Práctica III: Máquinas de vectores soporte y redes neuronales

Aprendizaje Automático II 2025-2026

La entrega será antes de las 10:00 del día 11 de diciembre para el grupo del jueves y del día 12 para el del viernes. se realizará una prueba práctica de evaluación relacionada con la entrega.

Objetivos

- Comprender en profundidad las SVM y su aplicación en problemas reales.
- Comprender en profundidad las distintas tipos de redes neuronales:
  - Feedfordward neural networks.
  - Redes convolucionales.

1. SVM

1.1. Clasificación

Vamos a resolver el problema de clasificación covtype, puede descargarlo como:

```
from sklearn.datasets import fetch_covtype
from sklearn.model_selection import train_test_split

# Cargar el conjunto de datos de dataset
dataset = fetch_covtype()
X, y = dataset.data, dataset.target

# Separación train test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5,
random_state=42)
reduced_size = 1000
```

```
# Selecciónamos la muestra de manera estratificadds
X_train_reduced, _, y_train_reduced, _ = train_test_split(X_train, y_train,
                        train_size= reduced_size /X_train.shape[0],
                        random_state=42, stratify=y_train)

X.shape, X_train.shape, X_train_reduced.shape
```

Vamos a construir un modelo de clasificación basado en SVM, primero nos basaremos en `from sklearn.svm import LinearSVC`. Responda razonadamente a las siguientes preguntas:

1. ¿Cuál es la utilidad de los parámetros *loss*, *penalty*, *dual*, *tol* y C en nuestro problema? Escriba sus expresiones matemáticas si fuera necesario y relacione los distintos parámetros con *loss*.

2. ¿Cuál es el coste computacional de resolver el problema primal de una SVM? ¿Y el dual? Acompañe la respuesta con un estudio experimental.

3. ¿Cuál es el coste computacional de predecir el problema? Acompañe de nuevo con un estudio experimental.

4. ¿Por qué cree que en el enunciado se ha reducido el tamaño de entrenamiento a mil muestras?

Resuelva el problema utilizando *Pipeline* y *GridSearchCV*, tenga presente una construcción adecuada de la rejilla de búsqueda y si los hiperparámetros que se optimizan verdaderamente contribuyen a la calidad del modelo. Finalmente, dé el accuracy en test encontrado.

Trate de nuevo de resolver el problema (es decir, busque el mejor modelo con sus mejores hiperparámetros), utilizando como SVM ahora la implementación `from sklearn.svm import SVC` y responda a las siguientes preguntas:

5. ¿Qué importancia tiene el parámetro *kernel*?

6. Explique la diferencia entre los distintos tipos de kernel que se encuentran disponibles en sklearn, dando expresiones analíticas en caso de ser necesario.

7. Busque el mejor accuracy posible para los kernels lineales, gaussiano y polinómico.

8. ¿Qué devuelve el atributo *support_vectors_*?

9. ¿Qué porcentaje de muestras del conjunto de entrenamiento son vectores soporte?

10. Suponga que de una SVM h_1, se selecciona el atributo *support_vectors_* y sus respectivas etiquetas, sean estos los conjuntos S, t. Se entrena una segunda SVM, h_2; con los mismos hiperparámetros que la primera, pero ahora, con el conjunto de entrenamiento descrito S, t. ¿Qué diferencia habría respecto a la función de predicción de h_1 y h_2?

11. ¿En qué situaciones utilizaría la implementación de *LinearSVC* frente a *SVC*?

12. ¿Por qué *SVC*, a diferencia de *LinearSVC*, no tiene el parámetro *dual*?

1.2. Regresión

A continuación trataremos de resolver el problema de regresión de *Diabetes*:

```
from sklearn.datasets import load_diabetes
dataset = load_diabetes()

# Cargar el conjunto de datos de dataset
X, y = dataset.data, dataset.target

# Separación train test
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.5, random_state=42)
```

En lo relativo al método `sklearn.svm.SVR`, responda a las siguientes cuestiones:

1. ¿Qué utilidad tiene el nuevo parámetro épsilon?

2. ¿Cuáles son las similitudes y diferencias entre utilizar una SVM y KNN? ¿En qué contexto utilizaría uno o otro?.

3. ¿Cuáles son las similitudes y diferencias entre utilizar una SVM y árboles de decisión? ¿En qué contexto utilizaría uno o otro?.

