Advanced Machine Learning for Land Cover Change Detection

Ivan Innocent Sekibenga

2026-02-14

1 Introduction

This document extends the categorical change detection workflow by applying three advanced analytical methods to the same Central Valley Landsat imagery (2001 and 2011):

1. **XGBoost Classification** — Gradient boosted trees, often the top performer on tabular data
2. **Support Vector Machine (SVM)** — RBF kernel SVM, a classic choice for spectral classification
3. **Principal Component Analysis (PCA)** — Dimensionality reduction on the combined 12-band stack to reveal spectral change patterns

All three methods are compared against the Random Forest baseline from the original analysis.

2 Load Packages and Data

```
library(terra)
library(sf)
library(tidyverse)
library(tidymodels)
library(kableExtra)
```

```
## systemfonts and textshaping have been compiled with different versions of Freetype. Because of this, textshaping
will not use the font cache provided by systemfonts
```

```
library(vip)
library(xgboost)
library(kernlab)
```

> **Note:** If `xgboost` or `kernlab` are not installed, run the following in your console before knitting:
>
> ```
> install.packages(c("xgboost", "kernlab"))
> ```

2.1 Load Raster Imagery and Training Data

We reuse the same data loading and extraction pipeline from the original analysis.

```
# Load rasters
raster_t1 <- rast("data/rs/centralvalley-2001LE7.tif")
raster_t2 <- rast("data/rs/centralvalley-2011LT5.tif")

# Load and reproject training polygons
training_polys <- readRDS("data/rs/lcsamples.rds")
training_reproj <- project(training_polys, crs(raster_t1))

# Extract spectral values for Time 1
extract_t1 <- terra::extract(raster_t1, training_reproj, df = TRUE) |>
  left_join(
    tibble(ID = seq_len(nrow(training_polys)),
           class = training_polys$class),
    by = "ID"
  ) |>
  mutate(class = as.factor(class)) |>
  select(class, B1:B7) |>
  drop_na()

# Extract spectral values for Time 2
extract_t2 <- terra::extract(raster_t2, training_reproj, df = TRUE) |>
  left_join(
    tibble(ID = seq_len(nrow(training_polys)),
           class = training_polys$class),
    by = "ID"
  ) |>
  mutate(class = as.factor(class)) |>
  select(class, B1:B7) |>
  drop_na()

cat("Time 1 pixels:", nrow(extract_t1), "\n")
```

```
## Time 1 pixels: 68831
```

```
cat("Time 2 pixels:", nrow(extract_t2), "\n")
```

```
## Time 2 pixels: 68831
```

2.2 Create Common Train/Test Splits

To ensure a fair comparison across all models, we use the same train/test split and cross-validation folds for each method.

```
set.seed(6142)

# Time 1 split
split_t1 <- initial_split(extract_t1, prop = 0.7, strata = class)
train_t1 <- training(split_t1)
test_t1  <- testing(split_t1)

# Time 2 split
split_t2 <- initial_split(extract_t2, prop = 0.7, strata = class)
train_t2 <- training(split_t2)
test_t2  <- testing(split_t2)

# Cross-validation folds (Time 1 — used for all model comparisons)
set.seed(3917)
cv_folds_t1 <- vfold_cv(extract_t1, v = 5, strata = class)

cat("Training pixels (T1):", nrow(train_t1), "\n")
```

```
## Training pixels (T1): 48180
```

```
cat("Testing pixels (T1):",  nrow(test_t1), "\n")
```

```
## Testing pixels (T1): 20651
```

Interpretation: Using identical splits and folds across all models ensures that performance differences reflect model quality, not data sampling variation.

3 Random Forest Baseline

We first reproduce the Random Forest model as a baseline for comparison.

```
rf_spec <- rand_forest(trees = 500) |>
  set_engine("randomForest") |>
  set_mode("classification")

rf_wf <- workflow() |>
  add_recipe(recipe(class ~ ., data = train_t1)) |>
  add_model(rf_spec) |>
  fit(train_t1)

# Test set predictions
rf_preds <- augment(rf_wf, test_t1)

cat("=== Random Forest — Test Set ===\n")
```

```
## === Random Forest — Test Set ===
```

```
conf_mat(rf_preds, truth = class, estimate = .pred_class)
```

```
##           Truth
## Prediction built cropland fallow open water
##   built     1378       14      5   20     0
##   cropland    30     1985     64    0     0
##   fallow      20      126   2851    2     0
##   open       126        1      0 5581     0
##   water        0        0      0    0  8448
```

```
rf_test_metrics <- rf_preds |>
  metrics(truth = class, estimate = .pred_class) |>
  mutate(model = "Random Forest")

rf_test_metrics |>
  select(model, .metric, .estimate) |>
  kable(caption = "Random Forest — Test Set Metrics", digits = 4) |>
  kable_styling(bootstrap_options = c("striped", "hover"), full_width = FALSE)
```

Random Forest — Test Set Metrics

model	.metric	.estimate
Random Forest	accuracy	0.9802
Random Forest	kap	0.9726

```
# Cross-validation
rf_cv <- workflow() |>
  add_recipe(recipe(class ~ ., data = extract_t1)) |>
  add_model(rf_spec) |>
  fit_resamples(resamples = cv_folds_t1, metrics = metric_set(accuracy, kap))

rf_cv_metrics <- collect_metrics(rf_cv) |> mutate(model = "Random Forest")

rf_cv_metrics |>
  select(model, .metric, mean, std_err) |>
  kable(caption = "Random Forest — 5-Fold CV Metrics", digits = 4) |>
  kable_styling(bootstrap_options = c("striped", "hover"), full_width = FALSE)
```

Random Forest — 5-Fold CV Metrics

model	.metric	mean	std_err
Random Forest	accuracy	0.9823	3e-04
Random Forest	kap	0.9755	4e-04

Interpretation: These metrics establish the baseline performance. Random Forest is a strong default classifier for spectral data because it handles non-linear relationships, is robust to outliers, and requires no feature scaling. The cross-validation results provide a more stable performance estimate than the single test split.

4 XGBoost Classification

4.1 Model Specification

XGBoost (eXtreme Gradient Boosting) builds trees sequentially, where each tree corrects errors made by the ensemble so far. Key differences from Random Forest:

- **Sequential learning**: Trees are added one at a time, each focusing on previously misclassified samples
- **Regularization**: Built-in L1/L2 regularization helps prevent overfitting
- **Learning rate**: Controls how much each new tree contributes, allowing fine-grained optimization

```
xgb_spec <- boost_tree(
  trees = 500,
  tree_depth = 6,
  learn_rate = 0.1,
  min_n = 5
) |>
  set_engine("xgboost") |>
  set_mode("classification")

xgb_wf <- workflow() |>
  add_recipe(recipe(class ~ ., data = train_t1)) |>
  add_model(xgb_spec) |>
  fit(train_t1)

xgb_wf
```

```
## ══ Workflow [trained] ══════════════════════════════════════════════════════
## Preprocessor: Recipe
## Model: boost_tree()
##
## ── Preprocessor ────────────────────────────────────────────────────────────
## 0 Recipe Steps
##
## ── Model ───────────────────────────────────────────────────────────────────
## ##### xgb.Booster
## call:
##   xgboost::xgb.train(params = list(eta = 0.1, max_depth = 6, gamma = 0,
##     colsample_bytree = 1, colsample_bynode = 1, min_child_weight = 5,
##     subsample = 1, num_class = 5L, nthread = 1, objective = "multi:softprob"),
##     data = x$data, nrounds = 500, evals = x$watchlist, verbose = 0)
## # of features: 6
## # of rounds:  500
## callbacks:
##    evaluation_log
## evaluation_log:
##   iter training_mlogloss
##  <num>             <num>
##      1        1.24171162
##      2        1.10379551
##    ---               ---
##    499        0.02543657
##    500        0.02540265
```

4.2 XGBoost Accuracy Assessment

```
xgb_preds <- augment(xgb_wf, test_t1)

cat("=== XGBoost — Test Set ===\n")
```

```
## === XGBoost — Test Set ===
```

```
conf_mat(xgb_preds, truth = class, estimate = .pred_class)
```

```
##           Truth
## Prediction built cropland fallow open water
##   built     1386       15      9   39     0
##   cropland    36     1986     93    0     0
##   fallow      21      124   2818    2     0
##   open       110        1      0 5562     0
##   water        1        0      0    0  8448
```

```
xgb_test_metrics <- xgb_preds |>
  metrics(truth = class, estimate = .pred_class) |>
  mutate(model = "XGBoost")

xgb_test_metrics |>
  select(model, .metric, .estimate) |>
  kable(caption = "XGBoost — Test Set Metrics", digits = 4) |>
  kable_styling(bootstrap_options = c("striped", "hover"), full_width = FALSE)
```

XGBoost — Test Set Metrics

model	.metric	.estimate
XGBoost	accuracy	0.9782
XGBoost	kap	0.9698

```
xgb_cv <- workflow() |>
  add_recipe(recipe(class ~ ., data = extract_t1)) |>
  add_model(xgb_spec) |>
  fit_resamples(resamples = cv_folds_t1, metrics = metric_set(accuracy, kap))

xgb_cv_metrics <- collect_metrics(xgb_cv) |> mutate(model = "XGBoost")

xgb_cv_metrics |>
  select(model, .metric, mean, std_err) |>
  kable(caption = "XGBoost — 5-Fold CV Metrics", digits = 4) |>
  kable_styling(bootstrap_options = c("striped", "hover"), full_width = FALSE)
```

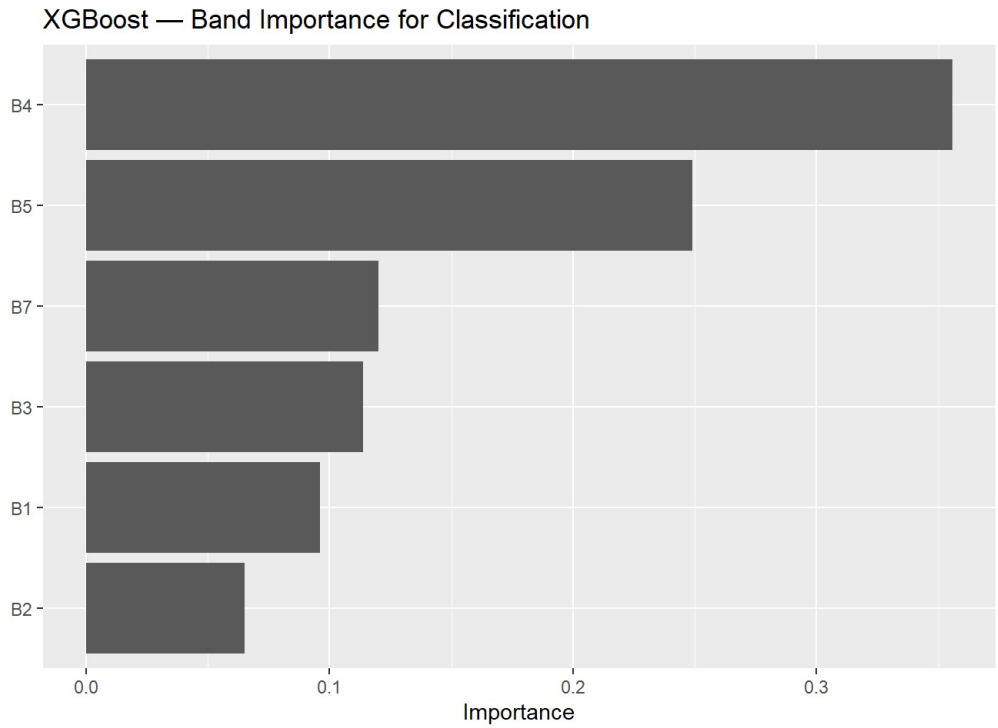
XGBoost — 5-Fold CV Metrics

model	.metric	mean	std_err
XGBoost	accuracy	0.9796	5e-04
XGBoost	kap	0.9717	7e-04

Interpretation: Compare these metrics with the Random Forest baseline. XGBoost often achieves similar or slightly better accuracy on tabular data due to its sequential error-correction mechanism. However, with small training samples, the improvement may be modest since both methods already fit the data well.

4.3 XGBoost Variable Importance

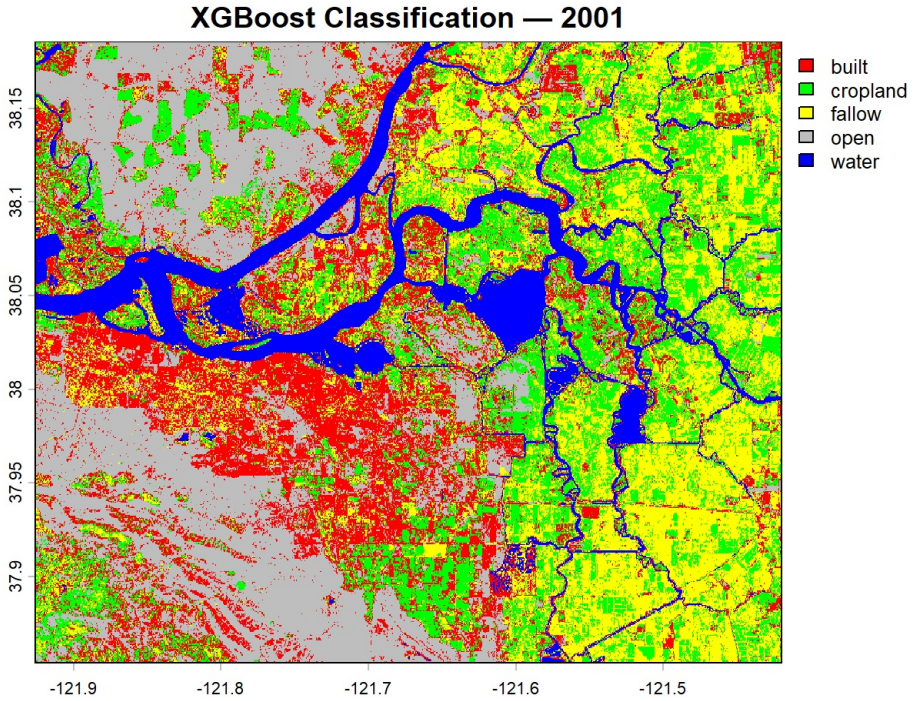
```
xgb_wf |>
  extract_fit_parsnip() |>
  vip(num_features = 6) +
  labs(title = "XGBoost — Band Importance for Classification")
```



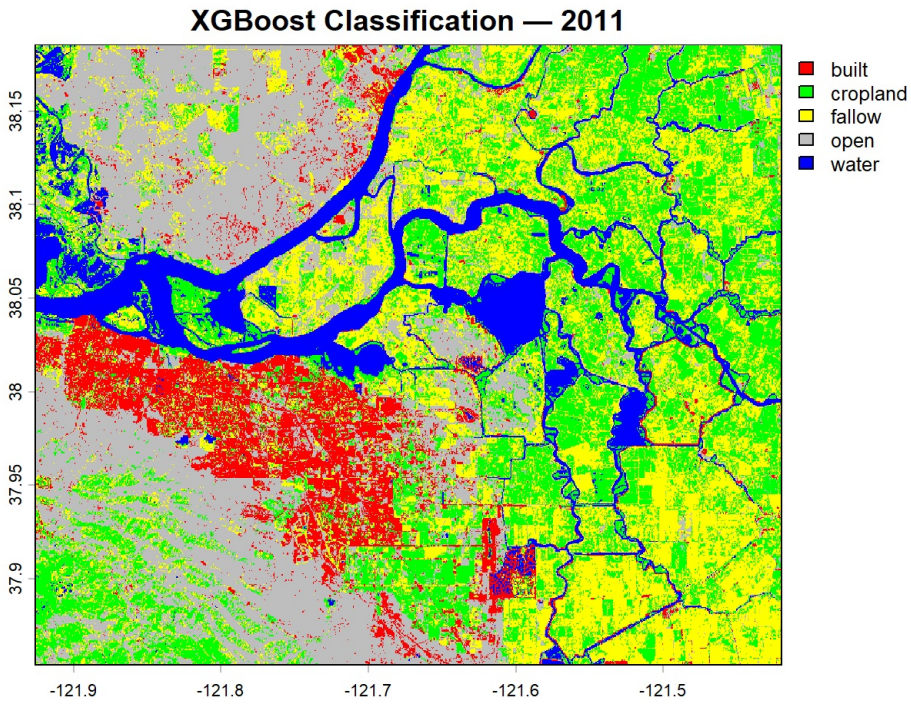
Interpretation: XGBoost importance rankings may differ from Random Forest because boosting focuses on bands that help correct residual errors. Bands ranked highly by both methods are consistently informative, while discrepancies highlight complementary information used by different learning strategies.

4.4 Classify Rasters with XGBoost

```
# Train XGBoost for Time 2 as well
xgb_wf_t2 <- workflow() |>
  add_recipe(recipe(class ~ ., data = train_t2)) |>
  add_model(xgb_spec) |>
  fit(train_t2)

# Classify both time periods
classified_xgb_t1 <- terra::predict(raster_t1, xgb_wf, na.rm = TRUE)
classified_xgb_t2 <- terra::predict(raster_t2, xgb_wf_t2, na.rm = TRUE)

class_colors <- c("built" = "red", "cropland" = "green",
                  "fallow" = "yellow", "open" = "grey", "water" = "blue")

par(mfrow = c(1, 1))
plot(classified_xgb_t1, main = "XGBoost Classification — 2001",
     col = class_colors, type = "classes")
```



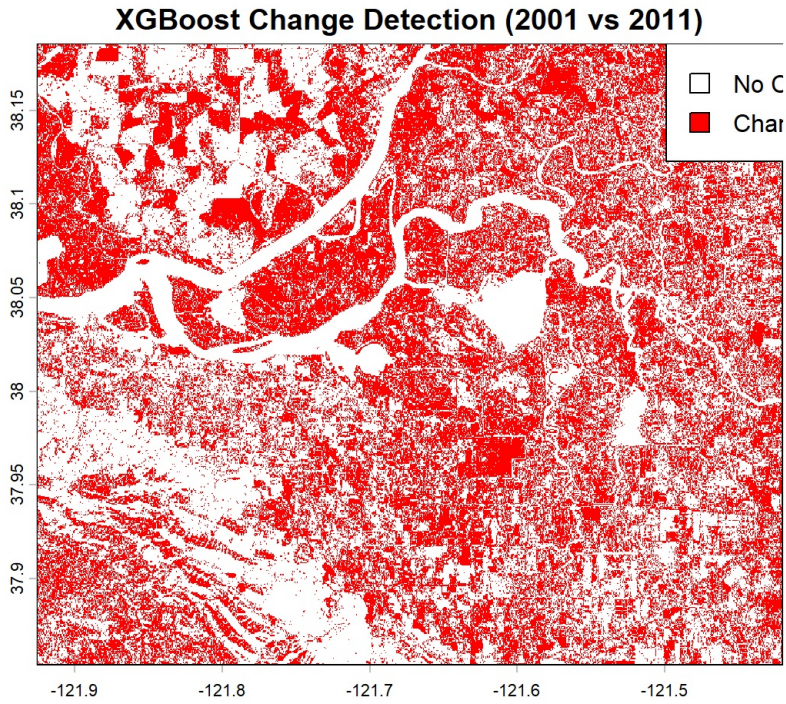
```
plot(classified_xgb_t2, main = "XGBoost Classification — 2011",
     col = class_colors, type = "classes")
```




```
par(mfrow = c(1, 1))
```

4.5 XGBoost Change Detection

```
# Binary change map
xgb_change <- classified_xgb_t1 != classified_xgb_t2

plot(xgb_change, main = "XGBoost Change Detection (2001 vs 2011)",
     col = c("white", "red"), legend = FALSE)
legend("topright", legend = c("No Change", "Change"),
       fill = c("white", "red"))
```



```
# Transition matrix with class names
class_names <- levels(extract_t1$class)

xgb_vals_t1 <- values(classified_xgb_t1) |> as.vector()
xgb_vals_t2 <- values(classified_xgb_t2) |> as.vector()
xgb_vals_t1 <- class_names[xgb_vals_t1]
xgb_vals_t2 <- class_names[xgb_vals_t2]

xgb_valid <- !is.na(xgb_vals_t1) & !is.na(xgb_vals_t2)

xgb_transition <- table(
  "From (2001)" = xgb_vals_t1[xgb_valid],
  "To (2011)"   = xgb_vals_t2[xgb_valid]
)

xgb_transition |>
  as.data.frame.matrix() |>
  kable(caption = "XGBoost — Transition Matrix (pixel counts)") |>
  kable_styling(bootstrap_options = c("striped", "hover"))
```

XGBoost — Transition Matrix (pixel counts)

	built	cropland	fallow	open	water
built	131813	127788	116793	102098	17996
cropland	9603	202231	191797	59585	7589
fallow	19206	198777	272638	35830	10705
open	47028	35474	48954	444222	2511
water	358	4531	359	283	220539

Interpretation: The XGBoost transition matrix can be compared directly with the Random Forest transition matrix from the original analysis. Differences in detected transitions reflect how the two classifiers treat ambiguous pixels. Consistent transitions across both methods are more trustworthy, while discrepancies highlight uncertain areas.

5 SVM Classification

5.1 Model Specification

Support Vector Machines with a Radial Basis Function (RBF) kernel map spectral data into a higher-dimensional space where classes become linearly separable. SVMs are particularly effective when:

- The number of features is moderate relative to sample size
- Class boundaries are non-linear in the original feature space
- Training data is limited (SVMs maximize the margin between classes)

Unlike tree-based methods, SVMs require feature scaling, which we handle via `step_normalize()`.

```
svm_spec <- svm_rbf(cost = 10, rbf_sigma = 0.1) |>
  set_engine("kernlab") |>
  set_mode("classification")

# SVM requires normalized features
svm_recipe <- recipe(class ~ ., data = train_t1) |>
  step_normalize(all_predictors())

svm_wf <- workflow() |>
  add_recipe(svm_recipe) |>
  add_model(svm_spec) |>
  fit(train_t1)

svm_wf
```

```
## ══ Workflow [trained] ══════════════════════════════════════════════════════
## Preprocessor: Recipe
## Model: svm_rbf()
##
## ── Preprocessor ────────────────────────────────────────────────────────────
## 1 Recipe Step
##
## • step_normalize()
##
## ── Model ───────────────────────────────────────────────────────────────────
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 10
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.1
##
## Number of Support Vectors : 9429
##
## Objective Function Value : -9818.556 -5156.783 -15626.95 -34.0963 -61621.35 -1101.865 -6.6055 -846.296 -8.2058 -
5.0551
## Training error : 0.065567
## Probability model included.
```

Interpretation: The cost parameter (C = 10) controls the trade-off between maximizing the margin and minimizing classification errors. Higher cost penalizes misclassifications more heavily. The RBF sigma parameter controls how far the influence of a single training example reaches; smaller values mean each point has a more local influence. Feature normalization (`step_normalize`) is essential for SVM because the algorithm is sensitive to feature scales.

5.2 SVM Accuracy Assessment

```
svm_preds <- augment(svm_wf, test_t1)

cat("=== SVM (RBF) — Test Set ===\n")
```

```
## === SVM (RBF) — Test Set ===
```

```
conf_mat(svm_preds, truth = class, estimate = .pred_class)
```

```
##           Truth
## Prediction built cropland fallow open water
##   built     1253       36     13   19     0
##   cropland    52     1254    103    0     0
##   fallow      36      833   2804    4     0
##   open       213        3      0 5580     0
##   water        0        0      0    0  8448
```

```
svm_test_metrics <- svm_preds |>
  metrics(truth = class, estimate = .pred_class) |>
  mutate(model = "SVM (RBF)")

svm_test_metrics |>
  select(model, .metric, .estimate) |>
  kable(caption = "SVM (RBF) — Test Set Metrics", digits = 4) |>
  kable_styling(bootstrap_options = c("striped", "hover"), full_width = FALSE)
```

SVM (RBF) — Test Set Metrics

model	.metric	.estimate
SVM (RBF)	accuracy	0.9365
SVM (RBF)	kap	0.9117

```
svm_cv <- workflow() |>
  add_recipe(recipe(class ~ ., data = extract_t1) |> step_normalize(all_predictors())) |>
  add_model(svm_spec) |>
  fit_resamples(resamples = cv_folds_t1, metrics = metric_set(accuracy, kap))

svm_cv_metrics <- collect_metrics(svm_cv) |> mutate(model = "SVM (RBF)")

svm_cv_metrics |>
  select(model, .metric, mean, std_err) |>
  kable(caption = "SVM (RBF) — 5-Fold CV Metrics", digits = 4) |>
  kable_styling(bootstrap_options = c("striped", "hover"), full_width = FALSE)
```

