

```
In [22]: import numpy as np
import time
from nilearn.connectome import ConnectivityMeasure
from nilearn.datasets import fetch_abide_pcp
from nilearn.image import load_img
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix
import warnings
warnings.filterwarnings('ignore')

# Download ABIDE dataset with CC400 ROIs
print("Downloading ABIDE dataset with CC400 ROIs...")
abide_data = fetch_abide_pcp(
    derivatives=["rois_cc400"],
    pipeline="cpac",
    band_pass_filtering=True,
    global_signal_regression=True
)
```

Downloading ABIDE dataset with CC400 ROIs...  
[fetch\_abide\_pcp] Dataset found in /Users/xinyang/nilearn\_data/ABIDE\_p

```
In [25]: print("\nExtracting functional connectivity features...")

# 1. Create a functional connectivity estimator (correlation-based)
connectivity_measure = ConnectivityMeasure(kind='correlation')

# 2. Look at the first subject to infer number of ROIs
first_ts = np.asarray(abide_data.rois_cc400[0])
n_time_first, n_rois = first_ts.shape

print(f"Every subject has number of ROIs : {n_rois}")
# Every subject will have shape: (time_points, n_rois)

X = []
y = []

total_subjects = len(abide_data.rois_cc400)
successful = 0

for idx, (ts_obj, label) in enumerate(
    zip(abide_data.rois_cc400, abide_data.phenotypic['DX_GROUP']), 1
):

    # Convert time series to numpy array
    time_series = np.asarray(ts_obj)

    # 4. Compute correlation-based functional connectivity (n_rois x n_rois)
    conn_matrix = connectivity_measure.fit_transform([time_series])[0]

    # 5. Vectorize upper triangle
    upper_triangle = conn_matrix[np.triu_indices_from(conn_matrix, k=1)]
    X.append(upper_triangle)
```

```

# 6. Convert label to: autism=1, control=0
y_val = 1 if label == 1 else 0
y.append(y_val)

# Convert label to text
label_name = "autism" if y_val == 1 else "control"

successful += 1

# 7. One-line progress message (with label)
print(f"Subject {idx} | {label_name} | shape: {time_series.shape}
      | Connectivity OK | Done")

# ---- Summary ----
print(f"\nSuccessfully processed {successful}/{total_subjects} subjects")

X = np.array(X)
y = np.array(y)

print(f"Number of samples: {len(X)}")
if len(X) > 0:
    print(f"Feature dimension (length of each feature vector): {X.shape[1]}")
print(f"Class distribution (counts of [control, autism]): {np.bincount(y)}")

```



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

Subject 839	control	shape: (146, 392)	Connectivity OK	Done
Subject 840	control	shape: (146, 392)	Connectivity OK	Done
Subject 841	control	shape: (146, 392)	Connectivity OK	Done
Subject 842	control	shape: (146, 392)	Connectivity OK	Done
Subject 843	control	shape: (146, 392)	Connectivity OK	Done
Subject 844	control	shape: (196, 392)	Connectivity OK	Done
Subject 845	control	shape: (196, 392)	Connectivity OK	Done
Subject 846	control	shape: (196, 392)	Connectivity OK	Done
Subject 847	control	shape: (196, 392)	Connectivity OK	Done
Subject 848	control	shape: (196, 392)	Connectivity OK	Done
Subject 849	control	shape: (196, 392)	Connectivity OK	Done
Subject 850	control	shape: (196, 392)	Connectivity OK	Done
Subject 851	control	shape: (196, 392)	Connectivity OK	Done
Subject 852	control	shape: (196, 392)	Connectivity OK	Done
Subject 853	control	shape: (196, 392)	Connectivity OK	Done
Subject 854	control	shape: (196, 392)	Connectivity OK	Done
Subject 855	control	shape: (196, 392)	Connectivity OK	Done
Subject 856	control	shape: (196, 392)	Connectivity OK	Done
Subject 857	control	shape: (196, 392)	Connectivity OK	Done
Subject 858	autism	shape: (196, 392)	Connectivity OK	Done
Subject 859	autism	shape: (196, 392)	Connectivity OK	Done
Subject 860	autism	shape: (196, 392)	Connectivity OK	Done
Subject 861	autism	shape: (196, 392)	Connectivity OK	Done
Subject 862	autism	shape: (196, 392)	Connectivity OK	Done
Subject 863	autism	shape: (196, 392)	Connectivity OK	Done
Subject 864	autism	shape: (196, 392)	Connectivity OK	Done
Subject 865	autism	shape: (196, 392)	Connectivity OK	Done
Subject 866	autism	shape: (196, 392)	Connectivity OK	Done
Subject 867	autism	shape: (196, 392)	Connectivity OK	Done
Subject 868	autism	shape: (196, 392)	Connectivity OK	Done
Subject 869	autism	shape: (196, 392)	Connectivity OK	Done
Subject 870	autism	shape: (116, 392)	Connectivity OK	Done
Subject 871	autism	shape: (116, 392)	Connectivity OK	Done

