Clase 07: Machine Learning Geoespacial

Inteligencia Artificial aplicada al Análisis Territorial

Profesor: Francisco Parra O.

22 de septiembre de 2025

USACH - Ingeniería Civil en Informática

Agenda

Introducción al ML Geoespacial

Preparación de Datos Geoespaciales

Algoritmos de Machine Learning

Deep Learning para Análisis Territorial

Casos de Uso y Aplicaciones

Herramientas y Frameworks

Mejores Prácticas y Consideraciones

Introducción al ML Geoespacial

¿Por qué Machine Learning en Geografía?

Desafíos tradicionales:

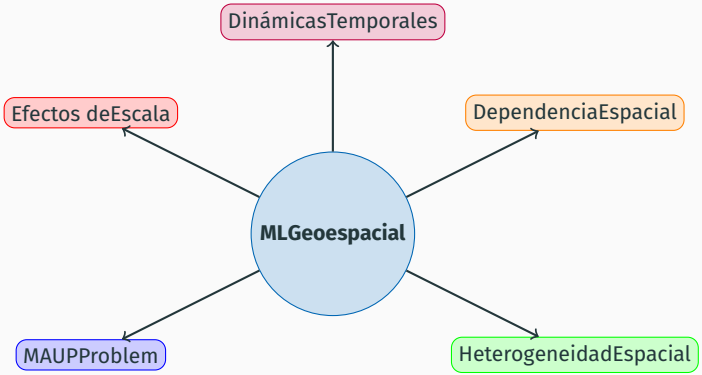
- Volumen masivo de datos
- Patrones complejos no lineales
- Múltiples escalas espaciales
- Heterogeneidad espacial
- Dimensionalidad alta

Oportunidades del ML:

- Detección automática de patrones
- Predicción precisa
- Escalabilidad
- Manejo de no linealidad
- Integración multimodal

El ML permite extraer conocimiento de datos geoespaciales masivos

Particularidades del ML Geoespacial



Preparación de Datos Geoespaciales

Tipos de Datos Geoespaciales

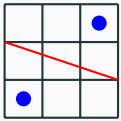
Vectoriales

- Puntos
- Líneas
- Polígonos
- Atributos

Raster

- Imágenes satelitales
- DEM/DSM
- Mapas de calor
- Bandas espectrales

Híbridos



- Nubes de puntos
- Grafos espaciales
- Trayectorias
- Series temporales

Feature Engineering Espacial

Características espaciales clave:

1. **Coordenadas:** X, Y, Z
2. **Distancias:** Euclidiana, Manhattan, Haversine
3. **Vecindad:** K-NN, Buffer, Voronoi
4. **Contexto:** Estadísticas focales
5. **Topología:** Conectividad, adyacencia
6. **Morfología:** Forma, área, perímetro

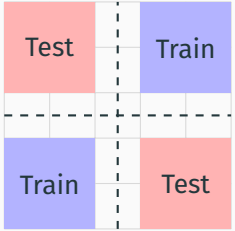
```
Distancia a POI gdf['dist_centro'] = gdf.geometry.distance(centro)
Densidad vecinal gdf['densidad_5km'] = gdf.buffer(5000).apply(lambda x: gdf[gdf.within(x)].shape[0])
Estadsticas focales gdf['mean_value_1km'] = focal_statistics(gdf, 'value', radius=1000, stat='mean')
```

Manejo de la Autocorrelación Espacial

Problema: Los datos espaciales violan el supuesto de independencia

Estrategias de mitigación:

- Spatial Cross-Validation
- Block sampling
- Spatial features explícitas
- Modelos espacialmente conscientes



Spatial CV con buffer zones

Algoritmos de Machine Learning

Clasificación de Imágenes Satelitales

Algoritmos principales:

- **Random Forest:** Robusto, interpretable
- **SVM:** Bueno para alta dimensionalidad
- **XGBoost:** Alto rendimiento
- **CNN:** Estado del arte para imágenes

Aplicaciones:

- Uso del suelo
- Detección de cambios
- Clasificación de cultivos
- Análisis urbano

```
1 Cargar imagen multiespectral with rasterio.open('sentinel2.tif') as src: img = src.read()
2 Preparar datos (bandas como features) X = img.reshape(n_bands, -1).T y =
        training_labels.ravel()
3 Entrenar clasificador rf = RandomForestClassifier(n_estimators =
        100).rf.fit(X[train_idx], y[train_idx])
4 Prediccin y_pred = rf
5   predict(X) classified = y_pred.reshape(height, width)..
```