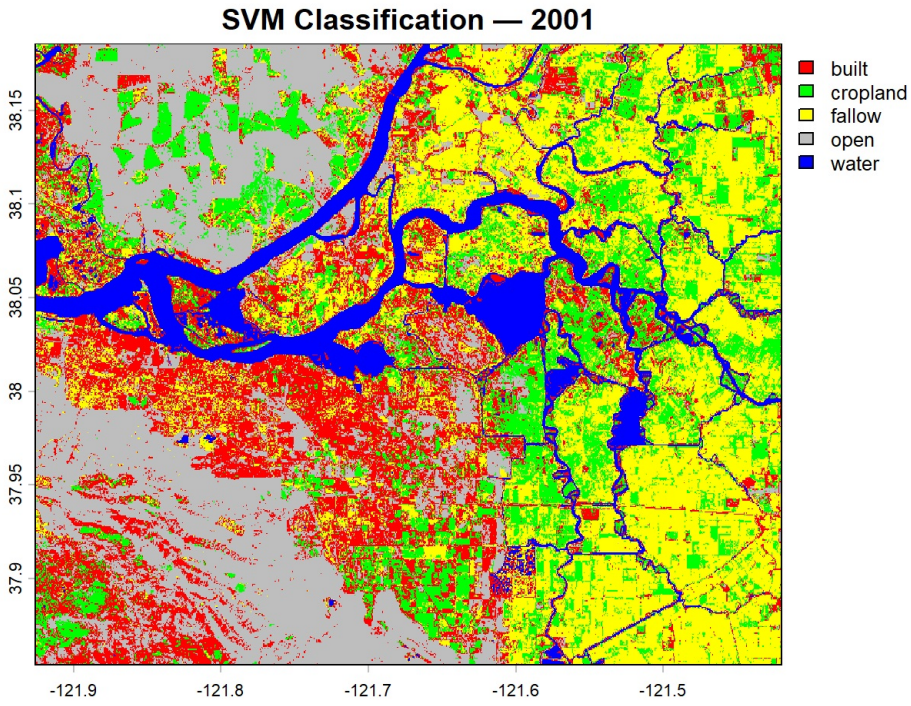
SVM (RBF) — 5-Fold CV Metrics

model	.metric	mean	std_err
SVM (RBF)	accuracy	0.9355	0.0010
SVM (RBF)	kap	0.9103	0.0015

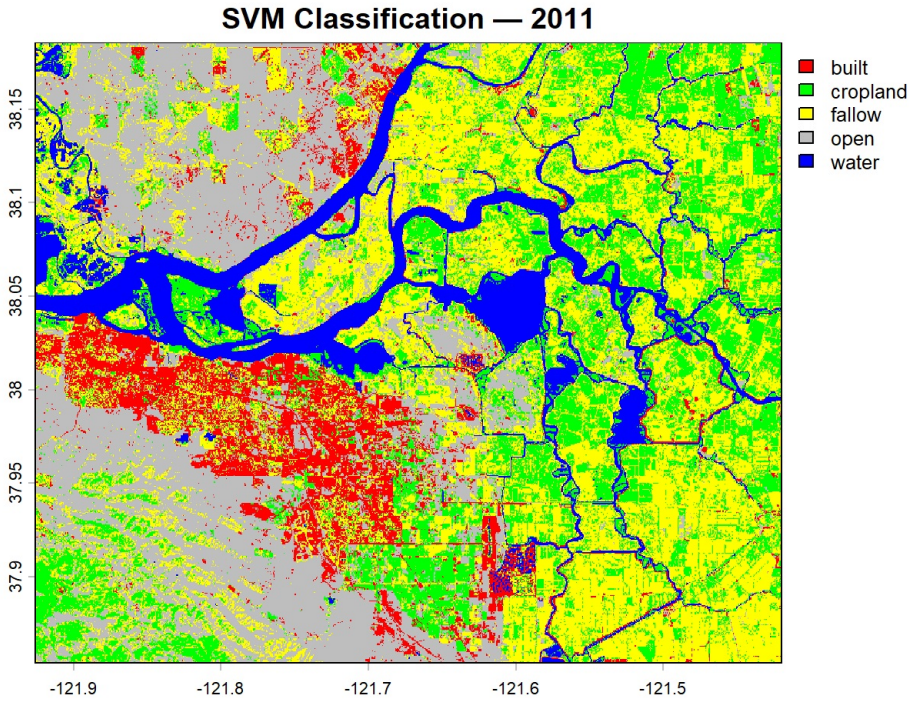
Interpretation: SVM performance should be compared with both Random Forest and XGBoost. SVMs often perform well with limited training data because the decision boundary depends only on support vectors (the most informative samples near class boundaries), rather than all training points. However, SVM classification of large rasters is computationally slower than tree-based methods.

5.3 Classify Rasters with SVM

```
# Train SVM for Time 2
svm_wf_t2 <- workflow() |>
  add_recipe(recipe(class ~ ., data = train_t2) |> step_normalize(all_predictors())) |>
  add_model(svm_spec) |>
  fit(train_t2)

# Classify both time periods
classified_svm_t1 <- terra::predict(raster_t1, svm_wf, na.rm = TRUE)
classified_svm_t2 <- terra::predict(raster_t2, svm_wf_t2, na.rm = TRUE)

par(mfrow = c(1, 1))
plot(classified_svm_t1, main = "SVM Classification — 2001",
     col = class_colors, type = "classes")
```



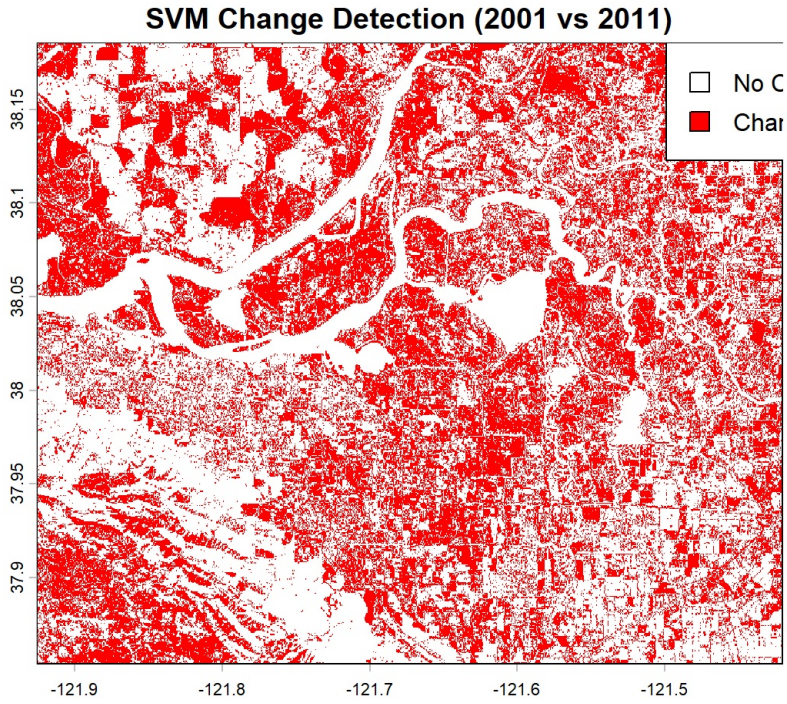

```
plot(classified_svm_t2, main = "SVM Classification — 2011",
     col = class_colors, type = "classes")
```




```
par(mfrow = c(1, 1))
```

5.4 SVM Change Detection

```
svm_change <- classified_svm_t1 != classified_svm_t2

plot(svm_change, main = "SVM Change Detection (2001 vs 2011)",
     col = c("white", "red"), legend = FALSE)
legend("topright", legend = c("No Change", "Change"),
       fill = c("white", "red"))
```



```
svm_vals_t1 <- values(classified_svm_t1) |> as.vector()
svm_vals_t2 <- values(classified_svm_t2) |> as.vector()
svm_vals_t1 <- class_names[svm_vals_t1]
svm_vals_t2 <- class_names[svm_vals_t2]

svm_valid <- !is.na(svm_vals_t1) & !is.na(svm_vals_t2)

svm_transition <- table(
  "From (2001)" = svm_vals_t1[svm_valid],
  "To (2011)"   = svm_vals_t2[svm_valid]
)

svm_transition |>
  as.data.frame.matrix() |>
  kable(caption = "SVM — Transition Matrix (pixel counts)") |>
  kable_styling(bootstrap_options = c("striped", "hover"))
```

SVM — Transition Matrix (pixel counts)

	built	cropland	fallow	open	water
built	118068	126033	144557	73213	13225
cropland	15791	147312	162161	57208	3667
fallow	19211	184215	402221	26234	3461
open	49469	25455	70450	431998	2068
water	568	8120	670	142	223191

Interpretation: Compare the SVM transition matrix with both Random Forest and XGBoost versions. SVMs may classify boundary pixels differently because the decision boundary geometry differs fundamentally from tree-based splits. Transitions that appear consistently across all three methods represent the most robust change detections.

6 Principal Component Analysis of Spectral Change

6.1 Motivation

Instead of classifying each date independently, PCA on the combined 12-band stack (6 bands from each date) can reveal:

- **Spectral change components**: PCs that capture the variance between dates highlight change patterns
- **Dimensionality reduction**: Compress 12 bands into a few informative components
- **Change visualization**: Map the dominant change patterns spatially without requiring training data

6.2 Create Combined 12-Band Stack

```
# Combine both dates into a single 12-band stack
band_names_t1 <- paste0("T1_", names(raster_t1))
band_names_t2 <- paste0("T2_", names(raster_t2))

combined_stack <- c(raster_t1, raster_t2)
names(combined_stack) <- c(band_names_t1, band_names_t2)

cat("Combined stack:\n")
```

```
## Combined stack:
```

```
combined_stack
```

```
## class       : SpatRaster
## size        : 1230, 1877, 12  (nrow, ncol, nlyr)
## resolution  : 0.0002694946, 0.0002694946  (x, y)
## extent      : -121.9258, -121.42, 37.85402, 38.1855  (xmin, xmax, ymin, ymax)
## coord. ref. : lon/lat WGS 84 (EPSG:4326)
## sources     : centralvalley-2001LE7.tif  (6 layers)
##               centralvalley-2011LT5.tif  (6 layers)
## names       : T1_B1, T1_B2, T1_B3, T1_B4, T1_B5, T1_B7, ...
```

Interpretation: The 12-band stack pairs each spectral band across both dates (e.g., T1_B1 with T2_B1). PCA will find linear combinations of these 12 bands that capture the most variance. Components driven by between-date differences will effectively highlight spectral change.

6.3 Compute PCA

```
# Sample pixels for PCA (full raster may be too large)
set.seed(7263)
n_pixels <- ncell(combined_stack)
sample_size <- min(50000, n_pixels)
sample_idx <- sample(n_pixels, sample_size)

# Extract sample values
sample_vals <- combined_stack[sample_idx]
sample_vals <- na.omit(as.data.frame(sample_vals))

cat("Sample size for PCA:", nrow(sample_vals), "\n")
```

```
## Sample size for PCA: 50000
```

```
# Run PCA
pca_result <- prcomp(sample_vals, center = TRUE, scale. = TRUE)

# Variance explained
pca_summary <- summary(pca_result)
pca_var <- data.frame(
  PC = paste0("PC", 1:12),
  Variance_Pct = round(pca_summary$importance[2, ] * 100, 2),
  Cumulative_Pct = round(pca_summary$importance[3, ] * 100, 2)
)

pca_var |>
  kable(caption = "PCA — Variance Explained by Component") |>
  kable_styling(bootstrap_options = c("striped", "hover"), full_width = FALSE)
```

PCA — Variance Explained by Component

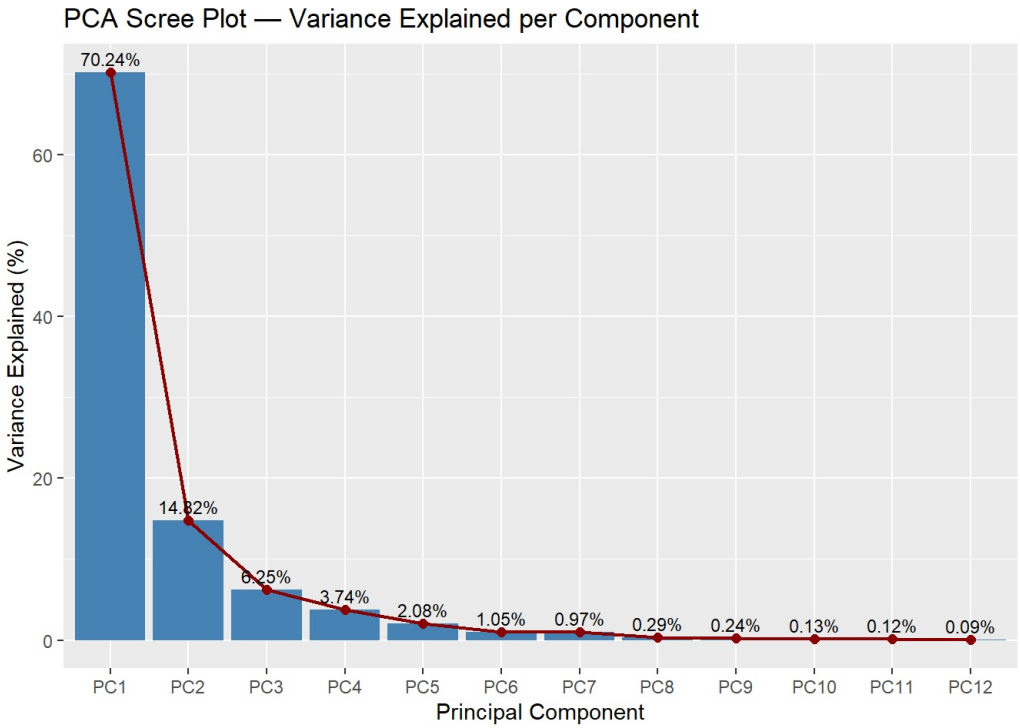
	PC	Variance_Pct	Cumulative_Pct
PC1	PC1	70.24	70.24
PC2	PC2	14.82	85.06
PC3	PC3	6.25	91.31
PC4	PC4	3.74	95.05
PC5	PC5	2.08	97.12
PC6	PC6	1.05	98.17
PC7	PC7	0.97	99.14
PC8	PC8	0.29	99.43
PC9	PC9	0.24	99.67
PC10	PC10	0.13	99.79
PC11	PC11	0.12	99.91
PC12	PC12	0.09	100.00

Interpretation: The table shows how much of the total spectral variance each principal component captures:

- **PC1** typically captures overall brightness differences (high variance, common to both dates)
- **PC2–PC3** often capture vegetation vs. non-vegetation contrasts or sensor differences
- **Later PCs** may isolate subtle change signals or noise
- **Cumulative %**: The first 3–4 PCs usually capture >90% of total variance, meaning the remaining 8–9 bands contain mostly redundant information or noise

6.4 Scree Plot

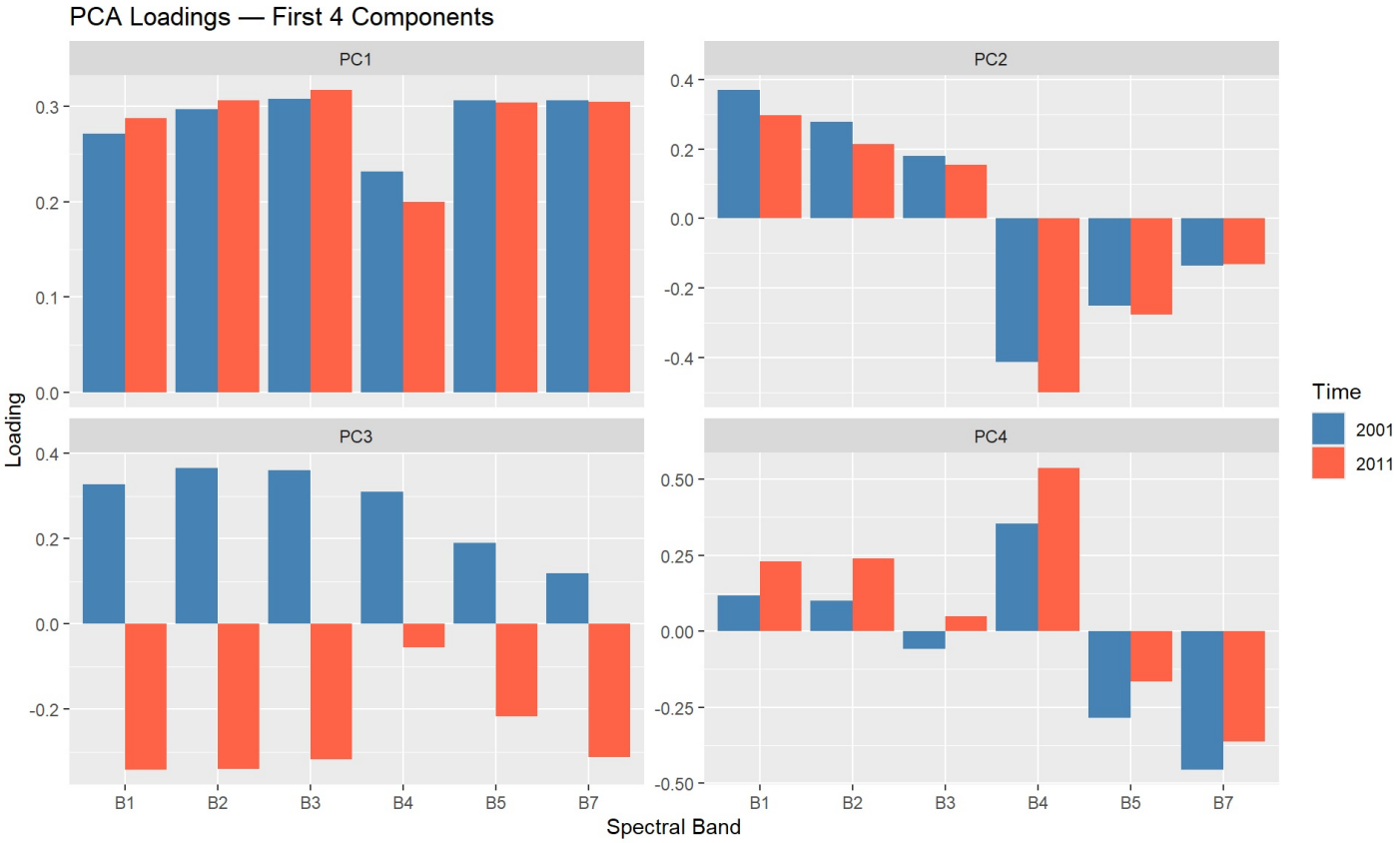
```
ggplot(pca_var, aes(x = reorder(PC, -Variance_Pct), y = Variance_Pct)) +
  geom_col(fill = "steelblue") +
  geom_line(aes(group = 1), color = "darkred", linewidth = 0.8) +
  geom_point(color = "darkred", size = 2) +
  labs(title = "PCA Scree Plot — Variance Explained per Component",
       x = "Principal Component", y = "Variance Explained (%)") +
  geom_text(aes(label = paste0(Variance_Pct, "%")), vjust = -0.5, size = 3)
```



