorbar=None,
    width=.5,
    #capsize=.2,
    #hue_order=df_sorted['model_name'].unique()
)#, palette="vlag")
for container in ax.containers:
    ax.bar_label(container, fmt='%.3f')
plt.title('Comparación de modelo QDA sobre datasets para distintas seed de␣
↪randomness', fontsize=16)
plt.ylabel('Dataset_Split sobre el que corre el modelo', fontsize = 10)
plt.xlabel('Error rate en inferencia', fontsize = 10)
plt.legend(fontsize=8)
#plt.savefig('img/mem_allocation_algs.png', dpi='figure', bbox_inches='tight')
```

[48]: <matplotlib.legend.Legend at 0x7c55d62203d0>



No se observan diferencias significativas. Cabe destacar que cuando se realizaron pruebas realizando el split sin stratify, es decir, sin garantizar que se mantenia la proporcion de muestras de cada clase al particionar el dataset, si se observaban diferencias lo que era esperable ya que es equivalente a cambiar las probabilidades a priori (aunque tampoco eran diferencias significativas).

1.2.6 1.5: Estimar y comparar los tiempos de predicción de las clases QDA y TensorizedQDA. De haber diferencias ¿Cuáles pueden ser las causas?

[49]:
```
# SPLIT DATASETS AGAIN WITH ORIGINAL SEED
train_x_iris, train_y_iris, test_x_iris, test_y_iris =␣
↪split_transpose(X_full_iris, y_full_iris, TEST_SIZE, RNG_SEED)
train_x_penguin, train_y_penguin, test_x_penguin, test_y_penguin =␣
↪split_transpose(X_full_penguin, y_full_penguin, TEST_SIZE, RNG_SEED)

qda_tensor = TensorizedQDA()

print("### DATASET IRIS ###")
qda_tensor.fit(train_x_iris, train_y_iris)
train_acc = accuracy(train_y_iris, qda_tensor.predict(train_x_iris))
test_acc = accuracy(test_y_iris, qda_tensor.predict(test_x_iris))
print(f"Train (apparent) error is {1-train_acc:.4f} while test error is␣
↪{1-test_acc:.4f}")

# Model: QDA_TENSOR - Dataset: IRIS - SPLIT: Train - A PRIORI: None
model_name="qda_tensor"
dataset_name="iris_train"
seed=RNG_SEED
number=1

silence = dispatcher(perdict_method=qda_tensor.predict,
           dataset_x=train_x_iris,
           dataset_y=train_y_iris,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )

# Model: QDA_TENSOR - Dataset: IRIS - SPLIT: Test - A PRIORI: None
dataset_name="iris_test"

silence = dispatcher(perdict_method=qda_tensor.predict,
           dataset_x=test_x_iris,
           dataset_y=test_y_iris,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
```

```
        )
```

```
### DATASET IRIS ###
Train (apparent) error is 0.0333 while test error is 0.0000
Decorator parameters: qda_tensor, iris_train, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_tensor'  timestamp: '2024-10-10
23:19:24.030337'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: qda_tensor, iris_test, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_tensor'  timestamp: '2024-10-10
23:19:25.455048'>' into DB.
Add successful.
Closing all connections…
```

[50]:

```
# DATASET PENGUIN
print("### DATASET PENGUIN ###")
qda_tensor.fit(train_x_penguin, train_y_penguin)
train_acc = accuracy(train_y_penguin, qda_tensor.predict(train_x_penguin))
test_acc = accuracy(test_y_penguin, qda_tensor.predict(test_x_penguin))
print(f"Train (apparent) error is {1-train_acc:.4f} while test error is␣
 ↪{1-test_acc:.4f}")

# Model: QDA_TENSOR - Dataset: PENGUIN - SPLIT: Train - A PRIORI: None
model_name="qda_tensor"
dataset_name="penguin_train"
seed=RNG_SEED
number=1

silence = dispatcher(perdict_method=lda.predict,
           dataset_x=train_x_penguin,
           dataset_y=train_y_penguin,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )

# Model: QDA_TENSOR - Dataset: PENGUIN - SPLIT: Test - A PRIORI: None
dataset_name="penguin_test"

silence = dispatcher(perdict_method=lda.predict,
           dataset_x=test_x_penguin,
           dataset_y=test_y_penguin,
```

```
            model_name=model_name,
            dataset_name=dataset_name,
            seed=seed,
            number=number,
            repeat=repeat
        )
```

```
### DATASET PENGUIN ###
Train (apparent) error is 0.0098 while test error is 0.0219
Decorator parameters: qda_tensor, penguin_train, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_tensor'  timestamp: '2024-10-10
23:19:28.573697'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: qda_tensor, penguin_test, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_tensor'  timestamp: '2024-10-10
23:19:31.018352'>' into DB.
Add successful.
Closing all connections…
```

[51]:
```
query = text(f'SELECT * FROM {metrics_table} WHERE model_name=:pattern or␣
 ↪model_name=:pattern2')
df = pd.read_sql_query(query, con=engine, params={'pattern': 'qda', 'pattern2':␣
 ↪'qda_tensor'})
```

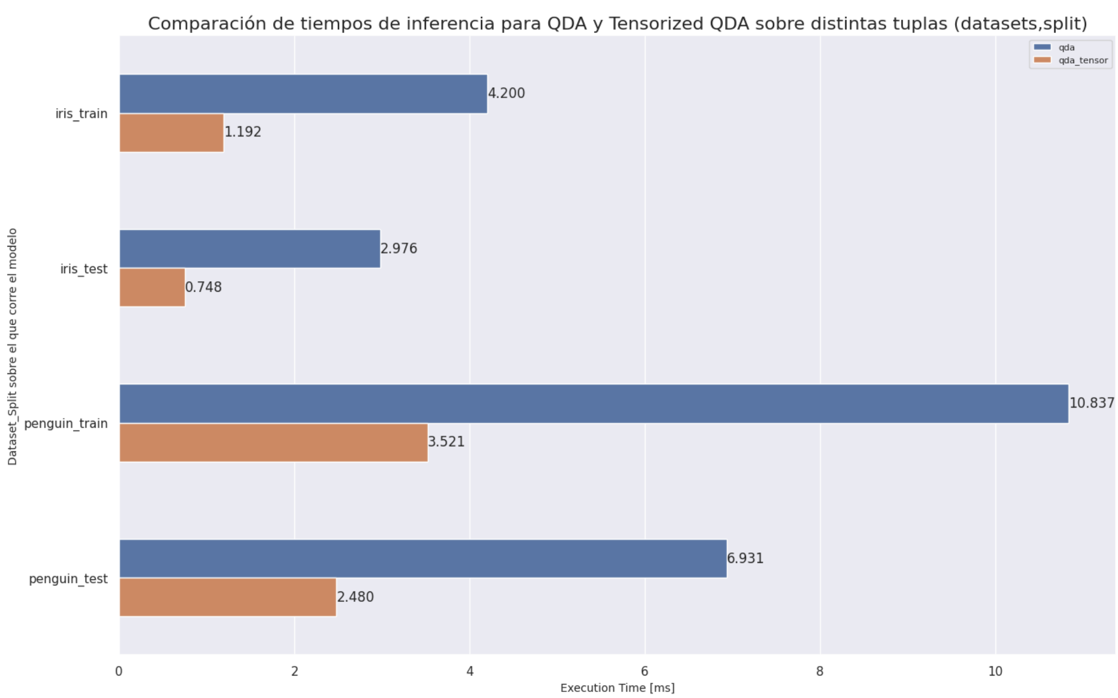
[52]:
```
df.head()
```

[52]:
```
   id                   timestamp model_name   dataset_name  seed     error  \
0   1 2024-10-10 23:16:14.630317        qda     iris_train  6543  0.033333
1   2 2024-10-10 23:16:18.312860        qda      iris_test  6543  0.000000
2  11 2024-10-10 23:16:49.046225        qda  penguin_train  6543  0.009756
3  12 2024-10-10 23:16:52.906735        qda   penguin_test  6543  0.021898
4  25 2024-10-10 23:18:16.391133        qda     iris_train  4111  0.033333

   accuracy  memory_allocation  execution_time_ms  execution_time_dv_ms  \
0  0.966667           0.005965           6.411409              0.894431
1  1.000000           0.004926           4.485744              0.731992
2  0.990244           0.004096           6.968637              0.751504
3  0.978102           0.003682           4.810729              0.516173
4  0.966667           0.005965           2.933711              0.250066

  comments
0
1
2
3
```

4

[53]:
```
sns.set(rc={'figure.figsize':(16,10)})
ax = sns.barplot(
    x='execution_time_ms',
    y='dataset_name',
    data=df,
    hue='model_name',
    errorbar=None,
    width=.5,
    #capsize=.2,
    #hue_order=df_sorted['model_name'].unique()
)#, palette="vlag")
for container in ax.containers:
    ax.bar_label(container, fmt='%.3f')
plt.title('Comparación de tiempos de inferencia para QDA y Tensorized QDA sobre␣
 ↪distintas tuplas (datasets,split)', fontsize=16)
plt.ylabel('Dataset_Split sobre el que corre el modelo', fontsize = 10)
plt.xlabel('Execution Time [ms]', fontsize = 10)
plt.legend(fontsize=8)
#plt.savefig('img/mem_allocation_algs.png', dpi='figure', bbox_inches='tight')
```

[53]: <matplotlib.legend.Legend at 0x7c55d66bc1f0>



Se observan claras mejorías en términos de tiempo de ejecución para el modelo Tensorizado. Esto se puede explicar ya que el método predict de la clase **BaseBayesianClassifier** tiene un bucle que itera sobre cada observación(m_obs) llamando, a su vez, al método _predict_one que itera sobre cada clase (clase $g \in \mathcal{G}$) para generar su output.

Métodos de **BaseBayesianClassifier**, donde para m_obs = 90 y len($\mathcal{G}$=3, se realizarán 90∗3 = 270 iteraciones.

