syde_572_project

November 25, 2025

1 Breast Cancer Diagnostic Dataset

SYDE 572 Final Project

Matthew Keller: 20931412 Leo Lau: 20933924

For this project, we are investigating the Breast Cancer Wisconsin (Diagnostic) dataset. The features in this dataset are computed from the characteristics of the cell nuclei of a fine needle aspirate of a breast mass. There are 30 features within the dataset along with ID numbers of the specific samples and the Diagnosis of the breast mass.

1.1 Preprocessing

Firstly, we want to make sure the dataset is complete. We will inspect the dataset to see if it has the number of features it claims to have and is not missing any data. We also want to check if there is any categorical data that we will need to convert.

[12]:
```
import pandas as pd
import seaborn as sns
from ucimlrepo import fetch_ucirepo
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import numpy as np
from sklearn.manifold import TSNE
import umap
from sklearn.manifold import Isomap
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import time
import os
import tracemalloc
```

[13]:
```
USE_KAGGLE = True

# The UCI Database was down but we were able to find the same dataset on Kaggle␣
 ↪uploaded by the same people. The only difference is the columns names are␣
 ↪more descriptive instead of mean1, mean2
```

```
if USE_KAGGLE:
    import kagglehub

    path = kagglehub.dataset_download("uciml/breast-cancer-wisconsin-data")
    df = pd.read_csv(os.path.join(path, "data.csv"))
    df = df.drop(columns=["Unnamed: 32"], axis=1)
    X = df.drop(columns=["id", "diagnosis"])
    y = df[["diagnosis"]]
    y["Diagnosis"] = y["diagnosis"]
    y = y.drop(columns=["diagnosis"])

else:
    lung_cancer = fetch_ucirepo(id=17)
    X = lung_cancer.data.features
    y = lung_cancer.data.targets
    print(lung_cancer.metadata)
    print(lung_cancer.variables)
```

```
/var/folders/4m/l4mr8hbd6q7_2qryv48yb8z00000gn/T/ipykernel_91943/2401575608.py:1
1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  y["Diagnosis"] = y["diagnosis"]
```

[14]:
```
X.describe()
```

[14]:

	radius_mean	texture_mean	perimeter_mean	area_mean	\
count	569.000000	569.000000	569.000000	569.000000	
mean	14.127292	19.289649	91.969033	654.889104	
std	3.524049	4.301036	24.298981	351.914129	
min	6.981000	9.710000	43.790000	143.500000	
25%	11.700000	16.170000	75.170000	420.300000	
50%	13.370000	18.840000	86.240000	551.100000	
75%	15.780000	21.800000	104.100000	782.700000	
max	28.110000	39.280000	188.500000	2501.000000	

	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	\
count	569.000000	569.000000	569.000000	569.000000	
mean	0.096360	0.104341	0.088799	0.048919	
std	0.014064	0.052813	0.079720	0.038803	
min	0.052630	0.019380	0.000000	0.000000	
25%	0.086370	0.064920	0.029560	0.020310	
50%	0.095870	0.092630	0.061540	0.033500	
75%	0.105300	0.130400	0.130700	0.074000	
max	0.163400	0.345400	0.426800	0.201200	

```
       symmetry_mean  fractal_dimension_mean  …  radius_worst  \
count     569.000000              569.000000  …    569.000000
mean        0.181162                0.062798  …     16.269190
std         0.027414                0.007060  …      4.833242
min         0.106000                0.049960  …      7.930000
25%         0.161900                0.057700  …     13.010000
50%         0.179200                0.061540  …     14.970000
75%         0.195700                0.066120  …     18.790000
max         0.304000                0.097440  …     36.040000

       texture_worst  perimeter_worst   area_worst  smoothness_worst  \
count     569.000000       569.000000   569.000000        569.000000
mean       25.677223       107.261213   880.583128          0.132369
std         6.146258        33.602542   569.356993          0.022832
min        12.020000        50.410000   185.200000          0.071170
25%        21.080000        84.110000   515.300000          0.116600
50%        25.410000        97.660000   686.500000          0.131300
75%        29.720000       125.400000  1084.000000          0.146000
max        49.540000       251.200000  4254.000000          0.222600

       compactness_worst  concavity_worst  concave points_worst  \
count         569.000000       569.000000            569.000000
mean            0.254265         0.272188              0.114606
std             0.157336         0.208624              0.065732
min             0.027290         0.000000              0.000000
25%             0.147200         0.114500              0.064930
50%             0.211900         0.226700              0.099930
75%             0.339100         0.382900              0.161400
max             1.058000         1.252000              0.291000

       symmetry_worst  fractal_dimension_worst
count      569.000000               569.000000
mean         0.290076                 0.083946
std          0.061867                 0.018061
min          0.156500                 0.055040
25%          0.250400                 0.071460
50%          0.282200                 0.080040
75%          0.317900                 0.092080
max          0.663800                 0.207500

[8 rows x 30 columns]
```

[15]:
```
X.isna().sum().sum()
```

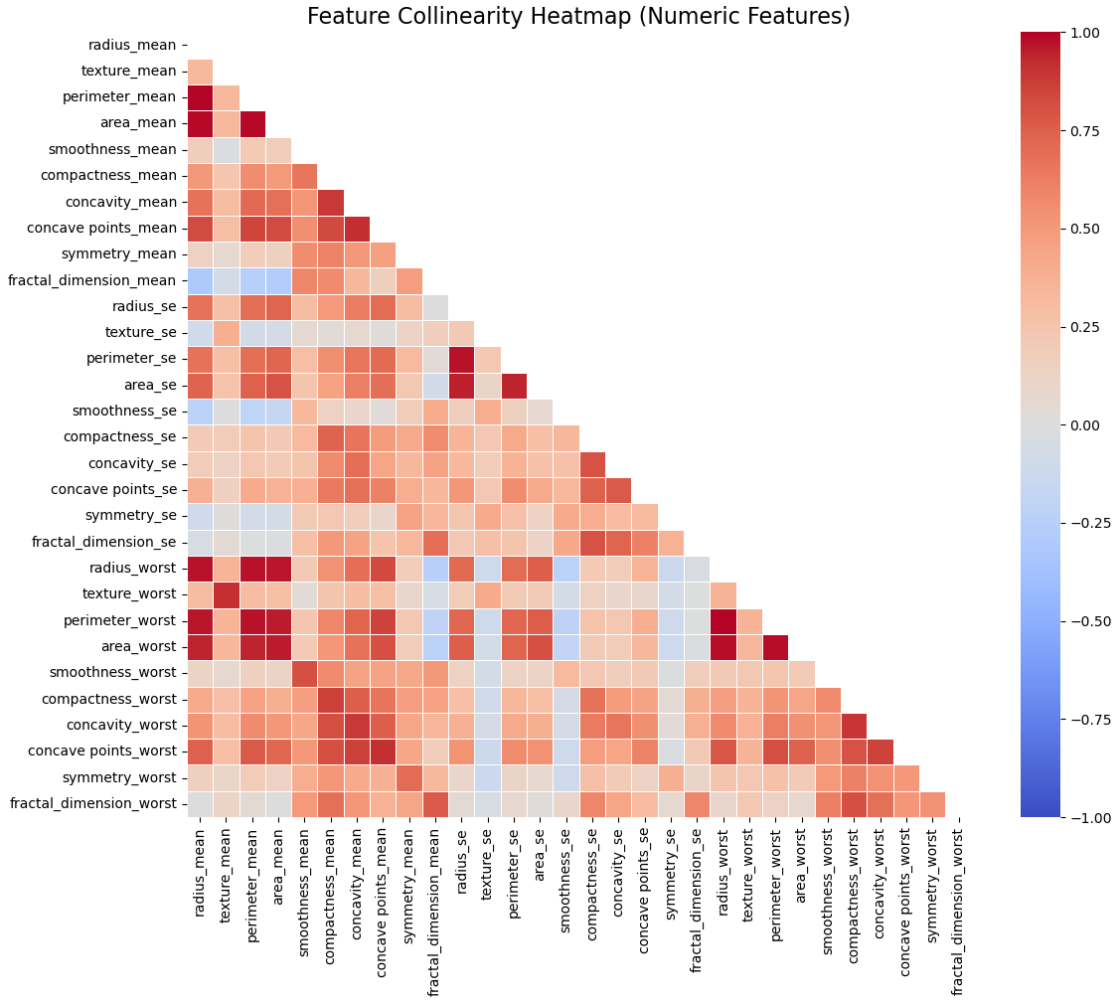
[15]:
```
np.int64(0)
```

This confirms that the dataset is complete with no missing data. There is also no categorical data

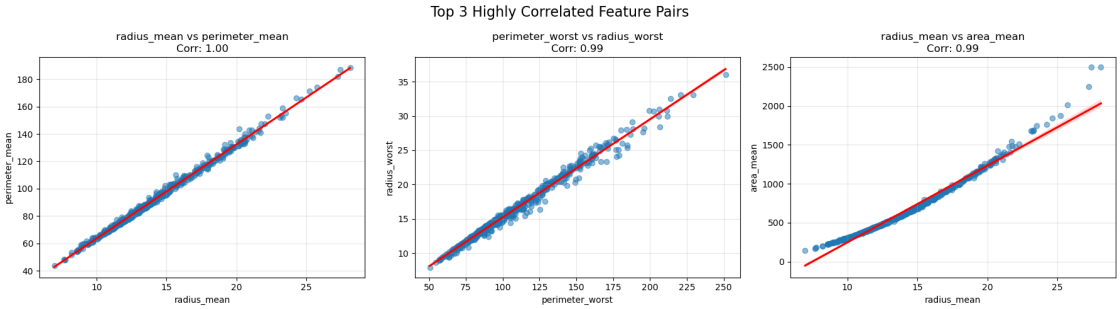
except for the labels so we will use the dataset as it.

[16]:
```
def analyze_correlations(df):

    # Correlations only make sense for numerical data
    numeric_df = df.select_dtypes(include=[np.number])

    if numeric_df.empty:
        print("No numeric columns found in the dataframe.")
        return

    # Calculate correlation matrix
    corr_matrix = numeric_df.corr()


    plt.figure(figsize=(12, 10))
    mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

    sns.heatmap(corr_matrix,
                mask=mask,
                annot=False,
                cmap='coolwarm',
                vmin=-1,
                vmax=1,
                center=0,
                square=True,
                linewidths=.5)

    plt.title('Feature Collinearity Heatmap (Numeric Features)', fontsize=16)
    plt.tight_layout()
    plt.show()


    # Unstack the matrix to get pairs
    corr_pairs = corr_matrix.abs().unstack()

    # Sort descending
    sorted_pairs = corr_pairs.sort_values(ascending=False)

    # Filter out self-correlations (which equal 1.0)
    sorted_pairs = sorted_pairs[sorted_pairs < 1.0]

    # Remove duplicates
    # We do this by iterating and tracking seen pairs
    unique_pairs = []
    seen_cols = set()
```

```
    for index, value in sorted_pairs.items():
        col1, col2 = index

        # Create a frozenset to handle (A, B) same as (B, A)
        pair_set = frozenset([col1, col2])

        if pair_set not in seen_cols:
            unique_pairs.append((col1, col2, value, corr_matrix.loc[col1,␣
 ↪col2]))
            seen_cols.add(pair_set)

        if len(unique_pairs) >= 3:
            break

    print("\n--- Top 3 Correlated Pairs ---")
    for col1, col2, abs_corr, real_corr in unique_pairs:
        print(f"{col1} vs {col2}: Correlation = {real_corr:.4f}")

    if unique_pairs:
        fig, axes = plt.subplots(1, 3, figsize=(18, 5))
        fig.suptitle('Top 3 Highly Correlated Feature Pairs', fontsize=16)

        for i, (col1, col2, _, real_corr) in enumerate(unique_pairs):
            # Create a scatter plot with a regression line
            sns.regplot(data=df, x=col1, y=col2, ax=axes[i],
                        scatter_kws={'alpha':0.5}, line_kws={'color':'red'})

            axes[i].set_title(f'{col1} vs {col2}\nCorr: {real_corr:.2f}')
            axes[i].grid(True, alpha=0.3)

        plt.tight_layout()
        plt.show()
    else:
        print("Not enough correlations found to plot top 3.")

np.random.seed(42)

analyze_correlations(X)
```




```
--- Top 3 Correlated Pairs ---
radius_mean vs perimeter_mean: Correlation = 0.9979
perimeter_worst vs radius_worst: Correlation = 0.9937
radius_mean vs area_mean: Correlation = 0.9874
```



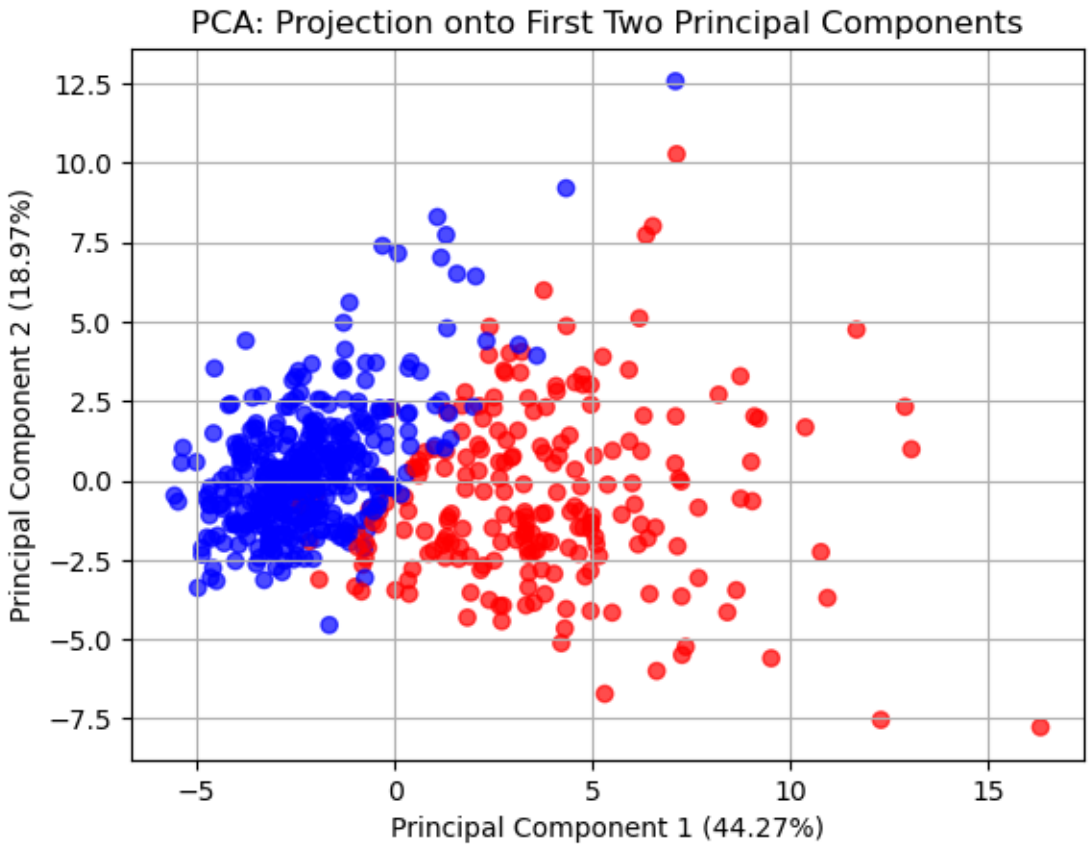
1.2 Principal Component Analysis

[17]:
```
# Setting seed
RANDOM_STATE = 42
```

Lets look at PCA in 2D by taking 2 components

[18]:
```
scaler = StandardScaler()
x_scaled = scaler.fit_transform(X)


pca = PCA(n_components=2, random_state=RANDOM_STATE)
principal_components = pca.fit_transform(x_scaled)

print("\n--- Explained Variance ---")
print(f"Variance explained by PC1: {pca.explained_variance_ratio_[0]:.2%}")
print(f"Variance explained by PC2: {pca.explained_variance_ratio_[1]:.2%}")
print(f"Total variance explained by 2 components: {pca.
 ↪explained_variance_ratio_.sum():.2%}")

scatter = plt.scatter(principal_components[:, 0], principal_components[:, 1],
                      c=["red" if label == "M" else "blue" for label in␣
 ↪y["Diagnosis"]],
                      alpha=0.7)

# Add labels and title
plt.xlabel(f"Principal Component 1 ({pca.explained_variance_ratio_[0]:.2%})")
plt.ylabel(f"Principal Component 2 ({pca.explained_variance_ratio_[1]:.2%})")
plt.title("PCA: Projection onto First Two Principal Components")
plt.grid(True)
```

```
--- Explained Variance ---
Variance explained by PC1: 44.27%
Variance explained by PC2: 18.97%
Total variance explained by 2 components: 63.24%
```