Interpretation: The scree plot helps identify the “elbow” point where additional components contribute diminishing returns. Components before the elbow capture meaningful spectral patterns; those after it are primarily noise. This guides how many PCs to retain for visualization and analysis.

6.5 PCA Loadings

```
# Extract loadings for the first 4 PCs
loadings_df <- as.data.frame(pca_result$rotation[, 1:4]) |>
  rownames_to_column("Band") |>
  pivot_longer(cols = PC1:PC4, names_to = "PC", values_to = "Loading") |>
  mutate(
    Time = ifelse(str_starts(Band, "T1"), "2001", "2011"),
    Band_Name = str_remove(Band, "T[12]_")
  )

ggplot(loadings_df, aes(x = Band_Name, y = Loading, fill = Time)) +
  geom_col(position = "dodge") +
  facet_wrap(~PC, scales = "free_y") +
  labs(title = "PCA Loadings — First 4 Components",
       x = "Spectral Band", y = "Loading") +
  scale_fill_manual(values = c("2001" = "steelblue", "2011" = "tomato"))
```



Interpretation: The loading plot reveals what each PC represents:

- **Loadings with same sign for both dates**: The PC captures a spectral property common to both images (e.g., brightness, vegetation index)
- **Loadings with opposite signs between dates**: The PC captures **change** between 2001 and 2011 — this is the most informative for change detection
- **Band-specific patterns**: Which spectral bands drive each component (e.g., if PC2 is driven by NIR, it captures vegetation change)

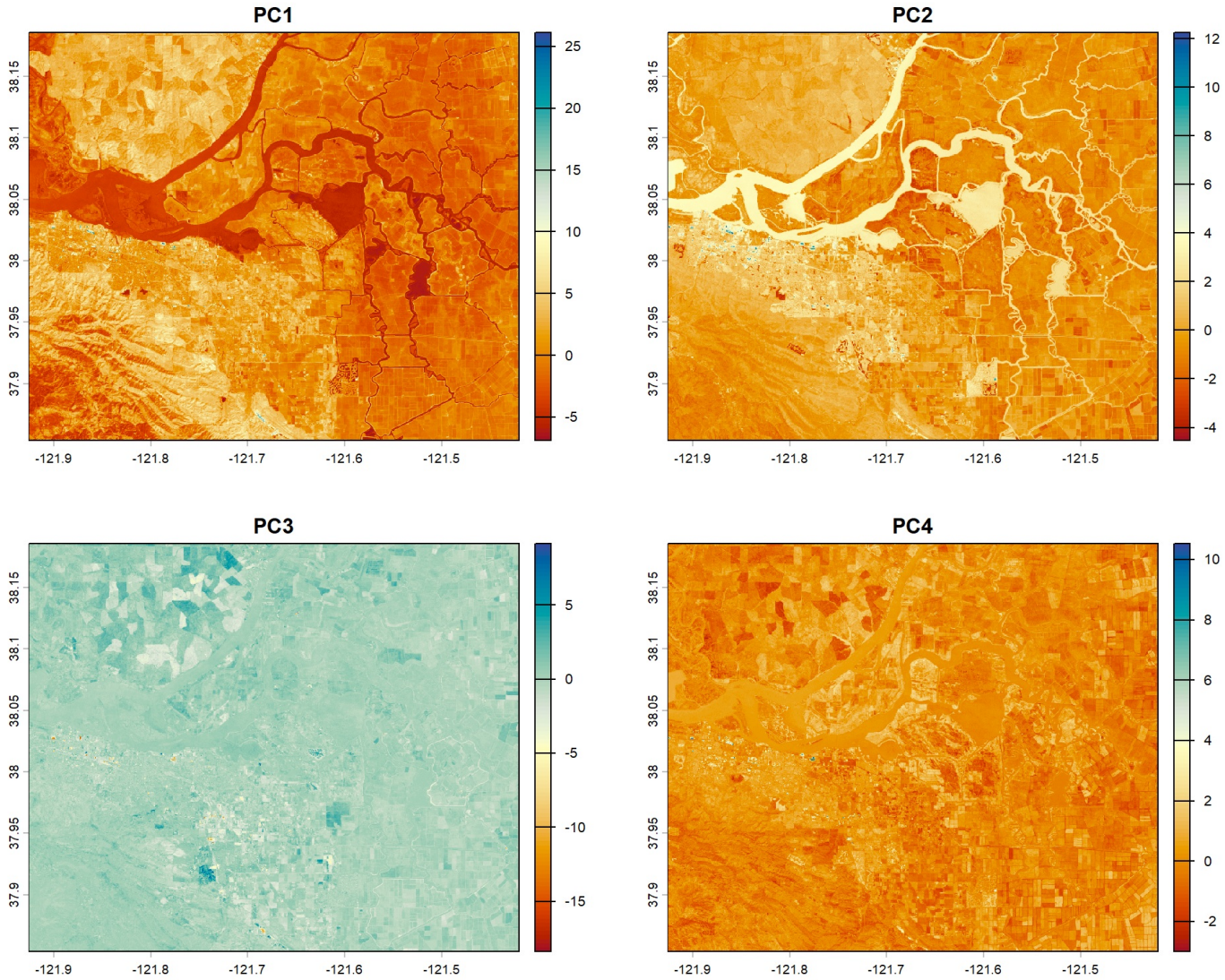
PCs with strong opposing loadings between T1 and T2 bands are the most useful change indicators.

6.6 Map PCA Components Spatially

```
# Project the full raster through PCA
pca_raster <- terra::predict(combined_stack, pca_result)
```

```
## |---------|---------|---------|---------|=========================================
```

```
# Plot first 4 PCs
par(mfrow = c(2, 2))
plot(pca_raster[[1]], main = "PC1", col = hcl.colors(100, "RdYlBu"))
plot(pca_raster[[2]], main = "PC2", col = hcl.colors(100, "RdYlBu"))
plot(pca_raster[[3]], main = "PC3", col = hcl.colors(100, "RdYlBu"))
plot(pca_raster[[4]], main = "PC4", col = hcl.colors(100, "RdYlBu"))
```




```
par(mfrow = c(1, 1))
```