Successfully processed 871/871 subjects

Number of samples: 871

Feature dimension (length of each feature vector): 76636

Class distribution (counts of [control, autism]): [468 403]

```
In [26]: # Grid search for optimal SVM parameters
print("\n" + "="*60)
print("Grid Search for Optimal SVM Parameters")
print("="*60)

# Define parameter grid
param_grid = {
    'C': [0.01, 1, 10, 100],
    'kernel': ['rbf', 'linear']
}

# Define 5-fold cross-validation
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Initialize GridSearchCV
svm = SVC(random_state=42)
```

```

# Grid Search
grid_search = GridSearchCV(
    svm,
    param_grid,
    cv=skf,
    scoring='accuracy',
    n_jobs=-1,
    verbose=1
)

# Fit grid search
print("\nPerforming grid search...")
start_time_grid = time.time()
grid_search.fit(X, y)
total_time_grid = time.time() - start_time_grid

# Print best parameters and best score
print(f"\nBest parameters: {grid_search.best_params_}")
print(f"Best cross-validation score: {grid_search.best_score_:.4f}")
print(f"Total time for grid search: {total_time_grid/60:.2f} minutes ({total_time_grid:.2f} seconds)")

# Get best model
best_svm = grid_search.best_estimator_

```

=====  
Grid Search for Optimal SVM Parameters  
=====

Performing grid search...

Fitting 5 folds for each of 8 candidates, totalling 40 fits

Best parameters: {'C': 10, 'kernel': 'rbf'}

Best cross-validation score: 0.6911

Total time for grid search: 3.09 minutes (185.36s)

```

In [27]: # Store metrics for each fold using best model
print("\n" + "="*60)
print("5-Fold Cross-Validation Results with Best Model")
print("="*60)

accuracies = []
sensitivities = []
specificities = []
f1_scores = []
fold_times = []

fold = 1
total_cv_time = 0 # track total time for CV

for train_idx, test_idx in skf.split(X, y):
    fold_start = time.time() # start timing this fold

    X_train, X_test = X[train_idx], X[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]

```

```

# Train best SVM model
best_svm.fit(X_train, y_train)

# Predict
y_pred = best_svm.predict(X_test)

# Calculate metrics
acc = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred, average='binary')

# Sensitivity & Specificity
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
sens = tp / (tp + fn)
spec = tn / (tn + fp)

fold_time = time.time() - fold_start # calculate fold time
total_cv_time += fold_time

# Store metrics
accuracies.append(acc)
sensitivities.append(sens)
specificities.append(spec)
f1_scores.append(f1)
fold_times.append(fold_time)

fold += 1

# ---- Print table for each fold ----
print("\nFold Accuracy Sensitivity Specificity F1 Score Time (s)")
for i in range(5):
    print(f"{i+1:<5} {accuracies[i]:.4f} {sensitivities[i]:.4f} "
          f"{specificities[i]:.4f} {f1_scores[i]:.4f} {fold_times[i]:.4f}")

# Calculate and display mean and std
print("\n" + "="*60)
print("Mean Results Across 5 Folds")
print("="*60)
print(f"Accuracy: {np.mean(accuracies):.4f} ± {np.std(accuracies):.4f}")
print(f"Sensitivity: {np.mean(sensitivities):.4f} ± {np.std(sensitivities):.4f}")
print(f"Specificity: {np.mean(specificities):.4f} ± {np.std(specificities):.4f}")
print(f"F1 Score: {np.mean(f1_scores):.4f} ± {np.std(f1_scores):.4f}")
print("="*60)

# Print total CV time
print(f"\nTotal time for 5-fold evaluation: {total_cv_time/60:.2f} minutes (")
print("="*60)

```

=====

5-Fold Cross-Validation Results with Best Model

=====

Fold	Accuracy	Sensitivity	Specificity	F1 Score	Time (s)
1	0.7086	0.6914	0.7234	0.6871	6.74
2	0.7011	0.5926	0.7957	0.6486	6.71
3	0.6954	0.5679	0.8065	0.6345	6.74
4	0.6897	0.6125	0.7553	0.6447	6.70
5	0.6609	0.6125	0.7021	0.6242	6.81

=====

Mean Results Across 5 Folds

=====

Accuracy: 0.6911 ± 0.0164  
Sensitivity: 0.6154 ± 0.0414  
Specificity: 0.7566 ± 0.0402  
F1 Score: 0.6478 ± 0.0214

=====

Total time for 5-fold evaluation: 0.56 minutes (33.71 seconds)

=====

In [ ]: