For the wine data set in the below link: https://archive.ics.uci.edu/dataset/109/wine what different models can i fit? only list me the models

**ChatGPT said:**

Here’s a list of different models you can fit on the Wine dataset:

1. Logistic Regression
2. k-Nearest Neighbors (k-NN)
3. Support Vector Machine (SVM)
4. Decision Tree
5. Random Forest
6. Gradient Boosting (e.g., XGBoost, LightGBM)
7. Naive Bayes
8. Linear Discriminant Analysis (LDA)
9. Quadratic Discriminant Analysis (QDA)
10. Multi-layer Perceptron (Neural Network)
11. Ridge Classifier
12. Bagging Classifier
13. Extra Trees Classifier

Let me know if you'd like a comparison of model performance, code examples, or a visual workflow.

**You said:**

can i build a simple neural network?

**ChatGPT said:**

Yes, you can absolutely build a simple neural network for the Wine dataset. Since it's a multiclass classification problem with 3 classes and 13 numerical features, a small feedforward neural network (multi-layer perceptron) with:

* **Input layer:** 13 neurons (one for each feature)
* **Hidden layer(s):** 1 or 2 layers with 8–32 neurons (ReLU activation)
* **Output layer:** 3 neurons with softmax activation

would be sufficient.

Let me know if you'd like a PyTorch example, TensorFlow/Keras version, or a training visualization.

**You said:**

build me a simple NN. Use 50 Epochs at max.

**ChatGPT said:**

Here’s a simple neural network using **Keras (TensorFlow)** to classify the Wine dataset, training for **up to 50 epochs**:

import numpy as np

from sklearn.datasets import load\_wine

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.utils import to\_categorical

# Load data

data = load\_wine()

X = data.data

y = data.target

# Preprocess

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

y\_cat = to\_categorical(y) # one-hot encoding for multiclass

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y\_cat, test\_size=0.2, random\_state=42)

# Build model

model = Sequential([

Dense(16, input\_shape=(X.shape[1],), activation='relu'),

Dense(8, activation='relu'),

Dense(3, activation='softmax') # 3 classes

])

# Compile

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Train

history = model.fit(X\_train, y\_train, epochs=50, validation\_data=(X\_test, y\_test), verbose=1)

# Evaluate

test\_loss, test\_acc = model.evaluate(X\_test, y\_test, verbose=0)

print(f'Test Accuracy: {test\_acc:.4f}')

Let me know if you want a plot of the training history, a PyTorch version, or model export options.

**You said:**

use PyTorch

**ChatGPT said:**

Here’s a simple neural network built with **PyTorch** for the Wine dataset, trained for **50 epochs max**:

import torch

import torch.nn as nn

import torch.optim as optim

from sklearn.datasets import load\_wine

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from torch.utils.data import TensorDataset, DataLoader

# Load and preprocess data

data = load\_wine()

X = data.data

y = data.target

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

X\_train = torch.tensor(X\_train, dtype=torch.float32)

X\_test = torch.tensor(X\_test, dtype=torch.float32)

y\_train = torch.tensor(y\_train, dtype=torch.long)

y\_test = torch.tensor(y\_test, dtype=torch.long)

train\_ds = TensorDataset(X\_train, y\_train)

test\_ds = TensorDataset(X\_test, y\_test)

train\_loader = DataLoader(train\_ds, batch\_size=16, shuffle=True)

test\_loader = DataLoader(test\_ds, batch\_size=16)

# Define model

class WineNN(nn.Module):

def \_\_init\_\_(self):

super(WineNN, self).\_\_init\_\_()

self.model = nn.Sequential(

nn.Linear(13, 16),

nn.ReLU(),

nn.Linear(16, 8),

nn.ReLU(),

nn.Linear(8, 3)

)

def forward(self, x):

return self.model(x)

model = WineNN()

# Training setup

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=0.01)

epochs = 50

# Training loop

for epoch in range(epochs):

model.train()

for xb, yb in train\_loader:

preds = model(xb)

loss = criterion(preds, yb)

optimizer.zero\_grad()

loss.backward()

optimizer.step()

# Evaluation

model.eval()

correct = 0

total = 0

with torch.no\_grad():

for xb, yb in test\_loader:

preds = model(xb)

\_, predicted = torch.max(preds, 1)

total += yb.size(0)

correct += (predicted == yb).sum().item()

print(f'Test Accuracy: {correct / total:.4f}')

Let me know if you'd like to add validation accuracy tracking, plot training loss, or apply early stopping.

**You said:**

Print me loss and accuracy for every 10 epochs.

**ChatGPT said:**

Here's the modified PyTorch code that **prints loss and accuracy every 10 epochs** during training:

import torch

import torch.nn as nn

import torch.optim as optim

from sklearn.datasets import load\_wine

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from torch.utils.data import TensorDataset, DataLoader

# Load and preprocess data

data = load\_wine()

X = data.data

y = data.target

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

X\_train = torch.tensor(X\_train, dtype=torch.float32)

X\_test = torch.tensor(X\_test, dtype=torch.float32)

y\_train = torch.tensor(y\_train, dtype=torch.long)

y\_test = torch.tensor(y\_test, dtype=torch.long)

train\_ds = TensorDataset(X\_train, y\_train)

test\_ds = TensorDataset(X\_test, y\_test)

train\_loader = DataLoader(train\_ds, batch\_size=16, shuffle=True)

test\_loader = DataLoader(test\_ds, batch\_size=16)

# Define model

class WineNN(nn.Module):

def \_\_init\_\_(self):

super(WineNN, self).\_\_init\_\_()

self.model = nn.Sequential(

nn.Linear(13, 16),

nn.ReLU(),

nn.Linear(16, 8),

nn.ReLU(),

nn.Linear(8, 3)

)

def forward(self, x):

return self.model(x)

model = WineNN()

# Training setup

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=0.01)

epochs = 50

# Training loop

for epoch in range(1, epochs + 1):

model.train()

running\_loss = 0.0

for xb, yb in train\_loader:

preds = model(xb)

loss = criterion(preds, yb)

optimizer.zero\_grad()

loss.backward()

optimizer.step()

running\_loss += loss.item()

# Print every 10 epochs

if epoch % 10 == 0:

model.eval()

correct = 0

total = 0

with torch.no\_grad():

for xb, yb in test\_loader:

preds = model(xb)

\_, predicted = torch.max(preds, 1)

total += yb.size(0)

correct += (predicted == yb).sum().item()

avg\_loss = running\_loss / len(train\_loader)

accuracy = correct / total

print(f"Epoch {epoch:2d}: Loss = {avg\_loss:.4f}, Accuracy = {accuracy:.4f}")

Let me know if you’d like to see a training curve plot, add learning rate decay, or compare with a baseline model.