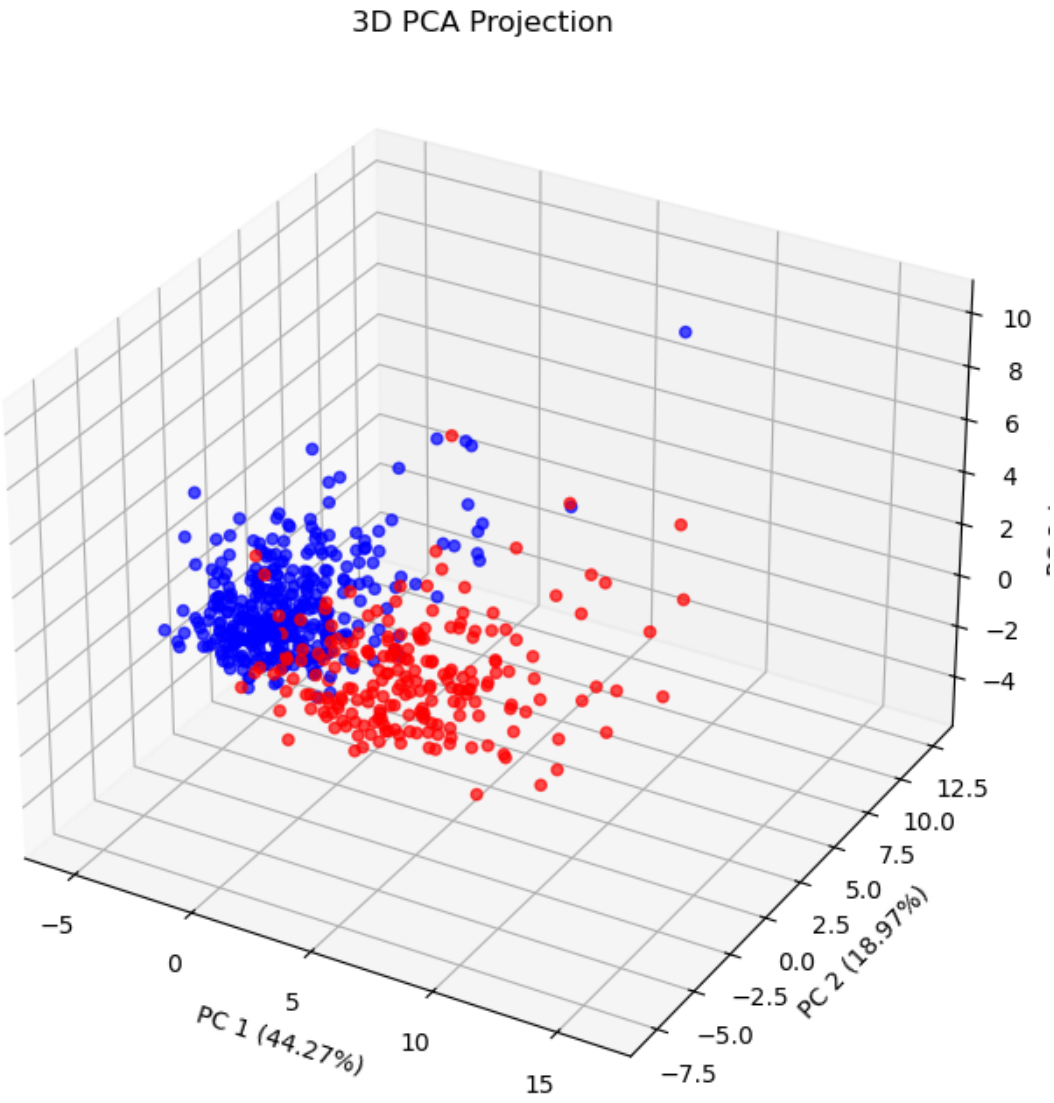
Lets take a look in 3D

[19]:
```
pca = PCA(n_components=3, random_state=RANDOM_STATE)
principal_components_3d = pca.fit_transform(x_scaled)


print("\n--- Explained Variance (3 Components) ---")
print(f"Variance explained by PC1: {pca.explained_variance_ratio_[0]:.2%}")
print(f"Variance explained by PC2: {pca.explained_variance_ratio_[1]:.2%}")
print(f"Variance explained by PC3: {pca.explained_variance_ratio_[2]:.2%}") #␣
 ↪Added PC3
print(f"Total variance explained by 3 components: {pca.
 ↪explained_variance_ratio_.sum():.2%}")

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

scatter = ax.scatter(principal_components_3d[:, 0], # PC1 on x-axis
                     principal_components_3d[:, 1], # PC2 on y-axis
                     principal_components_3d[:, 2], # PC3 on z-axis
```

```
                    c=["red" if label == "M" else "blue" for label in␣
 ↪y["Diagnosis"]],
                    cmap='viridis',
                    alpha=0.7)

ax.set_xlabel(f"PC 1 ({pca.explained_variance_ratio_[0]:.2%})")
ax.set_ylabel(f"PC 2 ({pca.explained_variance_ratio_[1]:.2%})")
ax.set_zlabel(f"PC 3 ({pca.explained_variance_ratio_[2]:.2%})")
ax.set_title("3D PCA Projection")

plt.show()
```

```
--- Explained Variance (3 Components) ---
Variance explained by PC1: 44.27%
Variance explained by PC2: 18.97%
Variance explained by PC3: 9.39%
Total variance explained by 3 components: 72.64%
```

```
/var/folders/4m/l4mr8hbd6q7_2qryv48yb8z00000gn/T/ipykernel_91943/71686734.py:14:
UserWarning: No data for colormapping provided via 'c'. Parameters 'cmap' will
be ignored
  scatter = ax.scatter(principal_components_3d[:, 0], # PC1 on x-axis
```

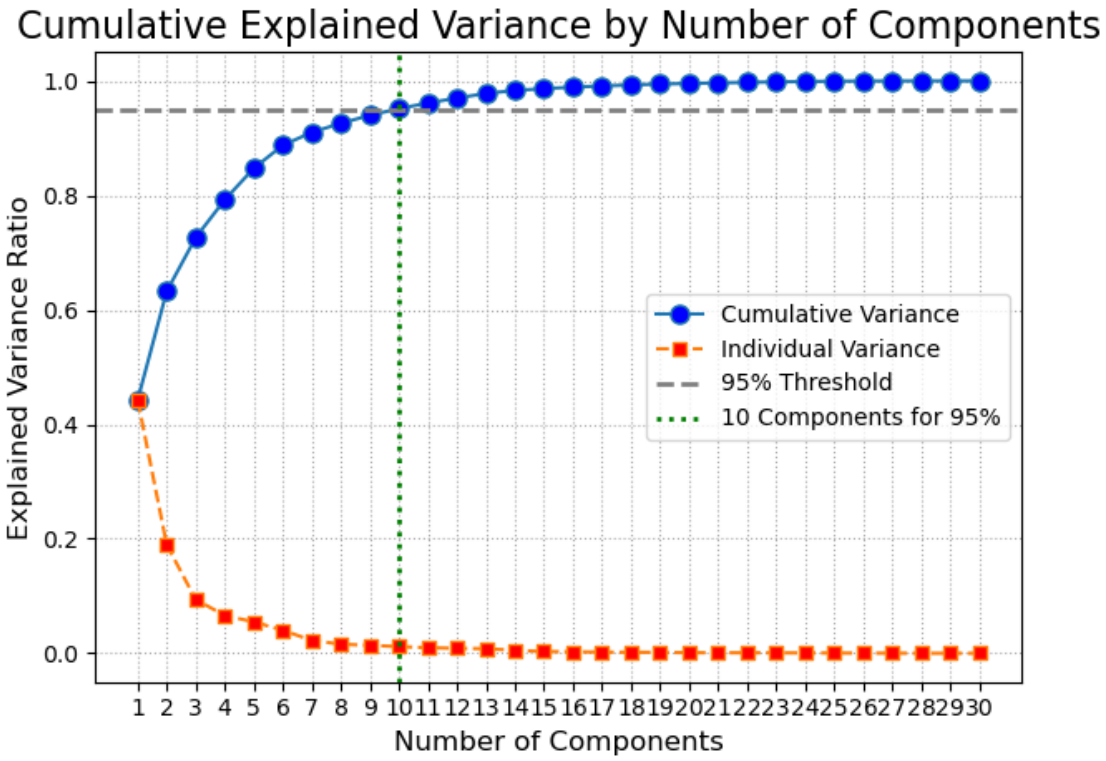


Taking a look at the variance explained by each PC

[20]:
```
pca = PCA(n_components=None, random_state=RANDOM_STATE)
pca.fit(x_scaled)

explained_variance = pca.explained_variance_ratio_
cumulative_variance = np.cumsum(explained_variance)

components_for_95_variance = np.argmax(cumulative_variance >= 0.95) + 1 # Add 1␣
↪for 0-based index
```

```
print(f"Number of components needed for 95% variance:␣
 ↪{components_for_95_variance}")

num_components = len(explained_variance)
component_numbers = np.arange(1, num_components + 1)

plt.plot(component_numbers, cumulative_variance, 'o-', markerfacecolor='blue',␣
 ↪markersize=8, label='Cumulative Variance')
plt.plot(component_numbers, explained_variance, 's--', markerfacecolor='red',␣
 ↪markersize=6, label='Individual Variance')

plt.title('Cumulative Explained Variance by Number of Components', fontsize=16)
plt.xlabel('Number of Components', fontsize=12)
plt.ylabel('Explained Variance Ratio', fontsize=12)
plt.xticks(component_numbers)

plt.axhline(y=0.95, color='gray', linestyle='--', linewidth=2, label='95%␣
 ↪Threshold')

plt.axvline(x=components_for_95_variance, color='green', linestyle=':',␣
 ↪linewidth=2,
            label=f'{components_for_95_variance} Components for 95%')

plt.grid(True, linestyle=':')
plt.legend(loc='center right')
plt.tight_layout()
```

```
Number of components needed for 95% variance: 10
```



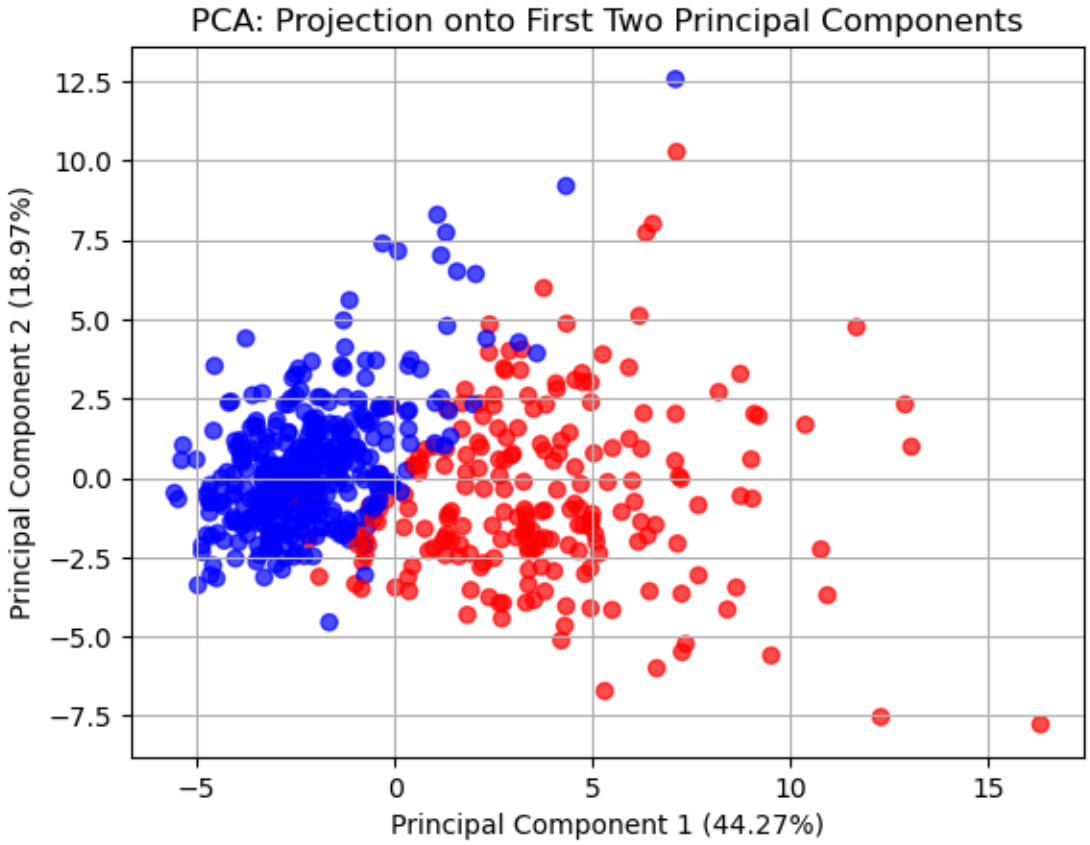
10 PCA components result in a variance of 95%, but there seems to be a sharper cuttoff at ~18 components

