KNN_Classification

May 28, 2026

[30]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report
```

[31]:
```
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = data.target

print("Shape:", X.shape)
print("Target classes:", np.unique(y))
```

```
Shape: (569, 30)
Target classes: [0 1]
```

[33]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[34]:
```
knn_model = KNeighborsClassifier(n_neighbors=5, metric="minkowski", p=2)
knn_model.fit(X_train_scaled, y_train)
```

[34]:
```
KNeighborsClassifier()
```

[35]:
```
y_pred = knn_model.predict(X_test_scaled)

acc = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
```

```
print("Accuracy:", round(acc, 4))
print("Confusion Matrix:\n", cm)
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.9561
Confusion Matrix:
 [[39  3]
 [ 2 70]]
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.93      0.94        42
           1       0.96      0.97      0.97        72

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

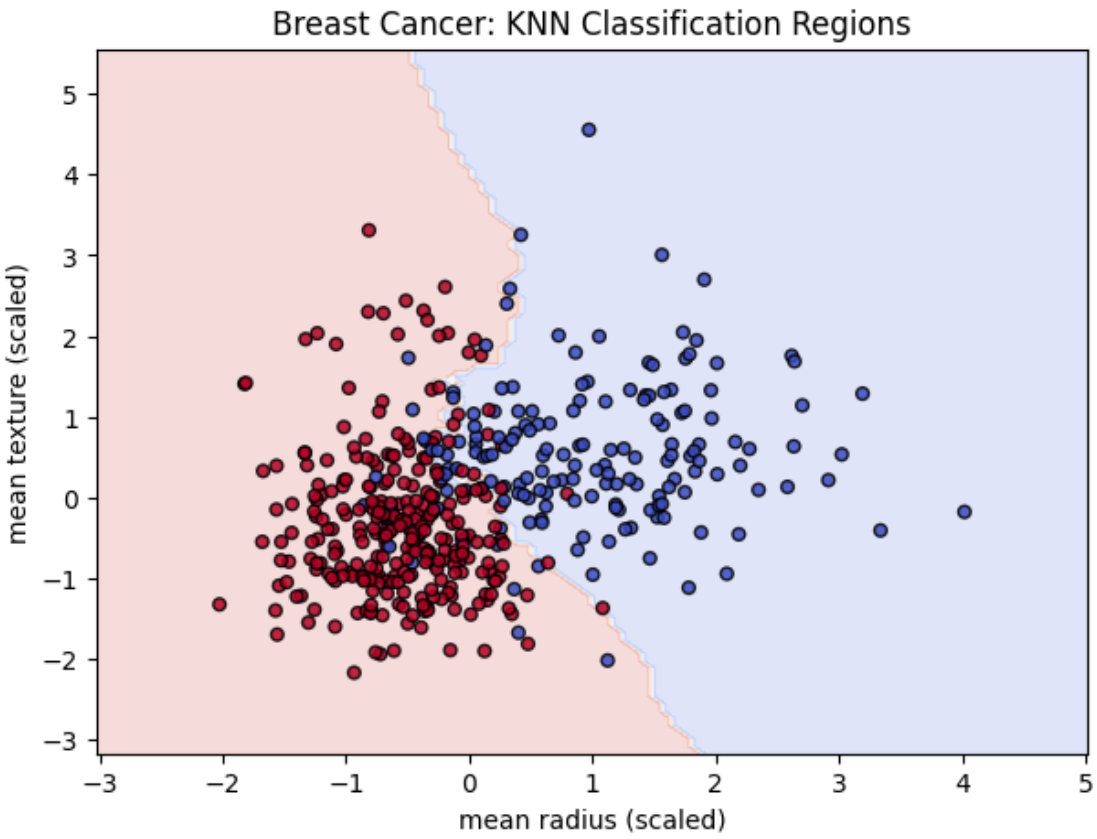
[36]:
```
for k in [1, 3, 5, 7, 11, 15]:
    model = KNeighborsClassifier(n_neighbors=k, metric="minkowski", p=2)
    model.fit(X_train_scaled, y_train)
    pred = model.predict(X_test_scaled)
    print(f"k={k}: accuracy={accuracy_score(y_test, pred):.4f}")
```

```
k=1: accuracy=0.9386
k=3: accuracy=0.9825
k=5: accuracy=0.9561
k=7: accuracy=0.9737
k=11: accuracy=0.9737
k=15: accuracy=0.9737
```

1 Graphs - for KNN

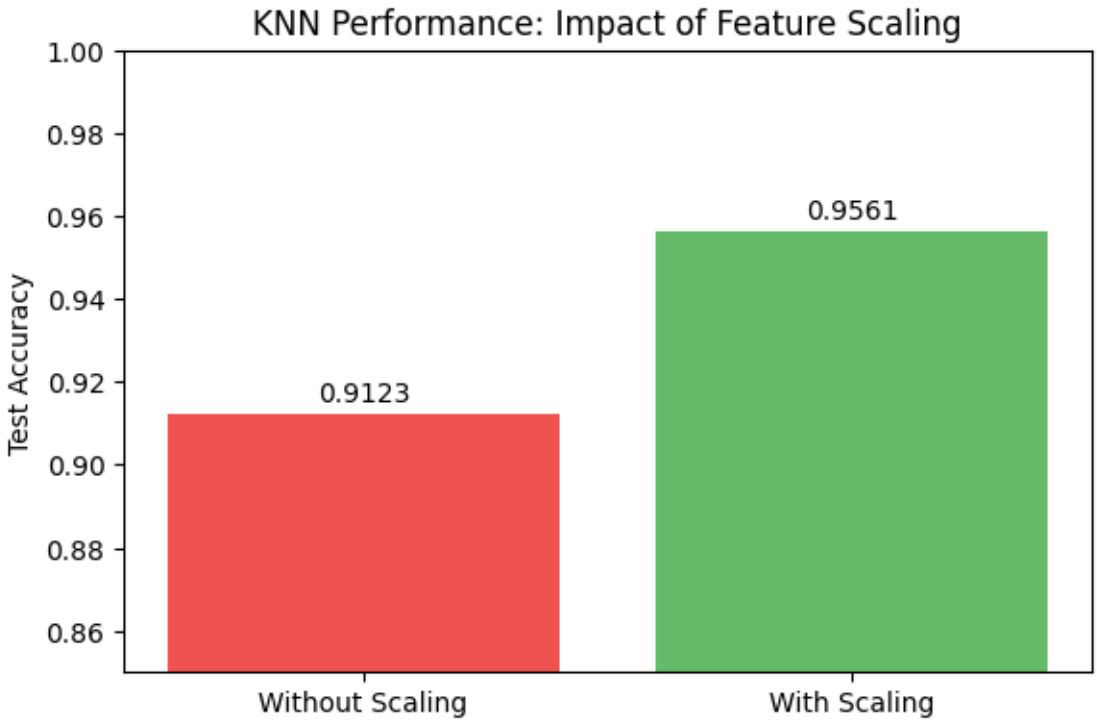
[37]:
```
# Classification visual using two breast-cancer features
from sklearn.inspection import DecisionBoundaryDisplay

feature_names = list(data.feature_names)

def find_feature_index(name, all_names):
    # Robust match for variations like spaces vs underscores or case changes
    key = name.strip().lower().replace("_", " ")
    normalized = [n.strip().lower().replace("_", " ") for n in all_names]
    if key in normalized:
        return normalized.index(key)
    raise ValueError(f"Feature '{name}' not found. Available: {all_names[:5]} ..
 ↪.")
```

```
ix = find_feature_index("mean radius", feature_names)
iy = find_feature_index("mean texture", feature_names)

feat_x = feature_names[ix]
feat_y = feature_names[iy]

X2 = data.data[:, [ix, iy]]
y2 = data.target

X2_train, X2_test, y2_train, y2_test = train_test_split(
    X2, y2, test_size=0.2, random_state=42, stratify=y2
)

scaler2 = StandardScaler()
X2_train_scaled = scaler2.fit_transform(X2_train)
X2_test_scaled = scaler2.transform(X2_test)

knn2 = KNeighborsClassifier(n_neighbors=9, metric="minkowski", p=2)
knn2.fit(X2_train_scaled, y2_train)

fig, ax = plt.subplots(figsize=(7, 5))
DecisionBoundaryDisplay.from_estimator(
    knn2, X2_train_scaled, response_method="predict", cmap="coolwarm", alpha=0.
 ↪18, ax=ax
)
ax.scatter(
    X2_train_scaled[:, 0],
    X2_train_scaled[:, 1],
    c=y2_train,
    cmap="coolwarm",
    s=24,
    edgecolor="k",
    alpha=0.85
)
ax.set_xlabel(feat_x + " (scaled)")
ax.set_ylabel(feat_y + " (scaled)")
ax.set_title("Breast Cancer: KNN Classification Regions")
plt.show()
```