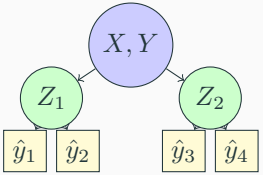
Predicción Espacial con ML

Random Forest Espacial

- Incorpora coordenadas como features
- Captura no linealidad espacial
- Maneja interacciones complejas

Ventajas sobre Kriging:

- No requiere estacionariedad
- Múltiples covariables
- Relaciones no lineales
- Robustez a outliers



RF con features espaciales

Clustering Espacial

Algoritmos especializados:

📍 DBSCAN Espacial

- Densidad espacial
- Formas irregulares
- Ruido y outliers

📊 Regionalization

- Clusters contiguos
- Restricciones espaciales
- Homogeneidad interna

🗂️ Hierarchical Clustering

- Múltiples escalas
- Dendrogramas espaciales

```
DBSCAN espacial coords = np.column_stack([gdf['x'], gdf['y']])
db = DBSCAN(eps=0.5, min_samples=5).fit(coords)
gdf['cluster'] = db.labels_
Regionalizacin con restricciones from pysal.lib.region import max_p_regions
w = weights.Queen.from_dataframe(gdf)attrs = gdf[['income', 'education']].values
gdf['region'] = max_p_regions(w, attrs, threshold=1000, solver='greedy')
```

Deep Learning para Análisis Territorial

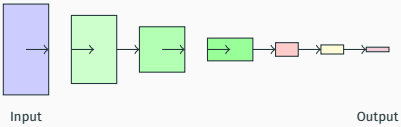
Redes Neuronales Convolucionales (CNN)

Arquitecturas para imágenes satelitales:

- **U-Net:** Segmentación semántica
- **ResNet:** Clasificación profunda
- **YOLO:** Detección de objetos
- **Mask R-CNN:** Segmentación de instancias

Aplicaciones:

- Detección de edificios
- Mapeo de carreteras
- Análisis de vegetación
- Monitoreo de cambios

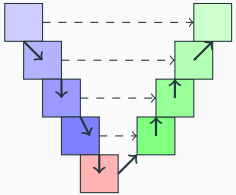


Arquitectura CNN simplificada

Segmentación Semántica con U-Net

```python
import tensorflow as tf
from tensorflow.keras import layers

def unet_model(input_shape):
    inputs = layers.Input(input_shape)

    # Encoder
    c1 = layers.Conv2D(64, 3, activation='relu', padding='same')(inputs)
    p1 = layers.MaxPooling2D(2)(c1)

    c2 = layers.Conv2D(128, 3, activation='relu', padding='same')(p1)
    p2 = layers.MaxPooling2D(2)(c2)

    # Bottleneck
    b = layers.Conv2D(256, 3, activation='relu', padding='same')(p2)

    # Decoder
    u2 = layers.UpSampling2D(2)(b)
    u2 = layers.concatenate([u2, c2])
    c3 = layers.Conv2D(128, 3, activation='relu', padding='same')(u2)

    u1 = layers.UpSampling2D(2)(c3)
    u1 = layers.concatenate([u1, c1])
    c4 = layers.Conv2D(64, 3, activation='relu', padding='same')(u1)

    outputs = layers.Conv2D(n_classes, 1, activation='softmax')(c4)
```

Arquitectura U-Net



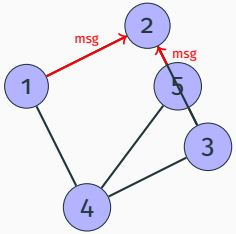
Graph Neural Networks para Datos Espaciales

¿Por qué grafos para datos espaciales?

- Relaciones no euclidianas
- Topología compleja
- Propagación de información
- Efectos de red

Aplicaciones:

- Predicción de tráfico
- Análisis de redes urbanas
- Difusión espacial
- Sistemas de transporte



Message Passing

Casos de Uso y Aplicaciones

Caso 1: Predicción de Precios Inmobiliarios

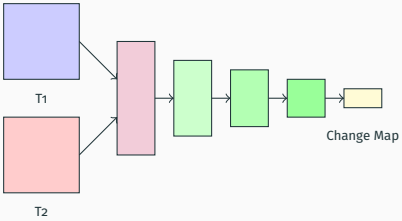
Pipeline completo:

1. **Datos:**
   - Características propiedades
   - POIs cercanos
   - Conectividad vial
   - Datos socioeconómicos
2. **Features espaciales:**
   - Distancia a metro
   - Densidad comercial
   - Índice de vegetación
3. **Modelo:** XGBoost + features espaciales
4. **Validación:** Spatial CV

```
Feature engineering espacial gdf['dist_metro
'] = gdf.geometry.apply( lambda x:
    metro_stations.distance(x).min())
gdf['comercios_500m'] =
gdf.buffer(500).apply(lambda x :
comercios[comercios.within(x)].shape[0])
Modelo con bsqueda de hiperparmetros
    param_grid =
    {'max_depth': [3, 5, 7], 'n_estimators':
    [100, 200, 300], 'learning_rate': [001, 01, 03]
xgb_model = xgb.XGBRegressor() spatial_cv =
    SpatialKFold(n_splits = 5)
grid = GridSearchCV(xgb_model, param_grid, cv =
    spatial_cv) grid.fit(X_train, y_train).
```

Caso 2: Detección de Cambios en Uso del Suelo

Arquitectura CNN para change detection:



Métricas de evaluación:

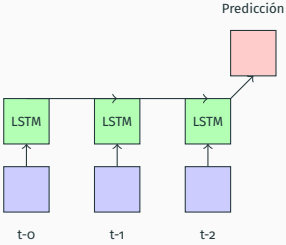
- Overall Accuracy
- Kappa coefficient
- F1-score por clase
- IoU (Intersection over Union)

10 Feature extraction
features$_t1 = vgg16_encoder(input_t1) features_t2 = vgg16_encoder(input_t2)$
11 Difference learning diff =
layers.subtract([features$_t1, features_t2]$)
12 Decoder x = decoder$_block(diff)$
13 Output: binary change map output = Conv2D(1, 1,
activation='sigmoid')(x)
14 return Model([input$_t1, input_t2], output)$
15 Training con data augmentation espacial datagen =
SpatialAugmentation(rotation$_range = 20, horizontal_flip = True, vertical_flip = True)$

Caso 3: Predicción de Demanda de Transporte

LSTM Espaciotemporal:

- Captura patrones temporales
- Considera dependencia espacial
- Maneja múltiples series




```
Preparar datos espaciotemporales def prepare_st_data(gdf, time_steps = 24) :
    X, y = [], [] for location in gdf location_i d unique() :
    loc_data = gdf[gdf location_i d == location]
for i in range(len(loc_data) − time_steps) : X append(loc_data tail oc[i :
    i + time_steps]values)y append(loc_data tail oc[i +
    time_steps]demand)
return np.array(X), np.array(y)
Modelo LSTM model = Sequential([ LSTM(50,
    return_sequences = True, input_shape =
    (time_steps, n_features)), LSTM(50), Dense(25, activation =
'relu'), Dense(1) ])
model.compile(optimizer='adam', loss='mse')
```

Herramientas y Frameworks

Ecosistema de Herramientas

Python Stack:

- **Scikit-learn:** ML clásico
- **TensorFlow/PyTorch:** Deep Learning
- **GeoPandas:** Datos vectoriales
- **Rasterio:** Datos raster
- **PySAL:** Análisis espacial
- **EarthEngine:** Big data satelital

Cloud Platforms:

- Google Earth Engine
- AWS SageMaker
- Azure ML
- Planetary Computer

</> Frameworks especializados:

TorchGeo

- Datasets geoespaciales
- Modelos preentrenados
- Augmentation espacial

Raster Vision

- Pipeline completo
- Chips generation
- Model zoo

GDAL/OGR

- Procesamiento raster/vector
- Transformaciones CRS
- I/O formatos

Pipeline Completo con TorchGeo

```python
import torch
from torchgeo.datasets import EuroSAT
from torchgeo.models import ResNet50
from torchgeo.samplers import RandomGeoSampler
from torch.utils.data import DataLoader

# Dataset geoespacial
dataset = EuroSAT(root='data/', download=True)

# Sampler espacial
sampler = RandomGeoSampler(
    dataset,
    size=256,
    length=10000,
    roi=region_of_interest
)

# DataLoader
dataloader = DataLoader(
    dataset,
    batch_size=32,
    sampler=sampler,
    collate_fn=stack_samples
)

# Modelo preentrenado
model = ResNet50(weights='eurosat', num_classes=10)

# Fine-tuning
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
criterion = torch.nn.CrossEntropyLoss()
for epoch in range(num_epochs):
```

Mejores Prácticas y Consideraciones

Validación en ML Geoespacial

⚠ Errores comunes:

- Random split ignora autocorrelación
- Data leakage espacial
- Overfitting a patrones locales
- Ignorar efectos de escala

✅ Buenas prácticas:

- Spatial Cross-Validation
- Leave-One-Region-Out
- Temporal + Spatial splits
- Métricas espacialmente explícitas

```
class SpatialKFold(BaseCrossValidator): def __init__(self, n_splits=5, buffer_size=1000): self.n_splits = n_splits self.buffer_size
def split(self, X, y=None, groups=None):  Crear clusters espaciales kmeans = KMeans(n_clusters = self.n_splits).clusters = kmeans.fit_predict(X[['x', 'y']])
for i in range(self.n_splits) : Test_indices test_idx = np.where(clusters == i)[0]
Train indices (con buffer)
    test_points = X.iloc[test_idx].geometry.buffer = test_points.buffer(self.buffer_size)
train_mask = X.geometry.within(buffer.unary_union).train_idx = np.where(train_mask)[0]
yield train_idx, test_idx
```

Interpretabilidad y Explicabilidad

Técnicas de interpretación:

Feature Importance

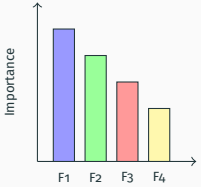
- Permutation importance
- SHAP values espaciales
- Partial dependence plots

Mapas de explicación

- Grad-CAM para CNNs
- Attention maps
- Local interpretations

Análisis de errores

- Mapeo de residuos
- Patrones espaciales de error
- Análisis por región



Feature Importance

Escalabilidad y Big Data

Estrategias de escalamiento:

- **Dask:** Paralelización local
- **Spark:** Procesamiento distribuido
- **Cloud computing:** GEE, AWS
- **GPU acceleration:** RAPIDS

Optimizaciones:

- Tile-based processing
- Lazy evaluation
- Data pyramids
- COG (Cloud Optimized GeoTIFF)

```
Cliente Dask distribuido client = Client('scheduler-address:8786')
GeoDataFrame distribuido ddf = dgpd.read_parquet(
 'geodata/*.parquet')
Operaciones lazy ddf['buffer_1km'] =
        ddf.geometry.buffer(1000) ddf['..'area'] =
        ddf.buffer_1km.area
Feature engineering paralelo def compute_spatial_features(partition) :
        Operaciones por partic in partition[  'neighbor_count'] =
        return partition
ddf = ddf.map_partitions(compute_spatial_features)
Computar cuando sea necesario result = ddf.compute()
```

Tendencias y Futuro

Tendencias Emergentes

Nuevas direcciones:

Foundation Models

- Modelos preentrenados masivos
- Transfer learning geoespacial
- Few-shot learning

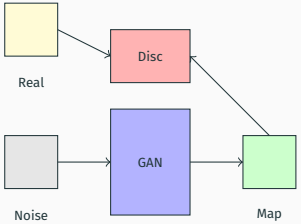
AutoML Espacial

- Optimización automática
- Neural Architecture Search
- Feature engineering automático

Física-informada

- Physics-informed neural networks
- Restricciones espaciales
- Modelos híbridos

IA Generativa Espacial:



Aplicaciones:

- Síntesis de mapas
- Super-resolución satelital
- Simulación urbana

Desafíos y Oportunidades

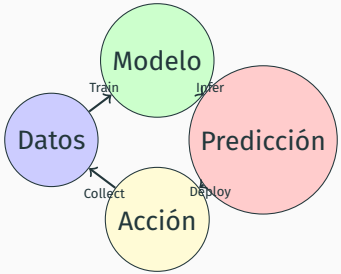
❗ Desafíos actuales:

- Datos etiquetados escasos
- Transferibilidad geográfica
- Sesgo espacial
- Incertidumbre cuantificación
- Cambio de distribución

💡 Oportunidades:

- Integración multimodal
- Digital twins urbanos
- Monitoreo en tiempo real
- Predicción de riesgos
- Planificación adaptativa

Ciclo virtuoso del ML Geoespacial



Resumen y Conclusiones

Puntos clave:

- ML transforma el análisis geoespacial
- Considerar siempre dependencia espacial
- Validación espacial es crítica
- Deep Learning abre nuevas posibilidades
- Interpretabilidad sigue siendo esencial

El futuro del análisis territorial es inteligente, automatizado y espacialmente consciente

Para profundizar:

- Documentación TorchGeo
- Tutoriales Google Earth Engine
- Papers en IGARSS, NeurIPS
- Cursos de especialización online

francisco.parra.o@usach.cl
github.com/franciscoparrao

Próximos Pasos

</> Ejercicios prácticos:

1. Clasificación de uso del suelo con RF
2. Segmentación de edificios con U-Net
3. Predicción espacial con XGBoost
4. Clustering espacial con DBSCAN

Lecturas recomendadas:

- ”Deep Learning for the Earth Sciences”
- ”Spatial Data Science with Python”
- Papers de CVPR Earth Vision

Proyecto final:

Desarrollar pipeline completo de ML para problema geoespacial real:

- Definir problema
- Preparar datos
- Entrenar modelo
- Validar espacialmente
- Interpretar resultados
- Desplegar solución