Lets project the target class onto the PC space

[21]:
```
def plot_pca_2d(principal_components, pca_model, y=None):

    # Get the variance explained by the first two components
    pc1_var = pca_model.explained_variance_ratio_[0]
    pc2_var = pca_model.explained_variance_ratio_[1]


    scatter = plt.scatter(principal_components[:, 0],
                          principal_components[:, 1],
                          c=["red" if label == "M" else "blue" for label in␣
↪y["Diagnosis"]] if y is not None else None,
                          cmap='viridis', # A common colormap
                          alpha=0.7)

    plt.xlabel(f"Principal Component 1 ({pc1_var:.2%})")
    plt.ylabel(f"Principal Component 2 ({pc2_var:.2%})")
    plt.title("PCA: Projection onto First Two Principal Components")
```

```
    plt.grid(True)
    plt.show()


plot_pca_2d(principal_components, pca, y=y)
```

```
/var/folders/4m/l4mr8hbd6q7_2qryv48yb8z00000gn/T/ipykernel_91943/1231580005.py:1
6: UserWarning: No data for colormapping provided via 'c'. Parameters 'cmap'
will be ignored
  scatter = plt.scatter(principal_components[:, 0],
```



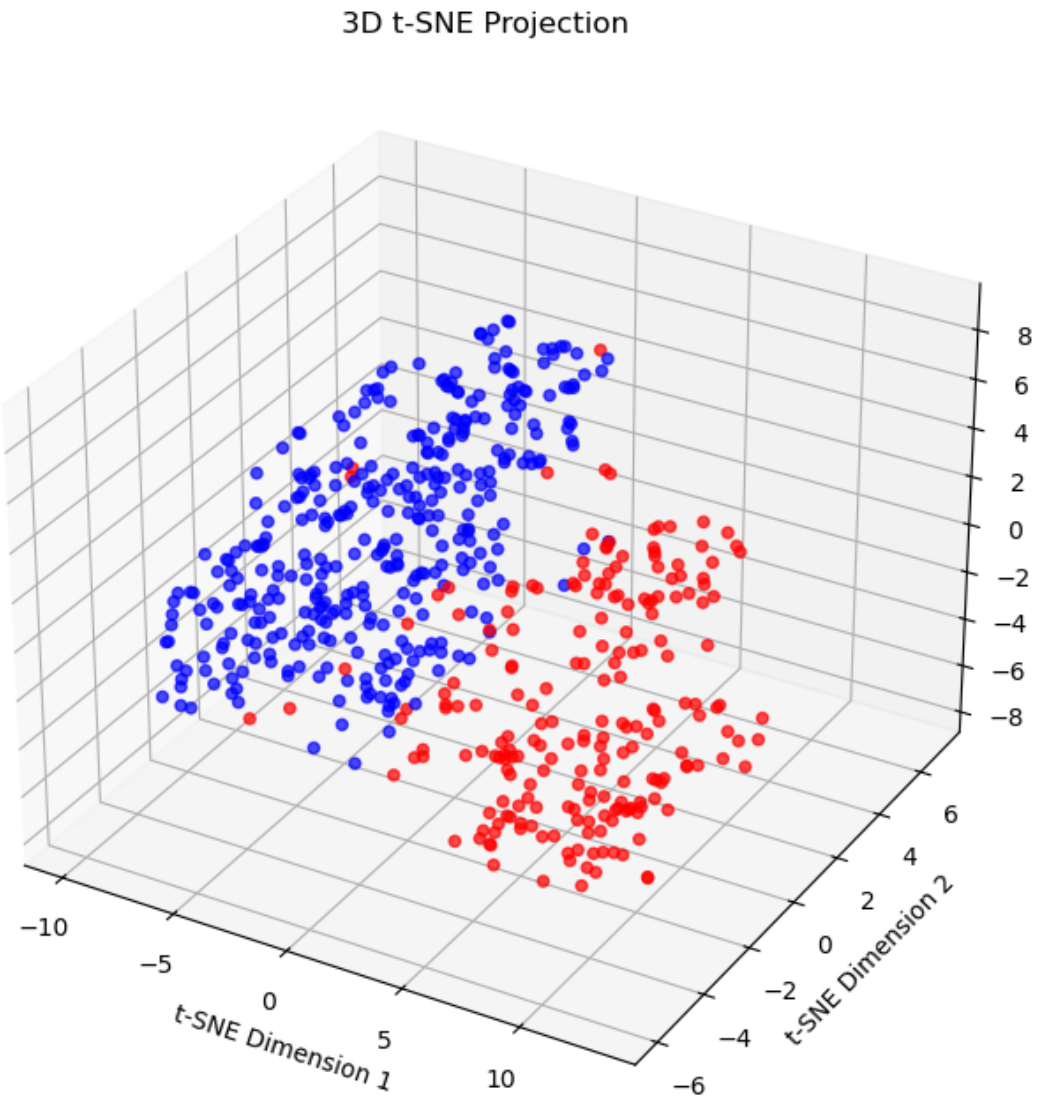
1.3 TSNE

Lets take a look at using TSNE for dimensionality reduction

[22]:
```
def fit_tsne(tsne_kwargs, x_data):
    tsne = TSNE(**tsne_kwargs)
    tsne_results = tsne.fit_transform(x_data)
    return tsne_results
```

```
def plot_components_3d(tsne_results, y=None):
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='3d')

    scatter = ax.scatter(tsne_results[:, 0],
                         tsne_results[:, 1],
                         tsne_results[:, 2],
                         c=["red" if label == "M" else "blue" for label in␣
 ↪y["Diagnosis"]] if y is not None else None,
                         cmap='viridis', # A common colormap
                         alpha=0.7)
    ax.set_xlabel("t-SNE Dimension 1")
    ax.set_ylabel("t-SNE Dimension 2")
    ax.set_zlabel("t-SNE Dimension 3")
    ax.set_title("3D t-SNE Projection")

tsne_results = fit_tsne({"n_components": 3, "random_state": RANDOM_STATE},␣
 ↪x_scaled)

plot_components_3d(tsne_results, y)
```

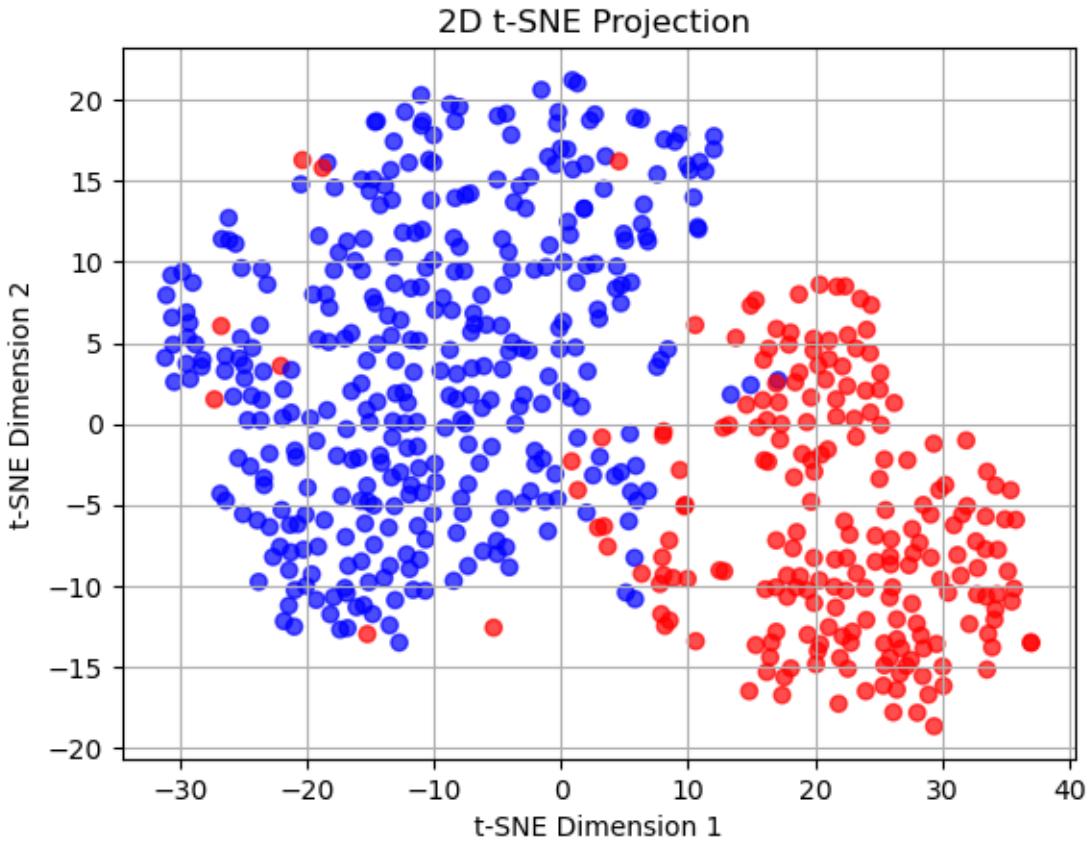
```
/var/folders/4m/l4mr8hbd6q7_2qryv48yb8z00000gn/T/ipykernel_91943/1358799948.py:1
0: UserWarning: No data for colormapping provided via 'c'. Parameters 'cmap'
will be ignored
  scatter = ax.scatter(tsne_results[:, 0],
```



[23]:
```
def plot_2d(tsne_results, y=None, additional_title=""):

    scatter = plt.scatter(tsne_results[:, 0],
                          tsne_results[:, 1],
                          c=["red" if label == "M" else "blue" for label in␣
↪y["Diagnosis"]] if y is not None else None,
                          cmap='viridis', # A common colormap
                          alpha=0.7)

    plt.xlabel("t-SNE Dimension 1")
    plt.ylabel("t-SNE Dimension 2")
    plt.title(f"2D t-SNE Projection {additional_title}")
```

```
    plt.grid(True)
    plt.show()

tsne_2d = fit_tsne({"n_components": 2, "random_state": RANDOM_STATE}, x_scaled)
plot_2d(tsne_2d, y)
```

```
/var/folders/4m/l4mr8hbd6q7_2qryv48yb8z00000gn/T/ipykernel_91943/3705409882.py:3
: UserWarning: No data for colormapping provided via 'c'. Parameters 'cmap' will
be ignored
  scatter = plt.scatter(tsne_results[:, 0],
```

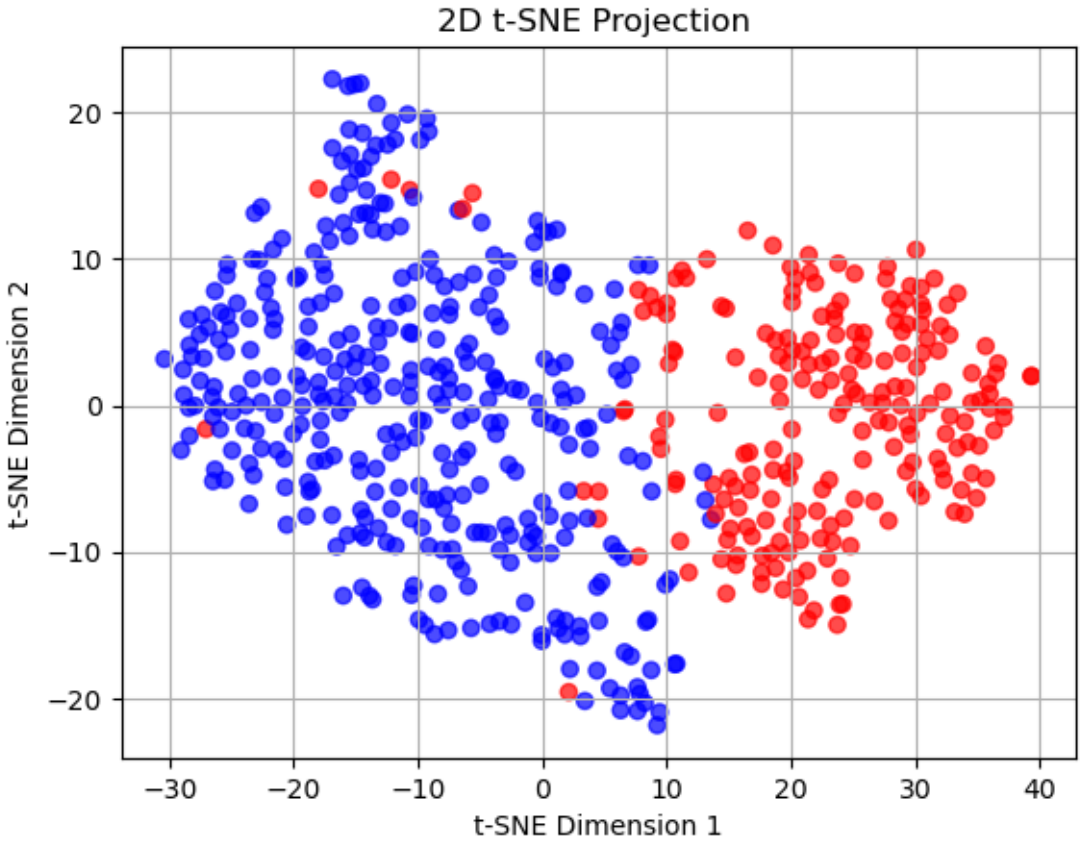


We can also use TSNE to reduce PCA components

[24]:
```
pca = PCA(n_components=None, random_state=RANDOM_STATE)
x_pca_all_components = pca.fit_transform(x_scaled)

x_pca_10 = x_pca_all_components[:, :10]
```

```
tsne_pca_2d = fit_tsne({"n_components": 2, "random_state": RANDOM_STATE},␣
 ↪x_pca_10)
plot_2d(tsne_pca_2d, y)
```

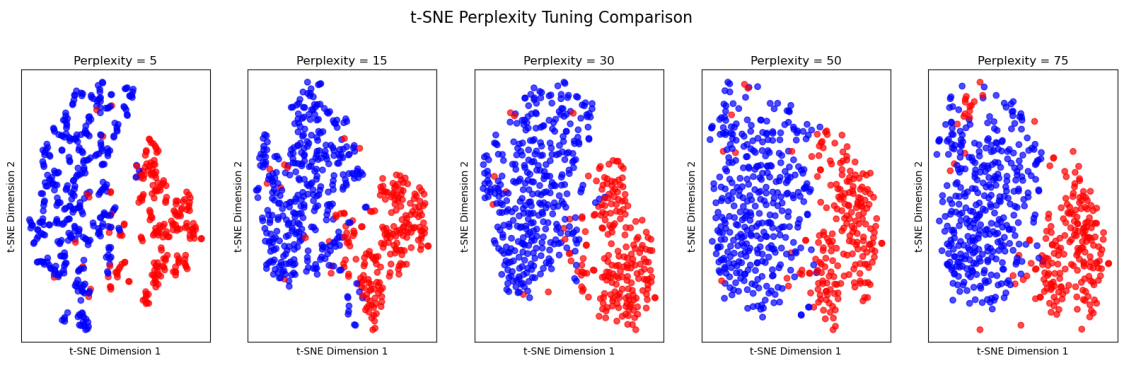
```
/var/folders/4m/l4mr8hbd6q7_2qryv48yb8z00000gn/T/ipykernel_91943/3705409882.py:3
: UserWarning: No data for colormapping provided via 'c'. Parameters 'cmap' will
be ignored
  scatter = plt.scatter(tsne_results[:, 0],
```



Tuning the perplexity variable in the TSNE function creates different results. The perplexity variable relates to the number of nearest neighbors used in the manifold learning algorithm

[25]:
```
perplexity_values = [5, 15, 30, 50, 75]

fig, axes = plt.subplots(1, len(perplexity_values), figsize=(20, 5))


colours = ["red" if label == "M" else "blue" for label in y["Diagnosis"]]
```

```
def subplot_2d(ax, tsne_results, hyperparam_name, hyperparam_value, model_name,␣
 ↪colours=None):
    ax.scatter(tsne_results[:, 0],
               tsne_results[:, 1],
               c=colours,
               alpha=0.7)

    ax.set_title(f"{hyperparam_name} = {hyperparam_value}")
    ax.set_xlabel(f"{model_name} Dimension 1")
    ax.set_ylabel(f"{model_name} Dimension 2")
    ax.set_xticks([])
    ax.set_yticks([])

for i, perplexity in enumerate(perplexity_values):
    tsne_kwargs = {
        "n_components": 2,
        "random_state": RANDOM_STATE,
        "perplexity": perplexity,
    }
    tsne_results = fit_tsne(tsne_kwargs, x_scaled)


    subplot_2d(axes[i], tsne_results, hyperparam_name="Perplexity",␣
 ↪hyperparam_value=perplexity, model_name="t-SNE", colours=colours)


plt.suptitle("t-SNE Perplexity Tuning Comparison", fontsize=16, y=1.05)
plt.show()

print("Tuning complete.")
```



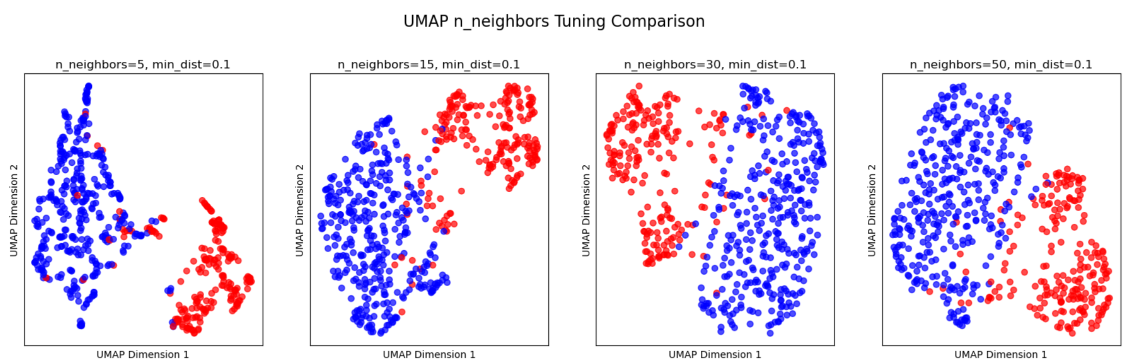

```
Tuning complete.
```

1.4 UMAP

[26]:
```
def plot_components_2d_umap(ax, umap_results, y, n_neighbors, min_dist):

    colors = ["red" if label == "M" else "blue" for label in y["Diagnosis"]]

    ax.scatter(
        umap_results[:, 0],
        umap_results[:, 1],
        c=colors,
        alpha=0.7
    )

    ax.set_title(f"n_neighbors={n_neighbors}, min_dist={min_dist}")
    ax.set_xlabel("UMAP Dimension 1")
    ax.set_ylabel("UMAP Dimension 2")
    ax.set_xticks([])
    ax.set_yticks([])


# --------------------------------
# Hyperparameter tuning options
# --------------------------------
n_neighbors_list = [5, 15, 30, 50]
min_dist = 0.1

fig, axes = plt.subplots(1, len(n_neighbors_list), figsize=(20, 5))

print("Running UMAP for different n_neighbors values...")

for i, n_neighbors in enumerate(n_neighbors_list):

    reducer = umap.UMAP(
        n_neighbors=n_neighbors,
        min_dist=min_dist,
        n_components=2,
        random_state=42
    )

    umap_results = reducer.fit_transform(x_scaled)

    plot_components_2d_umap(
        axes[i],
        umap_results,
        y,
        n_neighbors=n_neighbors,
        min_dist=min_dist
    )
```

```
plt.suptitle("UMAP n_neighbors Tuning Comparison", fontsize=16, y=1.05)
plt.show()
```

```
Running UMAP for different n_neighbors values…

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(
/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(
/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(
/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(
```



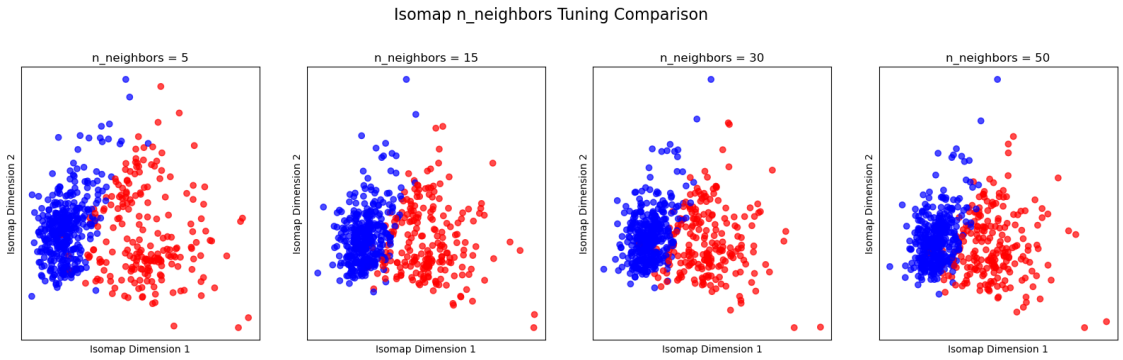
1.5 Isomap

We can similarly tune the parameter to n_neighbors

[27]:
```
def plot_components_2d_isomap(ax, isomap_results, y, n_neighbors):
    colors = ["red" if label == "M" else "blue" for label in y["Diagnosis"]]

    ax.scatter(isomap_results[:, 0],
               isomap_results[:, 1],
               c=colors,
               alpha=0.7)
```

```
    ax.set_title(f"n_neighbors = {n_neighbors}")
    ax.set_xlabel("Isomap Dimension 1")
    ax.set_ylabel("Isomap Dimension 2")
    ax.set_xticks([])
    ax.set_yticks([])

n_neighbors_list = [5, 15, 30, 50]

fig, axes = plt.subplots(1, len(n_neighbors_list), figsize=(20, 5))

print("Running Isomap for different n_neighbors values...")

for i, n_neighbors in enumerate(n_neighbors_list):
    isomap = Isomap(n_neighbors=n_neighbors, n_components=2)

    isomap_results = isomap.fit_transform(x_scaled)

    subplot_2d(axes[i],
               isomap_results,
               hyperparam_name="n_neighbors",
               hyperparam_value=n_neighbors,
               model_name="Isomap",
               colours=colours)

plt.suptitle("Isomap n_neighbors Tuning Comparison", fontsize=16, y=1.05)
plt.show()
```

```
Running Isomap for different n_neighbors values…
```



1.6 Training the classification model

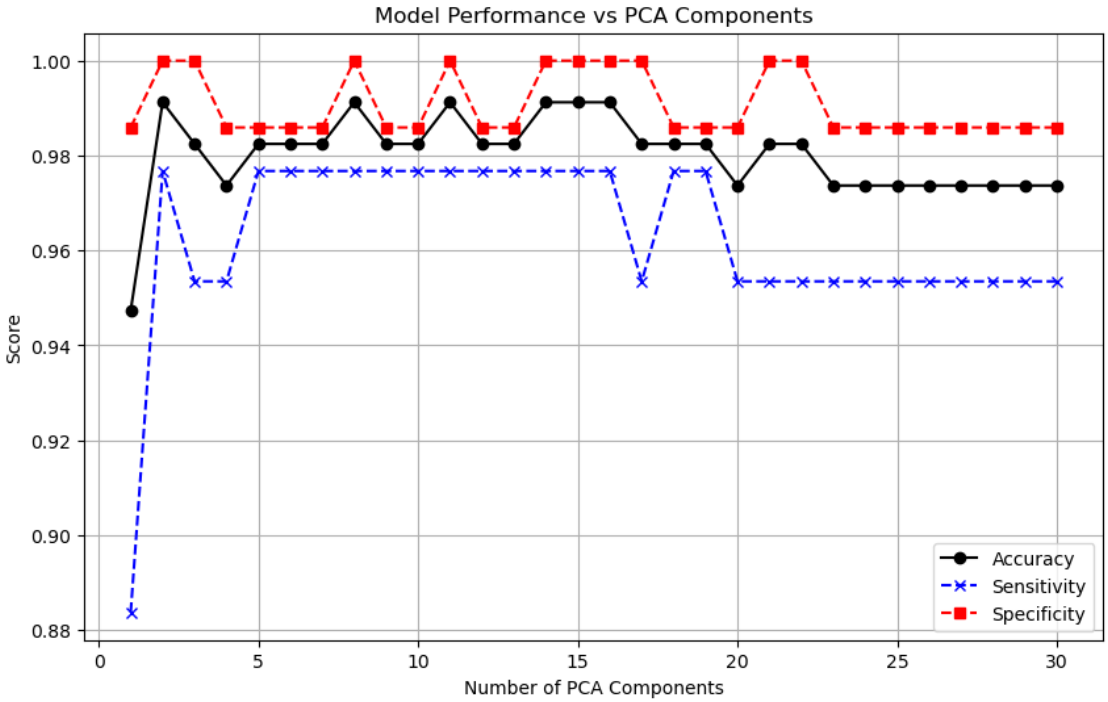
Using PCA components to train the model. Lets see how many PCs produce the best test accuracy

[28]:
```
start_time = time.time()
tracemalloc.start()

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

best_acc = 0
best_components = None
best_pca_model = None
best_metrics = {} # To store sens/spec of the winning model

# Lists to store history for plotting
history = {
    "n": [],
    "accuracy": [],
    "sensitivity": [],
    "specificity": []
}

print(f"{'PCA':<5} | {'Accuracy':<10} | {'Sens':<10} | {'Spec':<10}")
print("-" * 45)

for n in range(1, X.shape[1] + 1):

    # Apply PCA
    pca = PCA(n_components=n)
    X_pca = pca.fit_transform(X_scaled)

    # Train/test split
    X_train, X_test, y_train, y_test = train_test_split(
        X_pca, y.values.ravel(), test_size=0.2, random_state=42
    )

    # Train classifier
    clf = LogisticRegression(max_iter=1500)
    clf.fit(X_train, y_train)

    # Evaluate
    y_pred = clf.predict(X_test)

    acc = accuracy_score(y_test, y_pred)

    # Get True Negatives, False Positives, False Negatives, True Positives
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

    # Calculate Sensitivity (Recall) and Specificity
    sensitivity = tp / (tp + fn) if (tp + fn) > 0 else 0
```

```
    specificity = tn / (tn + fp) if (tn + fp) > 0 else 0

    # Store for plotting
    history["n"].append(n)
    history["accuracy"].append(acc)
    history["sensitivity"].append(sensitivity)
    history["specificity"].append(specificity)

    print(f"{n:<5d} | {acc:.5f}    | {sensitivity:.5f}    | {specificity:.5f}")

    if acc > best_acc:
        best_acc = acc
        best_components = n
        best_pca_model = pca
        best_metrics = {'sens': sensitivity, 'spec': specificity}


print("\n==========================")
print(f" Best PCA components: {best_components}")
print(f" Best Accuracy:       {best_acc:.5f}")
print(f" Sensitivity:         {best_metrics['sens']:.5f}")
print(f" Specificity:         {best_metrics['spec']:.5f}")
print("==========================")

end_time = time.time()
elapsed = end_time - start_time

current, peak = tracemalloc.get_traced_memory()
tracemalloc.stop()

peak_mb = peak / 10**6

print(f"PCA took: Time={elapsed:.2f}s, Memory={peak_mb:.2f}MB")

# --- PLOTTING ALL THREE METRICS ---
plt.figure(figsize=(10, 6))

plt.plot(history["n"], history["accuracy"], marker="o", label="Accuracy",␣
 ↪color='black')
plt.plot(history["n"], history["sensitivity"], marker="x", linestyle="--",␣
 ↪label="Sensitivity", color='blue')
plt.plot(history["n"], history["specificity"], marker="s", linestyle="--",␣
 ↪label="Specificity", color='red')

plt.xlabel("Number of PCA Components")
plt.ylabel("Score")
plt.title("Model Performance vs PCA Components")
```

```
plt.legend()
plt.grid(True)
plt.show()
```

```
PCA   | Accuracy   | Sens       | Spec
-------------------------------------------
1     | 0.94737    | 0.88372    | 0.98592
2     | 0.99123    | 0.97674    | 1.00000
3     | 0.98246    | 0.95349    | 1.00000
4     | 0.97368    | 0.95349    | 0.98592
5     | 0.98246    | 0.97674    | 0.98592
6     | 0.98246    | 0.97674    | 0.98592
7     | 0.98246    | 0.97674    | 0.98592
8     | 0.99123    | 0.97674    | 1.00000
9     | 0.98246    | 0.97674    | 0.98592
10    | 0.98246    | 0.97674    | 0.98592
11    | 0.99123    | 0.97674    | 1.00000
12    | 0.98246    | 0.97674    | 0.98592
13    | 0.98246    | 0.97674    | 0.98592
14    | 0.99123    | 0.97674    | 1.00000
15    | 0.99123    | 0.97674    | 1.00000
16    | 0.99123    | 0.97674    | 1.00000
17    | 0.98246    | 0.95349    | 1.00000
18    | 0.98246    | 0.97674    | 0.98592
19    | 0.98246    | 0.97674    | 0.98592
20    | 0.97368    | 0.95349    | 0.98592
21    | 0.98246    | 0.95349    | 1.00000
22    | 0.98246    | 0.95349    | 1.00000
23    | 0.97368    | 0.95349    | 0.98592
24    | 0.97368    | 0.95349    | 0.98592
25    | 0.97368    | 0.95349    | 0.98592
26    | 0.97368    | 0.95349    | 0.98592
27    | 0.97368    | 0.95349    | 0.98592
28    | 0.97368    | 0.95349    | 0.98592
29    | 0.97368    | 0.95349    | 0.98592
30    | 0.97368    | 0.95349    | 0.98592

==========================
 Best PCA components: 2
 Best Accuracy:       0.99123
 Sensitivity:         0.97674
 Specificity:         1.00000
==========================
PCA took: Time=0.49s, Memory=1.60MB
```



Lets investigate what features comprise PC1 and PC2

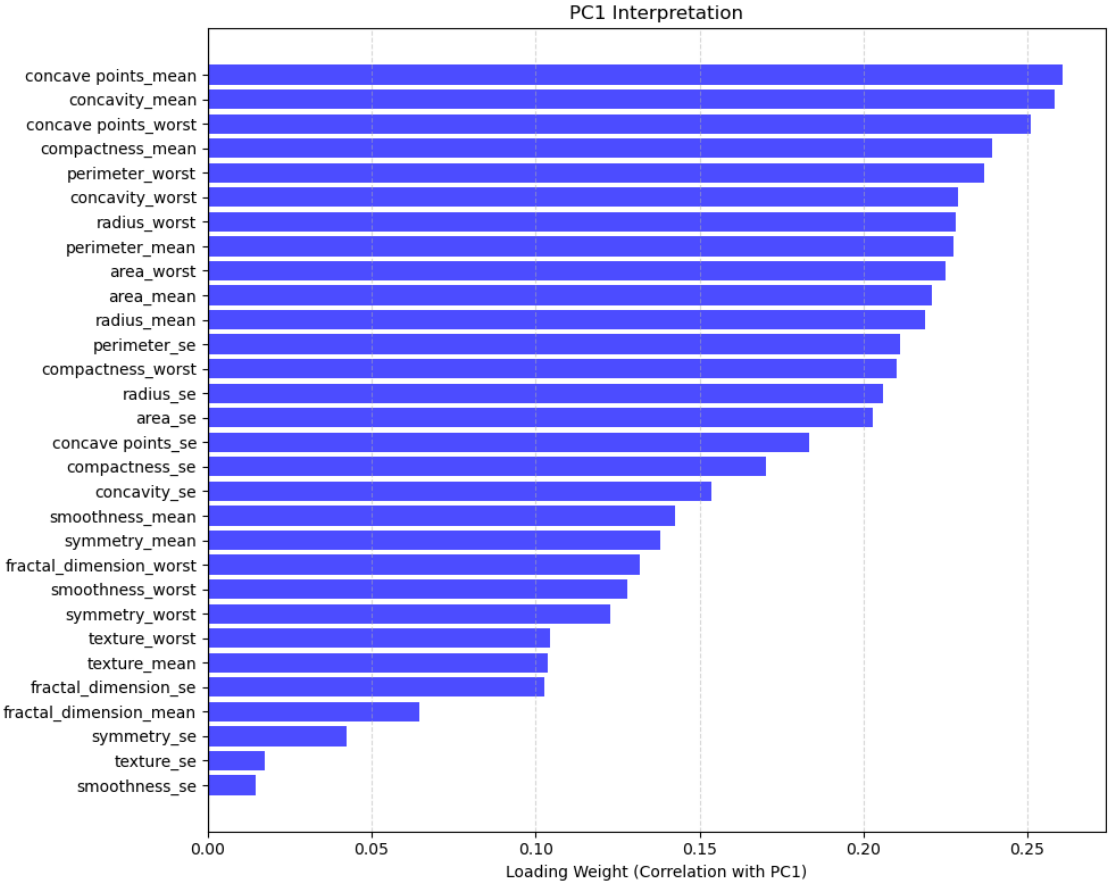
[29]:
```
pc_column_names = [f"PC{i+1}" for i in range(best_components)]
print(f"\nBest PC Column Names: {pc_column_names}")

# We create a DataFrame where rows = Original Features, Columns = Best PCs
print("\n--- Top Contributing Features (Loadings) ---")
loadings_df = pd.DataFrame(
    best_pca_model.components_.T,  # Transpose to get (Features x PCs)
    columns=pc_column_names,
    index=X.columns # Assumes X is a Pandas DataFrame
)

# Print the full loadings matrix, or just the top factors for PC1
print(loadings_df)
print("============================")

def get_top_features_per_pc(pca_model, feature_names, top_n=2):
    """
    Prints the top N original features that contribute most to each Principal␣
 ↪Component.
    """
    print(f"\n--- Top {top_n} Original Features per Principal Component ---")
```

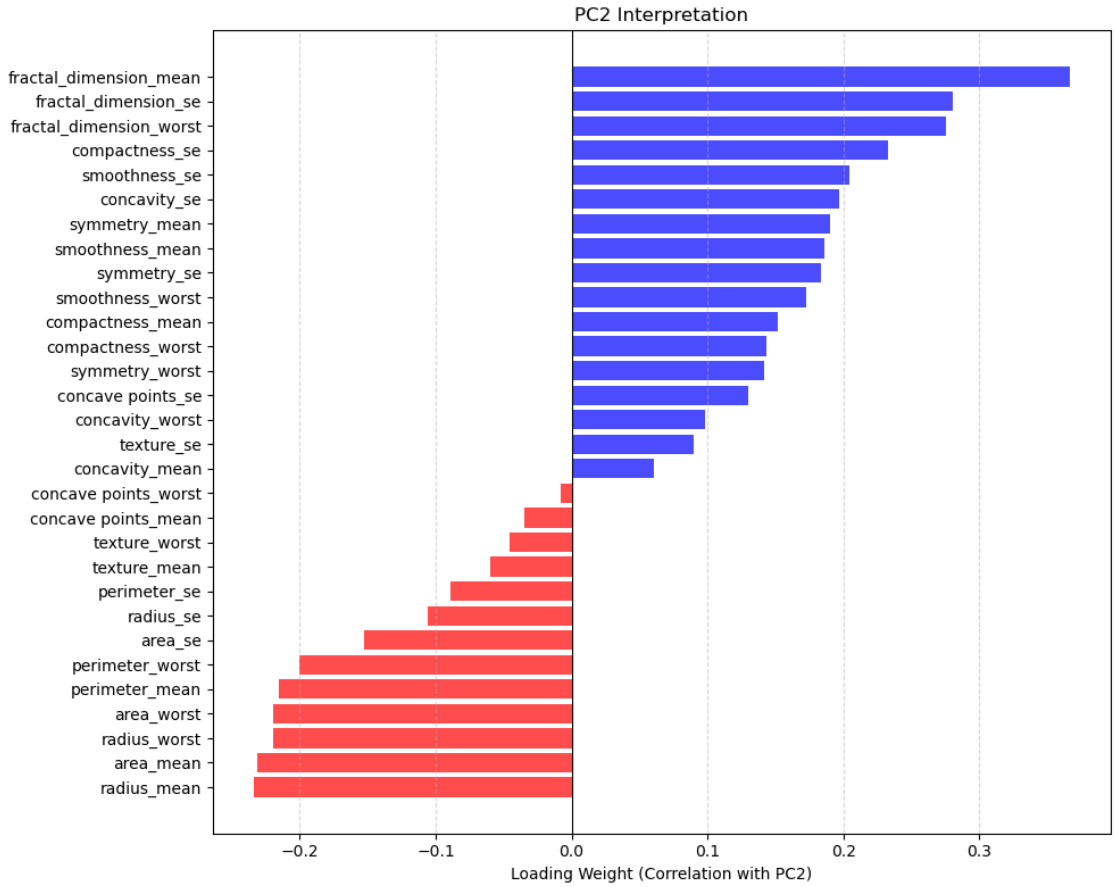
```
    # pca_model.components_ has shape (n_components, n_features)
    for i, component in enumerate(pca_model.components_):
        # Zip feature names with their weights
        feature_weights = list(zip(feature_names, component))

        # Sort by absolute value of the weight (magnitude indicates importance)
        sorted_features = sorted(feature_weights, key=lambda x: abs(x[1]),␣
 ↪reverse=True)

        # Print the top N
        top_features = sorted_features[:top_n]

        feature_str = ", ".join([f"{name} ({weight:.3f})" for name, weight in␣
 ↪top_features])
        print(f"PC{i+1}: {feature_str}")


if best_pca_model is not None:
    get_top_features_per_pc(best_pca_model, X.columns, top_n=3)
else:
    print("No best model found (did accuracy improve?).")

sorted_loadings = loadings_df.sort_values(by='PC1', ascending=True)

plt.figure(figsize=(10, 8))

# Plot PC2
plt.barh(
    y=sorted_loadings.index,
    width=sorted_loadings['PC1'],
    color=['red' if x < 0 else 'blue' for x in sorted_loadings['PC1']],
    alpha=0.7
)

plt.title("PC1 Interpretation")
plt.xlabel("Loading Weight (Correlation with PC1)")
plt.axvline(0, color='black', linewidth=0.8) # Line at 0
plt.grid(axis='x', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()

sorted_loadings = loadings_df.sort_values(by='PC2', ascending=True)

plt.figure(figsize=(10, 8))

# Plot PC2
plt.barh(
```

```
    y=sorted_loadings.index,
    width=sorted_loadings['PC2'],
    color=['red' if x < 0 else 'blue' for x in sorted_loadings['PC2']],
    alpha=0.7
)

plt.title("PC2 Interpretation")
plt.xlabel("Loading Weight (Correlation with PC2)")
plt.axvline(0, color='black', linewidth=0.8) # Line at 0
plt.grid(axis='x', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```

```
Best PC Column Names: ['PC1', 'PC2']

--- Top Contributing Features (Loadings) ---
                            PC1       PC2
radius_mean              0.218902 -0.233857
texture_mean             0.103725 -0.059706
perimeter_mean           0.227537 -0.215181
area_mean                0.220995 -0.231077
smoothness_mean          0.142590  0.186113
compactness_mean         0.239285  0.151892
concavity_mean           0.258400  0.060165
concave points_mean      0.260854 -0.034768
symmetry_mean            0.138167  0.190349
fractal_dimension_mean   0.064363  0.366575
radius_se                0.205979 -0.105552
texture_se               0.017428  0.089980
perimeter_se             0.211326 -0.089457
area_se                  0.202870 -0.152293
smoothness_se            0.014531  0.204430
compactness_se           0.170393  0.232716
concavity_se             0.153590  0.197207
concave points_se        0.183417  0.130322
symmetry_se              0.042498  0.183848
fractal_dimension_se     0.102568  0.280092
radius_worst             0.227997 -0.219866
texture_worst            0.104469 -0.045467
perimeter_worst          0.236640 -0.199878
area_worst               0.224871 -0.219352
smoothness_worst         0.127953  0.172304
compactness_worst        0.210096  0.143593
concavity_worst          0.228768  0.097964
concave points_worst     0.250886 -0.008257
symmetry_worst           0.122905  0.141883
fractal_dimension_worst  0.131784  0.275339
```

```
==========================

--- Top 3 Original Features per Principal Component ---
PC1: concave points_mean (0.261), concavity_mean (0.258), concave points_worst
(0.251)
PC2: fractal_dimension_mean (0.367), fractal_dimension_se (0.280),
fractal_dimension_worst (0.275)
```





Doing the same thing for TSNE

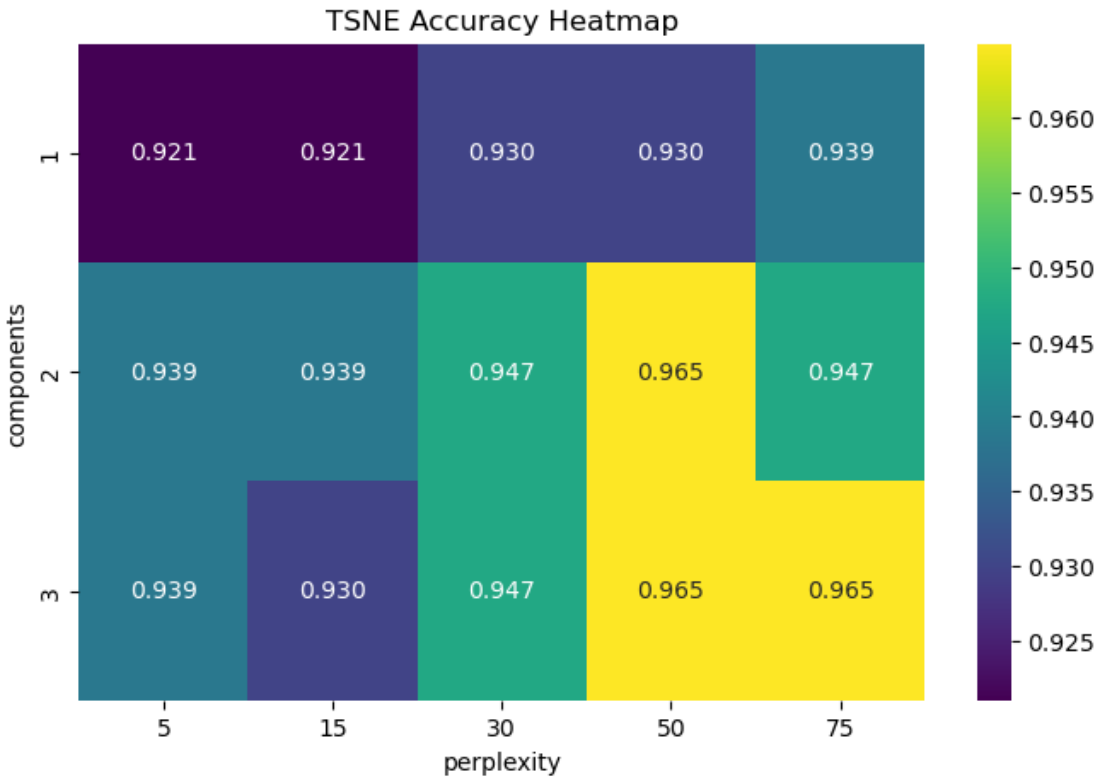
[30]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

best_acc = 0
best_params = None
# Initialize variables to store the metrics of the best run
best_sensitivity = 0.0
best_specificity = 0.0

accuracies = []
perplexity_values = [5, 15, 30, 50, 75]

print("Running TSNE sweeps...\n")

start_time = time.time()
tracemalloc.start()
for n in range(1, 4):  # 1 to 3 components
```

```
    for p in perplexity_values:

        print(f"Testing: n_components={n}, perplexity={p} ...")

        tsne = TSNE(
            n_components=n,
            perplexity=p,
            random_state=42
        )

        X_tsne = tsne.fit_transform(X_scaled)

        # Train/test split
        X_train, X_test, y_train, y_test = train_test_split(
            X_tsne, y.values.ravel(), test_size=0.2, random_state=42
        )

        # Train classifier
        clf = LogisticRegression(max_iter=1500)
        clf.fit(X_train, y_train)

        # Evaluate
        y_pred = clf.predict(X_test)
        acc = accuracy_score(y_test, y_pred)

        tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

        sensitivity = tp / (tp + fn) if (tp + fn) > 0 else 0
        specificity = tn / (tn + fp) if (tn + fp) > 0 else 0
        # ----------------------------------------------------

        accuracies.append((n, p, acc))

        print(f" -> Accuracy = {acc:.5f}")

        # Track best
        if acc > best_acc:
            best_acc = acc
            best_params = (n, p)
            # Save the specific metrics associated with this best result
            best_sensitivity = sensitivity
            best_specificity = specificity


# Print best result
print("\n==============================")
print(f" Best TSNE parameters:")
```

```
print(f"   n_components = {best_params[0]}")
print(f"   perplexity   = {best_params[1]}")
print(f" Best accuracy  = {best_acc:.5f}")
print(f" Sensitivity    = {best_sensitivity:.5f}")
print(f" Specificity    = {best_specificity:.5f}")
print("==============================")

end_time = time.time()
elapsed = end_time - start_time

current, peak = tracemalloc.get_traced_memory()
tracemalloc.stop()

peak_mb = peak / 10**6

print(f"TSNE took: Time={elapsed:.2f}s, Memory={peak_mb:.2f}MB")

df = pd.DataFrame(accuracies, columns=["components", "perplexity", "accuracy"])

pivot = df.pivot(index="components", columns="perplexity", values="accuracy")

plt.figure(figsize=(8, 5))
sns.heatmap(pivot, annot=True, fmt=".3f", cmap="viridis")
plt.title("TSNE Accuracy Heatmap")
plt.show()
```

```
Running TSNE sweeps…

Testing: n_components=1, perplexity=5 …
 -> Accuracy = 0.92105
Testing: n_components=1, perplexity=15 …
 -> Accuracy = 0.92105
Testing: n_components=1, perplexity=30 …
 -> Accuracy = 0.92982
Testing: n_components=1, perplexity=50 …
 -> Accuracy = 0.92982
Testing: n_components=1, perplexity=75 …
 -> Accuracy = 0.93860
Testing: n_components=2, perplexity=5 …
 -> Accuracy = 0.93860
Testing: n_components=2, perplexity=15 …
 -> Accuracy = 0.93860
Testing: n_components=2, perplexity=30 …
 -> Accuracy = 0.94737
Testing: n_components=2, perplexity=50 …
 -> Accuracy = 0.96491
Testing: n_components=2, perplexity=75 …
 -> Accuracy = 0.94737
```

```
Testing: n_components=3, perplexity=5 …
 -> Accuracy = 0.93860
Testing: n_components=3, perplexity=15 …
 -> Accuracy = 0.92982
Testing: n_components=3, perplexity=30 …
 -> Accuracy = 0.94737
Testing: n_components=3, perplexity=50 …
 -> Accuracy = 0.96491
Testing: n_components=3, perplexity=75 …
 -> Accuracy = 0.96491

==============================
 Best TSNE parameters:
   n_components = 2
   perplexity   = 50
 Best accuracy  = 0.96491
 Sensitivity    = 0.95349
 Specificity    = 0.97183
==============================
TSNE took: Time=74.02s, Memory=7.95MB
```



Same thing for UMAP

[31]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

best_acc = 0
best_components = None

best_sensitivity = 0.0
best_specificity = 0.0

accuracies = []
n_neighbours_list = [5, 15, 30, 50]

print("Running UMAP sweeps...\n")

start_time = time.time()
tracemalloc.start()
for n in range(1, 11):
    for p in n_neighbors_list:

        print(f"Testing: n_components={n}, n_neighbours={p} ...")

        umap_reducer = umap.UMAP(
            n_components=n,
            n_neighbors=p,
            random_state=42)

        X_umap = umap_reducer.fit_transform(X_scaled)

        # Train/test split
        X_train, X_test, y_train, y_test = train_test_split(
            X_umap, y.values.ravel(), test_size=0.2, random_state=42
        )

        # Train classifier
        clf = LogisticRegression(max_iter=1500)
        clf.fit(X_train, y_train)

        # Evaluate
        y_pred = clf.predict(X_test)
        acc = accuracy_score(y_test, y_pred)

        tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

        sensitivity = tp / (tp + fn) if (tp + fn) > 0 else 0
        specificity = tn / (tn + fp) if (tn + fp) > 0 else 0
        # ------------------------------------------------
```

```
        accuracies.append((n, p, acc))

        print(f" -> Accuracy = {acc:.5f}")

        # Track best
        if acc > best_acc:
            best_acc = acc
            best_params = (n, p)
            # Save the specific metrics associated with this best result
            best_sensitivity = sensitivity
            best_specificity = specificity


# Print best result
print("\n==============================")
print(f" Best UMAP parameters:")
print(f"   n_components = {best_params[0]}")
print(f"   n_neighbours = {best_params[1]}")
print(f" Best accuracy  = {best_acc:.5f}")
print(f" Sensitivity    = {best_sensitivity:.5f}")
print(f" Specificity    = {best_specificity:.5f}")
print("==============================")

end_time = time.time()
elapsed = end_time - start_time

current, peak = tracemalloc.get_traced_memory()
tracemalloc.stop()

peak_mb = peak / 10**6

print(f"UMAP took: Time={elapsed:.2f}s, Memory={peak_mb:.2f}MB")

df = pd.DataFrame(accuracies, columns=["components", "n_neighbours",␣
 ↪"accuracy"])

pivot = df.pivot(index="components", columns="n_neighbours", values="accuracy")

plt.figure(figsize=(8, 5))
sns.heatmap(pivot, annot=True, fmt=".3f", cmap="viridis")
plt.title("UMAP Accuracy Heatmap")
plt.show()
```

```
Running UMAP sweeps…

Testing: n_components=1, n_neighbours=5 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
```

```
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.88596
Testing: n_components=1, n_neighbours=15 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.92982
Testing: n_components=1, n_neighbours=30 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.93860
Testing: n_components=1, n_neighbours=50 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.95614
Testing: n_components=2, n_neighbours=5 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.94737
Testing: n_components=2, n_neighbours=15 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.92982
Testing: n_components=2, n_neighbours=30 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.95614
Testing: n_components=2, n_neighbours=50 …
```

```
/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.96491
Testing: n_components=3, n_neighbours=5 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.93860
Testing: n_components=3, n_neighbours=15 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.94737
Testing: n_components=3, n_neighbours=30 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.96491
Testing: n_components=3, n_neighbours=50 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.96491
Testing: n_components=4, n_neighbours=5 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.94737
Testing: n_components=4, n_neighbours=15 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(
```

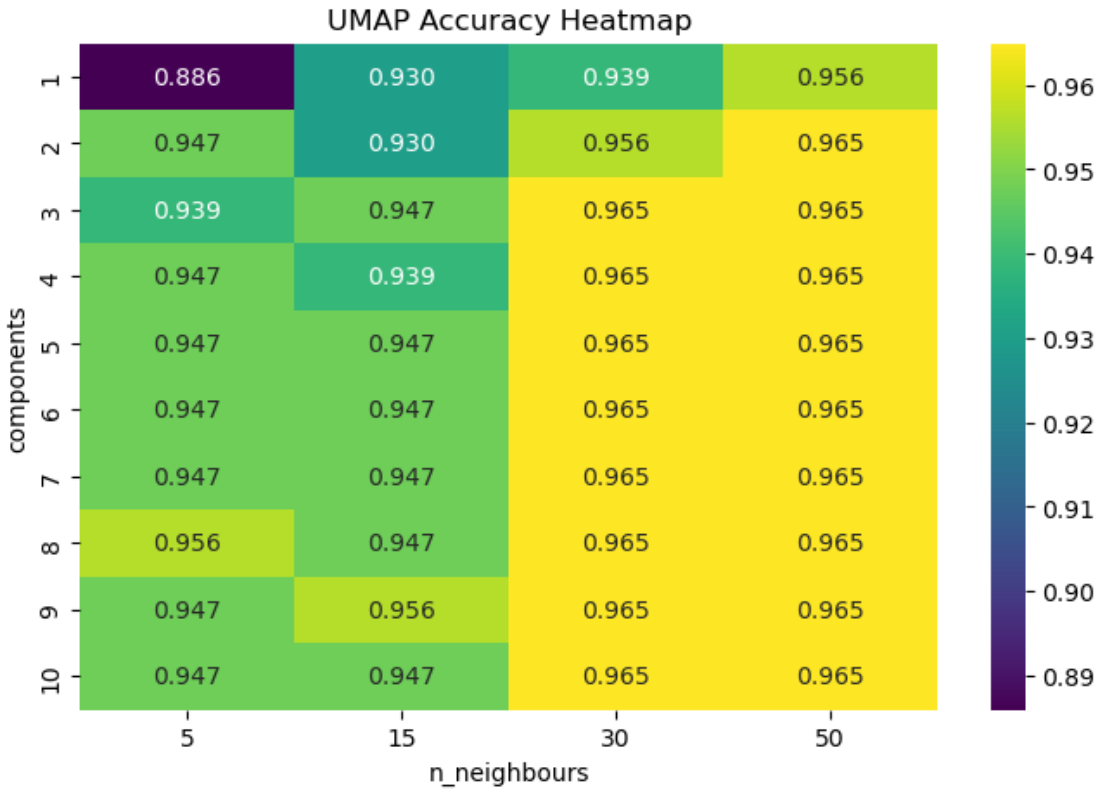
```
 -> Accuracy = 0.93860
Testing: n_components=4, n_neighbours=30 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.96491
Testing: n_components=4, n_neighbours=50 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.96491
Testing: n_components=5, n_neighbours=5 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.94737
Testing: n_components=5, n_neighbours=15 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.94737
Testing: n_components=5, n_neighbours=30 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.96491
Testing: n_components=5, n_neighbours=50 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.96491
Testing: n_components=6, n_neighbours=5 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
```

```
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.94737
Testing: n_components=6, n_neighbours=15 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.94737
Testing: n_components=6, n_neighbours=30 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.96491
Testing: n_components=6, n_neighbours=50 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.96491
Testing: n_components=7, n_neighbours=5 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.94737
Testing: n_components=7, n_neighbours=15 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.94737
Testing: n_components=7, n_neighbours=30 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.96491
Testing: n_components=7, n_neighbours=50 …
```

```
/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.96491
Testing: n_components=8, n_neighbours=5 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.95614
Testing: n_components=8, n_neighbours=15 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.94737
Testing: n_components=8, n_neighbours=30 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.96491
Testing: n_components=8, n_neighbours=50 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.96491
Testing: n_components=9, n_neighbours=5 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.94737
Testing: n_components=9, n_neighbours=15 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(
```

```
 -> Accuracy = 0.95614
Testing: n_components=9, n_neighbours=30 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.96491
Testing: n_components=9, n_neighbours=50 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.96491
Testing: n_components=10, n_neighbours=5 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.94737
Testing: n_components=10, n_neighbours=15 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.94737
Testing: n_components=10, n_neighbours=30 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.96491
Testing: n_components=10, n_neighbours=50 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.96491

==============================
 Best UMAP parameters:
   n_components = 2
   n_neighbours = 50
```

```
 Best accuracy  = 0.96491
 Sensitivity    = 0.95349
 Specificity    = 0.97183
==============================
UMAP took: Time=69.70s, Memory=6.36MB
```



And Isomap

[32]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

best_acc = 0
best_components = None

best_sensitivity = 0.0
best_specificity = 0.0

accuracies = []
n_neighbours_list = [5, 15, 30, 50]

print("Running Isomap sweeps...\n")
start_time = time.time()
```

