Dementia Classification Using Machine Learning: Logistic Regression and Support Vector Machines

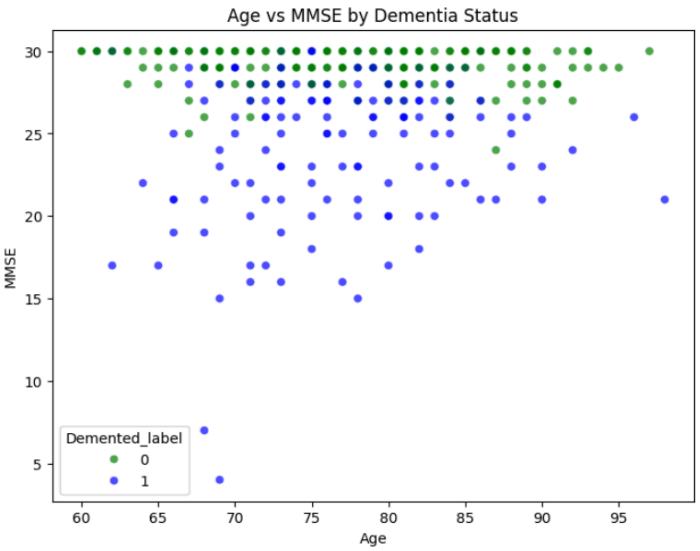
Code: [Github]

Introduction

Dementia is a progressive neurological condition that impairs cognitive functions such as memory, thinking, and reasoning, severely affecting daily life. Accurate and early diagnosis is crucial for patient care and treatment planning. Machine learning approaches like Logistic Regression and Support Vector Machines (SVM) are widely used to automate and improve the accuracy of dementia classification. These models analyze clinical and neuroimaging data to differentiate between demented and non-demented individuals efficiently.

Dataset Information

The dataset used for this study is the OASIS Longitudinal Demographics dataset, consisting of demographic, cognitive performance, and brain imaging measurements for subjects labeled as "Demented" or "Nondemented." Key features include Age, Education (EDUC), Socioeconomic Status (SES), Mini-Mental State Examination (MMSE), Clinical Dementia Rating (CDR), brain volume measurements (eTIV, nWBV, ASF), and demographic variables such as gender and handedness.



	Subject ID	MRI ID	Group	Visit	MR Delay	M/F	Hand	Age	EDUC	SES	MMSE	CDR	eTIV	nWBV	ASF
0	OAS2_0001	OAS2_0001_MR1	Nondemented	1	0	M	R	87	14	2.0	27.0	0.0	1986.550000	0.696106	0.883440
1	OAS2_0001	OAS2_0001_MR2	Nondemented	2	457	M	R	88	14	2.0	30.0	0.0	2004.479526	0.681062	0.875539
2	OAS2_0002	OAS2_0002_MR1	Demented	1	0	M	R	75	12	NaN	23.0	0.5	1678.290000	0.736336	1.045710
3	OAS2_0002	OAS2_0002_MR2	Demented	2	560	M	R	76	12	NaN	28.0	0.5	1737.620000	0.713402	1.010000
4	OAS2_0002	OAS2_0002_MR3	Demented	3	1895	M	R	80	12	NaN	22.0	0.5	1697.911134	0.701236	1.033623

Data Preprocessing

Categorical string variables, including the dementia status group, gender, and handedness, were converted to numerical labels to allow model compatibility. Missing values in the SES feature were imputed using the median value. Features were then standardized to zero mean and unit variance, and the dataset was split into training and testing sets using an 80:20 ratio.

Code:

```
df['Demented_label'] = df['Group'].apply(lambda x: 1 if
str(x).strip().lower() == 'demented' else 0)

df['M/F_encoded'] = df['M/F'].apply(lambda x: 1 if
str(x).strip().upper() == 'M' else 0)

df['Hand_encoded'] = df['Hand'].apply(lambda x: 0 if
str(x).strip().upper() == 'R' else 1)

df['SES'].fillna(df['SES'].median(), inplace=True)

df.head()
```

Output:

	Subject ID	MRI ID	Group	Visit	MR Delay	M/F	Hand	Age	EDUC	SES	MMSE	CDR	eTIV	nWBV	ASF	Demented_label	M/F_encoded	Hand_encoded
0	OAS2_0001	OAS2_0001_MR1	Nondemented	1	0	M	R	87	14	2.0	27.0	0.0	1986.550000	0.696106	0.883440	0	1	0
1	OAS2_0001	OAS2_0001_MR2	Nondemented	2	457	M	R	88	14	2.0	30.0	0.0	2004.479526	0.681062	0.875539	0	1	0
2	OAS2_0002	OAS2_0002_MR1	Demented	1	0	M	R	75	12	2.0	23.0	0.5	1678.290000	0.736336	1.045710	1	1	0
3	OAS2_0002	OAS2_0002_MR2	Demented	2	560	M	R	76	12	2.0	28.0	0.5	1737.620000	0.713402	1.010000	1	1	0
4	OAS2_0002	OAS2_0002_MR3	Demented	3	1895	M	R	80	12	2.0	22.0	0.5	1697.911134	0.701236	1.033623	1	1	0

Training and Testing:

The dataset is split into 80% for training and 20% for testing

Code:

```
from sklearn.model_selection import train_test_split

X = df[feature_cols]

Y = df['Demented_label']
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size=0.2, random_state=42, stratify=Y
)
print(f"Train samples: {len(X_train)}, Test samples: {len(X_test)}")
print(f"Train positive samples: {sum(y_train)}, Test positive samples:
{sum(y_test)}")
```

Logistic Regression: Theory Overview

Logistic Regression is a statistical model used for binary classification. It estimates the probability that a given input belongs to a particular class by applying the logistic function to a linear combination of input features. This model is interpretable and effective for problems where the relationship between features and the target is approximately linear. It predicts class membership by choosing a decision threshold on the output probability.

Code:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler

# Impute missing values
imputer = SimpleImputer(strategy='median')
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)

# Scale features
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train_imputed)

X_test_scaled = scaler.transform(X_test_imputed)



log_reg = LogisticRegression(max_iter=1000)

log_reg.fit(X_train_scaled, y_train)



y_pred = log_reg.predict(X_test_scaled)



print(classification_report(y_test, y_pred,
target_names=['Nondemented', 'Demented']))
```

Logistic Regression: Results

The logistic regression model was trained on the prepared dataset and evaluated on the test set. Its performance metrics are summarized below:

	precision	recall	f1-score	support
Nondemented	1.00	0.96	0.98	46
Demented	0.94	1.00	0.97	29
accuracy			0.97	75
macro avg	0.97	0.98	0.97	75
weighted avg	0.98	0.97	0.97	75

Support Vector Machines (SVM): Theory Overview

Support Vector Machine (SVM) is a supervised learning algorithm designed to classify data by finding the optimal hyperplane that maximally separates different classes. It focuses on the margin, defined as the distance from the hyperplane to the nearest data points (support vectors). The use of kernel functions allows SVM to handle non-linear class boundaries by transforming the data into higher-dimensional feature spaces.

Code:

```
from sklearn.impute import SimpleImputer

from sklearn.pipeline import Pipeline

from sklearn.svm import SVC

from sklearn.metrics import classification_report



preprocessing = Pipeline([

    ('imputer', SimpleImputer(strategy='median')),

    ('scaler', StandardScaler())

])



X_train_processed = preprocessing.fit_transform(X_train)

X_test_processed = preprocessing.transform(X_test)



svm_clf = SVC(kernel='rbf', gamma='scale', C=1)

svm_clf.fit(X_train_processed, y_train)



y_pred = svm_clf.predict(X_test_processed)

print(classification_report(y_test, y_pred,
target_names=['Nondemented', 'Demented']))
```

SVM: Results

An SVM with a radial basis function (RBF) kernel was trained and optimized using grid search hyperparameter tuning. The classification results for the test set are summarized below:

	precision	recall	f1-score	support
Nondemented	0.98	0.91	0.94	46
Demented	0.88	0.97	0.92	29
accuracy			0.93	75
macro avg	0.93	0.94	0.93	75
weighted avg	0.94	0.93	0.93	75

Principal Component Analysis (PCA)

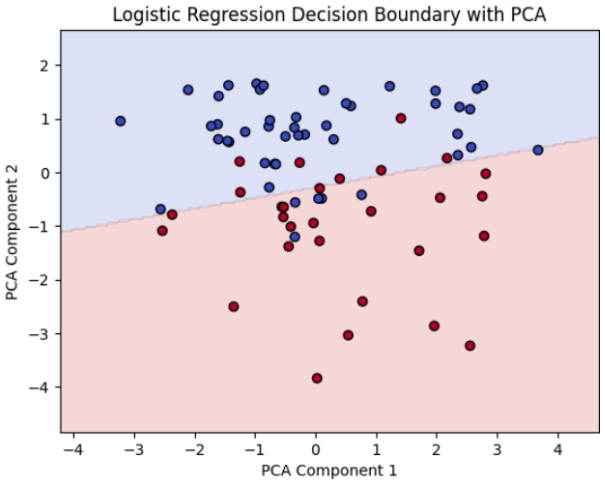
Principal Component Analysis (PCA) is a dimensionality reduction technique that transforms a high-dimensional dataset into a smaller set of new variables called principal components. These components capture the maximum variance present in the original data while being uncorrelated with each other. PCA helps to simplify complex datasets, reduce noise, and improve computational efficiency, particularly useful for visualization in two or three dimensions. By projecting data onto the principal components, PCA retains the most important information and allows models to work on simplified inputs.

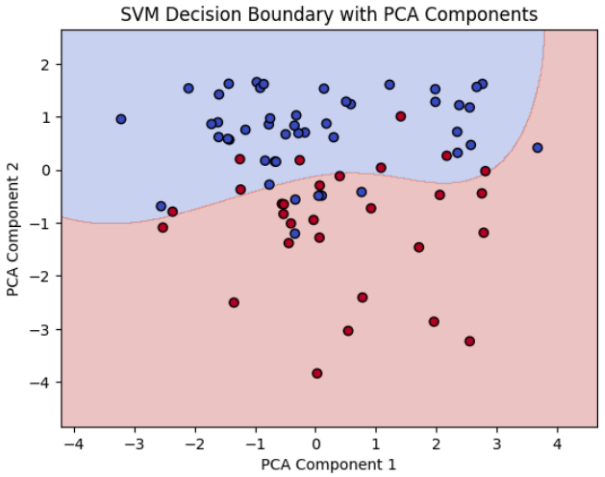
Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression


imputer = SimpleImputer(strategy='median')
scaler = StandardScaler()


X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)


X_train_scaled = scaler.fit_transform(X_train_imputed)
```

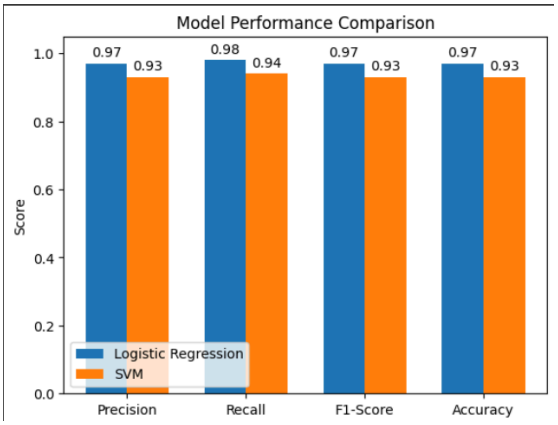
```
X_test_scaled = scaler.transform(X_test_imputed)


pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)


log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train_pca, y_train)


svm_clf = SVC(kernel='rbf', gamma='scale', C=1)
svm_clf.fit(X_train_pca, y_train)


def plot_logistic_decision_boundary(X, y, model):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 200),
                         np.linspace(y_min, y_max, 200))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)


    plt.contourf(xx, yy, Z, alpha=0.2, cmap=plt.cm.coolwarm)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm,
edgecolors='k')
    plt.xlabel('PCA Component 1')
    plt.ylabel('PCA Component 2')
    plt.title('Decision Boundary with PCA')
    plt.show()


plot_logistic_decision_boundary(X_test_pca, y_test, log_reg)
```

Output:





Comparing Logistic Regression and SVM for Classification



Both Logistic Regression and Support Vector Machines (SVM) are popular classification algorithms, but they differ in their approach and strengths. Logistic Regression models the probability of class membership using a logistic function and works well for linearly separable data with interpretable coefficients. SVM, on the other hand, aims to find the optimal decision boundary that maximizes the margin between classes and can handle both linear and non-linear separations through kernel functions. While Logistic Regression provides probabilistic outputs, SVMs focus on maximizing classification margins and tend to be more robust to outliers. The choice between them depends on the dataset characteristics, interpretability needs, and computational resources.

Conclusion

IIn this study, Logistic Regression and Support Vector Machines (SVM) were used to classify dementia from clinical and neuroimaging data. Both models showed strong performance, with SVM proving slightly more robust. PCA-based dimensionality reduction aided visualization, and future work may explore feature engineering, ensembles, and explainability to improve clinical relevance.