Métodos como SVM en regresión son sensibles a la distribución de las etiquetas, para solventar tales problemas puede basarse en *TransformedTargetRegressor* de la biblioteca *sklearn.compose*.

Utilizando un *Pipeline*, *TransformedTargetRegressor* y *GridSearchCV* busque los mejores hiperparámetros para predecir el modelo con el menor error posible.

ls

2. Redes neuronales

2.1. Redes neuronales *feedforward*

Complete el código proporcionado en el fichero `aa2_p3_v2.py`, que contiene la implementación de una red neuronal *feedforward*. Una vez completado, aplique la red neuronal para resolver el problema diabetes y responda a las siguientes preguntas:

1. ¿Cuál es el coste computacional de una red neuronal?
2. ¿Cuál es su coste en memoria?
3. Supongamos una red neuronal con una única capa oculta, esto es una función de la siguiente forma:

$$Y_0 = \sigma(W_0 X + \beta_0), \tag{1}$$
$$Y_1 = \theta(W_1 Y_0 + \beta_1), \tag{2}$$

   donde θ es la función de activación en la capa de salida y σ la función de activación en la capa oculta. ¿Qué interpretación geométrica le daría a los pesos de la red?

De manera habitual se suelen utilizar diferentes bibliotecas para la implementación de redes neuronales. Algunos ejemplos son:

- *MLP* de *sklearn*: Útil para problemas pequeños o cuando se quiere realizar la integración de las redes neuronales con otras herramientas de *sklearn* (como por ejemplo realizar ensembles de redes neuronales).
- *keras/Tensorflow* o *pytorch*: Para el diseño de redes neuronales más complejas. Estas librerías no solo tienen más flexibilidad que *MLP* de *sklearn*, sino que además permiten el uso de GPUs.

A continuación usaremos las bibliotecas anteriores para resolver distintos problemas.

Importante: En las secciones 2.3, 2.4 y 2.5 debe elegir solo una de las dos librerías, *PyTorch* o *Keras*, para hacer los ejercicios propuestos.

2.2. *MLP* de *sklearn*

Utilice MLPRegressor para resolver el problema diabetes. Después responda a las siguientes preguntas:

1. ¿Qué utilidad tienen los parámetros siguientes?
   - hidden_layer_sizes,
   - activation,
   - solver,
   - alpha,
   - batch_size,
   - learning_rate,
   - learning_rate_init,
   - max_iter.
2. ¿Qué es el optimizador *Adam*? ¿De qué parámetros depende?
3. ¿Podría entrenar los modelos con el dataset de entrenamiento completo en un tiempo prudencial?
4. Mida los tiempos de entrenamiento y predicción y compare con los de la sección 2.1.
5. Fije una estructura para la red con una única capa oculta y vaya incrementando el número de neuronas en dicha capa. Muestre en una gráfica el *score* frente al número de neuronas en la capa oculta.
6. Estudie la evolución del *score* y el tiempo de entrenamiento fijando todos los hiperparámetros y variando el número de capas ocultas de uno a cinco.
7. ¿Qué modelo considera mejor, una red neuronal o una SVM?
8. ¿Son las redes neuronales modelos deterministas? Es decir, ¿cada vez que se entrena la red se obtienen los mismos resultados? ¿De qué depende este determinismo?
9. ¿Tendría sentido construir un *ensemble* de redes neuronales? Implemente uno utilizando VotingRegressor de *sklearn.ensemble*. ¿Mejora el rendimiento del modelo?

2.3. PyTorch

En este apartado procederemos a resolver el problema covtype utilizando la biblioteca *PyTorch*. Para ello deberá instalar *PyTorch* siguiendo las instrucciones disponibles en `https://pytorch.org/get-started/locally/`. Recomendamos encarecidamente el uso de entornos virtuales o de conda para mantener una instalación limpia y controlada.

A modo de guía, se muestra a continuación un código sencillo para una red neuronal *feedforward* implementada con *PyTorch*:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import TensorDataset, DataLoader

# Definir el modelo de red neuronal
class Net(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(input_dim, 128)
        self.dropout = nn.Dropout(0.2)
        self.fc2 = nn.Linear(128, output_dim)
        self.relu = nn.ReLU()
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.softmax(self.fc2(x))
        return x

# Crear el modelo
model = Net(X_train.shape[1], y_train.shape[1])

# Definir la función de pérdida y el optimizador
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters())

# Entrenar el modelo
for epoch in range(50):  # número de epochs
```

```
    optimizer.zero_grad()
    outputs = model(X_train)
    loss = criterion(outputs, torch.argmax(y_train, dim=1))
    loss.backward()
    optimizer.step()

# Evaluar el modelo
with torch.no_grad():
    y_pred = model(X_test)
```

Responda a las siguientes cuestiones:

1. ¿Qué función cumple la clase *nn.Module*?
2. ¿Qué hace la capa *nn.Dropout*?
3. ¿En qué se parecen el uso de la capa *nn.Dropout* en *PyTorch* y el uso de un parámetro *alpha* distinto de 0 en *MLP*? ¿En qué se diferencian?
4. ¿Qué hace la función de pérdida *nn.CrossEntropyLoss* y qué papel cumple?
5. ¿Qué es el número de *epochs* de entrenamiento?

2.4. Keras

En este apartado vamos a resolver el problema covtype utilizando la biblioteca *keras*. Para ello deberá instalar *keras* siguiendo las instrucciones disponibles en `https://www.tensorflow.org/install`. Igual que en la sección anterior, recomendamos el uso de entornos virtuales o de conda para mantener una instalación limpia y controlada.

A modo de guía, se muestra a continuación un código sencillo para una red neuronal *feedforward* implementada con *keras*:

```
# Crear el modelo de red neuronal feedforward
model = Sequential()
model.add(Dense(128, input_dim=X_train.shape[1], activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(y_train.shape[1], activation='softmax'))

# Compilar el modelo
model.compile(optimizer='adam',
```

```
    loss='categorical_crossentropy',
    metrics=['accuracy']
    )
# Entrenar el modelo
history = model.fit(X_train, y_train, epochs=50,
batch_size=32)

# Evaluar el modelo en el conjunto de prueba
y_pred = model.predict(X_test)
```

Responda a las siguientes cuestiones:
Responda a las siguientes cuestiones:

1. ¿Qué función tiene la clase *Sequential*? ¿Qué elementos contiene?
2. ¿Qué función cumple la capa *Dropout*?
3. ¿En qué se parecen el uso de la capa *Dropout* en *keras* y el uso de un parámetro *alpha* distinto de 0 en *MLP*? ¿En qué se diferencian?
4. ¿Qué hace *categorical_crossentropy* y qué función cumple?
5. ¿Qué hace el parámetro *epochs*?

2.5. Redes neuronales convolucionales

Usando *keras* o *PyTorch*, trate de resolver el problema de clasificación de imágenes de dígitos MNIST a partir de redes neuronales convolucionales (CNNs). Para ello, en *keras* puede hacer uso de las capas *Conv2D* y *MaxPooling2D*:

```
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(MaxPooling2D((2, 2)))
```

También puede basarse en alguno de estos ejemplos:

- `https://www.tensorflow.org/datasets/keras_example`
- `https://www.digitalocean.com/community/tutorials/mnist-dataset-in-python`

Si decide utilizar *PyTorch*, a continuación se facilita el código de una CNN sencilla utilizando esta librería:

```
import torch
import torch.nn as nn
import torch.nn.functional as F

# Entrada esperada en PyTorch: (N, C, H, W) = (batch, 3, 32, 32)
# Nota: Keras usa (N, H, W, C). Si tus tensores vienen así,
# convierte con x = x.permute(0, 3, 1, 2) antes de pasarlos al modelo.

class SmallCNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv = nn.Conv2d(in_channels=3, out_channels=32,
                              kernel_size=3, stride=1, padding=0)

        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
    def forward(self, x):
        x = self.conv(x)          # (N, 32, 30, 30) tras conv 3x3 'valid' sobre 32x
        x = F.relu(x)             # activation='relu' en Keras
        x = self.pool(x)          # (N, 32, 15, 15)
        return x

model = SmallCNN()
```

Finalmente, responda razonadamente a las siguientes preguntas:

1. ¿Qué hacen las dos capas mencionadas?
2. ¿Qué diferencia hay entre *MaxPooling* y *Dropout*? ¿Tiene sentido utilizar *Dropout* en este problema? ¿Y *MaxPooling* en el anterior?