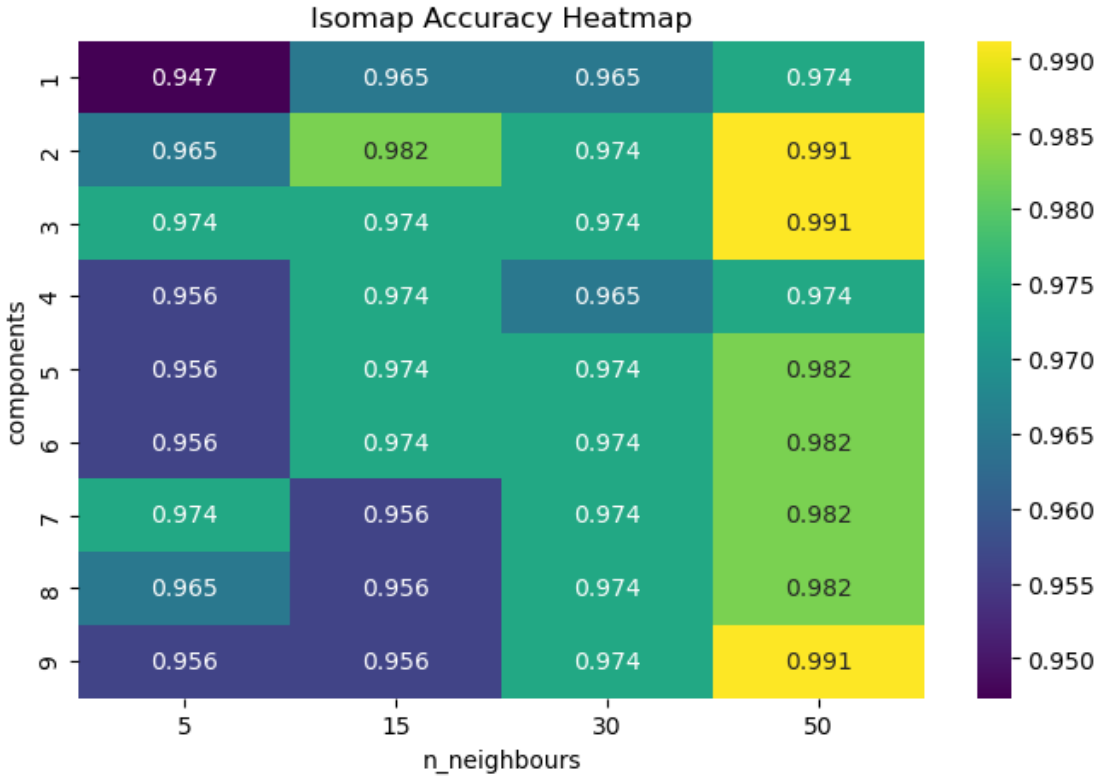
```
tracemalloc.start()
for n in range(1, 10):
    for p in n_neighbors_list:

        print(f"Testing: n_components={n}, n_neighbours={p} ...")

        isomap = Isomap(
            n_components=n,
            n_neighbors=p)

        X_isomap = isomap.fit_transform(X_scaled)

        # Train/test split
        X_train, X_test, y_train, y_test = train_test_split(
            X_isomap, y.values.ravel(), test_size=0.2, random_state=42
        )

        # Train classifier
        clf = LogisticRegression(max_iter=1500)
        clf.fit(X_train, y_train)

        # Evaluate
        y_pred = clf.predict(X_test)
        acc = accuracy_score(y_test, y_pred)

        tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

        sensitivity = tp / (tp + fn) if (tp + fn) > 0 else 0
        specificity = tn / (tn + fp) if (tn + fp) > 0 else 0
        # ----------------------------------------------------

        accuracies.append((n, p, acc))

        print(f" -> Accuracy = {acc:.5f}")

        # Track best
        if acc > best_acc:
            best_acc = acc
            best_params = (n, p)
            best_sensitivity = sensitivity
            best_specificity = specificity


# Print best result
print("\n==============================")
print(f" Best Isomap parameters:")
print(f"   n_components = {best_params[0]}")
```

```
print(f"   n_neighbours = {best_params[1]}")
print(f" Best accuracy  = {best_acc:.5f}")
print(f" Sensitivity    = {best_sensitivity:.5f}")
print(f" Specificity    = {best_specificity:.5f}")
print("================================")

end_time = time.time()
elapsed = end_time - start_time

current, peak = tracemalloc.get_traced_memory()
tracemalloc.stop()

peak_mb = peak / 10**6

print(f"Isomap took: Time={elapsed:.2f}s, Memory={peak_mb:.2f}MB")

df = pd.DataFrame(accuracies, columns=["components", "n_neighbours",␣
 ↪"accuracy"])

pivot = df.pivot(index="components", columns="n_neighbours", values="accuracy")

plt.figure(figsize=(8, 5))
sns.heatmap(pivot, annot=True, fmt=".3f", cmap="viridis")
plt.title("Isomap Accuracy Heatmap")
plt.show()
```

```
Running Isomap sweeps…

Testing: n_components=1, n_neighbours=5 …
 -> Accuracy = 0.94737
Testing: n_components=1, n_neighbours=15 …
 -> Accuracy = 0.96491
Testing: n_components=1, n_neighbours=30 …
 -> Accuracy = 0.96491
Testing: n_components=1, n_neighbours=50 …
 -> Accuracy = 0.97368
Testing: n_components=2, n_neighbours=5 …
 -> Accuracy = 0.96491
Testing: n_components=2, n_neighbours=15 …
 -> Accuracy = 0.98246
Testing: n_components=2, n_neighbours=30 …
 -> Accuracy = 0.97368
Testing: n_components=2, n_neighbours=50 …
 -> Accuracy = 0.99123
Testing: n_components=3, n_neighbours=5 …
 -> Accuracy = 0.97368
Testing: n_components=3, n_neighbours=15 …
 -> Accuracy = 0.97368
```

```
Testing: n_components=3, n_neighbours=30 …
 -> Accuracy = 0.97368
Testing: n_components=3, n_neighbours=50 …
 -> Accuracy = 0.99123
Testing: n_components=4, n_neighbours=5 …
 -> Accuracy = 0.95614
Testing: n_components=4, n_neighbours=15 …
 -> Accuracy = 0.97368
Testing: n_components=4, n_neighbours=30 …
 -> Accuracy = 0.96491
Testing: n_components=4, n_neighbours=50 …
 -> Accuracy = 0.97368
Testing: n_components=5, n_neighbours=5 …
 -> Accuracy = 0.95614
Testing: n_components=5, n_neighbours=15 …
 -> Accuracy = 0.97368
Testing: n_components=5, n_neighbours=30 …
 -> Accuracy = 0.97368
Testing: n_components=5, n_neighbours=50 …
 -> Accuracy = 0.98246
Testing: n_components=6, n_neighbours=5 …
 -> Accuracy = 0.95614
Testing: n_components=6, n_neighbours=15 …
 -> Accuracy = 0.97368
Testing: n_components=6, n_neighbours=30 …
 -> Accuracy = 0.97368
Testing: n_components=6, n_neighbours=50 …
 -> Accuracy = 0.98246
Testing: n_components=7, n_neighbours=5 …
 -> Accuracy = 0.97368
Testing: n_components=7, n_neighbours=15 …
 -> Accuracy = 0.95614
Testing: n_components=7, n_neighbours=30 …
 -> Accuracy = 0.97368
Testing: n_components=7, n_neighbours=50 …
 -> Accuracy = 0.98246
Testing: n_components=8, n_neighbours=5 …
 -> Accuracy = 0.96491
Testing: n_components=8, n_neighbours=15 …
 -> Accuracy = 0.95614
Testing: n_components=8, n_neighbours=30 …
 -> Accuracy = 0.97368
Testing: n_components=8, n_neighbours=50 …
 -> Accuracy = 0.98246
Testing: n_components=9, n_neighbours=5 …
 -> Accuracy = 0.95614
Testing: n_components=9, n_neighbours=15 …
 -> Accuracy = 0.95614
```

```
Testing: n_components=9, n_neighbours=30 …
 -> Accuracy = 0.97368
Testing: n_components=9, n_neighbours=50 …
 -> Accuracy = 0.99123

==============================
 Best Isomap parameters:
   n_components = 2
   n_neighbours = 50
 Best accuracy  = 0.99123
 Sensitivity    = 0.97674
 Specificity    = 1.00000
==============================
Isomap took: Time=4.72s, Memory=9.08MB
```



We can also first run PCA then use the non-linear dimensionality techniques. Lets use 3 PCs from PCA.

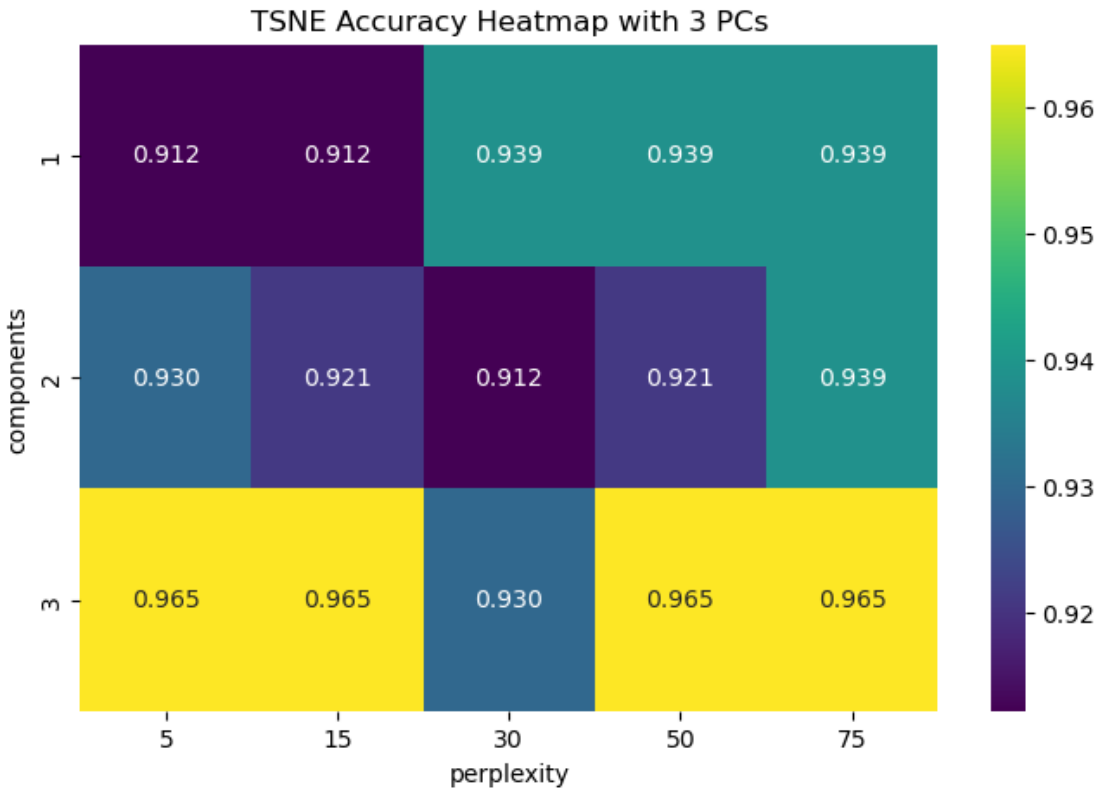
[33]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

pca = PCA(n_components=3)
```

```
X_pca = pca.fit_transform(X_scaled)

best_acc = 0
best_params = None
# Initialize variables to store the metrics of the best run
best_sensitivity = 0.0
best_specificity = 0.0

accuracies = []
perplexity_values = [5, 15, 30, 50, 75]

print("Running TSNE sweeps...\n")

start_time = time.time()
tracemalloc.start()
for n in range(1, 4):  # 1 to 3 components
    for p in perplexity_values:

        print(f"Testing: n_components={n}, perplexity={p} ...")

        tsne = TSNE(
            n_components=n,
            perplexity=p,
            random_state=42
        )

        X_tsne = tsne.fit_transform(X_pca)

        # Train/test split
        X_train, X_test, y_train, y_test = train_test_split(
            X_tsne, y.values.ravel(), test_size=0.2, random_state=42
        )

        # Train classifier
        clf = LogisticRegression(max_iter=1500)
        clf.fit(X_train, y_train)

        # Evaluate
        y_pred = clf.predict(X_test)
        acc = accuracy_score(y_test, y_pred)

        tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

        sensitivity = tp / (tp + fn) if (tp + fn) > 0 else 0
        specificity = tn / (tn + fp) if (tn + fp) > 0 else 0
        # ------------------------------------------------
```

```
        accuracies.append((n, p, acc))

        print(f" -> Accuracy = {acc:.5f}")

        # Track best
        if acc > best_acc:
            best_acc = acc
            best_params = (n, p)
            best_sensitivity = sensitivity
            best_specificity = specificity


# Print best result
print("\n==============================")
print(f" Best TSNE parameters:")
print(f"   n_components = {best_params[0]}")
print(f"   perplexity   = {best_params[1]}")
print(f" Best accuracy  = {best_acc:.5f}")
print(f" Sensitivity    = {best_sensitivity:.5f}")
print(f" Specificity    = {best_specificity:.5f}")
print("==============================")

end_time = time.time()
elapsed = end_time - start_time

current, peak = tracemalloc.get_traced_memory()

tracemalloc.stop()

peak_mb = peak / 10**6

print(f"TSNE + PCA took: Time={elapsed:.2f}s, Memory={peak_mb:.2f}MB")

df = pd.DataFrame(accuracies, columns=["components", "perplexity", "accuracy"])

pivot = df.pivot(index="components", columns="perplexity", values="accuracy")

plt.figure(figsize=(8, 5))
sns.heatmap(pivot, annot=True, fmt=".3f", cmap="viridis")
plt.title("TSNE Accuracy Heatmap with 3 PCs")
plt.show()
```

```
Running TSNE sweeps…

Testing: n_components=1, perplexity=5 …
 -> Accuracy = 0.91228
Testing: n_components=1, perplexity=15 …
 -> Accuracy = 0.91228
```

```
Testing: n_components=1, perplexity=30 …
 -> Accuracy = 0.93860
Testing: n_components=1, perplexity=50 …
 -> Accuracy = 0.93860
Testing: n_components=1, perplexity=75 …
 -> Accuracy = 0.93860
Testing: n_components=2, perplexity=5 …
 -> Accuracy = 0.92982
Testing: n_components=2, perplexity=15 …
 -> Accuracy = 0.92105
Testing: n_components=2, perplexity=30 …
 -> Accuracy = 0.91228
Testing: n_components=2, perplexity=50 …
 -> Accuracy = 0.92105
Testing: n_components=2, perplexity=75 …
 -> Accuracy = 0.93860
Testing: n_components=3, perplexity=5 …
 -> Accuracy = 0.96491
Testing: n_components=3, perplexity=15 …
 -> Accuracy = 0.96491
Testing: n_components=3, perplexity=30 …
 -> Accuracy = 0.92982
Testing: n_components=3, perplexity=50 …
 -> Accuracy = 0.96491
Testing: n_components=3, perplexity=75 …
 -> Accuracy = 0.96491

==============================
 Best TSNE parameters:
   n_components = 3
   perplexity   = 5
 Best accuracy  = 0.96491
 Sensitivity    = 0.93023
 Specificity    = 0.98592
==============================
TSNE + PCA took: Time=61.14s, Memory=8.00MB
```



Same thing with UMAP

[34]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

pca = PCA(n_components=3)
X_pca = pca.fit_transform(X_scaled)

best_acc = 0
best_components = None

best_sensitivity = 0.0
best_specificity = 0.0

accuracies = []
n_neighbours_list = [5, 15, 30, 50]

print("Running UMAP sweeps...\n")
start_time = time.time()
tracemalloc.start()
for n in range(1, 11):
    for p in n_neighbors_list:
```

```
        print(f"Testing: n_components={n}, n_neighbours={p} ...")

        umap_reducer = umap.UMAP(
            n_components=n,
            n_neighbors=p,
            random_state=42)

        X_umap = umap_reducer.fit_transform(X_pca)

        # Train/test split
        X_train, X_test, y_train, y_test = train_test_split(
            X_umap, y.values.ravel(), test_size=0.2, random_state=42
        )

        # Train classifier
        clf = LogisticRegression(max_iter=1500)
        clf.fit(X_train, y_train)

        # Evaluate
        y_pred = clf.predict(X_test)
        acc = accuracy_score(y_test, y_pred)

        tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

        sensitivity = tp / (tp + fn) if (tp + fn) > 0 else 0
        specificity = tn / (tn + fp) if (tn + fp) > 0 else 0
        # ------------------------------------------------

        accuracies.append((n, p, acc))

        print(f" -> Accuracy = {acc:.5f}")

        # Track best
        if acc > best_acc:
            best_acc = acc
            best_params = (n, p)
            best_sensitivity = sensitivity
            best_specificity = specificity


# Print best result
print("\n==============================")
print(f" Best UMAP parameters:")
print(f"   n_components = {best_params[0]}")
print(f"   n_neighbours = {best_params[1]}")
print(f" Best accuracy  = {best_acc:.5f}")
```

```
print(f" Sensitivity    = {best_sensitivity:.5f}")
print(f" Specificity    = {best_specificity:.5f}")
print("==============================")

end_time = time.time()
elapsed = end_time - start_time

current, peak = tracemalloc.get_traced_memory()

tracemalloc.stop()

peak_mb = peak / 10**6

print(f"UMAP + PCA took: Time={elapsed:.2f}s, Memory={peak_mb:.2f}MB")

df = pd.DataFrame(accuracies, columns=["components", "n_neighbours",␣
 ↪"accuracy"])

pivot = df.pivot(index="components", columns="n_neighbours", values="accuracy")

plt.figure(figsize=(8, 5))
sns.heatmap(pivot, annot=True, fmt=".3f", cmap="viridis")
plt.title("UMAP Accuracy Heatmap with 3 PCs")
plt.show()
```

```
Running UMAP sweeps…

Testing: n_components=1, n_neighbours=5 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.93860
Testing: n_components=1, n_neighbours=15 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.94737
Testing: n_components=1, n_neighbours=30 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(
```

```
 -> Accuracy = 0.94737
Testing: n_components=1, n_neighbours=50 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.94737
Testing: n_components=2, n_neighbours=5 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.92982
Testing: n_components=2, n_neighbours=15 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.92982
Testing: n_components=2, n_neighbours=30 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.92105
Testing: n_components=2, n_neighbours=50 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.92105
Testing: n_components=3, n_neighbours=5 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.93860
Testing: n_components=3, n_neighbours=15 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
```

