Laboratorio 2: Detección de Cambios Urbanos

Análisis Multitemporal con Imágenes Satelitales

Desarrollo de Aplicaciones Geoinformáticas

Prof. Francisco Parra O.
Departamento de Ingeniería Informática
Universidad de Santiago de Chile

Semestre 2, 2025

Contents

1 Introducción

1.1 Contexto

Las ciudades chilenas han experimentado transformaciones significativas en las últimas décadas. La expansión urbana, la pérdida de áreas verdes y los cambios en el uso de suelo tienen impactos directos en la calidad de vida, el medio ambiente y la planificación territorial.

Las imágenes satelitales, disponibles gratuitamente desde programas como Landsat (desde 1972) y Sentinel-2 (desde 2015), permiten analizar estos cambios de manera objetiva y cuantificable.

1.2 Objetivo General

Desarrollar un sistema completo de **detección y cuantificación de cambios urbanos** utilizando series temporales de imágenes satelitales, aplicando técnicas de teledetección y visualización interactiva.

1.3 Objetivos Específicos

1. Adquirir y procesar series temporales de imágenes Sentinel-2 o Landsat
2. Calcular índices espectrales (NDVI, NDBI, NDWI) para múltiples fechas
3. Implementar algoritmos de detección de cambios
4. Clasificar tipos de cambio (urbanización, pérdida de vegetación, etc.)
5. Cuantificar cambios por sector/zona
6. Desarrollar un dashboard interactivo para explorar los resultados

1.4 Competencias a Desarrollar

- Procesamiento de imágenes satelitales multitemporales
- Cálculo e interpretación de índices espectrales
- Técnicas de detección de cambios en teledetección
- Análisis zonal y estadísticas espaciales
- Desarrollo de aplicaciones de visualización interactiva
- Comunicación efectiva de resultados geoespaciales

2 Descripción del Proyecto

2.1 Área de Estudio

Cada grupo debe seleccionar **una zona urbana de Chile** que haya experimentado cambios significativos en los últimos 5-10 años.

Importante

El área de estudio debe:

- Tener entre 100 y 500 km² (una comuna o parte de ella)
- Mostrar cambios visibles (expansión urbana, nuevos proyectos, etc.)
- Tener disponibilidad de imágenes con baja nubosidad

Zonas sugeridas (con cambios documentados):

Zona	Tipo de cambio	Período sugerido
Colina / Chicureo	Expansión urbana acelerada	2017-2024
Pudahuel (aeropuerto)	Desarrollo industrial	2015-2024
San Pedro de la Paz	Urbanización costera	2016-2024
Puerto Montt	Expansión periurbana	2015-2024
Antofagasta norte	Expansión minera/urbana	2015-2024
Valdivia sur	Nuevos barrios	2017-2024

Table 1: Zonas con cambios urbanos significativos

2.2 Período de Análisis

El análisis debe cubrir **al menos 5 años**, con un mínimo de **4 fechas de análisis** distribuidas en el período. Se recomienda:

- Usar imágenes de la **misma época del año** (ej: todos los eneros o todos los veranos)
- Seleccionar imágenes con **menos de 10% de nubosidad** en el área de interés
- Incluir al menos una imagen **antes** del cambio principal y una **después**

2.3 Datos Requeridos

Dato	Fuente	Formato	Uso
Imágenes Sentinel-2	Copernicus Hub / GEE	GeoTIFF	Índices espectrales
Imágenes Landsat 8/9	USGS / GEE	GeoTIFF	Alternativa/complemento
Límites comunales	IDE Chile	Shapefile/GeoJSON	Delimitación
Manzanas censales	INE	Shapefile	Análisis zonal
Red vial	OpenStreetMap	GeoJSON	Contexto

Table 2: Fuentes de datos requeridas

3 Componentes del Proyecto

3.1 Parte 1: Adquisición de Datos (15%)

Objetivo

Descargar y organizar series temporales de imágenes satelitales para el área de estudio.