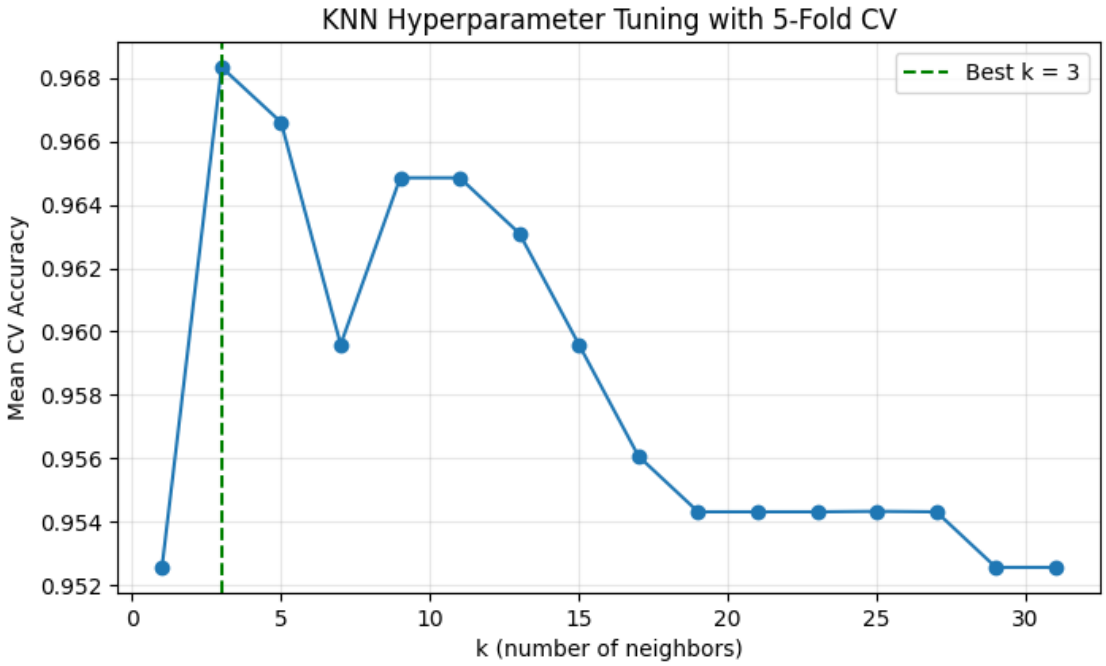
[38]:
```
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_breast_cancer

data = load_breast_cancer()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Model without scaling
knn_raw = KNeighborsClassifier(n_neighbors=5)
knn_raw.fit(X_train, y_train)
pred_raw = knn_raw.predict(X_test)
acc_raw = accuracy_score(y_test, pred_raw)
```

```
# Model with scaling
knn_scaled = make_pipeline(StandardScaler(),␣
 ↪KNeighborsClassifier(n_neighbors=5))
knn_scaled.fit(X_train, y_train)
pred_scaled = knn_scaled.predict(X_test)
acc_scaled = accuracy_score(y_test, pred_scaled)

plt.figure(figsize=(6.5, 4.2))
plt.bar(["Without Scaling", "With Scaling"], [acc_raw, acc_scaled],␣
 ↪color=["#ef5350", "#66bb6a"])
plt.ylim(0.85, 1.00)
plt.ylabel("Test Accuracy")
plt.title("KNN Performance: Impact of Feature Scaling")
for i, v in enumerate([acc_raw, acc_scaled]):
    plt.text(i, v + 0.003, f"{v:.4f}", ha="center")
plt.show()
```

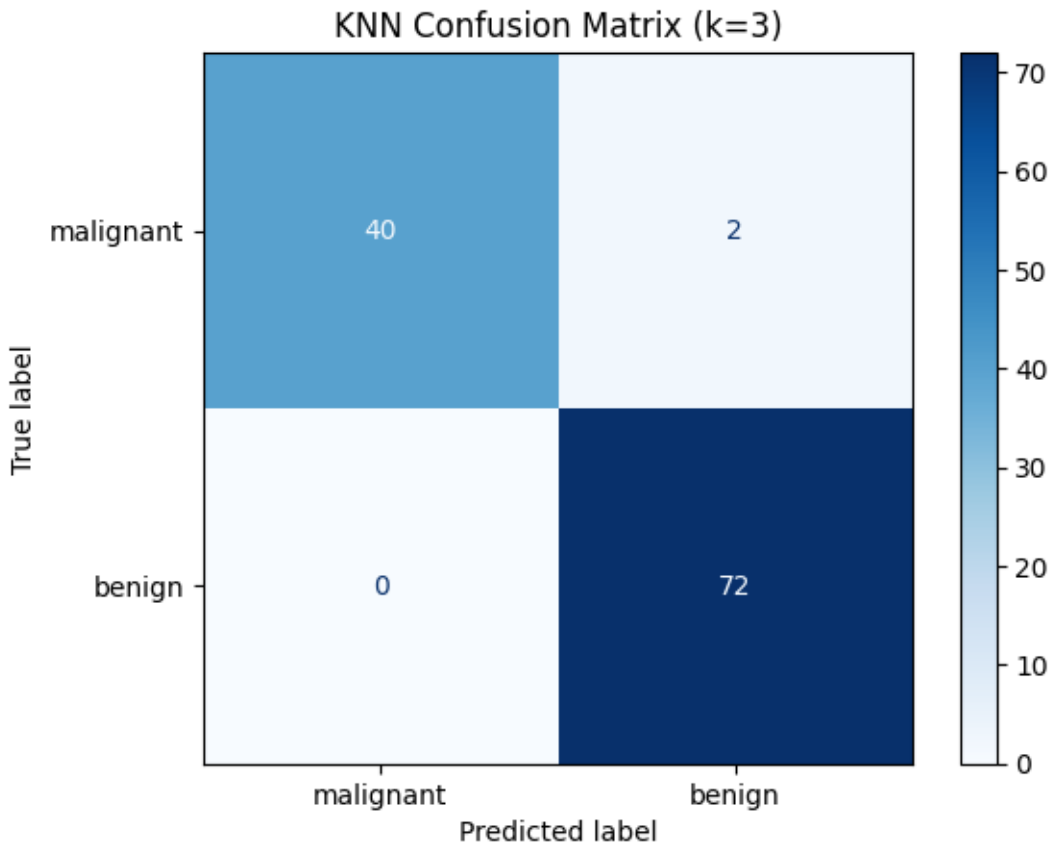


[39]:
```
from sklearn.model_selection import StratifiedKFold, cross_val_score

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```python
k_values = list(range(1, 32, 2))  # odd k values
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cv_scores = []

for k in k_values:
    model = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(model, X_scaled, y, cv=cv, scoring="accuracy")
    cv_scores.append(scores.mean())

best_idx = int(np.argmax(cv_scores))
best_k = k_values[best_idx]
best_score = cv_scores[best_idx]

plt.figure(figsize=(8, 4.5))
plt.plot(k_values, cv_scores, marker="o")
plt.axvline(best_k, color="green", linestyle="--", label=f"Best k = {best_k}")
plt.xlabel("k (number of neighbors)")
plt.ylabel("Mean CV Accuracy")
plt.title("KNN Hyperparameter Tuning with 5-Fold CV")
plt.legend()
plt.grid(alpha=0.3)
plt.show()
```



[40]:
```
from sklearn.metrics import ConfusionMatrixDisplay, classification_report

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

k_for_eval = best_k if "best_k" in globals() else 9
knn_best = KNeighborsClassifier(n_neighbors=k_for_eval)
knn_best.fit(X_train_scaled, y_train)
y_pred = knn_best.predict(X_test_scaled)

ConfusionMatrixDisplay.from_predictions(
    y_test, y_pred, display_labels=data.target_names, cmap="Blues",␣
 ↪values_format="d"
)
plt.title(f"KNN Confusion Matrix (k={k_for_eval})")
plt.show()

print(classification_report(y_test, y_pred, target_names=data.target_names))
```

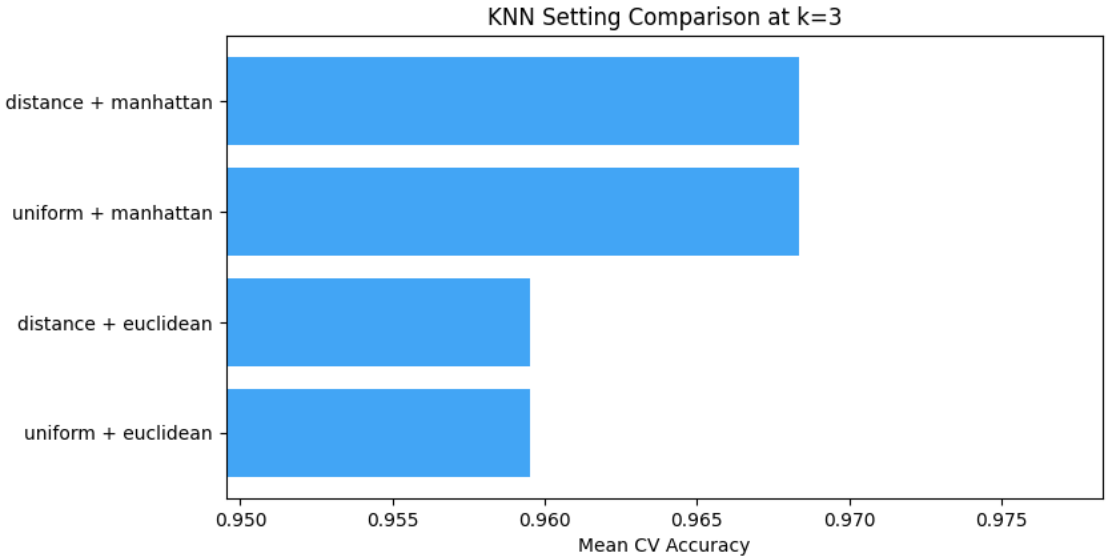


	precision	recall	f1-score	support
malignant	1.00	0.95	0.98	42
benign	0.97	1.00	0.99	72
accuracy			0.98	114
macro avg	0.99	0.98	0.98	114
weighted avg	0.98	0.98	0.98	114

[41]:
```
import pandas as pd

configs = [
    {"weights": "uniform", "p": 2, "label": "uniform + euclidean"},
    {"weights": "distance", "p": 2, "label": "distance + euclidean"},
    {"weights": "uniform", "p": 1, "label": "uniform + manhattan"},
    {"weights": "distance", "p": 1, "label": "distance + manhattan"},
]

rows = []
```

```python
k_for_compare = best_k if "best_k" in globals() else 9
for cfg in configs:
    pipe = make_pipeline(
        StandardScaler(),
        KNeighborsClassifier(
            n_neighbors=k_for_compare,
            weights=cfg["weights"],
            metric="minkowski",
            p=cfg["p"]
        )
    )
    score = cross_val_score(pipe, X, y, cv=5, scoring="accuracy").mean()
    rows.append({"setting": cfg["label"], "cv_accuracy": score})

result = pd.DataFrame(rows).sort_values("cv_accuracy", ascending=False)
print(result)

plt.figure(figsize=(8.5, 4.5))
plt.barh(result["setting"], result["cv_accuracy"], color="#42a5f5")
plt.xlabel("Mean CV Accuracy")
plt.title(f"KNN Setting Comparison at k={k_for_compare}")
plt.gca().invert_yaxis()
plt.xlim(result["cv_accuracy"].min() - 0.01, result["cv_accuracy"].max() + 0.01)
plt.show()
```

```
                setting  cv_accuracy
3  distance + manhattan     0.968343
2   uniform + manhattan     0.968343
1  distance + euclidean     0.959525
0   uniform + euclidean     0.959525
```



[ ]: