Machine Learning

In this jupyter notebook, we are training a machine learning model on the data that we exported from Matlab, with our features. We split the data into the 2 recordings, so that leakage was reduced - the first recording was used for the training dataset, and the second recording was used for testing. We compared usage of a Random Forest Classifier and Support Vector Machine. Our result is that the Random Forest Classifier has a higher accuracy then the support vector machine - therefore we will use it over the Support Vector Machine.

Importing the Data

In [79]:
```
import pandas as pd
```

In [80]:
```
df = pd.read_csv("ml_data.csv")
df.head()
```

Out[80]:

	avg_bpm	rmssd	sdnn	gsr_avg	mean_absolute_derivative	std_gs
0	71.287129	0.046690	0.046231	9339.129032	8.688751	552.34609
1	88.757396	0.012000	0.008485	8033.696062	10.214207	438.87837
2	84.905660	0.061400	0.053454	8002.003634	7.397931	410.89007
3	84.905660	0.061400	0.053454	7980.096348	7.805495	386.54236
4	81.190798	0.063676	0.078017	7962.944878	5.007646	365.67888

In [81]:
```
import sklearn.model_selection as ms
X = df.loc[:, df.columns != "stress"]
X.head()
```

Out[81]:

	avg_bpm	rmssd	sdnn	gsr_avg	mean_absolute_derivative	std_gs
0	71.287129	0.046690	0.046231	9339.129032	8.688751	552.34609
1	88.757396	0.012000	0.008485	8033.696062	10.214207	438.87837
2	84.905660	0.061400	0.053454	8002.003634	7.397931	410.89007
3	84.905660	0.061400	0.053454	7980.096348	7.805495	386.54236
4	81.190798	0.063676	0.078017	7962.944878	5.007646	365.67888

In [82]:
```
Y = df.loc[:, df.columns == "stress"]
Y.head()
```

Out[82]:

	stress
0	0
1	0
2	0
3	0
4	0

In [83]:
```
from sklearn.model_selection import train_test_split

# split it so that the first recording is training dataset, and 2nd recor
# data comes like this, all calm recordings 1 and 2, then all stress reco
# We want all calm recordings 1 and all stress recordings 1 in X_train
from collections import Counter
import pandas as pd
from sklearn.utils import shuffle

print(Counter(Y["stress"]))

calm_recording_1_length = int(1168/4)
calm_recording_2_length = int(1200/4)
stress_recording_1_length = int(1284/4)
stress_recording_2_length = int(1520/4)

# Define slice indices
calm1_start = 0
calm1_end = calm1_start + calm_recording_1_length

calm2_start = calm1_end
calm2_end = calm2_start + calm_recording_2_length

stress1_start = calm2_end
stress1_end = stress1_start + stress_recording_1_length

stress2_start = stress1_end
stress2_end = stress2_start + stress_recording_2_length

X_train = pd.concat([
    X.iloc[calm1_start:calm1_end, :],
    X.iloc[stress1_start:stress1_end, :]
])

X_test = pd.concat([
    X.iloc[calm2_start:calm2_end, :],
    X.iloc[stress2_start:stress2_end, :]
])

Y_train = pd.concat([
    Y.iloc[calm1_start:calm1_end],
    Y.iloc[stress1_start:stress1_end]
])

Y_test = pd.concat([
    Y.iloc[calm2_start:calm2_end],
    Y.iloc[stress2_start:stress2_end]
```

```
])

X_train, Y_train = shuffle(X_train, Y_train, random_state=42)
X_test, Y_test = shuffle(X_test, Y_test, random_state=42)

print(X_train.shape, X_test.shape)
print(Y_train.shape, Y_test.shape)
```

```
Counter({1: 699, 0: 590})
(613, 6) (676, 6)
(613, 1) (676, 1)
```

In [84]:
```
X_train.head()
```

Out[84]:

	avg_bpm	rmssd	sdnn	gsr_avg	mean_absolute_derivative	std_
670	77.888360	0.098752	0.119164	3359.891429	1.531244	348.858
101	69.870845	0.088546	0.113002	4828.571973	2.764526	824.900
131	66.683132	0.146319	0.149301	6124.757865	2.462128	396.242
693	85.376532	0.048758	0.053750	2257.462345	1.308668	381.997
909	72.727273	0.106623	0.103216	754.154143	2.022058	433.567

In [85]:
```
Y_train.head()
```

Out[85]:

	stress
670	1
101	0
131	0
693	1
909	1

Training the SVM (Support Vector Machine)

In [86]:
```
from sklearn.svm import SVC
model = SVC()
model
```

Out[86]:

▾ SVC ⓘ ⓒ

▸ Parameters

	C	1.0
	kernel	'rbf'
	degree	3
	gamma	'scale'
	coef0	0.0
	shrinking	True
	probability	False
	tol	0.001
	cache_size	200
	class_weight	None
	verbose	False
	max_iter	-1
	decision_function_shape	'ovr'
	break_ties	False
	random_state	None

In [87]:

```
import numpy as np
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
model.fit(X_train_scaled, np.ravel(Y_train))
```

Out[87]:

▾ SVC ⓘ ?

▸ Parameters

	C	1.0
	kernel	'rbf'
	degree	3
	gamma	'scale'
	coef0	0.0
	shrinking	True
	probability	False
	tol	0.001
	cache_size	200
	class_weight	None
	verbose	False
	max_iter	-1
	decision_function_shape	'ovr'
	break_ties	False
	random_state	None

In [88]:
```
model.score(X_test_scaled, Y_test)
```

Out[88]: 0.8683431952662722

In [89]:
```
from sklearn.metrics import classification_report, confusion_matrix

predictions = model.predict(X_test_scaled)
print(classification_report(Y_test, predictions, zero_division=0))
```

```
              precision    recall  f1-score   support

           0       0.77      1.00      0.87       298
           1       1.00      0.76      0.87       378

    accuracy                           0.87       676
   macro avg       0.89      0.88      0.87       676
weighted avg       0.90      0.87      0.87       676
```

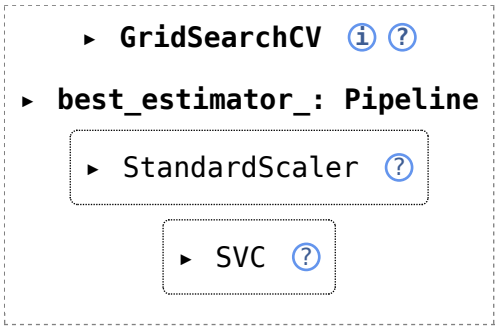
In [90]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline

pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('svm', SVC())
])
```

```
param_grid = {'svm__C': [0.1, 1, 10, 100, 1000],
                          'svm__gamma': [1, 0.1, 0.01, 0.001, 0.0001, 'scal
                          'svm__kernel': ['rbf']}
grid = GridSearchCV(pipeline, param_grid, verbose=0)
```

In [91]:
```
grid.fit(X_train, np.ravel(Y_train))
```

Out[91]:



In [92]:
```
print(grid.best_params_)
print(grid.best_estimator_)
print(grid.best_score_)
```

```
{'svm__C': 0.1, 'svm__gamma': 1, 'svm__kernel': 'rbf'}
Pipeline(steps=[('scaler', StandardScaler()), ('svm', SVC(C=0.1, gamma=
1))])
1.0
```

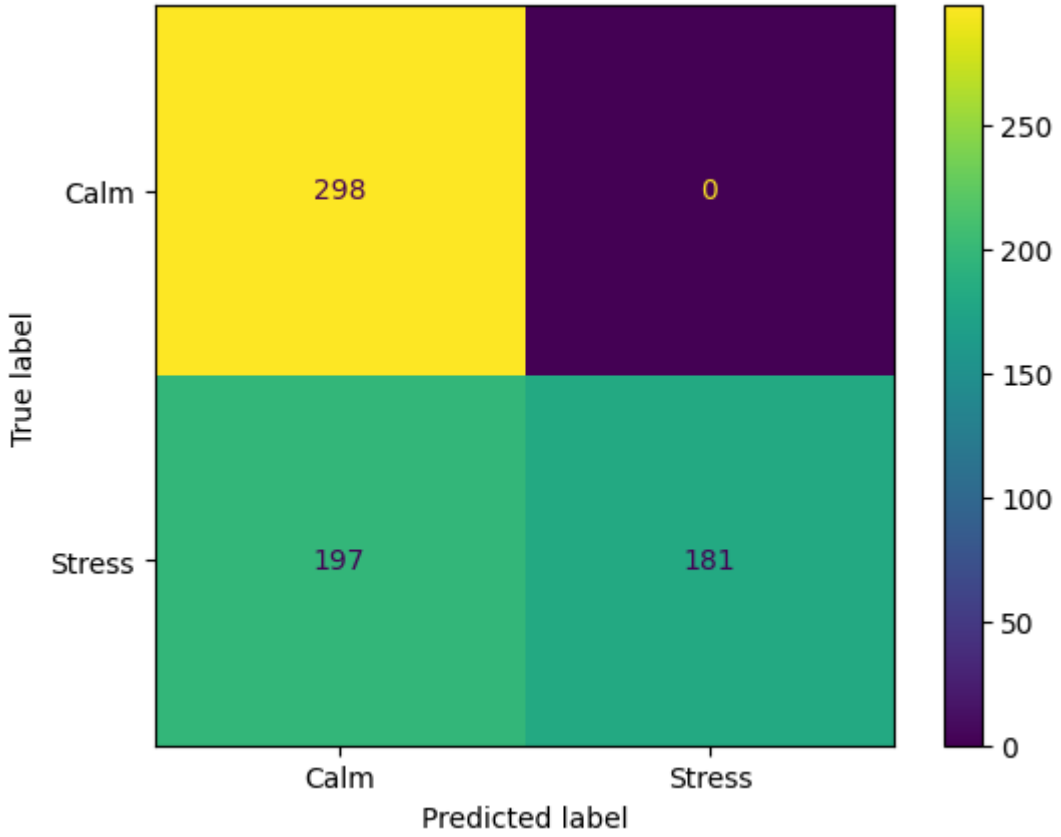
In [93]:
```
grid_predictions = grid.predict(X_test)
print(grid.score(X_test, Y_test))
print(classification_report(Y_test, grid_predictions, zero_division=0))
```

```
0.7085798816568047
              precision    recall  f1-score   support

           0       0.60      1.00      0.75       298
           1       1.00      0.48      0.65       378

    accuracy                           0.71       676
   macro avg       0.80      0.74      0.70       676
weighted avg       0.82      0.71      0.69       676
```

In [94]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
cm = confusion_matrix(Y_test, grid_predictions)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Calm"
disp.plot()
plt.show()
```



Results of SVM

Our results for SVM was an accuracy 71%

Training Random Forest Classifier

In [95]:
```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()
```

In [96]:
```
param_grid_rf = {
    'max_depth': [3,5,7,10],
    'n_estimators': [100, 200, 300, 400, 500],
    'max_features': [10, 20, 30 , 40],
    'min_samples_leaf': [1, 2, 4]
}

grid_rf = GridSearchCV(rf, param_grid_rf, scoring='f1', n_jobs=-1, verbos
grid_rf.fit(X_train, np.ravel(Y_train))
```

Out[96]:

▸ **GridSearchCV** ⓘ ?

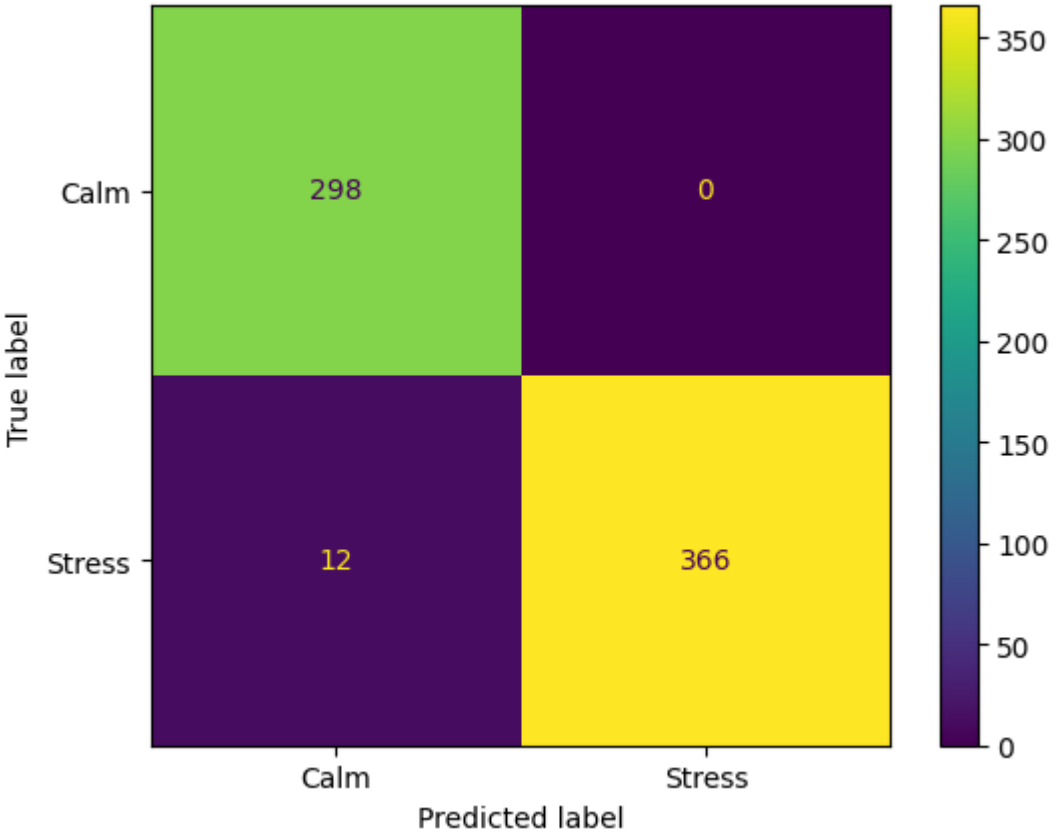
▸ **best_estimator_: RandomForestClassifier**

▸ RandomForestClassifier ?

In [97]:
```
grid_rf.score(X_test, Y_test)
```

Out[97]: 0.9838709677419355

In [98]:
```
cm_rf = confusion_matrix(Y_test, grid_rf.predict(X_test))
disp = ConfusionMatrixDisplay(cm_rf, display_labels=["Calm", "Stress"])
disp.plot()
plt.show()
```



Results

Random Forest Classifier has a far better score at 98% compared to Support Vector Machine's 71%. Therefore we will use a Random Forest Classifier over the Support Vector Machine for this dataset, due to the improved performance.

In [99]:
```
import joblib
joblib.dump(grid_rf, "model.pkl")
```

Out[99]: ['model.pkl']