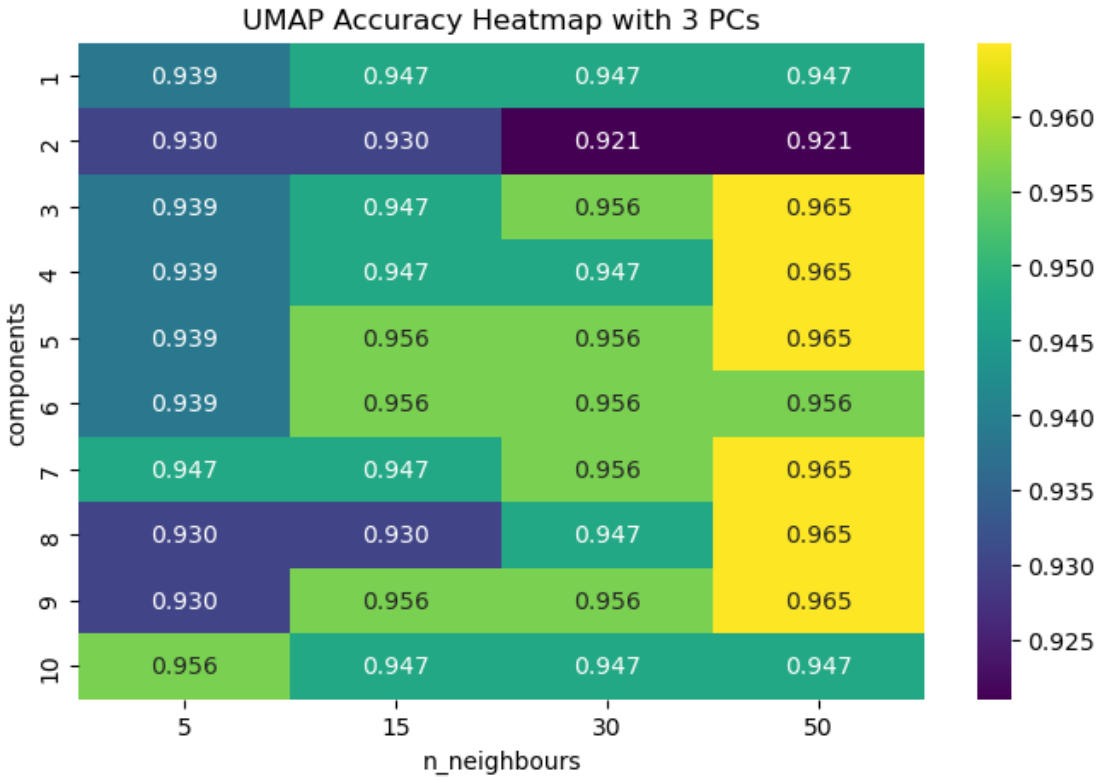
```
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.94737
Testing: n_components=3, n_neighbours=30 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.95614
Testing: n_components=3, n_neighbours=50 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.96491
Testing: n_components=4, n_neighbours=5 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.93860
Testing: n_components=4, n_neighbours=15 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.94737
Testing: n_components=4, n_neighbours=30 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.94737
Testing: n_components=4, n_neighbours=50 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.96491
Testing: n_components=5, n_neighbours=5 …
```

```
/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.93860
Testing: n_components=5, n_neighbours=15 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.95614
Testing: n_components=5, n_neighbours=30 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.95614
Testing: n_components=5, n_neighbours=50 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.96491
Testing: n_components=6, n_neighbours=5 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.93860
Testing: n_components=6, n_neighbours=15 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.95614
Testing: n_components=6, n_neighbours=30 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(
```

```
 -> Accuracy = 0.95614
Testing: n_components=6, n_neighbours=50 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.95614
Testing: n_components=7, n_neighbours=5 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.94737
Testing: n_components=7, n_neighbours=15 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.94737
Testing: n_components=7, n_neighbours=30 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.95614
Testing: n_components=7, n_neighbours=50 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.96491
Testing: n_components=8, n_neighbours=5 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.92982
Testing: n_components=8, n_neighbours=15 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
```

```
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.92982
Testing: n_components=8, n_neighbours=30 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.94737
Testing: n_components=8, n_neighbours=50 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.96491
Testing: n_components=9, n_neighbours=5 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.92982
Testing: n_components=9, n_neighbours=15 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.95614
Testing: n_components=9, n_neighbours=30 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.95614
Testing: n_components=9, n_neighbours=50 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.96491
Testing: n_components=10, n_neighbours=5 …
```

```
/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.95614
Testing: n_components=10, n_neighbours=15 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.94737
Testing: n_components=10, n_neighbours=30 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.94737
Testing: n_components=10, n_neighbours=50 …

/opt/homebrew/anaconda3/envs/syde575/lib/python3.12/site-
packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by
setting random_state. Use no seed for parallelism.
  warn(

 -> Accuracy = 0.94737

==============================
 Best UMAP parameters:
   n_components = 3
   n_neighbours = 50
 Best accuracy  = 0.96491
 Sensitivity    = 0.93023
 Specificity    = 0.98592
==============================
UMAP + PCA took: Time=66.76s, Memory=6.05MB
```



And lastly Isomap again

[35]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

pca = PCA(n_components=3)
X_pca = pca.fit_transform(X_scaled)

best_acc = 0
best_components = None

best_sensitivity = 0.0
best_specificity = 0.0

accuracies = []
n_neighbours_list = [5, 15, 30, 50]

print("Running Isomap sweeps...\n")

start_time = time.time()
tracemalloc.start()
for n in range(1, 10):
```

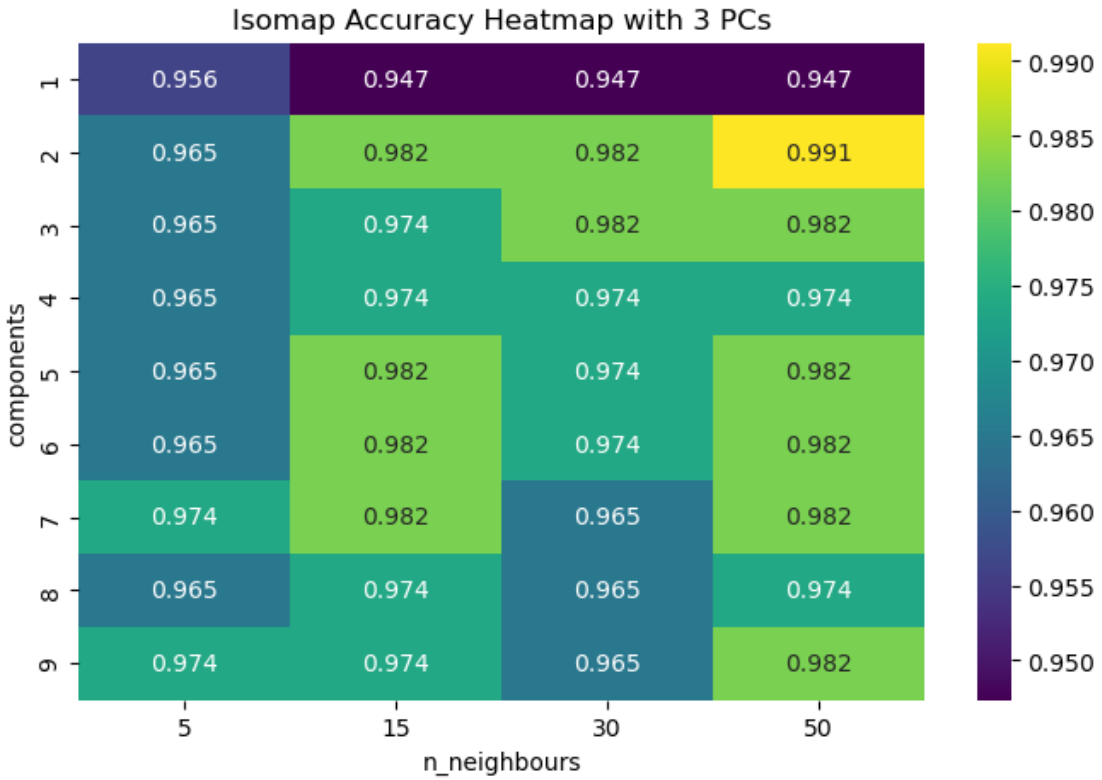
```
    for p in n_neighbors_list:

        print(f"Testing: n_components={n}, n_neighbours={p} ...")

        isomap = Isomap(
            n_components=n,
            n_neighbors=p)

        X_isomap = isomap.fit_transform(X_pca)

        # Train/test split
        X_train, X_test, y_train, y_test = train_test_split(
            X_isomap, y.values.ravel(), test_size=0.2, random_state=42
        )

        # Train classifier
        clf = LogisticRegression(max_iter=1500)
        clf.fit(X_train, y_train)

        # Evaluate
        y_pred = clf.predict(X_test)
        acc = accuracy_score(y_test, y_pred)

        tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

        sensitivity = tp / (tp + fn) if (tp + fn) > 0 else 0
        specificity = tn / (tn + fp) if (tn + fp) > 0 else 0
        # ------------------------------------------------

        accuracies.append((n, p, acc))

        print(f" -> Accuracy = {acc:.5f}")

        # Track best
        if acc > best_acc:
            best_acc = acc
            best_params = (n, p)
            best_sensitivity = sensitivity
            best_specificity = specificity


# Print best result
print("\n==============================")
print(f" Best Isomap parameters:")
print(f"   n_components = {best_params[0]}")
print(f"   n_neighbours = {best_params[1]}")
print(f" Best accuracy  = {best_acc:.5f}")
```

```
print(f" Sensitivity    = {best_sensitivity:.5f}")
print(f" Specificity    = {best_specificity:.5f}")
print("==============================")

end_time = time.time()
elapsed = end_time - start_time

current, peak = tracemalloc.get_traced_memory()

tracemalloc.stop()

peak_mb = peak / 10**6

print(f"Isomap + PCA took: Time={elapsed:.2f}s, Memory={peak_mb:.2f}MB")

df = pd.DataFrame(accuracies, columns=["components", "n_neighbours",␣
 ↪"accuracy"])

pivot = df.pivot(index="components", columns="n_neighbours", values="accuracy")

plt.figure(figsize=(8, 5))
sns.heatmap(pivot, annot=True, fmt=".3f", cmap="viridis")
plt.title("Isomap Accuracy Heatmap with 3 PCs")
plt.show()
```

```
Running Isomap sweeps…

Testing: n_components=1, n_neighbours=5 …
 -> Accuracy = 0.95614
Testing: n_components=1, n_neighbours=15 …
 -> Accuracy = 0.94737
Testing: n_components=1, n_neighbours=30 …
 -> Accuracy = 0.94737
Testing: n_components=1, n_neighbours=50 …
 -> Accuracy = 0.94737
Testing: n_components=2, n_neighbours=5 …
 -> Accuracy = 0.96491
Testing: n_components=2, n_neighbours=15 …
 -> Accuracy = 0.98246
Testing: n_components=2, n_neighbours=30 …
 -> Accuracy = 0.98246
Testing: n_components=2, n_neighbours=50 …
 -> Accuracy = 0.99123
Testing: n_components=3, n_neighbours=5 …
 -> Accuracy = 0.96491
Testing: n_components=3, n_neighbours=15 …
 -> Accuracy = 0.97368
Testing: n_components=3, n_neighbours=30 …
```

```
 -> Accuracy = 0.98246
Testing: n_components=3, n_neighbours=50 …
 -> Accuracy = 0.98246
Testing: n_components=4, n_neighbours=5 …
 -> Accuracy = 0.96491
Testing: n_components=4, n_neighbours=15 …
 -> Accuracy = 0.97368
Testing: n_components=4, n_neighbours=30 …
 -> Accuracy = 0.97368
Testing: n_components=4, n_neighbours=50 …
 -> Accuracy = 0.97368
Testing: n_components=5, n_neighbours=5 …
 -> Accuracy = 0.96491
Testing: n_components=5, n_neighbours=15 …
 -> Accuracy = 0.98246
Testing: n_components=5, n_neighbours=30 …
 -> Accuracy = 0.97368
Testing: n_components=5, n_neighbours=50 …
 -> Accuracy = 0.98246
Testing: n_components=6, n_neighbours=5 …
 -> Accuracy = 0.96491
Testing: n_components=6, n_neighbours=15 …
 -> Accuracy = 0.98246
Testing: n_components=6, n_neighbours=30 …
 -> Accuracy = 0.97368
Testing: n_components=6, n_neighbours=50 …
 -> Accuracy = 0.98246
Testing: n_components=7, n_neighbours=5 …
 -> Accuracy = 0.97368
Testing: n_components=7, n_neighbours=15 …
 -> Accuracy = 0.98246
Testing: n_components=7, n_neighbours=30 …
 -> Accuracy = 0.96491
Testing: n_components=7, n_neighbours=50 …
 -> Accuracy = 0.98246
Testing: n_components=8, n_neighbours=5 …
 -> Accuracy = 0.96491
Testing: n_components=8, n_neighbours=15 …
 -> Accuracy = 0.97368
Testing: n_components=8, n_neighbours=30 …
 -> Accuracy = 0.96491
Testing: n_components=8, n_neighbours=50 …
 -> Accuracy = 0.97368
Testing: n_components=9, n_neighbours=5 …
 -> Accuracy = 0.97368
Testing: n_components=9, n_neighbours=15 …
 -> Accuracy = 0.97368
Testing: n_components=9, n_neighbours=30 …
```

```
 -> Accuracy = 0.96491
Testing: n_components=9, n_neighbours=50 …
 -> Accuracy = 0.98246

==============================
 Best Isomap parameters:
   n_components = 2
   n_neighbours = 50
 Best accuracy  = 0.99123
 Sensitivity    = 0.97674
 Specificity    = 1.00000
==============================
Isomap + PCA took: Time=3.66s, Memory=8.80MB
```



The best accuracy for each non-linear technique was achieved when running it on the regularly scaled dataset as well as 3 PC components from PCA. However, PCA can help filter out some initial noise and reduce the size of the initial dataset as TSNE and UMAP are computationally intensive. We can see from the results, applying PCA then the non-linear technique preserves the accuracy, and in most cases, it reduces the compuational time and memory usage.