3.1.1 Opción A: Google Earth Engine (Recomendada)

```
import ee
import geemap

# Inicializar Earth Engine
ee.Initialize()

# Definir area de estudio (ejemplo: Colina)
area = ee.Geometry.Rectangle([-70.75, -33.25, -70.55, -33.10])

# Funcion para enmascarar nubes en Sentinel-2
def mask_clouds_s2(image):
    qa = image.select('QA60')
    cloud_mask = qa.bitwiseAnd(1 << 10).eq(0).And(
                 qa.bitwiseAnd(1 << 11).eq(0))
    return image.updateMask(cloud_mask)

# Coleccion Sentinel-2 para multiples anios
years = [2018, 2020, 2022, 2024]
composites = []

for year in years:
    # Filtrar por fecha (verano: dic-feb)
    start = f'{year}-01-01'
    end = f'{year}-02-28'

    collection = (ee.ImageCollection('COPERNICUS/S2_SR_HARMONIZED')
        .filterBounds(area)
        .filterDate(start, end)
        .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 10))
        .map(mask_clouds_s2))

    # Crear composito mediana
    composite = collection.median().clip(area)
    composites.append(composite)

    # Exportar
    task = ee.batch.Export.image.toDrive(
        image=composite.select(['B2','B3','B4','B8','B11','B12']),
        description=f'sentinel2_{year}',
        folder='cambio_urbano',
        region=area,
        scale=10,
        maxPixels=1e9
```

```
    )
    task.start()
    print(f"Exportando {year}...")
```

3.1.2 Opción B: Descarga Directa (Copernicus Browser)

Para quienes prefieran descarga manual:

1. Acceder a `https://browser.dataspace.copernicus.eu/`
2. Registrarse (gratuito)
3. Dibujar área de interés
4. Filtrar por fecha y nubosidad
5. Descargar productos L2A (reflectancia de superficie)

Entregable

Entregable Parte 1:

- Mínimo 4 imágenes de diferentes años
- Script de descarga documentado
- Metadatos de cada imagen (fecha, nubosidad, sensor)
- Justificación de fechas seleccionadas

3.2 Parte 2: Procesamiento y Cálculo de Índices (20%)

Objetivo

Calcular índices espectrales relevantes para detectar cambios urbanos y de vegetación.

3.2.1 Índices Espectrales Requeridos

Índice	Fórmula	Detecta
NDVI	$\frac{NIR-Red}{NIR+Red}$	Vegetación
NDBI	$\frac{SWIR-NIR}{SWIR+NIR}$	Áreas construidas
NDWI	$\frac{Green-NIR}{Green+NIR}$	Cuerpos de agua
BSI	$\frac{(SWIR+Red)-(NIR+Blue)}{(SWIR+Red)+(NIR+Blue)}$	Suelo desnudo

Table 3: Índices espectrales a calcular

3.2.2 Implementación

```python
import rasterio
import numpy as np
from pathlib import Path

def calcular_indices(ruta_imagen, ruta_salida):
    """
    Calcula indices espectrales para una imagen Sentinel-2.

    Bandas Sentinel-2 (10m): B2=Blue, B3=Green, B4=Red, B8=NIR
    Bandas Sentinel-2 (20m): B11=SWIR1, B12=SWIR2
    """
    with rasterio.open(ruta_imagen) as src:
        # Leer bandas (ajustar indices segun estructura del archivo)
        blue = src.read(1).astype(float) / 10000  # Escalar a reflectancia
        green = src.read(2).astype(float) / 10000
        red = src.read(3).astype(float) / 10000
        nir = src.read(4).astype(float) / 10000
        swir = src.read(5).astype(float) / 10000

        profile = src.profile

    # Evitar division por cero
    eps = 1e-10

    # NDVI: Vegetacion
    ndvi = (nir - red) / (nir + red + eps)

    # NDBI: Areas construidas
    ndbi = (swir - nir) / (swir + nir + eps)

    # NDWI: Agua
    ndwi = (green - nir) / (green + nir + eps)

    # BSI: Suelo desnudo
    bsi = ((swir + red) - (nir + blue)) / ((swir + red) + (nir + blue) +
   eps)

    # Guardar indices
    profile.update(count=4, dtype='float32')

    with rasterio.open(ruta_salida, 'w', **profile) as dst:
        dst.write(ndvi.astype('float32'), 1)
        dst.write(ndbi.astype('float32'), 2)
        dst.write(ndwi.astype('float32'), 3)
        dst.write(bsi.astype('float32'), 4)
        dst.descriptions = ('NDVI', 'NDBI', 'NDWI', 'BSI')

    return {'ndvi': ndvi, 'ndbi': ndbi, 'ndwi': ndwi, 'bsi': bsi}

# Procesar todas las imagenes
imagenes = sorted(Path('data/raw/').glob('sentinel2_*.tif'))
for img in imagenes:
```

```
    year = img.stem.split('_')[1]
    salida = Path(f'data/processed/indices_{year}.tif')
    indices = calcular_indices(img, salida)
    print(f"Procesado {year}: NDVI medio = {np.nanmean(indices['ndvi']):.3
    f}")
```