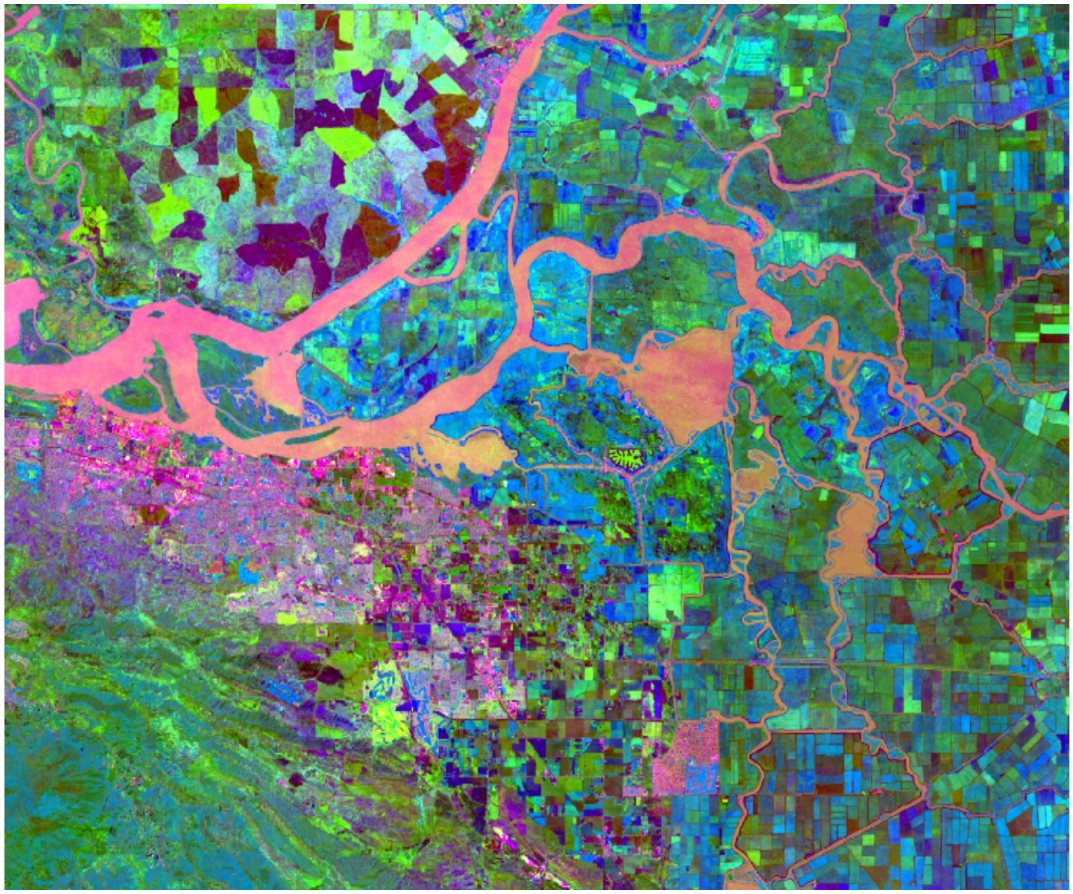
Interpretation: Each PC map shows the spatial distribution of that component's spectral pattern:

- **PC1**: Usually shows overall landscape structure (brightness/albedo)
- **PC2–PC3**: May highlight vegetation patterns, water bodies, or temporal change
- **Change-sensitive PCs**: Components identified from the loadings as having opposite-sign T1/T2 loadings will show spatial patterns of change — bright/dark areas correspond to locations of spectral change

Areas with extreme values in change-sensitive PCs are candidate change locations, identified purely from spectral patterns without requiring training data.

6.7 False Color PCA Composite

```
# Create a false color composite using PCs 1-3
plotRGB(pca_raster, r = 2, g = 3, b = 4, stretch = "lin",
        main = "PCA False Color Composite (PC2=R, PC3=G, PC4=B)")
```



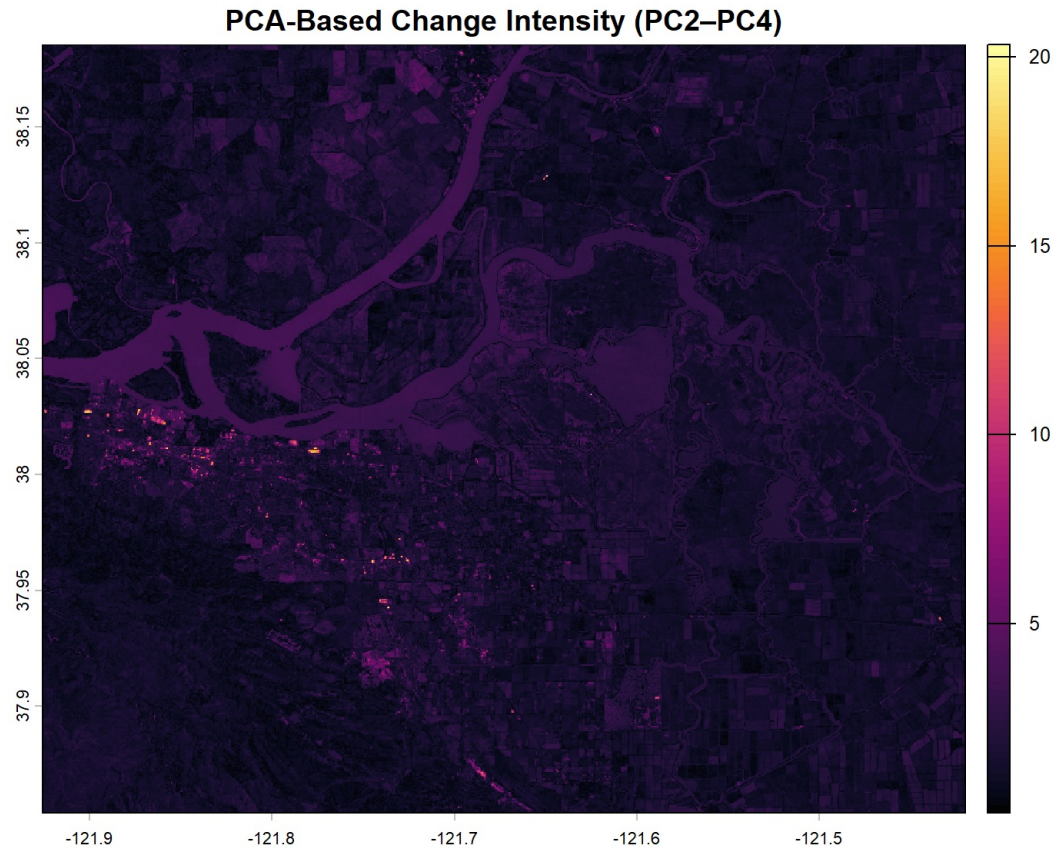
Interpretation: This false color composite assigns PCs 2, 3, and 4 to red, green, and blue channels (PC1 is excluded because it usually represents overall brightness rather than change). Different colors highlight different types of spectral change:

- **Distinct color clusters** correspond to different land cover types or change trajectories
- **Bright, saturated colors** indicate areas of strong spectral change
- **Neutral/grey tones** indicate stability (low variance in change-related components)

This unsupervised visualization can reveal change patterns that the supervised classifiers might miss, and is useful for identifying areas that warrant further investigation.

6.8 PCA-Based Change Intensity Map

```
# Compute change intensity as the Euclidean distance in PCA space
# using change-sensitive components (PC2 onward)
change_intensity <- sqrt(pca_raster[[2]]^2 + pca_raster[[3]]^2 + pca_raster[[4]]^2)
names(change_intensity) <- "Change_Intensity"

plot(change_intensity, main = "PCA-Based Change Intensity (PC2–PC4)",
     col = hcl.colors(100, "Inferno"))
```



Interpretation: The change intensity map combines the information from multiple change-sensitive PCs into a single continuous measure. Higher values (bright/yellow) indicate locations with greater spectral change between 2001 and 2011. Unlike the binary change maps from post-classification comparison, this is a **continuous** measure that:

- Does not require training data or classification
- Captures gradual change (e.g., vegetation degradation) not just categorical transitions
- Can be thresholded at different levels to define change vs. no-change at varying sensitivities

7 Model Comparison

7.1 Test Set Performance

```
# Combine all test set metrics
all_test_metrics <- bind_rows(rf_test_metrics, xgb_test_metrics, svm_test_metrics) |>
  select(model, .metric, .estimate) |>
  pivot_wider(names_from = .metric, values_from = .estimate)

all_test_metrics |>
  kable(caption = "Test Set Performance — All Models", digits = 4) |>
  kable_styling(bootstrap_options = c("striped", "hover"), full_width = FALSE)
```

Test Set Performance — All Models

model	accuracy	kap
Random Forest	0.9802	0.9726
XGBoost	0.9782	0.9698
SVM (RBF)	0.9365	0.9117