```
def predict(self, X):
    # this is actually an individual prediction encased in a for-loop
    m_obs = X.shape[1]
    y_hat = np.empty(m_obs, dtype=self.encoder.fmt)

    # m_obs = 90 (in iris test for instance)
    for i in range(m_obs):
      encoded_y_hat_i = self._predict_one(X[:,i].reshape(-1,1)) # for each row predict_log_cond
      y_hat[i] = self.encoder.names[encoded_y_hat_i]

    # return prediction as a row vector (matching y)
    return y_hat.reshape(1,-1)

  def _predict_one(self, x):
    # calculate all log posteriori probabilities (actually, +C)
    log_posteriori = [ log_a_priori_i + self._predict_log_conditional(x, idx) for idx, log_a_pr
                  in enumerate(self.log_a_priori) ] # iter for each class

    # return the class that has maximum a posteriori probability
    return np.argmax(log_posteriori)
```

Método **_predict_one** de la clase TensorizedQDA(BaseBayesianClassifier). Aqui se observa que se sobrecarga el método **_predict_one** evitando iterar sobre cada clase cada vez que se lo llama desde **predict**, esto hace que para el mismo ejemplo anterior de m_obs = 90 y len($\mathcal{G}$=3, se realizarán 90 iteraciones. Mientras mas clases haya, mas diferencia en la performance se observará.

```
def _predict_one(self, x):
    # return the class that has maximum a posteriori probability
    return np.argmax(self.log_a_priori + self._predict_log_conditionals(x))
```

1.3 Consigna 2: Optimización Matemática

1.3.1 2.1: QDA

Sugerencia: considerar combinaciones adecuadas de `transpose`, `reshape` y, ocasionalmente, `flatten`. Explorar la dimensionalidad de cada elemento antes de implementar las clases.

Debido a la forma cuadrática de QDA, no se puede predecir para n observaciones en una sola pasada (utilizar $X \in \mathbb{R}^{p \times n}$ en vez de $x \in \mathbb{R}^p$) sin pasar por una matriz de n x n en donde se computan todas las interacciones entre observaciones. Se puede acceder al resultado recuperando sólo la diagonal de dicha matriz, pero resulta ineficiente en tiempo y (especialmente) en memoria. Aún así, es *posible* que el modelo funcione más rápido.

1. Implementar el modelo `FasterQDA` (se recomienda heredarlo de TensorizedQDA) de manera

de eliminar el ciclo for en el método predict.

2. Comparar los tiempos de predicción de `FasterQDA` con `TensorizedQDA` y `QDA`.
3. Mostrar (puede ser con un print) dónde aparece la mencionada matriz de $n \ x \ n$, donde n es la cantidad de observaciones a predecir.
4. Demostrar que

$$diag(A \cdot B) = \sum_{cols} A \odot B^T = np.sum(A \odot B^T, axis = 1)$$

es decir, que se puede "esquivar" la matriz de $n \ x \ n$ usando matrices de $n \ x \ p$.

5. Utilizar la propiedad antes demostrada para reimplementar la predicción del modelo `FasterQDA` de forma eficiente. ¿Hay cambios en los tiempos de predicción?

2.1.1: Implementación de FasterQDA Estas implementaciones se pueden ver en la sección **Clases Base y Modelos -> QDA**

2.1.3: Mostrar donde aparece la matriz de n x n

[54]:
```
qda = FasterQDA(ultra_faster=False)
qda.fit(train_x_iris, train_y_iris)

r = qda.predict(train_x_iris)

# Imprimimos la Matriz de 90x90
print(f"Shape del train_x_iris dataset split: {train_x_iris.shape}") # 4␣
 ↪features and 90 samples
print(f"Shape de matriz de n x n: {qda.get_n_x_n_matrix().shape}")
print(f"Matriz de n x n: {qda.get_n_x_n_matrix()}")
```

```
Shape del train_x_iris dataset split: (4, 90)
Shape de matriz de n x n: (90, 90)
Matriz de n x n: [[ 1.61261328e+02  1.36800065e+01 -3.07037829e+00 …
5.62786248e+01
  -1.11732837e+00  5.79312220e+01]
 [ 1.36800065e+01  1.32357482e+00  1.11003409e-01 …  4.63498125e+00
  -4.67985705e-01  5.07397518e+00]
 [-3.07037829e+00  1.11003409e-01  9.37331622e-01 … -1.16120426e+00
  -6.19100996e-01 -2.87668955e-01]
 …
 [ 5.62786248e+01  4.63498125e+00 -1.16120426e+00 …  2.13600427e+01
   1.28902863e+00  2.33261819e+01]
 [-1.11732837e+00 -4.67985705e-01 -6.19100996e-01 …  1.28902863e+00
   2.63838036e+00  1.64131463e+00]
 [ 5.79312220e+01  5.07397518e+00 -2.87668955e-01 …  2.33261819e+01
   1.64131463e+00  2.77440511e+01]]
```

2.1.4: Desarrollo teórico La distancia de *Mahalanobis* es lo que utiliza QDA para calcular la **verosimilitud**.

$$d_M(x,\mu) = \sqrt{\frac{(x-\mu)^2}{\sigma^2}} \tag{Caso una variable}$$

$$d_M(x,\mu) = \sqrt{(x-\mu)^T\Sigma^{-1}(x-\mu)} \tag{Caso multivariable}$$

Para el caso ideal en el que las features no tengan correlación, obtendriamos una matriz de covarianza diagonal. En este caso, podriamos utilizar una optimización matemática que nos permita reemplazar el computo de productos de matrices de $(n*p)\times(p*p)\times(p*n)$ por operaciones equivalentes elemento a elemento. En el caso que la correlación no sea nula, podríamos igualmente optar por utilizar esta simplificación apelando a un trade-off entre complejidad computacional y performance en predicción ya que de todas formas los elementos fuera de la diagonal serán los menos predominantes de la matriz.

Demostrar que

$$diag(A\cdot B) = \sum_{cols} A\odot B^T = np.sum(A\odot B^T, axis=1)$$

es decir, que se puede "esquivar" la matriz de $n\ x\ n$ usando matrices de $n\ x\ p$. p features y n muestras.

La idea es demostrar que la diagonal de una matriz de n x n resultado del producto de matrices n*p se puede obtener con un producto* element-wise* optimizando la cantidad de operaciones a realizar. Mientras mas grande las dimensiones de la matriz mayor cantidad de operaciones se optimizarán.

En vez de calcular la operacion A @ B, solo se necesitan obtener las contribuciones a los elementos de la diagonal.

$$A\in\mathbb{R}^{n\times p}, B\in\mathbb{R}^{p\times n}, A.B\in\mathbb{R}^{n\times n}$$

Los elementos i,j de la matriz resultado se pueden pensar como la sumatoria de productos de la i-esima fila de A con la j-esima columna de B, dando como resultado la expresión:

$$(A\cdot B)_{ij} = \sum_{k=1}^{p} A_{ik}B_{kj} \tag{1}$$

$$diag(A\cdot B)_i = \sum_{k=1}^{p} A_{ik}B_{ki} \tag{2}$$

Ahora trabajaremos sobre la operación de producto *element-wise* para ver si podemos llegar a una equivalencia.

Si A pertenece a $n\times p$ y B a $p\times n$, la operación *element-wise* requiere misma dimensionalidad, por lo que necesitamos transponer B para tener 2 matrices de nxp y garantizar consistencia de la operación. Además, al transponer B invertimos indices, lo que necesitamos según la expresion (2).

$$(A\circ B^T)_{ik} = A_{ik}B_{ki} \tag{3}$$

Ahora, hacemos la sumatoria en k para cada fila i y llegamos a la misma expresión de la diagonal que tenemos en (2)

$$\mathrm{diag}(A \cdot B)_i = \sum_{k=1}^{p} (A \circ B^T)_{ik} = \sum_{k=1}^{p} A_{ik} B_{ki} \tag{4}$$

$$\mathrm{diag}(A \cdot B) = \mathrm{np.sum}(A \circ B^T, \text{axis} = 1) \tag{5}$$

Veamos un ejemplo:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}, \quad B = \begin{pmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix}$$

El producto de (A) y (B) es:

$$A \cdot B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \cdot \begin{pmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix} = \begin{pmatrix} (1 \cdot 7 + 2 \cdot 10) & (1 \cdot 8 + 2 \cdot 11) & (1 \cdot 9 + 2 \cdot 12) \\ (3 \cdot 7 + 4 \cdot 10) & (3 \cdot 8 + 4 \cdot 11) & (3 \cdot 9 + 4 \cdot 12) \\ (5 \cdot 7 + 6 \cdot 10) & (5 \cdot 8 + 6 \cdot 11) & (5 \cdot 9 + 6 \cdot 12) \end{pmatrix} = \begin{pmatrix} 27 & 30 & 33 \\ 61 & 68 & 75 \\ 95 & 106 & 117 \end{pmatrix}$$

Donde vemos que la diagonal es:

$$\mathrm{diag}(A \cdot B) = \begin{pmatrix} 27 \\ 68 \\ 117 \end{pmatrix}$$

Ahora veamos la propiedad demostrada:

$$B^T = \begin{pmatrix} 7 & 10 \\ 8 & 11 \\ 9 & 12 \end{pmatrix}$$

Calculamos producto de Hadamard \$ A B^T \$:

$$(A \circ B^T)_{ik} = A_{ik} B_{ki}$$

Obtenemos:

$$A \circ B^T = \begin{pmatrix} 1 \cdot 7 & 2 \cdot 10 \\ 3 \cdot 8 & 4 \cdot 11 \\ 5 \cdot 9 & 6 \cdot 12 \end{pmatrix} = \begin{pmatrix} 7 & 20 \\ 24 & 44 \\ 45 & 72 \end{pmatrix}$$

Sumando a lo largo de las columnas (axis=1) obtenemos como resultado:

$$7 + 20 = 27$$

$$24 + 44 = 68$$

$$45 + 72 = 117$$

$$\text{Resultado} = \begin{pmatrix} 27 \\ 68 \\ 117 \end{pmatrix}$$

2.1.2/5: Implementación de FasterQDA y ejecuciones con comparaciones Estas implementaciones se pueden ver en la sección **Clases Base y Modelos -> QDA**

[55]:
```
qda_optimized = FasterQDA()

print("### DATASET IRIS ###")
qda_optimized.fit(train_x_iris, train_y_iris)
train_acc = accuracy(train_y_iris, qda_optimized.predict(train_x_iris))
test_acc = accuracy(test_y_iris, qda_optimized.predict(test_x_iris))
print(f"Train (apparent) error is {1-train_acc:.4f} while test error is␣
 ↪{1-test_acc:.4f}")

# Model: QDA_OPTIMIZED - Dataset: IRIS - SPLIT: Train - A PRIORI: None
model_name="qda_optimized_ultra"
dataset_name="iris_train"
seed=RNG_SEED
number=1

silence = dispatcher(perdict_method=qda_optimized.predict,
           dataset_x=train_x_iris,
           dataset_y=train_y_iris,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )

# Model: QDA_OPTIMIZED - Dataset: IRIS - SPLIT: Test - A PRIORI: None
dataset_name="iris_test"

silence = dispatcher(perdict_method=qda_optimized.predict,
           dataset_x=test_x_iris,
           dataset_y=test_y_iris,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
```

```
            number=number,
            repeat=repeat
         )

print("### DATASET PENGUIN ###")
qda_optimized.fit(train_x_penguin, train_y_penguin)
train_acc = accuracy(train_y_penguin, qda_optimized.predict(train_x_penguin))
test_acc = accuracy(test_y_penguin, qda_optimized.predict(test_x_penguin))
print(f"Train (apparent) error is {1-train_acc:.4f} while test error is␣
 ↪{1-test_acc:.4f}")

# Model: QDA_OPTIMIZED - Dataset: PENGUIN - SPLIT: Train - A PRIORI: None
model_name="qda_optimized_ultra"
dataset_name="penguin_train"
seed=RNG_SEED
number=1

silence = dispatcher(perdict_method=qda_optimized.predict,
            dataset_x=train_x_penguin,
            dataset_y=train_y_penguin,
            model_name=model_name,
            dataset_name=dataset_name,
            seed=seed,
            number=number,
            repeat=repeat
         )

# Model: QDA_OPTIMIZED - Dataset: PENGUIN - SPLIT: Test - A PRIORI: None
dataset_name="penguin_test"

silence = dispatcher(perdict_method=qda_optimized.predict,
            dataset_x=test_x_penguin,
            dataset_y=test_y_penguin,
            model_name=model_name,
            dataset_name=dataset_name,
            seed=seed,
            number=number,
            repeat=repeat
         )
```

```
### DATASET IRIS ###
Train (apparent) error is 0.0222 while test error is 0.0000
Decorator parameters: qda_optimized_ultra, iris_train, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_optimized_ultra'  timestamp:
'2024-10-10 23:19:33.186804'>' into DB.
Add successful.
Closing all connections…
```

```
Decorator parameters: qda_optimized_ultra, iris_test, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_optimized_ultra'  timestamp:
'2024-10-10 23:19:34.171125'>' into DB.
Add successful.
Closing all connections…
### DATASET PENGUIN ###
Train (apparent) error is 0.0098 while test error is 0.0219
Decorator parameters: qda_optimized_ultra, penguin_train, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_optimized_ultra'  timestamp:
'2024-10-10 23:19:35.153615'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: qda_optimized_ultra, penguin_test, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_optimized_ultra'  timestamp:
'2024-10-10 23:19:36.137815'>' into DB.
Add successful.
Closing all connections…
```

[56]:

```python
qda_optimized = FasterQDA(ultra_faster=False)

print("### DATASET IRIS ###")
qda_optimized.fit(train_x_iris, train_y_iris)
train_acc = accuracy(train_y_iris, qda_optimized.predict(train_x_iris))
test_acc = accuracy(test_y_iris, qda_optimized.predict(test_x_iris))
print(f"Train (apparent) error is {1-train_acc:.4f} while test error is␣
 ↪{1-test_acc:.4f}")

# Model: QDA_OPTIMIZED - Dataset: IRIS - SPLIT: Train - A PRIORI: None
model_name="qda_optimized"
dataset_name="iris_train"
seed=RNG_SEED
number=1

silence = dispatcher(perdict_method=qda_optimized.predict,
           dataset_x=train_x_iris,
           dataset_y=train_y_iris,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )

# Model: QDA_OPTIMIZED - Dataset: IRIS - SPLIT: Test - A PRIORI: None
```

```
dataset_name="iris_test"

silence = dispatcher(perdict_method=qda_optimized.predict,
           dataset_x=test_x_iris,
           dataset_y=test_y_iris,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )

print("### DATASET PENGUIN ###")
qda_optimized.fit(train_x_penguin, train_y_penguin)
train_acc = accuracy(train_y_penguin, qda_optimized.predict(train_x_penguin))
test_acc = accuracy(test_y_penguin, qda_optimized.predict(test_x_penguin))
print(f"Train (apparent) error is {1-train_acc:.4f} while test error is␣
 ↪{1-test_acc:.4f}")

# Model: QDA_OPTIMIZED - Dataset: PENGUIN - SPLIT: Train - A PRIORI: None
model_name="qda_optimized"
dataset_name="penguin_train"
seed=RNG_SEED
number=1

silence = dispatcher(perdict_method=qda_optimized.predict,
           dataset_x=train_x_penguin,
           dataset_y=train_y_penguin,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )

# Model: QDA_OPTIMIZED - Dataset: PENGUIN - SPLIT: Test - A PRIORI: None
dataset_name="penguin_test"

silence = dispatcher(perdict_method=qda_optimized.predict,
           dataset_x=test_x_penguin,
           dataset_y=test_y_penguin,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )
```

```
### DATASET IRIS ###
Train (apparent) error is 0.0222 while test error is 0.0000
Decorator parameters: qda_optimized, iris_train, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_optimized'  timestamp: '2024-10-10
23:19:37.129466'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: qda_optimized, iris_test, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_optimized'  timestamp: '2024-10-10
23:19:38.097664'>' into DB.
Add successful.
Closing all connections…
### DATASET PENGUIN ###
Train (apparent) error is 0.0098 while test error is 0.0219
Decorator parameters: qda_optimized, penguin_train, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_optimized'  timestamp: '2024-10-10
23:19:39.096021'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: qda_optimized, penguin_test, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'qda_optimized'  timestamp: '2024-10-10
23:19:40.077645'>' into DB.
Add successful.
Closing all connections…
```

[57]:
```
query = text(f'SELECT * FROM {metrics_table} WHERE model_name LIKE :pattern and␣
↪model_name NOT LIKE :pattern3 and seed=:pattern2')
df = pd.read_sql_query(query, con=engine, params={'pattern': '%qda%',␣
↪'pattern2': '6543', 'pattern3': '%_a_priori_%'})
```

[58]:
```
df.head()
```

[58]:
```
   id                   timestamp  model_name   dataset_name  seed     error  \
0   1 2024-10-10 23:16:14.630317         qda     iris_train  6543  0.033333
1   2 2024-10-10 23:16:18.312860         qda      iris_test  6543  0.000000
2  11 2024-10-10 23:16:49.046225         qda  penguin_train  6543  0.009756
3  12 2024-10-10 23:16:52.906735         qda   penguin_test  6543  0.021898
4  41 2024-10-10 23:19:24.030337  qda_tensor     iris_train  6543  0.033333

   accuracy  memory_allocation  execution_time_ms  execution_time_dv_ms  \
0  0.966667           0.005965           6.411409              0.894431
1  1.000000           0.004926           4.485744              0.731992
2  0.990244           0.004096           6.968637              0.751504
```

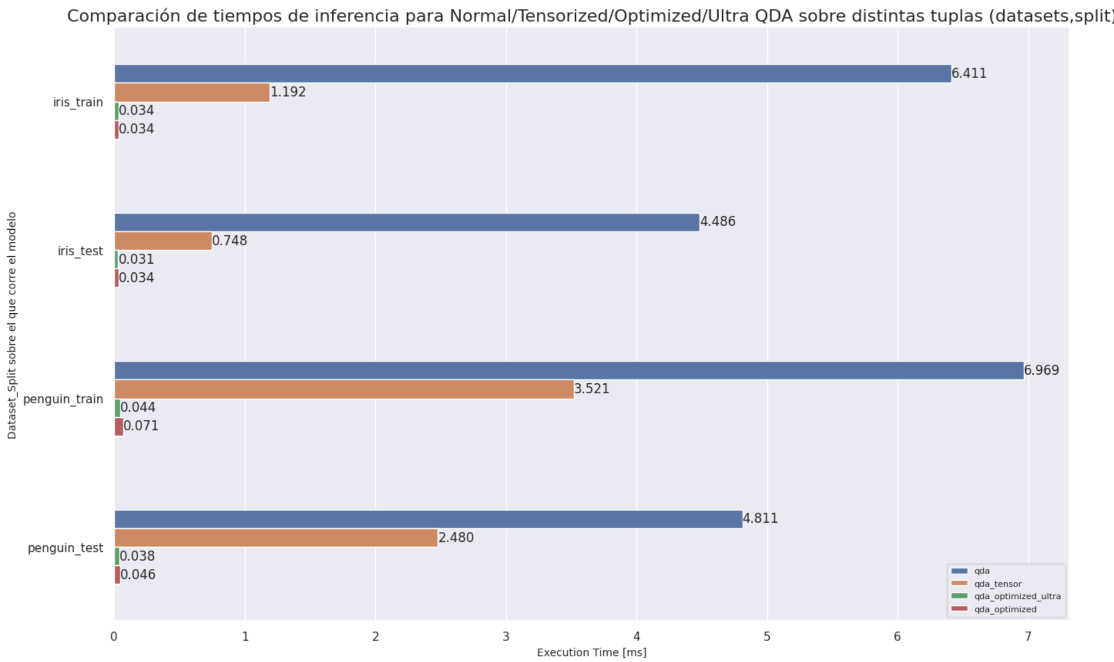
```
3  0.978102           0.003682           4.810729             0.516173
4  0.966667           0.005623           1.192369             0.148245

  comments
0
1
2
3
4
```

[59]:

```python
sns.set(rc={'figure.figsize':(16,10)})
ax = sns.barplot(
    x='execution_time_ms',
    y='dataset_name',
    data=df,
    hue='model_name',
    errorbar=None,
    width=.5,
    #capsize=.2,
    #hue_order=df_sorted['model_name'].unique()
)#, palette="vlag")
for container in ax.containers:
    ax.bar_label(container, fmt='%.3f')
plt.title('Comparación de tiempos de inferencia para Normal/Tensorized/
↪Optimized/Ultra QDA sobre distintas tuplas (datasets,split)', fontsize=16)
plt.ylabel('Dataset_Split sobre el que corre el modelo', fontsize = 10)
plt.xlabel('Execution Time [ms]', fontsize = 10)
plt.legend(fontsize=8)
#plt.savefig('img/mem_allocation_algs.png', dpi='figure', bbox_inches='tight')
```

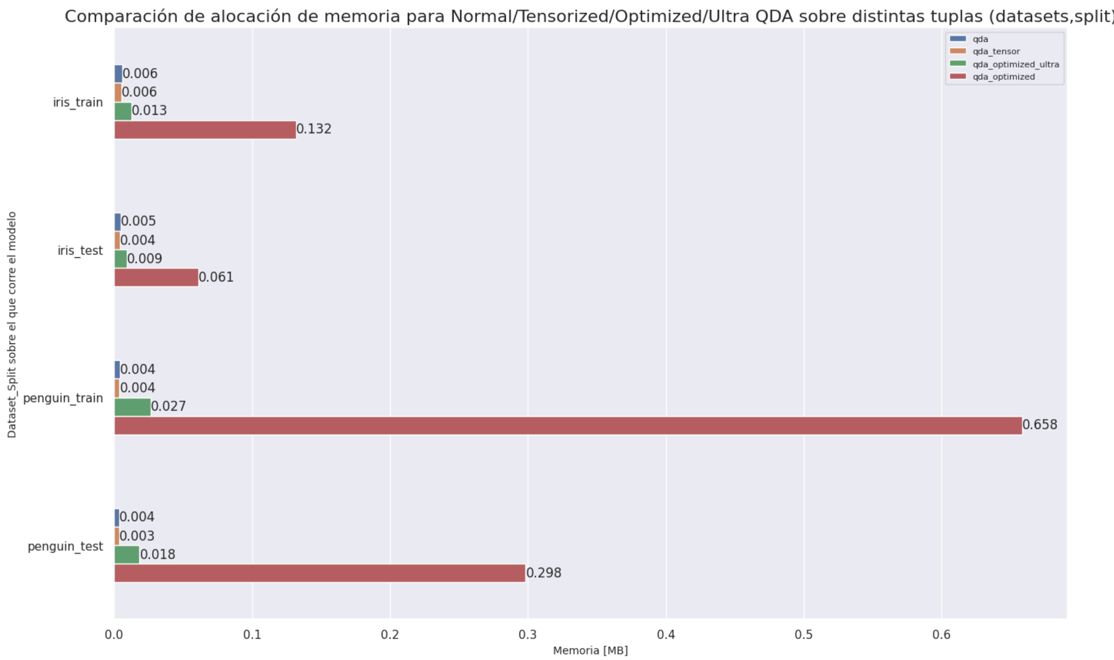
[59]:

```
<matplotlib.legend.Legend at 0x7c55d66789a0>
```



[60]:
```python
sns.set(rc={'figure.figsize':(16,10)})
ax = sns.barplot(
    x='memory_allocation',
    y='dataset_name',
    data=df,
    hue='model_name',
    errorbar=None,
    width=.5,
    #capsize=.2,
    #hue_order=df_sorted['model_name'].unique()
)#, palette="vlag")
for container in ax.containers:
    ax.bar_label(container, fmt='%.3f')
plt.title('Comparación de alocación de memoria para Normal/Tensorized/Optimized/
 ↪Ultra QDA sobre distintas tuplas (datasets,split)', fontsize=16)
plt.ylabel('Dataset_Split sobre el que corre el modelo', fontsize = 10)
plt.xlabel('Memoria [MB]', fontsize = 10)
plt.legend(fontsize=8)
#plt.savefig('img/mem_allocation_algs.png', dpi='figure', bbox_inches='tight')
```

[60]: <matplotlib.legend.Legend at 0x7c55d71e1060>



En este caso podemos ver que los tiempos de ejecución de los modelos siguen el orden siguiente:

$$qda > qda\_tensorized > qda\_optimized > qda\_optimized\_ultra$$

Para los primeros dos casos ya se ha hecho un análisis en secciones previas de este trabajo. Para los casos de qda_optimized y ultra, estos resultados tienen sentido ya que precisamente se simplifica la obtención de la diagonal, en la optimización estándar se realizan los productos de matrices completos mientras que en el optimized_ultra se utiliza la aproximación que utiliza el producto **element-wise** de Hadamard para una de las operaciones, realizando menos cuentas.

Si bien el tiempo que se optimiza para datasets pequeños de pocas features es poco, si se puede ver que hay una gran diferencia en términos de utilización de memoria, ya que en el caso de optimización estándar se expande una matriz de $n \times n$ con n muestras mientras que en el segundo de $n \times p$. Y esto a su vez sucede por cada clase.

1.3.2 2.2: LDA

1. "Tensorizar" el modelo LDA y comparar sus tiempos de predicción con el modelo antes implementado. *Notar que, en modo tensorizado, se puede directamente precomputar* $\mu^T \cdot \Sigma^{-1} \in \mathbb{R}^{k \times 1 \times p}$ *y guardar eso en vez de* Σ^{-1}.
2. LDA no sufre del problema antes descrito de QDA debido a que no computa productos internos, por lo que no tiene un verdadero costo extra en memoria predecir "en batch". Implementar el modelo `FasterLDA` y comparar sus tiempos de predicción con las versiones anteriores de LDA.

2.2.1: Tensorizar LDA Estas implementaciones se pueden ver en la sección **Clases Base y Modelos -> LDA**

2.2.2: Implementar FasterLDA y comparar resultados Estas implementaciones se pueden ver en la sección **Clases Base y Modelos -> LDA**

[61]:
```
lda_tensor = TensorizedLDA()

print("### DATASET IRIS ###")
lda_tensor.fit(train_x_iris, train_y_iris)
train_acc = accuracy(train_y_iris, lda_tensor.predict(train_x_iris))
test_acc = accuracy(test_y_iris, lda_tensor.predict(test_x_iris))
print(f"Train (apparent) error is {1-train_acc:.4f} while test error is␣
 ↪{1-test_acc:.4f}")

# Model: LDA_TENSOR - Dataset: IRIS - SPLIT: Train - A PRIORI: None
model_name="lda_tensor"
dataset_name="iris_train"
seed=RNG_SEED
number=1

silence = dispatcher(perdict_method=lda_tensor.predict,
           dataset_x=train_x_iris,
           dataset_y=train_y_iris,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )

# Model: LDA_TENSOR - Dataset: IRIS - SPLIT: Test - A PRIORI: None
dataset_name="iris_test"

silence = dispatcher(perdict_method=lda_tensor.predict,
           dataset_x=test_x_iris,
           dataset_y=test_y_iris,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )

print("### DATASET PENGUIN ###")
lda_tensor.fit(train_x_penguin, train_y_penguin)
train_acc = accuracy(train_y_penguin, lda_tensor.predict(train_x_penguin))
test_acc = accuracy(test_y_penguin, lda_tensor.predict(test_x_penguin))
print(f"Train (apparent) error is {1-train_acc:.4f} while test error is␣
 ↪{1-test_acc:.4f}")
```

```
# Model: LDA_TENSOR - Dataset: PENGUIN - SPLIT: Train - A PRIORI: None
model_name="lda_tensor"
dataset_name="penguin_train"
seed=RNG_SEED
number=1

silence = dispatcher(perdict_method=lda_tensor.predict,
           dataset_x=train_x_penguin,
           dataset_y=train_y_penguin,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )

# Model: LDA_TENSOR - Dataset: PENGUIN - SPLIT: Test - A PRIORI: None
dataset_name="penguin_test"

silence = dispatcher(perdict_method=lda_tensor.predict,
           dataset_x=test_x_penguin,
           dataset_y=test_y_penguin,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )
```

```
### DATASET IRIS ###
Train (apparent) error is 0.0333 while test error is 0.0000
Decorator parameters: lda_tensor, iris_train, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'lda_tensor'  timestamp: '2024-10-10
23:19:42.940719'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: lda_tensor, iris_test, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'lda_tensor'  timestamp: '2024-10-10
23:19:44.183908'>' into DB.
Add successful.
Closing all connections…
### DATASET PENGUIN ###
Train (apparent) error is 0.0098 while test error is 0.0146
Decorator parameters: lda_tensor, penguin_train, 6543
Initializing DatabaseService instance
```

```
Adding '<Metrics id: 'None' model_name: 'lda_tensor'  timestamp: '2024-10-10
23:19:46.046088'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: lda_tensor, penguin_test, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'lda_tensor'  timestamp: '2024-10-10
23:19:47.595259'>' into DB.
Add successful.
Closing all connections…
```

[62]:

```
lda_optimized = FasterLDA()

print("### DATASET IRIS ###")
lda_optimized.fit(train_x_iris, train_y_iris)
train_acc = accuracy(train_y_iris, lda_optimized.predict(train_x_iris))
test_acc = accuracy(test_y_iris, lda_optimized.predict(test_x_iris))
print(f"Train (apparent) error is {1-train_acc:.4f} while test error is␣
 ↪{1-test_acc:.4f}")

# Model: LDA_OPTIMIZED - Dataset: IRIS - SPLIT: Train - A PRIORI: None
model_name="lda_optimized_ultra"
dataset_name="iris_train"
seed=RNG_SEED
number=1

silence = dispatcher(perdict_method=lda_optimized.predict,
           dataset_x=train_x_iris,
           dataset_y=train_y_iris,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )

# Model: LDA_OPTIMIZED - Dataset: IRIS - SPLIT: Test - A PRIORI: None
dataset_name="iris_test"

silence = dispatcher(perdict_method=lda_optimized.predict,
           dataset_x=test_x_iris,
           dataset_y=test_y_iris,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
```

```python
        )

print("### DATASET PENGUIN ###")
lda_optimized.fit(train_x_penguin, train_y_penguin)
train_acc = accuracy(train_y_penguin, lda_optimized.predict(train_x_penguin))
test_acc = accuracy(test_y_penguin, lda_optimized.predict(test_x_penguin))
print(f"Train (apparent) error is {1-train_acc:.4f} while test error is␣
 ↪{1-test_acc:.4f}")

# Model: LDA_OPTIMIZED - Dataset: PENGUIN - SPLIT: Train - A PRIORI: None
model_name="lda_optimized_ultra"
dataset_name="penguin_train"
seed=RNG_SEED
number=1

silence = dispatcher(perdict_method=lda_optimized.predict,
           dataset_x=train_x_penguin,
           dataset_y=train_y_penguin,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )

# Model: LDA_OPTIMIZED - Dataset: PENGUIN - SPLIT: Test - A PRIORI: None
dataset_name="penguin_test"

silence = dispatcher(perdict_method=lda_optimized.predict,
           dataset_x=test_x_penguin,
           dataset_y=test_y_penguin,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )
```

```
### DATASET IRIS ###
Train (apparent) error is 0.0333 while test error is 0.0000
Decorator parameters: lda_optimized_ultra, iris_train, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'lda_optimized_ultra'  timestamp:
'2024-10-10 23:19:48.582593'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: lda_optimized_ultra, iris_test, 6543
Initializing DatabaseService instance
```

```
Adding '<Metrics id: 'None' model_name: 'lda_optimized_ultra'  timestamp:
'2024-10-10 23:19:49.570127'>' into DB.
Add successful.
Closing all connections…
### DATASET PENGUIN ###
Train (apparent) error is 0.0098 while test error is 0.0146
Decorator parameters: lda_optimized_ultra, penguin_train, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'lda_optimized_ultra'  timestamp:
'2024-10-10 23:19:50.557855'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: lda_optimized_ultra, penguin_test, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'lda_optimized_ultra'  timestamp:
'2024-10-10 23:19:51.569547'>' into DB.
Add successful.
Closing all connections…
```

[63]:

```python
lda_optimized = FasterLDA(ultra_faster=False)

print("### DATASET IRIS ###")
lda_optimized.fit(train_x_iris, train_y_iris)
train_acc = accuracy(train_y_iris, lda_optimized.predict(train_x_iris))
test_acc = accuracy(test_y_iris, lda_optimized.predict(test_x_iris))
print(f"Train (apparent) error is {1-train_acc:.4f} while test error is␣
 ↪{1-test_acc:.4f}")

# Model: LDA_OPTIMIZED - Dataset: IRIS - SPLIT: Train - A PRIORI: None
model_name="lda_optimized"
dataset_name="iris_train"
seed=RNG_SEED
number=1

silence = dispatcher(perdict_method=lda_optimized.predict,
           dataset_x=train_x_iris,
           dataset_y=train_y_iris,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )

# Model: LDA_OPTIMIZED - Dataset: IRIS - SPLIT: Test - A PRIORI: None
dataset_name="iris_test"
```

```
silence = dispatcher(perdict_method=lda_optimized.predict,
           dataset_x=test_x_iris,
           dataset_y=test_y_iris,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )

print("### DATASET PENGUIN ###")
lda_optimized.fit(train_x_penguin, train_y_penguin)
train_acc = accuracy(train_y_penguin, lda_optimized.predict(train_x_penguin))
test_acc = accuracy(test_y_penguin, lda_optimized.predict(test_x_penguin))
print(f"Train (apparent) error is {1-train_acc:.4f} while test error is␣
 ↪{1-test_acc:.4f}")

# Model: LDA_OPTIMIZED - Dataset: PENGUIN - SPLIT: Train - A PRIORI: None
model_name="lda_optimized"
dataset_name="penguin_train"
seed=RNG_SEED
number=1

silence = dispatcher(perdict_method=lda_optimized.predict,
           dataset_x=train_x_penguin,
           dataset_y=train_y_penguin,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )

# Model: LDA_OPTIMIZED - Dataset: PENGUIN - SPLIT: Test - A PRIORI: None
dataset_name="penguin_test"

silence = dispatcher(perdict_method=lda_optimized.predict,
           dataset_x=test_x_penguin,
           dataset_y=test_y_penguin,
           model_name=model_name,
           dataset_name=dataset_name,
           seed=seed,
           number=number,
           repeat=repeat
        )
```

```
### DATASET IRIS ###
Train (apparent) error is 0.0333 while test error is 0.0000
```

```
Decorator parameters: lda_optimized, iris_train, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'lda_optimized'  timestamp: '2024-10-10
23:19:52.542667'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: lda_optimized, iris_test, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'lda_optimized'  timestamp: '2024-10-10
23:19:53.538043'>' into DB.
Add successful.
Closing all connections…
### DATASET PENGUIN ###
Train (apparent) error is 0.0098 while test error is 0.0146
Decorator parameters: lda_optimized, penguin_train, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'lda_optimized'  timestamp: '2024-10-10
23:19:54.533293'>' into DB.
Add successful.
Closing all connections…
Decorator parameters: lda_optimized, penguin_test, 6543
Initializing DatabaseService instance
Adding '<Metrics id: 'None' model_name: 'lda_optimized'  timestamp: '2024-10-10
23:19:55.529748'>' into DB.
Add successful.
Closing all connections…
```

[64]:
```
query = text(f'SELECT * FROM {metrics_table} WHERE model_name LIKE :pattern and␣
  ↪model_name NOT LIKE :pattern3 and seed=:pattern2')
df = pd.read_sql_query(query, con=engine, params={'pattern': '%lda%',␣
  ↪'pattern2': '6543', 'pattern3': '%_a_priori_%'})
```

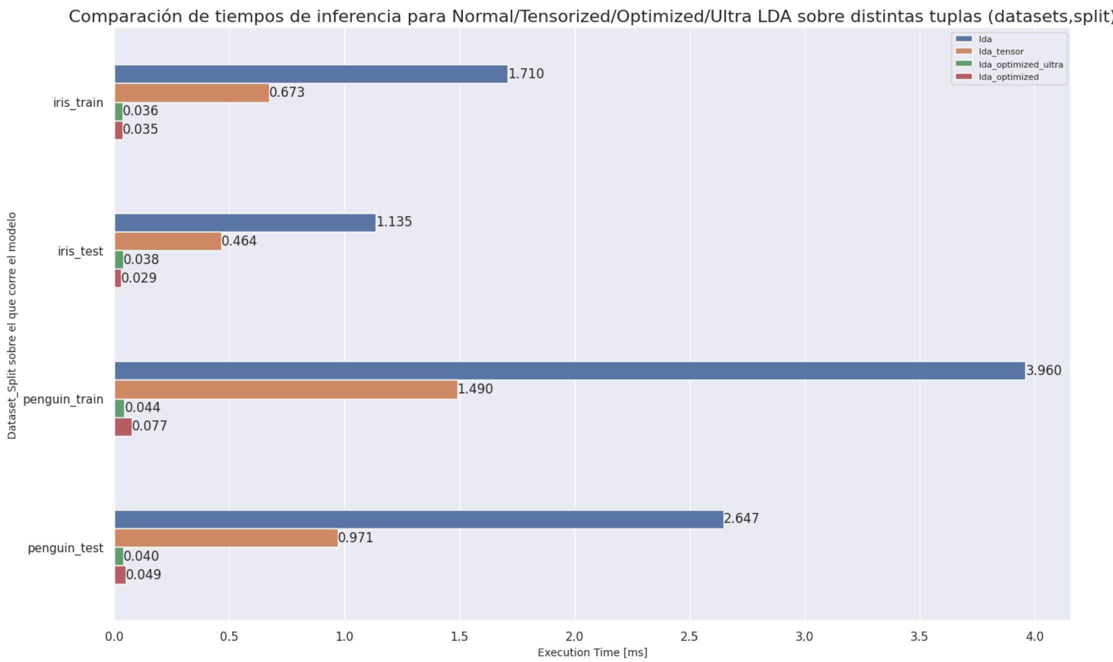
[65]:
```
df.head()
```

[65]:
```
   id                  timestamp  model_name   dataset_name  seed     error  \
0  21 2024-10-10 23:18:04.661040         lda     iris_train  6543  0.033333
1  22 2024-10-10 23:18:06.315344         lda      iris_test  6543  0.000000
2  23 2024-10-10 23:18:09.795230         lda  penguin_train  6543  0.009756
3  24 2024-10-10 23:18:12.352630         lda   penguin_test  6543  0.014599
4  53 2024-10-10 23:19:42.940719  lda_tensor     iris_train  6543  0.033333

   accuracy  memory_allocation  execution_time_ms  execution_time_dv_ms  \
0  0.966667           0.005867           1.709647              0.178541
1  1.000000           0.004723           1.135281              0.068964
2  0.990244           0.003998           3.960398              0.340679
3  0.985401           0.003479           2.646981              0.273339
4  0.966667           0.005806           0.673441              0.062675
```

```
  comments
0
1
2
3
4
```

```
[66]: sns.set(rc={'figure.figsize':(16,10)})
ax = sns.barplot(
    x='execution_time_ms',
    y='dataset_name',
    data=df,
    hue='model_name',
    errorbar=None,
    width=.5,
    #capsize=.2,
    #hue_order=df_sorted['model_name'].unique()
)#, palette="vlag")
for container in ax.containers:
    ax.bar_label(container, fmt='%.3f')
plt.title('Comparación de tiempos de inferencia para Normal/Tensorized/
 ↪Optimized/Ultra LDA sobre distintas tuplas (datasets,split)', fontsize=16)
plt.ylabel('Dataset_Split sobre el que corre el modelo', fontsize = 10)
plt.xlabel('Execution Time [ms]', fontsize = 10)
plt.legend(fontsize=8)
#plt.savefig('img/mem_allocation_algs.png', dpi='figure', bbox_inches='tight')
```

```
[66]: <matplotlib.legend.Legend at 0x7c55d85e6170>
```



[67]:
```
query = text(f'SELECT * FROM {metrics_table} WHERE model_name LIKE :pattern and␣
↪model_name NOT LIKE :pattern3 and seed=:pattern2')
df = pd.read_sql_query(query, con=engine, params={'pattern': '%lda%',␣
↪'pattern2': '6543', 'pattern3': '%_a_priori_%'})
```

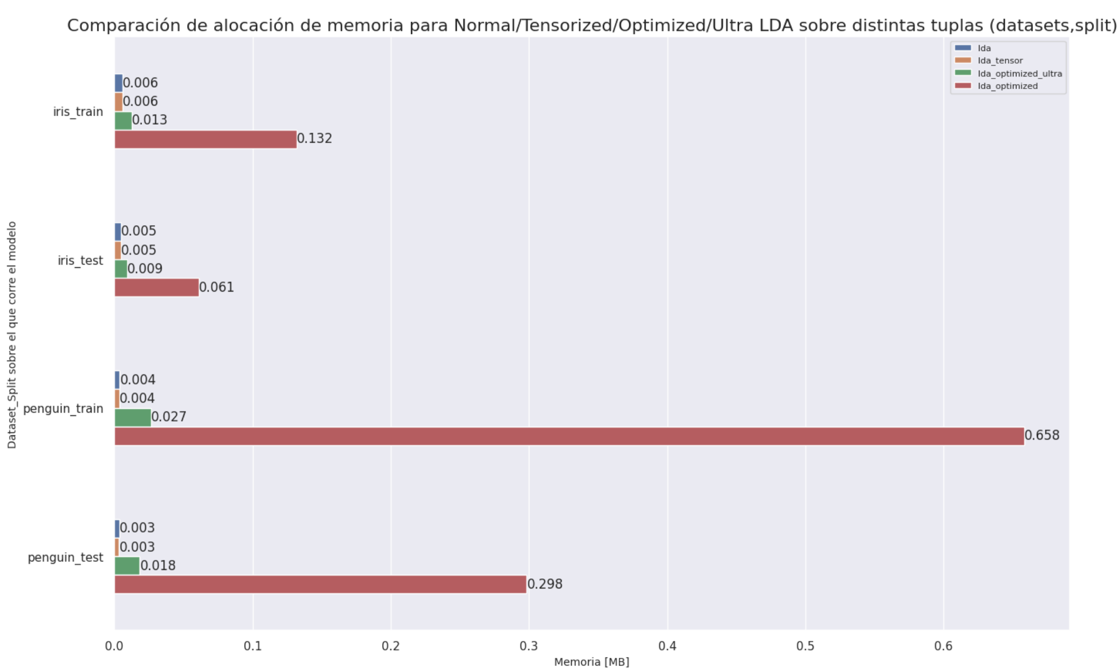
[68]:
```
df.head()
```

[68]:
```
   id                  timestamp  model_name   dataset_name  seed     error  \
0  21 2024-10-10 23:18:04.661040         lda     iris_train  6543  0.033333
1  22 2024-10-10 23:18:06.315344         lda      iris_test  6543  0.000000
2  23 2024-10-10 23:18:09.795230         lda  penguin_train  6543  0.009756
3  24 2024-10-10 23:18:12.352630         lda   penguin_test  6543  0.014599
4  53 2024-10-10 23:19:42.940719  lda_tensor     iris_train  6543  0.033333

   accuracy  memory_allocation  execution_time_ms  execution_time_dv_ms  \
0  0.966667           0.005867           1.709647              0.178541
1  1.000000           0.004723           1.135281              0.068964
2  0.990244           0.003998           3.960398              0.340679
3  0.985401           0.003479           2.646981              0.273339
4  0.966667           0.005806           0.673441              0.062675

  comments
0
1
2
```

3
4

[69]:
```
sns.set(rc={'figure.figsize':(16,10)})
ax = sns.barplot(
    x='memory_allocation',
    y='dataset_name',
    data=df,
    hue='model_name',
    errorbar=None,
    width=.5,
    #capsize=.2,
    #hue_order=df_sorted['model_name'].unique()
)#, palette="vlag")
for container in ax.containers:
    ax.bar_label(container, fmt='%.3f')
plt.title('Comparación de alocación de memoria para Normal/Tensorized/Optimized/
↪Ultra LDA sobre distintas tuplas (datasets,split)', fontsize=16)
plt.ylabel('Dataset_Split sobre el que corre el modelo', fontsize = 10)
plt.xlabel('Memoria [MB]', fontsize = 10)
plt.legend(fontsize=8)
#plt.savefig('img/mem_allocation_algs.png', dpi='figure', bbox_inches='tight')
```

[69]: <matplotlib.legend.Legend at 0x7c55d85e4160>



En este caso podemos ver que los tiempos de ejecución de los modelos siguen el orden siguiente:

$$lda > lda\_tensorized > lda\_optimized > lda\_optimized\_ultra$$

Para los primeros dos casos ya se ha hecho un análisis en secciones previas de este trabajo. Para los casos de lda_optimized y ultra, estos resultados tienen sentido ya que precisamente se simplifica la obtención de la diagonal, en la optimización estándar se realizan los productos de matrices completos mientras que en el optimized_ultra se utiliza la aproximación que utiliza el producto **element-wise** de Hadamard para una de las operaciones, realizando menos cuentas.

Si bien el tiempo que se optimiza para datasets pequeños de pocas features es poco, si se puede ver que hay una gran diferencia en términos de utilización de memoria, ya que en el caso de optimización estándar se expande una matriz de $n \times n$ con n muestras mientras que en el segundo de $n \times p$.

1.4 Consigna 3: Preguntas Teóricas

1. En LDA se menciona que la función a maximizar puede ser, mediante operaciones, convertida en:

$$\log f_j(x) = \mu_j^T \Sigma^{-1}(x - \frac{1}{2}\mu_j) + C'$$

   Mostrar los pasos por los cuales se llega a dicha expresión.

2. Explicar, utilizando las respectivas funciones a maximizar, por qué QDA y LDA son "quadratic" y "linear".

3. La implementación de QDA estima la probabilidad condicional utilizando $0.5 * np.log(det(inv\_cov)) - 0.5 * unbiased\_x.T@inv\_cov@unbiased\_x$ que no es *exactamente* lo descrito en el apartado teórico ¿Cuáles son las diferencias y por qué son expresiones equivalentes?

El espíritu de esta componente práctica es la de establecer un mínimo de trabajo aceptable para su entrega; se invita al alumno a explorar otros aspectos que generen curiosidad, sin sentirse de ninguna manera limitado por la consigna.

1.4.1 3.1

En el caso de LDA se hace una suposición extra, que es $X|_{G=j} \sim \mathcal{N}_p(\mu_j, \Sigma)$, es decir que las poblaciones no sólo siguen una distribución normal sino que son de igual matriz de covarianzas.

$$\boxed{\log f_j(x) = -\frac{1}{2}\log|\Sigma| - \frac{1}{2}(x - \mu_j)^T \Sigma^{-1}(x - \mu_j) + C}$$

Ahora, como $-\frac{1}{2}\log|\Sigma|$ es común a todas las clases se puede incorporar a la constante aditiva, quedando:

$$\log f_j(x) = -\frac{1}{2}(x - \mu_j)^T \Sigma^{-1}(x - \mu_j) + C$$

Realizando distributiva de la transpuesta y aplicando también la propiedad distributiva del producto matricial por izquierda y por derecha, se obtiene:

$$\log f_j(x) = -\frac{1}{2}(x^T\Sigma^{-1} - \mu_j^T\Sigma^{-1})(x-\mu_j) + C'$$

$$\log f_j(x) = -\frac{1}{2}(x^T\Sigma^{-1}x - \mu_j^T\Sigma^{-1}x - x^T\Sigma^{-1}\mu_j + \mu_j^T\Sigma^{-1}\mu_j) + C'$$

Similar a lo ocurrido con $-\frac{1}{2}\log|\Sigma|$, el término $x^T\Sigma^{-1}x$ es común a todas las clases por depender únicamente de la inversa de la matriz de covarianza y de la entrada. Es por este motivo que puede también incorporarse a la constante aditiva:

$$\log f_j(x) = -\frac{1}{2}(-\mu_j^T\Sigma^{-1}x - x^T\Sigma^{-1}\mu_j + \mu_j^T\Sigma^{-1}\mu_j) + C' \tag{A}$$

Para los términos $-\mu_j^T\Sigma^{-1}x$ y $x^T\Sigma^{-1}\mu_j$ se considera la siguiente demostración que los relaciona. Para esto se parte de un ejemplo utilizando matrices genéricas y variables simbólicas. Se utilizarán matrices de tamaño (2×2), aunque este razonamiento es extrapolable a matrices de orden $(n\times n)$.

Se tiene:

- Un vector x de dimensión (2×1):

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

- Un vector μ_j de dimensión (2×1):

$$\mu_j = \begin{pmatrix} \mu_{j1} \\ \mu_{j2} \end{pmatrix}$$

- Una matriz Σ^{-1} de dimensión (2×2), que es simétrica (Como también lo son las matrices de covarianzas en el modelo LDA):

$$\Sigma^{-1} = \begin{pmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{pmatrix}$$

Se calculan ambos productos $x^T\Sigma^{-1}\mu_j$ y $\mu_j^T\Sigma^{-1}x$.

- $x^T\Sigma^{-1}\mu_j$

$$x^T\Sigma^{-1} = \begin{pmatrix} x_1a_{11} + x_2a_{12} & x_1a_{12} + x_2a_{22} \end{pmatrix}$$

$$x^T\Sigma^{-1}\mu_j = \begin{pmatrix} x_1a_{11} + x_2a_{12} & x_1a_{12} + x_2a_{22} \end{pmatrix}\begin{pmatrix} \mu_{j1} \\ \mu_{j2} \end{pmatrix}$$

$$x^T\Sigma^{-1}\mu_j = (x_1a_{11} + x_2a_{12})\mu_{j1} + (x_1a_{12} + x_2a_{22})\mu_{j2}$$

$$x^T\Sigma^{-1}\mu_j = x_1a_{11}\mu_{j1} + x_2a_{12}\mu_{j1} + x_1a_{12}\mu_{j2} + x_2a_{22}\mu_{j2}$$

- $\mu_j^T \Sigma^{-1} x$

$$\mu_j^T \Sigma^{-1} = \begin{pmatrix} \mu_{j1} & \mu_{j2} \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{pmatrix}$$

$$\mu_j^T \Sigma^{-1} = \begin{pmatrix} \mu_{j1} a_{11} + \mu_{j2} a_{12} & \mu_{j1} a_{12} + \mu_{j2} a_{22} \end{pmatrix}$$

$$\mu_j^T \Sigma^{-1} x = \begin{pmatrix} \mu_{j1} a_{11} + \mu_{j2} a_{12} & \mu_{j1} a_{12} + \mu_{j2} a_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

$$\mu_j^T \Sigma^{-1} x = (\mu_{j1} a_{11} + \mu_{j2} a_{12}) x_1 + (\mu_{j1} a_{12} + \mu_{j2} a_{22}) x_2$$

$$\mu_j^T \Sigma^{-1} x = \mu_{j1} a_{11} x_1 + \mu_{j2} a_{12} x_1 + \mu_{j1} a_{12} x_2 + \mu_{j2} a_{22} x_2$$

Se comparan las dos expresiones obtenidas:

$$x^T \Sigma^{-1} \mu_j = x_1 a_{11} \mu_{j1} + x_2 a_{12} \mu_{j1} + x_1 a_{12} \mu_{j2} + x_2 a_{22} \mu_{j2}$$

$$\mu_j^T \Sigma^{-1} x = \mu_{j1} a_{11} x_1 + \mu_{j2} a_{12} x_1 + \mu_{j1} a_{12} x_2 + \mu_{j2} a_{22} x_2$$

Observamos que las dos expresiones son **idénticas**. Esto prueba que:

$$x^T \Sigma^{-1} \mu_j = \mu_j^T \Sigma^{-1} x$$

Con lo obtenido previamente puede reescribirse la expresión (A) quedando:

$$\log f_j(x) = -\frac{1}{2}(-2\mu_j^T \Sigma^{-1} x + \mu_j^T \Sigma^{-1} \mu_j) + C'$$

Se saca factor común $\mu_j^T \Sigma^{-1}$, quedando:

$$\log f_j(x) = -\frac{1}{2}\mu_j^T \Sigma^{-1}(-2x + \mu_j) + C'$$

Finalmente, se distribuye el $-\frac{1}{2}$ quedando:

$$\boxed{\log f_j(x) = \mu_j^T \Sigma^{-1}(x - \frac{1}{2}\mu_j) + C'}$$

1.4.2 3.2

QDA: Como las matrices de covarianzas (Σ_j) son distintas para cada clase, se conserva el término cuadrático ($(x-\mu_j)^T\Sigma_j^{-1}(x-\mu_j)$), lo que hace que la función discriminante sea cuadrática en (x). Por eso se llama *quadratic*.

LDA: Al suponer que todas las clases tienen la misma matriz de covarianzas (Σ), los términos cuadráticos que dependen de (x) se cancelan o se ignoran, y el discriminante resultante es lineal en (x). Por eso se llama *linear*, como se observa en las expresiones (1) con una feature y (2) con n features:

$$\delta_k(x) = x \cdot \frac{\widehat{\mu}_k}{\hat{\sigma}^2} - \frac{\widehat{\mu}_k^2}{2\hat{\sigma}^2} + \log(\hat{\pi}_k) \tag{1}$$

$$\boxed{\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \log(\hat{\pi}_k)} \tag{2}$$

1.4.3 3.3

$$\log f_j(x) = \frac{1}{2}\log|\Sigma_j^{-1}| - \frac{1}{2}(x-\mu_j)^T\Sigma_j^{-1}(x-\mu_j) \tag{Equivalencia implementada}$$

$$\log f_j(x) = -\frac{1}{2}\log|\Sigma_j| - \frac{1}{2}(x-\mu_j)^T\Sigma_j^{-1}(x-\mu_j) + C \tag{Teórica}$$

Usando las siguientes propiedades del determinante y del logaritmo vemos que:

$$\det(\Sigma^{-1}) = \frac{1}{\det(\Sigma)} \tag{1}$$

$$\log\left(\frac{1}{a}\right) = -\log(a) \tag{2}$$

Vemos que de (1)

$$\log(\det(\Sigma^{-1})) = \log\left(\frac{1}{\det(\Sigma)}\right)$$

Aplicando (2) a ambos lados

$$\log\left(\frac{1}{\det(\Sigma)}\right) = -\log(\det(\Sigma))$$

Por lo tanto

$$\therefore \boxed{\log(\det(\Sigma^{-1})) = -\log(\det(\Sigma))}$$

1.5 Consigna 4: Ejercicio Teórico

Sea una red neuronal de dos capas, la primera de 3 neuronas y la segunda de 1 con los parámetros inicializados con los siguientes valores:

$$w^{(1)} = \begin{pmatrix} 0.1 & -0.5 \\ -0.3 & -0.9 \\ 0.8 & 0.02 \end{pmatrix}, b^{(1)} = \begin{pmatrix} 0.1 \\ 0.5 \\ 0.8 \end{pmatrix}, w^{(2)} = \begin{pmatrix} -0.4 & 0.2 & -0.5 \end{pmatrix}, b^{(2)} = 0.7$$

y donde cada capa calcula su salida vía

$$y^{(i)} = \sigma(w^{(i)} \cdot x^{(i)} + b^{(i)})$$

donde $\sigma(z) = \frac{1}{1+e^{-z}}$ es la función sigmoidea .

Dada la observación $x = \begin{pmatrix} 1.8 \\ -3.4 \end{pmatrix}$, $y = 5$ y la función de costo $J(\theta) = \frac{1}{2}(\hat{y}_\theta - y)^2$, calcular las derivadas de J respecto de cada parámetro $w^{(1)}$, $w^{(2)}$, $b^{(1)}$, $b^{(2)}$.

Nota: Con una sigmoidea a la salida jamás va a poder estimar el 5 "pedido", pero eso no afecta al mecanismo de backpropagation!

La resolución de este ejercicio se adjunta en un PDF aparte en el campus.

1.5.1 Resolución en código y comprobación de resultados

Dependencias y clases base

[70]:
```python
import numpy as np
```

[71]:
```python
class Layer(object):
  def __init__(self, n_in, n_out, non_linearity_class, optimizer_factory, rng, w_init=None, b_init=None):
    self.activation = non_linearity_class()
    self.optim = optimizer_factory()
    #self.w = rng.standard_normal(size=(n_out, n_in))  * 0.1 # W shape is (n_out,n_in)
    #self.b = rng.uniform(size=(n_out, 1))                   # b shape is (n_out, 1)
    # Valores de peso y bias precalculados
    self.w = w_init if w_init is not None else rng.standard_normal(size=(n_out, n_in)) * 0.1
    self.b = b_init if b_init is not None else rng.uniform(size=(n_out, 1))
    self.last_output = None
    self.last_input = None

  def forward(self, X):
    self.last_input = X
    z = self.w @ X + self.b
    self.last_output = self.activation.f(z)
    print("--------------------------")
```

```
    print("Forward por capa z:")
    print(z)
    print("Despues de la f de activación:")
    print(self.last_output)
    print("--------------------------")
    return self.last_output

  def backwards(self, dY):
    dz = dY * self.activation.df()
    dW = dz @ self.last_input.T
    db = np.sum(dz, axis=1, keepdims=True)
    dX = self.w.T @ dz
    self.w, self.b = self.optim.update(self.w, self.b, dW, db)

    # Mostrar derivadas parciales
    print("--------------------------")
    print("Derivadas parciales respecto a W:")
    print(dW)
    print("Derivadas parciales respecto a b:")
    print("--------------------------")
    print(db)
    return dX
```

[72]:
```
class MLP(object):
  def __init__(self, dims, optimizer_factory, non_linearities, input_dim,␣
↪rng_seed = None, precalc_weights=None, precalc_biases=None):
    # check lengths
    if len(dims) != len(non_linearities):
      raise ValueError("dims' and Non_linearities' lengths do not match")
    # initialize RNG
    rng = np.random.default_rng(rng_seed)
    # construct a list of Layers with matching dimension and non-linear␣
↪activation function
    in_dims = [input_dim] + dims[:-1]
    #self.layers = [Layer(n_in, n_out, non_linearity, optimizer_factory, rng)
    #               for n_in,n_out,non_linearity in zip(in_dims,dims,␣
↪non_linearities)]
    self.layers = [Layer(n_in, n_out, non_linearity, optimizer_factory, rng,␣
↪w_init, b_init)
                   for n_in, n_out, non_linearity, w_init, b_init
                   in zip(in_dims, dims, non_linearities, precalc_weights,␣
↪precalc_biases)]


  def predict(self, X):
    # X can be interpreted as the output of a previous layer
    prediction = X
```

```
    # sequentially apply forward pass
    for layer in self.layers:
      prediction = layer.forward(prediction)
    return prediction

  def update(self, cost_gradient):
    # cost gradient is the cost derivative wrt last layer
    dY = cost_gradient
    # sequentially apply backwards update, in reversed order
    for layer in reversed(self.layers):
      dY = layer.backwards(dY)

  def __repr__(self):
    # super hardcoded
    return "MLP with layer sizes: "+ "-".join(str(layer.b.shape[0]) for layer␣
 ↪in self.layers)
```

[73]:
```
class Optimizer(object):
  def update(self, W, b, dW, db):
    raise NotImplementedError("optimizer update rule not implemented")

class VGD(Optimizer):
  def __init__(self, learning_rate):
    self.lr = learning_rate

  def update(self, W_old, b_old, dW, db):
    # vanilla GD: theta_t+1 = theta_t - alpha * gradient
    W_new = W_old - self.lr * dW
    b_new = b_old - self.lr * db
    return W_new, b_new

def factory_VGD(lr):
  return lambda : VGD(lr)
```

[74]:
```
class NonLinearity(object):
  def __init__(self):
    self.last_z = None
  def f(self, z):
    raise NotImplementedError("function evaluation not implemented")
  def df(self):
    raise NotImplementedError("function derivative not implemented")

class Sigmoid(NonLinearity):
  def __init__(self):
    super().__init__()
  def sigma():
    return "1 / (1 + np.exp(-z))"
```

```
    def f(self, z):
      self.last_z = z
      return 1 / (1 + np.exp(-z))

    def df(self):
      return np.exp(-self.last_z) / (1 + np.exp(-self.last_z))**2
```

Comprobación de resultados

[75]:
```
lr = 0.001
rng_seed = 6543

# Pesos y sesgos precalculados para cada capa
precalc_weights = [
    np.array([[0.1, -0.5], [-0.3, -0.9], [0.8, 0.02]]),  # Pesos para la␣
 ↪primera capa (3x2)
    np.array([[-0.4, 0.2, -0.5]])                         # Pesos para la␣
 ↪segunda capa (1x3)
]
precalc_biases = [
    np.array([[0.1], [0.5], [0.8]]),  # Bias para la primera capa (3x1)
    np.array([[0.7]])                 # Bias para la segunda capa (1x1)
]

# Configuración del MLP:
dims = [3, 1]  # 3 neuronas en la primera capa, 1 neurona en la segunda
input_dim = 2  # 2 entradas (dimensión de entrada)
optimizer_factory = lambda: VGD(learning_rate=lr) #No se usa porque el modelo␣
 ↪esta pre entrenado
non_linearities = [Sigmoid, Sigmoid]  # Activaciones sigmoides para ambas capas

# Crear la MLP con pesos y sesgos precalculados
mlp = MLP(dims, optimizer_factory, non_linearities, input_dim,␣
 ↪precalc_weights=precalc_weights, precalc_biases=precalc_biases)

# Input de ejemplo (2 características de entrada)
X = np.array([[1.8], [-3.4]])  # Tamaño del input (2x1)

# Predecir con el MLP
print("Forward de entrada a dalida")
output = mlp.predict(X)
print("--------------------------")
print("Salida: ",output)
print("--------------------------")
```

```
Forward de entrada a dalida
--------------------------
```

```
Forward por capa z:
[[1.98 ]
 [3.02 ]
 [2.172]]
Despues de la f de activación:
[[0.87868116]
 [0.95346953]
 [0.89770677]]
--------------------------
--------------------------
Forward por capa z:
[[0.09036805]]
Despues de la f de activación:
[[0.52257665]]
--------------------------
--------------------------
Salida:  [[0.52257665]]
--------------------------
```

[76]:
```
print("Backprop de salida a entrada")
y_true = 5
error = 1/2 * (output - y_true)**2
print("Error cuadratico medio: ", error)
mlp.update(output - y_true)
```

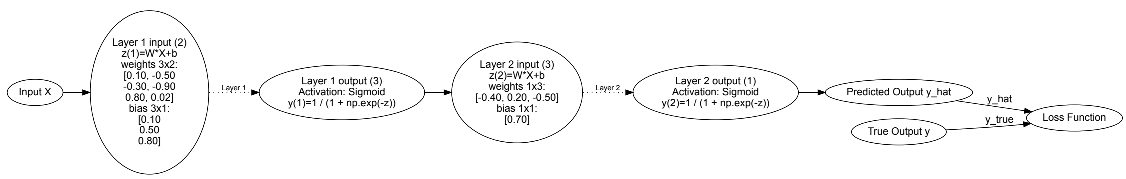
```
Backprop de salida a entrada
Error cuadratico medio:  [[10.02365992]]
--------------------------
Derivadas parciales respecto a W:
[[-0.98155159 -1.0650957  -1.0028046 ]]
Derivadas parciales respecto a b:
--------------------------
[[-1.11707367]]
--------------------------
Derivadas parciales respecto a W:
[[ 0.0857381  -0.16194975]
 [-0.01784139  0.0337004 ]
 [ 0.09232211 -0.1743862 ]]
Derivadas parciales respecto a b:
--------------------------
[[ 0.04763228]
 [-0.00991188]
 [ 0.05129006]]
```

[77]:
```
from graphviz import Digraph
import numpy as np

def matrix_to_text(matrix):
```

```python
    """Convierte una matriz numpy en una cadena de texto legible."""
    text_matrix = "\n["
    for row in matrix:
        text_matrix +=  ", ".join(f"{v:.2f}" for v in row) + "\n"
    text_matrix = text_matrix.rstrip("\n")
    text_matrix += "]"
    return text_matrix

def generate_mlp_graph(mlp):
    dot = Digraph()
    dot.attr(rankdir='LR', fontname="Arial")  # Configurar la dirección del grafo de izquierda a derecha

    # Crear el nodo de entrada X
    dot.node("X", "Input X", fontname="Arial")

    # Iterar a través de las capas de la red
    for i, layer in enumerate(mlp.layers):
        input_size = layer.w.shape[1]
        output_size = layer.w.shape[0]
        activation_name = layer.activation.__class__.__name__

        # Convertir los pesos y sesgos en un texto legible
        weights_text = matrix_to_text(layer.w)
        bias_text = matrix_to_text(layer.b)

        # Crear nodos para la capa actual con matrices en texto legible
        #dot.node(f"Layer {i+1} input", f"Layer {i+1} input ({input_size})")
        dot.node(f"Layer {i+1} input", f"Layer {i+1} input ({input_size})\nz({i+1})=W*X+b\nweights {layer.w.shape[0]}x{layer.w.shape[1]}: {weights_text}\nbias {layer.b.shape[0]}x{layer.b.shape[1]}: {bias_text}", fontname="Arial")
        #dot.node(f"Layer {i+1} output", f"Layer {i+1} output ({output_size})\nActivation: {activation_name}\nWeights {layer.w.shape[0]}x{layer.w.shape[1]}: {weights_text}\nBias {layer.b.shape[0]}x{layer.b.shape[1]}: {bias_text}")
        dot.node(f"Layer {i+1} output", f"Layer {i+1} output ({output_size})\nActivation: {activation_name}\ny({i+1})={Sigmoid.sigma()}", fontname="Arial")

        # Conectar la entrada con la salida de la misma capa
        dot.edge(f"Layer {i+1} input", f"Layer {i+1} output", label=f"Layer {i+1}", fontname="Arial", fontsize="10", style="dotted")

        # Conectar la salida de la capa anterior con la entrada de la actual
        if i > 0:
```

```
            dot.edge(f"Layer {i} output", f"Layer {i+1} input",␣
↪fontname="Arial")
        else:
            # Conectar el nodo de entrada X con la primera capa
            dot.edge("X", f"Layer {i+1} input", fontname="Arial")

    # Crear el nodo de salida y_hat para la predicción
    dot.node("y_hat", "Predicted Output y_hat", fontname="Arial")
    dot.edge(f"Layer {len(mlp.layers)} output", "y_hat", fontname="Arial")

    # Crear el nodo de salida verdadera y
    dot.node("y_true", "True Output y", fontname="Arial")

    # Crear el nodo de la función de pérdida que conecta y_hat y y_true
    dot.node("Loss", "Loss Function", fontname="Arial")
    dot.edge("y_hat", "Loss", label="y_hat", fontname="Arial")
    dot.edge("y_true", "Loss", label="y_true", fontname="Arial")

    return dot
```

[78]:
```
dot = generate_mlp_graph(mlp)
#dot.render('mlp_graph', format='png', view=True)
display(dot)
```



1.6 Anexo Teoría

1.6.1 Definición Clasificador Bayesiano

Sean k poblaciones, $x \in \mathbb{R}^p$ puede pertenecer a cualquiera $g \in \mathcal{G}$ de ellas. Bajo un esquema bayesiano, se define entonces $\pi_j \doteq P(G = j)$ la probabilidad *a priori* de que X pertenezca a la clase j, y se **asume conocida** la distribución condicional de cada observable dado su clase $f_j \doteq f_{X|G=j}$.

De esta manera dicha probabilidad *a posteriori* resulta

$$P(G|_{X=x} = j) = \frac{f_{X|G=j}(x) \cdot p_G(j)}{f_X(x)} \propto f_j(x) \cdot \pi_j$$

La regla de decisión de Bayes es entonces

$$H(x) \doteq \arg\max_{g \in \mathcal{G}} \{P(G|_{X=x} = j)\} = \arg\max_{g \in \mathcal{G}} \{f_j(x) \cdot \pi_j\}$$

es decir, se predice a x como perteneciente a la población j cuya probabilidad a posteriori es máxima.

Ojo, a no desesperar! π_j *no es otra cosa que una constante prefijada, y* f_j *es, en su esencia, un campo escalar de* x *a simplemente evaluar.*

1.6.2 Distribución condicional

Para los clasificadores de discriminante cuadrático y lineal (QDA/LDA) se asume que $X|_{G=j} \sim \mathcal{N}_p(\mu_j, \Sigma_j)$, es decir, se asume que cada población sigue una distribución normal.

Por definición, se tiene entonces que para una clase j:

$$f_j(x) = \frac{1}{(2\pi)^{\frac{p}{2}} \cdot |\Sigma_j|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu_j)^T \Sigma_j^{-1}(x-\mu_j)}$$

Aplicando logaritmo (que al ser una función estrictamente creciente no afecta el cálculo de máximos/mínimos), queda algo mucho más práctico de trabajar:

$$\log f_j(x) = -\frac{1}{2}\log|\Sigma_j| - \frac{1}{2}(x-\mu_j)^T \Sigma_j^{-1}(x-\mu_j) + C$$

Observar que en este caso $C = -\frac{p}{2}\log(2\pi)$, pero no se tiene en cuenta ya que al tener una constante aditiva en todas las clases, no afecta al cálculo del máximo.

1.6.3 LDA

En el caso de LDA se hace una suposición extra, que es $X|_{G=j} \sim \mathcal{N}_p(\mu_j, \Sigma)$, es decir que las poblaciones no sólo siguen una distribución normal sino que son de igual matriz de covarianzas. Reemplazando arriba se obtiene entonces:

$$\log f_j(x) = -\frac{1}{2}\log|\Sigma| - \frac{1}{2}(x-\mu_j)^T \Sigma^{-1}(x-\mu_j) + C$$

Ahora, como $-\frac{1}{2}\log|\Sigma|$ es común a todas las clases se puede incorporar a la constante aditiva y, distribuyendo y reagrupando términos sobre $(x-\mu_j)^T\Sigma^{-1}(x-\mu_j)$ se obtiene finalmente:

$$\log f_j(x) = \mu_j^T \Sigma^{-1}(x - \frac{1}{2}\mu_j) + C$$

1.6.4 Entrenamiento/Ajuste

Obsérvese que para ambos modelos, ajustarlos a los datos implica estimar los parámetros (μ_j, Σ_j) $\forall j = 1, \dots, k$ en el caso de QDA, y (μ_j, Σ) para LDA.

Estos parámetros se estiman por máxima verosimilitud, de manera que los estimadores resultan:

- $\hat{\mu}_j = \bar{x}_j$ el promedio de los x de la clase j
- $\hat{\Sigma}_j = s_j^2$ la matriz de covarianzas estimada para cada clase j
- $\hat{\pi}_j = f_{R_j} = \frac{n_j}{n}$ la frecuencia relativa de la clase j en la muestra

- $\hat{\Sigma} = \frac{1}{n}\sum_{j=1}^{k} n_j \cdot s_j^2$ el promedio ponderado (por frecs. relativas) de las matrices de covarianzas de todas las clases. *Observar que se utiliza el estimador de MV y no el insesgado*

Es importante notar que si bien todos los μ, Σ deben ser estimados, la distribución *a priori* puede no inferirse de los datos sino asumirse previamente, utilizándose como entrada del modelo.

1.6.5 Predicción

Para estos modelos, al igual que para cualquier clasificador Bayesiano del tipo antes visto, la estimación de la clase es por método *plug-in* sobre la regla de decisión $H(x)$, es decir devolver la clase que maximiza $\hat{f}_j(x) \cdot \hat{\pi}_j$, o lo que es lo mismo $\log \hat{f}_j(x) + \log \hat{\pi}_j$.

1.7 Anexo Persistencia y Análisis de datos

Dado que la idea de entrenar distintos modelos es poder comparar su performance tanto en términos de calidad de predicciones como en tiempo de ejecución y utilización de memoria es que se decidió utilizar una base de datos relacional externa para persistir todas las corridas de los modelos y poder realizar el análisis facilmente.

Modelo de datos:

- id = Column(Integer, primary_key=True, autoincrement=True)
- timestamp = Column(DateTime, unique=True)
- model_name = Column(String(200))
- dataset_name = Column(String(200))
- seed = Column(Integer)
- test_error = Column(Float)
- train_error = Column(Float)
- test_acc = Column(Float)
- train_acc = Column(Float)
- memory_allocation = Column(Float)
- execution_time = Column(Float)
- comments = Column(String(1000))