Entregable

Entregable Parte 2:

- Rasters de índices para cada fecha
- Visualización comparativa de índices (mapas lado a lado)
- Estadísticas descriptivas por fecha (media, std, histogramas)
- Notebook documentado con el procesamiento

3.3 Parte 3: Detección de Cambios (25%)

Objetivo

Implementar y comparar métodos de detección de cambios entre fechas.

3.3.1 Método 1: Diferencia de Índices

El método más simple: restar índices entre dos fechas.

```
import numpy as np
import rasterio

def detectar_cambio_diferencia(ruta_t1, ruta_t2, umbral=0.15):
    """
    Detecta cambios usando diferencia de NDVI.

    Returns:
        cambio: array con valores -1 (perdida), 0 (sin cambio), 1 (
    ganancia)
    """
    with rasterio.open(ruta_t1) as src1:
        ndvi_t1 = src1.read(1)  # Banda 1 = NDVI
        profile = src1.profile

    with rasterio.open(ruta_t2) as src2:
        ndvi_t2 = src2.read(1)

    # Diferencia
    diferencia = ndvi_t2 - ndvi_t1

    # Clasificar cambios
    cambio = np.zeros_like(diferencia, dtype=np.int8)
    cambio[diferencia < -umbral] = -1  # Perdida de vegetacion
    cambio[diferencia > umbral] = 1    # Ganancia de vegetacion
```

```
25
26      # Estadisticas
27      pixeles_total = np.sum(~np.isnan(diferencia))
28      perdida = np.sum(cambio == -1)
29      ganancia = np.sum(cambio == 1)
30
31      print(f"Pixeles con perdida: {perdida} ({100*perdida/pixeles_total:.1f
        }%)")
32      print(f"Pixeles con ganancia: {ganancia} ({100*ganancia/pixeles_total
        :.1f}%)")
33      print(f"Sin cambio significativo: {pixeles_total - perdida - ganancia}
        ")
34
35      return cambio, diferencia
36
37  # Detectar cambios 2018-2024
38  cambio, dif = detectar_cambio_diferencia(
39      'data/processed/indices_2018.tif',
40      'data/processed/indices_2024.tif',
41      umbral=0.15
42  )
```

3.3.2 Método 2: Clasificación de Cambio Urbano

Combinar múltiples índices para clasificar tipos de cambio:

```
1   import numpy as np
2
3   def clasificar_cambio_urbano(indices_t1, indices_t2, umbrales=None):
4       """
5       Clasifica el tipo de cambio urbano usando multiples indices.
6
7       Clases:
8           0: Sin cambio
9           1: Urbanizacion (vegetacion -> construido)
10          2: Perdida de vegetacion (otro tipo)
11          3: Ganancia de vegetacion
12          4: Nuevo cuerpo de agua
13          5: Perdida de agua
14      """
15      if umbrales is None:
16          umbrales = {
17              'ndvi_veg': 0.3,      # Umbral para considerar vegetacion
18              'ndbi_urbano': 0.0,   # Umbral para considerar urbano
19              'cambio_min': 0.1     # Cambio minimo significativo
20          }
21
22      # Extraer indices
23      ndvi_t1, ndbi_t1, ndwi_t1 = indices_t1['ndvi'], indices_t1['ndbi'],
        indices_t1['ndwi']
24      ndvi_t2, ndbi_t2, ndwi_t2 = indices_t2['ndvi'], indices_t2['ndbi'],
        indices_t2['ndwi']
25
26      # Inicializar clasificacion
```

```
    clase = np.zeros_like(ndvi_t1, dtype=np.uint8)

    # 1. Urbanizacion: era vegetacion, ahora es urbano
    era_vegetacion = ndvi_t1 > umbrales['ndvi_veg']
    es_urbano = ndbi_t2 > umbrales['ndbi_urbano']
    clase[era_vegetacion & es_urbano] = 1

    # 2. Perdida de vegetacion (no urbanizacion)
    perdio_veg = (ndvi_t1 - ndvi_t2) > umbrales['cambio_min']
    clase[(perdio_veg) & (clase == 0)] = 2

    # 3. Ganancia de vegetacion
    gano_veg = (ndvi_t2 - ndvi_t1) > umbrales['cambio_min']
    clase[gano_veg & (clase == 0)] = 3

    # 4. Nuevo cuerpo de agua
    era_no_agua = ndwi_t1 < 0
    es_agua = ndwi_t2 > 0.1
    clase[era_no_agua & es_agua & (clase == 0)] = 4

    # 5. Perdida de agua
    era_agua = ndwi_t1 > 0.1
    no_es_agua = ndwi_t2 < 0
    clase[era_agua & no_es_agua & (clase == 0)] = 5

    return clase

# Nombres de clases para visualizacion
CLASES_CAMBIO = {
    0: 'Sin cambio',
    1: 'Urbanizacion',
    2: 'Perdida vegetacion',
    3: 'Ganancia vegetacion',
    4: 'Nuevo cuerpo agua',
    5: 'Perdida agua'
}
```

3.3.3 Método 3: Análisis de Anomalías Temporales

Para series con más de 2 fechas:

```
import numpy as np
from scipy import stats

def detectar_anomalias_temporales(stack_indices, fecha_analisis=-1):
    """
    Detecta anomalias comparando con la serie historica.

    Args:
        stack_indices: array (n_fechas, height, width)
        fecha_analisis: indice de la fecha a analizar (-1 = ultima)

    Returns:
        z_score: anomalia estandarizada
```

```
        cambio_significativo: mascara booleana
    """
    # Calcular estadisticas historicas (excluyendo fecha de analisis)
    historico = np.delete(stack_indices, fecha_analisis, axis=0)
    media_hist = np.nanmean(historico, axis=0)
    std_hist = np.nanstd(historico, axis=0)

    # Imagen a analizar
    actual = stack_indices[fecha_analisis]

    # Z-score
    z_score = (actual - media_hist) / (std_hist + 1e-10)

    # Cambio significativo: |z| > 2
    cambio_significativo = np.abs(z_score) > 2

    # Clasificar direccion
    direccion = np.zeros_like(z_score, dtype=np.int8)
    direccion[z_score < -2] = -1  # Muy por debajo de lo normal
    direccion[z_score > 2] = 1    # Muy por encima de lo normal

    return z_score, cambio_significativo, direccion
```

Entregable

Entregable Parte 3:

- Implementación de al menos 2 métodos de detección
- Mapa de cambios clasificado
- Comparación de métodos (ventajas/desventajas)
- Justificación de umbrales utilizados
- Matriz de confusión si hay datos de validación

3.4 Parte 4: Cuantificación y Análisis Zonal (20%)

Objetivo

Cuantificar los cambios por zonas administrativas o de interés.

3.4.1 Análisis Zonal con GeoPandas

```
import geopandas as gpd
import rasterio
from rasterstats import zonal_stats
import pandas as pd

def analisis_zonal_cambios(ruta_cambios, ruta_zonas, columna_zona='
    NOM_ZONA'):
```

```
    """
    Calcula estadisticas de cambio por zona.
    """
    # Cargar zonas
    zonas = gpd.read_file(ruta_zonas)

    # Estadisticas zonales
    stats = zonal_stats(
        zonas,
        ruta_cambios,
        stats=['count', 'sum', 'mean'],
        categorical=True,
        category_map={0: 'sin_cambio', 1: 'urbanizacion',
                      2: 'perdida_veg', 3: 'ganancia_veg'}
    )

    # Convertir a DataFrame
    df_stats = pd.DataFrame(stats)
    df_stats['zona'] = zonas[columna_zona]

    # Calcular porcentajes
    total_pixeles = df_stats[['sin_cambio', 'urbanizacion',
                              'perdida_veg', 'ganancia_veg']].sum(axis=1)

    for col in ['urbanizacion', 'perdida_veg', 'ganancia_veg']:
        df_stats[f'{col}_pct'] = 100 * df_stats[col] / total_pixeles

    # Calcular area (asumiendo 10m x 10m = 100 m2 por pixel)
    pixel_area_ha = 100 / 10000  # hectareas
    for col in ['urbanizacion', 'perdida_veg', 'ganancia_veg']:
        df_stats[f'{col}_ha'] = df_stats[col] * pixel_area_ha

    return df_stats

# Ejecutar analisis
resultados = analisis_zonal_cambios(
    'data/processed/cambio_clasificado.tif',
    'data/vector/manzanas_censales.shp',
    columna_zona='MANZENT'
)

# Resumen
print("\n=== RESUMEN DE CAMBIOS POR ZONA ===")
print(f"Total urbanizacion: {resultados['urbanizacion_ha'].sum():.1f} ha")
print(f"Total perdida vegetacion: {resultados['perdida_veg_ha'].sum():.1f}
    ha")
print(f"Total ganancia vegetacion: {resultados['ganancia_veg_ha'].sum():.1
    f} ha")

# Top 10 zonas con mas urbanizacion
print("\nTop 10 zonas con mayor urbanizacion:")
print(resultados.nlargest(10, 'urbanizacion_ha')[['zona', 'urbanizacion_ha
    ']])
```

3.4.2 Análisis Temporal

```
import pandas as pd
import matplotlib.pyplot as plt

def analisis_temporal(lista_indices, fechas, mascara_area=None):
    """
    Analiza la evolucion temporal de indices en el area de estudio.
    """
    resultados = []

    for ruta, fecha in zip(lista_indices, fechas):
        with rasterio.open(ruta) as src:
            ndvi = src.read(1)
            ndbi = src.read(2)

        if mascara_area is not None:
            ndvi = ndvi[mascara_area]
            ndbi = ndbi[mascara_area]

        resultados.append({
            'fecha': fecha,
            'ndvi_mean': np.nanmean(ndvi),
            'ndvi_std': np.nanstd(ndvi),
            'ndbi_mean': np.nanmean(ndbi),
            'ndbi_std': np.nanstd(ndbi),
            'pct_vegetacion': 100 * np.sum(ndvi > 0.3) / np.sum(~np.isnan(
    ndvi)),
            'pct_urbano': 100 * np.sum(ndbi > 0) / np.sum(~np.isnan(ndbi))
        })

    df = pd.DataFrame(resultados)

    # Visualizar
    fig, axes = plt.subplots(2, 2, figsize=(12, 10))

    # NDVI temporal
    axes[0,0].errorbar(df['fecha'], df['ndvi_mean'], yerr=df['ndvi_std'],
                       marker='o', capsize=5)
    axes[0,0].set_ylabel('NDVI medio')
    axes[0,0].set_title('Evolucion del NDVI')
    axes[0,0].grid(True, alpha=0.3)

    # NDBI temporal
    axes[0,1].errorbar(df['fecha'], df['ndbi_mean'], yerr=df['ndbi_std'],
                       marker='s', color='brown', capsize=5)
    axes[0,1].set_ylabel('NDBI medio')
    axes[0,1].set_title('Evolucion del NDBI')
    axes[0,1].grid(True, alpha=0.3)

    # Porcentaje vegetacion
    axes[1,0].bar(df['fecha'], df['pct_vegetacion'], color='green', alpha
    =0.7)
    axes[1,0].set_ylabel('% Area con vegetacion')
```

```
    axes[1,0].set_title('Cobertura de vegetacion')

    # Porcentaje urbano
    axes[1,1].bar(df['fecha'], df['pct_urbano'], color='gray', alpha=0.7)
    axes[1,1].set_ylabel('% Area urbana')
    axes[1,1].set_title('Cobertura urbana')

    plt.tight_layout()
    plt.savefig('outputs/evolucion_temporal.png', dpi=150)

    return df
```

Entregable

Entregable Parte 4:

- Tabla resumen de cambios por zona (CSV)
- Gráficos de evolución temporal
- Mapa coroplético de intensidad de cambio por zona
- Análisis de patrones espaciales (¿dónde se concentran los cambios?)
- Interpretación de resultados (mínimo 1 página)

3.5 Parte 5: Dashboard Interactivo (20%)

Objetivo

Desarrollar una aplicación web para explorar los resultados de forma interactiva.

3.5.1 Estructura del Dashboard (Streamlit)

```
import streamlit as st
import geopandas as gpd
import pandas as pd
import folium
from streamlit_folium import st_folium
import plotly.express as px
import rasterio
from rasterio.plot import show
import numpy as np

st.set_page_config(page_title="Cambio Urbano", layout="wide")

st.title("Analisis de Cambio Urbano")
st.markdown("### Deteccion de cambios mediante imagenes satelitales")

# Sidebar: Controles
st.sidebar.header("Configuracion")
fecha_inicio = st.sidebar.selectbox("Fecha inicial", [2018, 2019, 2020])
```

```
fecha_fin = st.sidebar.selectbox("Fecha final", [2022, 2023, 2024])
tipo_cambio = st.sidebar.multiselect(
    "Tipos de cambio a mostrar",
    ["Urbanizacion", "Perdida vegetacion", "Ganancia vegetacion"],
    default=["Urbanizacion"]
)

# Cargar datos
@st.cache_data
def cargar_datos():
    zonas = gpd.read_file('data/vector/zonas_cambio.gpkg')
    stats = pd.read_csv('data/processed/estadisticas_cambio.csv')
    return zonas, stats

zonas, stats = cargar_datos()

# Layout en columnas
col1, col2 = st.columns([2, 1])

with col1:
    st.subheader("Mapa de Cambios")

    # Crear mapa Folium
    centro = [zonas.geometry.centroid.y.mean(),
              zonas.geometry.centroid.x.mean()]
    m = folium.Map(location=centro, zoom_start=12)

    # Agregar capa de cambios
    folium.GeoJson(
        zonas,
        style_function=lambda x: {
            'fillColor': 'red' if x['properties']['urbanizacion_ha'] > 10
   else 'yellow',
            'color': 'black',
            'weight': 1,
            'fillOpacity': 0.5
        },
        tooltip=folium.GeoJsonTooltip(
            fields=['zona', 'urbanizacion_ha', 'perdida_veg_ha'],
            aliases=['Zona:', 'Urbanizacion (ha):', 'Perdida veg (ha):']
        )
    ).add_to(m)

    st_folium(m, width=700, height=500)

with col2:
    st.subheader("Estadisticas")

    # Metricas principales
    total_urb = stats['urbanizacion_ha'].sum()
    total_perd = stats['perdida_veg_ha'].sum()

    st.metric("Total Urbanizacion", f"{total_urb:.1f} ha")
    st.metric("Perdida Vegetacion", f"{total_perd:.1f} ha")
```

```
72
73      # Grafico de barras
74      fig = px.bar(
75          stats.nlargest(10, 'urbanizacion_ha'),
76          x='zona', y='urbanizacion_ha',
77          title='Top 10 zonas con mas urbanizacion'
78      )
79      st.plotly_chart(fig, use_container_width=True)
80
81  # Seccion: Comparacion temporal
82  st.subheader("Comparacion Temporal")
83  col3, col4 = st.columns(2)
84
85  with col3:
86      st.image(f'outputs/ndvi_{fecha_inicio}.png', caption=f'NDVI {
        fecha_inicio}')
87
88  with col4:
89      st.image(f'outputs/ndvi_{fecha_fin}.png', caption=f'NDVI {fecha_fin}')
90
91  # Seccion: Evolucion temporal
92  st.subheader("Evolucion Temporal")
93  df_temporal = pd.read_csv('data/processed/evolucion_temporal.csv')
94  fig_temporal = px.line(
95      df_temporal, x='fecha', y=['pct_vegetacion', 'pct_urbano'],
96      title='Evolucion de cobertura',
97      labels={'value': 'Porcentaje', 'fecha': 'A o'}
98  )
99  st.plotly_chart(fig_temporal, use_container_width=True)
100
101 # Descarga de datos
102 st.sidebar.markdown("---")
103 st.sidebar.subheader("Descargar Datos")
104 csv = stats.to_csv(index=False)
105 st.sidebar.download_button(
106     "Descargar estadisticas (CSV)",
107     csv,
108     "estadisticas_cambio.csv",
109     "text/csv"
110 )
```

Entregable

Entregable Parte 5:

- Aplicación Streamlit funcional
- Mapa interactivo con capas de cambio
- Gráficos dinámicos (selección de fechas, zonas)
- Comparador visual antes/después
- Opción de descarga de resultados
- Instrucciones de ejecución (README)