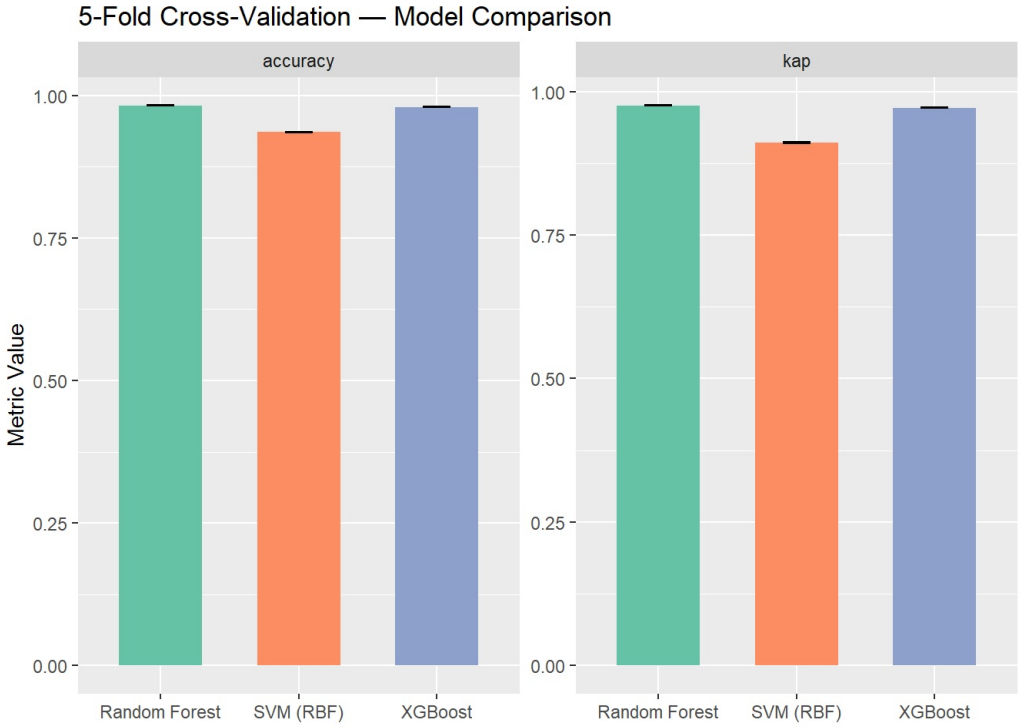
7.2 Cross-Validation Performance

```
all_cv_metrics <- bind_rows(rf_cv_metrics, xgb_cv_metrics, svm_cv_metrics) |>
  select(model, .metric, mean, std_err)

all_cv_metrics |>
  kable(caption = "5-Fold CV Performance — All Models", digits = 4) |>
  kable_styling(bootstrap_options = c("striped", "hover"), full_width = FALSE)
```

5-Fold CV Performance — All Models

model	.metric	mean	std_err
Random Forest	accuracy	0.9823	0.0003
Random Forest	kap	0.9755	0.0004
XGBoost	accuracy	0.9796	0.0005
XGBoost	kap	0.9717	0.0007
SVM (RBF)	accuracy	0.9355	0.0010
SVM (RBF)	kap	0.9103	0.0015

```
all_cv_metrics |>
  ggplot(aes(x = model, y = mean, fill = model)) +
  geom_col(width = 0.6) +
  geom_errorbar(aes(ymin = mean - std_err, ymax = mean + std_err), width = 0.2) +
  facet_wrap(~.metric, scales = "free_y") +
  labs(title = "5-Fold Cross-Validation — Model Comparison",
       x = NULL, y = "Metric Value") +
  theme(legend.position = "none") +
  scale_fill_brewer(palette = "Set2")
```



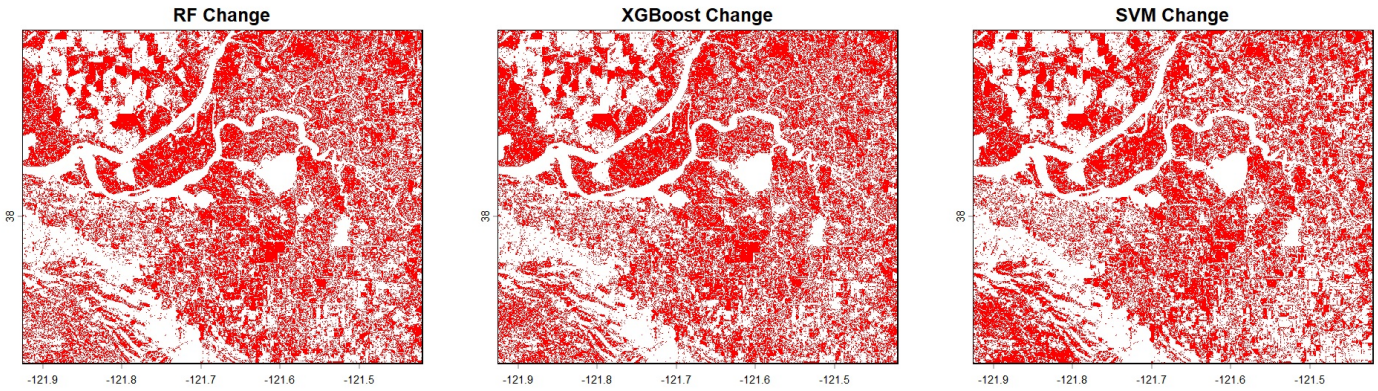
Interpretation: This comparison reveals the relative strengths of each method:

- **Random Forest**: Robust baseline; handles high-dimensional data well without preprocessing
- **XGBoost**: May show incremental improvement due to sequential error correction; strongest when there are complex class boundaries
- **SVM (RBF)**: Particularly effective with limited training samples; may outperform tree methods when classes are spectrally similar but separable in higher-dimensional space

The error bars (standard error) indicate how variable performance is across folds. Overlapping error bars suggest no meaningful difference between models.

7.3 Change Detection Comparison

```
# Load original RF classifications for comparison
rf_wf_t2 <- workflow() |>
  add_recipe(recipe(class ~ ., data = train_t2)) |>
  add_model(rf_spec) |>
  fit(train_t2)

classified_rf_t1 <- terra::predict(raster_t1, rf_wf, na.rm = TRUE)
classified_rf_t2 <- terra::predict(raster_t2, rf_wf_t2, na.rm = TRUE)
rf_change <- classified_rf_t1 != classified_rf_t2

# Side-by-side change maps
par(mfrow = c(1, 3))
plot(rf_change, main = "RF Change",
     col = c("white", "red"), legend = FALSE)
plot(xgb_change, main = "XGBoost Change",
     col = c("white", "red"), legend = FALSE)
plot(svm_change, main = "SVM Change",
     col = c("white", "red"), legend = FALSE)
```




```
par(mfrow = c(1, 1))
```

```
# Count changed pixels per method
change_counts <- tibble(
  Model = c("Random Forest", "XGBoost", "SVM (RBF)"),
  Changed_Pixels = c(
    sum(values(rf_change) == 1, na.rm = TRUE),
    sum(values(xgb_change) == 1, na.rm = TRUE),
    sum(values(svm_change) == 1, na.rm = TRUE)
  ),
  Total_Pixels = c(
    sum(!is.na(values(rf_change))),
    sum(!is.na(values(xgb_change))),
    sum(!is.na(values(svm_change)))
  )
) |>
  mutate(Change_Pct = round(Changed_Pixels / Total_Pixels * 100, 2))

change_counts |>
  kable(caption = "Change Detection Summary — All Models") |>
  kable_styling(bootstrap_options = c("striped", "hover"), full_width = FALSE)
```

Change Detection Summary — All Models

Model	Changed_Pixels	Total_Pixels	Change_Pct
Random Forest	1023163	2308708	44.32
XGBoost	1037265	2308708	44.93
SVM (RBF)	985918	2308708	42.70

Interpretation: This comparison reveals how different classifiers affect change detection outcomes:

- **Higher change percentage** may indicate a more sensitive classifier (detecting real changes) or one that introduces more classification noise
- **Agreement across methods** strengthens confidence in detected changes
- **Disagreement** highlights uncertain areas where classifier choice affects conclusions

In practice, an ensemble approach (majority vote across classifiers) can produce more robust change maps by reducing classifier-specific errors.

8 Summary

This analysis compared three classification approaches and an unsupervised method for land cover change detection:

Method	Type	Key Strengths
Random Forest	Supervised (ensemble)	Robust baseline, no preprocessing needed
XGBoost	Supervised (boosting)	Sequential error correction, often highest accuracy
SVM (RBF)	Supervised (kernel)	Effective with limited samples, smooth boundaries
PCA	Unsupervised	No training data needed, continuous change measure

Key takeaways:

- **Supervised methods** (RF, XGBoost, SVM) produce categorical change maps but require training data and are subject to error propagation
- **PCA** provides a complementary unsupervised perspective on spectral change intensity without requiring labeled samples
- **Model agreement** across multiple classifiers strengthens confidence in detected changes
- **No single method is universally best** — the optimal choice depends on data characteristics, training sample availability, and the specific change detection question