4 Estructura del Repositorio

```
laboratorio_cambio_urbano/
|-- README.md
|-- requirements.txt
|-- docker-compose.yml (opcional)
|
|-- data/
|   |-- raw/                    # Imagenes originales
|   |   |-- sentinel2_2018.tif
|   |   |-- sentinel2_2020.tif
|   |   |-- sentinel2_2022.tif
|   |   |-- sentinel2_2024.tif
|   |
|   |-- processed/              # Datos procesados
|   |   |-- indices_2018.tif
|   |   |-- indices_2024.tif
|   |   |-- cambio_clasificado.tif
|   |   |-- estadisticas_cambio.csv
|   |
|   |-- vector/                 # Datos vectoriales
|       |-- limite_comuna.gpkg
|       |-- manzanas_censales.shp
|       |-- zonas_cambio.gpkg
|
|-- notebooks/
|   |-- 01_descarga_datos.ipynb
|   |-- 02_calculo_indices.ipynb
|   |-- 03_deteccion_cambios.ipynb
|   |-- 04_analisis_zonal.ipynb
|   |-- 05_visualizacion.ipynb
|
|-- scripts/
|   |-- download_sentinel.py
|   |-- calculate_indices.py
|   |-- detect_changes.py
|   |-- zonal_analysis.py
```

```
|
|-- app/
|   |-- app.py                    # Streamlit app
|   |-- utils.py
|   |-- config.py
|
|-- outputs/
|   |-- figures/
|   |-- maps/
|   |-- reports/
|
|-- docs/
    |-- guia_laboratorio.pdf
    |-- informe_final.pdf
```

5 Criterios de Evaluación

5.1 Rúbrica General

Componente	Peso	Criterios principales
Adquisición de datos	15%	Calidad de imágenes, documentación de fuentes
Cálculo de índices	20%	Correctitud, visualización, interpretación
Detección de cambios	25%	Métodos implementados, justificación de parámetros
Análisis zonal	20%	Cuantificación, interpretación, patrones
Dashboard	20%	Funcionalidad, usabilidad, diseño

5.2 Criterios para Nota Máxima (7.0)

Para obtener nota 7.0, el proyecto debe incluir al menos **3 elementos adicionales**:

- Validación con datos externos (ortofotos, Google Earth histórico)
- Más de 6 fechas de análisis
- Clasificación supervisada de uso de suelo
- Análisis de series temporales con suavizado (Savitzky-Golay)
- Predicción de cambio futuro (modelo simple)
- Análisis de fragmentación del paisaje
- Integración con datos censales (correlación cambio-población)
- Deploy del dashboard en la nube (Streamlit Cloud, Heroku)
- Animación temporal (GIF/video de cambios)
- API REST para consulta de datos

5.3 Penalizaciones

- Código sin documentar: -0.5 por componente
- Entrega tardía: -1.0 por día
- Plagio: Nota mínima (1.0)
- Dashboard no funcional: -1.5
- Menos de 4 fechas de análisis: -1.0

6 Plazos y Entrega

6.1 Hitos

Semana	Hito	Entregable
1	Setup y datos	Área definida, imágenes descargadas, repositorio creado
2	Procesamiento	Índices calculados, primer análisis de cambios
3	Análisis y app	Dashboard funcional, informe final, video

6.2 Formato de Entrega

1. **Repositorio GitHub:** Link al repositorio con todo el código
2. **Informe PDF:** Máximo 10 páginas con:
   - Descripción del área y período de estudio
   - Metodología aplicada
   - Resultados principales (mapas, gráficos, tablas)
   - Discusión e interpretación
   - Conclusiones
3. **Video:** 5 minutos máximo mostrando el dashboard y resultados principales
4. **Link al dashboard:** Si está desplegado en la nube

7 Recursos Adicionales

7.1 Tutoriales Recomendados

- Google Earth Engine: `https://developers.google.com/earth-engine/tutorials`
- Rasterio: `https://rasterio.readthedocs.io/`
- Streamlit: `https://docs.streamlit.io/`
- Change Detection with Python: `https://www.earthdatascience.org/`

7.2 Papers de Referencia

- Zhu, Z. (2017). Change detection using Landsat time series. *Remote Sensing of Environment*.
- Kennedy, R. E. et al. (2010). Detecting trends in forest disturbance and recovery using yearly Landsat time series.
- Pesaresi, M. et al. (2016). Operating procedure for the production of the Global Human Settlement Layer.

7.3 Datos de Validación

- Google Earth Pro: Imágenes históricas de alta resolución
- IDE Chile: Ortofotos SAF (`https://www.ide.cl/`)
- Global Human Settlement Layer: `https://ghsl.jrc.ec.europa.eu/`

Tip

Consejo final: Comiencen temprano con la descarga de datos. Las imágenes satelitales pueden tardar en descargarse y es frustrante quedarse sin tiempo por problemas técnicos. ¡Planifiquen bien!