main-notebook-output

April 15, 2024

1 Data Loading

[ ]:
```
from scipy import stats
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import (
    train_test_split,
    StratifiedShuffleSplit,
    GridSearchCV
)
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,␣
 ↪RocCurveDisplay
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from utils import custom_calculate_f1_score, get_all_metrics,␣
 ↪TorchKFoldCrossValidation
import torch
from torch import nn
from torch.utils.data import TensorDataset
from torchinfo import summary
```

[ ]:
```
df = pd.read_csv('./dataset/Dry_Bean_Dataset.csv')
df.head()
```

[ ]:
```
    Area  Perimeter  MajorAxisLength  MinorAxisLength  AspectRation  \
0  28395    610.291       208.178117       173.888747      1.197191
1  28734    638.018       200.524796       182.734419      1.097356
2  29380    624.110       212.826130       175.931143      1.209713
3  30008    645.884       210.557999       182.516516      1.153638
4  30140    620.134       201.847882       190.279279      1.060798

   Eccentricity  ConvexArea  EquivDiameter    Extent  Solidity  roundness  \
0      0.549812       28715     190.141097  0.763923  0.988856   0.958027
1      0.411785       29172     191.272751  0.783968  0.984986   0.887034
2      0.562727       29690     193.410904  0.778113  0.989559   0.947849
```

```
3      0.498616       30724      195.467062  0.782681  0.976696   0.903936
4      0.333680       30417      195.896503  0.773098  0.990893   0.984877

   Compactness  ShapeFactor1  ShapeFactor2  ShapeFactor3  ShapeFactor4  Class
0     0.913358      0.007332      0.003147      0.834222      0.998724  SEKER
1     0.953861      0.006979      0.003564      0.909851      0.998430  SEKER
2     0.908774      0.007244      0.003048      0.825871      0.999066  SEKER
3     0.928329      0.007017      0.003215      0.861794      0.994199  SEKER
4     0.970516      0.006697      0.003665      0.941900      0.999166  SEKER
```

[ ]:
```
## Dataset is multi-class, but we will convert it to binary
df['Class'] = df['Class'].apply(lambda x: 'DERMASON' if x == 'DERMASON' else␣
↪'Non-DERMASON')
```

2 Data analysis

2.0.1 Check for nullish values

The isna function checks if there are any nullish values in the dataframe or not

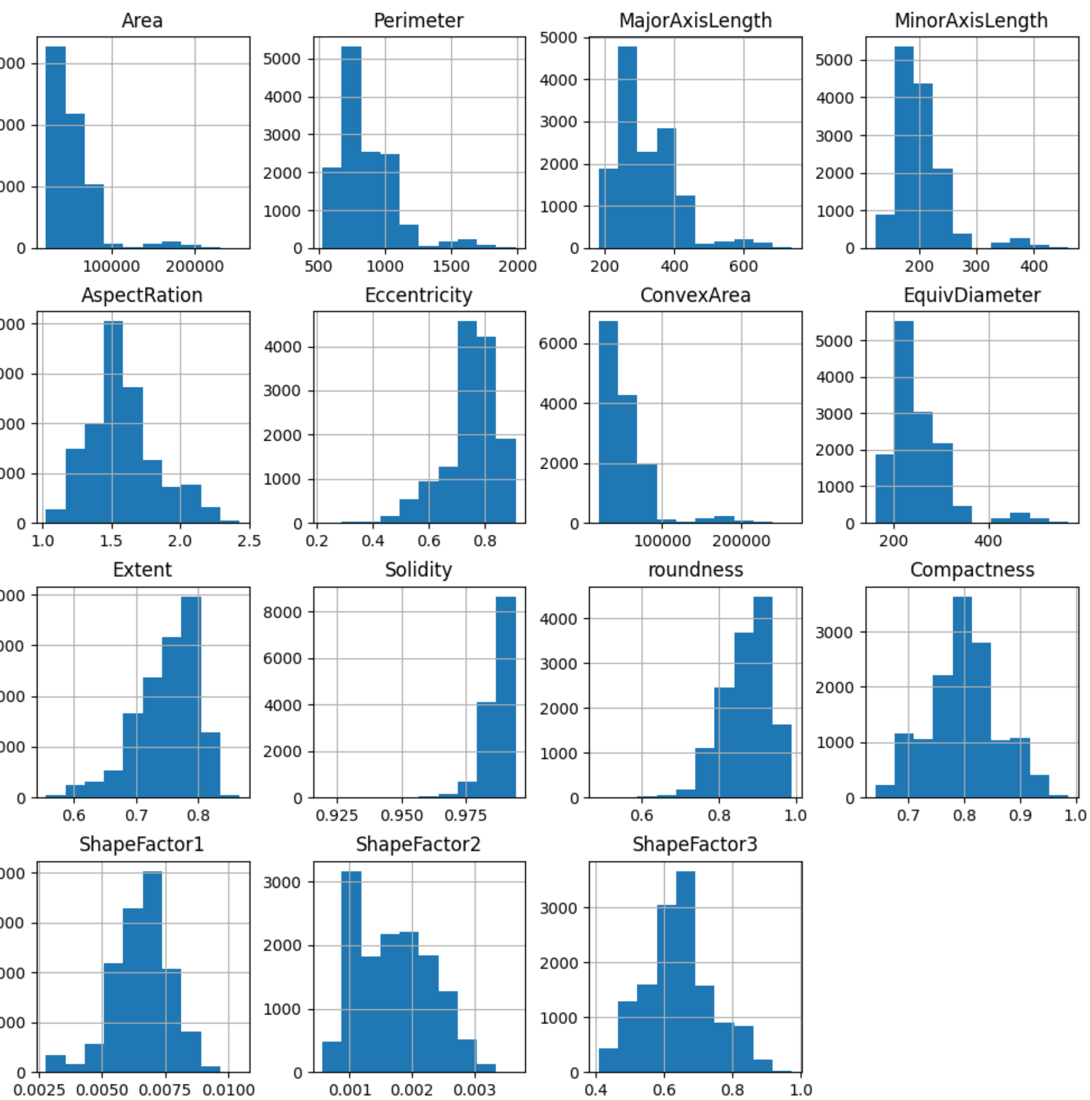
[ ]:
```
df.isna().any()
```

[ ]:
```
Area               False
Perimeter          False
MajorAxisLength    False
MinorAxisLength    False
AspectRation       False
Eccentricity       False
ConvexArea         False
EquivDiameter      False
Extent             False
Solidity           False
roundness          False
Compactness        False
ShapeFactor1       False
ShapeFactor2       False
ShapeFactor3       False
ShapeFactor4       False
Class              False
dtype: bool
```

2.0.2 Data distribution

From the histograms below, it can be observed that the value range of the attributes are high and needs to be standardized for ML models to converge faster

[ ]:
```
_ = df.iloc[:, :-2].hist(figsize=(12, 12))
```



2.0.3 Correlations

From the correlations heatmap below, it can be observed: 1. Some features have very high correlation with other variables (so they are not independent) 2. Some attributes like Area, Perimeter, MajorAxisLength, MinorAxisLength, etc have direct correlation to some extent with labels

[ ]:
```
class_factorized_df = pd.concat([df.iloc[:, :-1], pd.Series(df['Class'].
  ↪factorize()[0], name='Class')], axis=1)
class_factorized_df.corr().style.background_gradient(cmap='coolwarm')
```

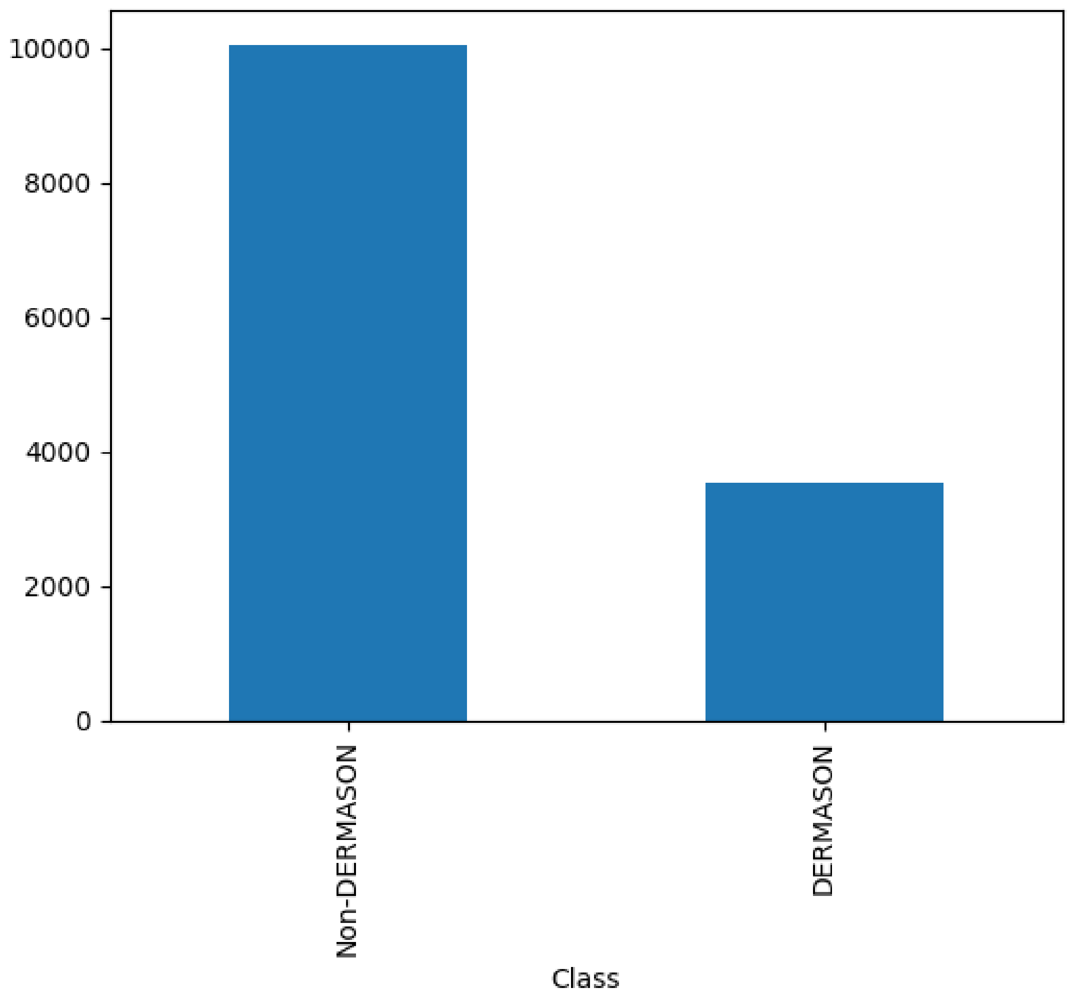
[ ]:
```
<pandas.io.formats.style.Styler at 0x31baa4b90>
```

2.0.4 Label distribution

From the below bar graph, we can infer following points: 1. There aren't any out-of-place/nullish class values 2. We need to convert the string values of class to numerical representation 3. The class distribution is unbalanced, so the training data needs to be split with stratification.

[ ]:
```
df['Class'].value_counts().plot(kind='bar')
```

[ ]:
```
<Axes: xlabel='Class'>
```



3 Data Preprocessing

3.0.1 Converting string to numerical representation

[ ]:
```
encoder = LabelEncoder()
df['Class_numerical'] = encoder.fit_transform(df['Class'])
```

[ ]:
```
encoder.inverse_transform([0, 1])
```

[ ]:
```
array(['DERMASON', 'Non-DERMASON'], dtype=object)
```

3.0.2 Normalizing dataset

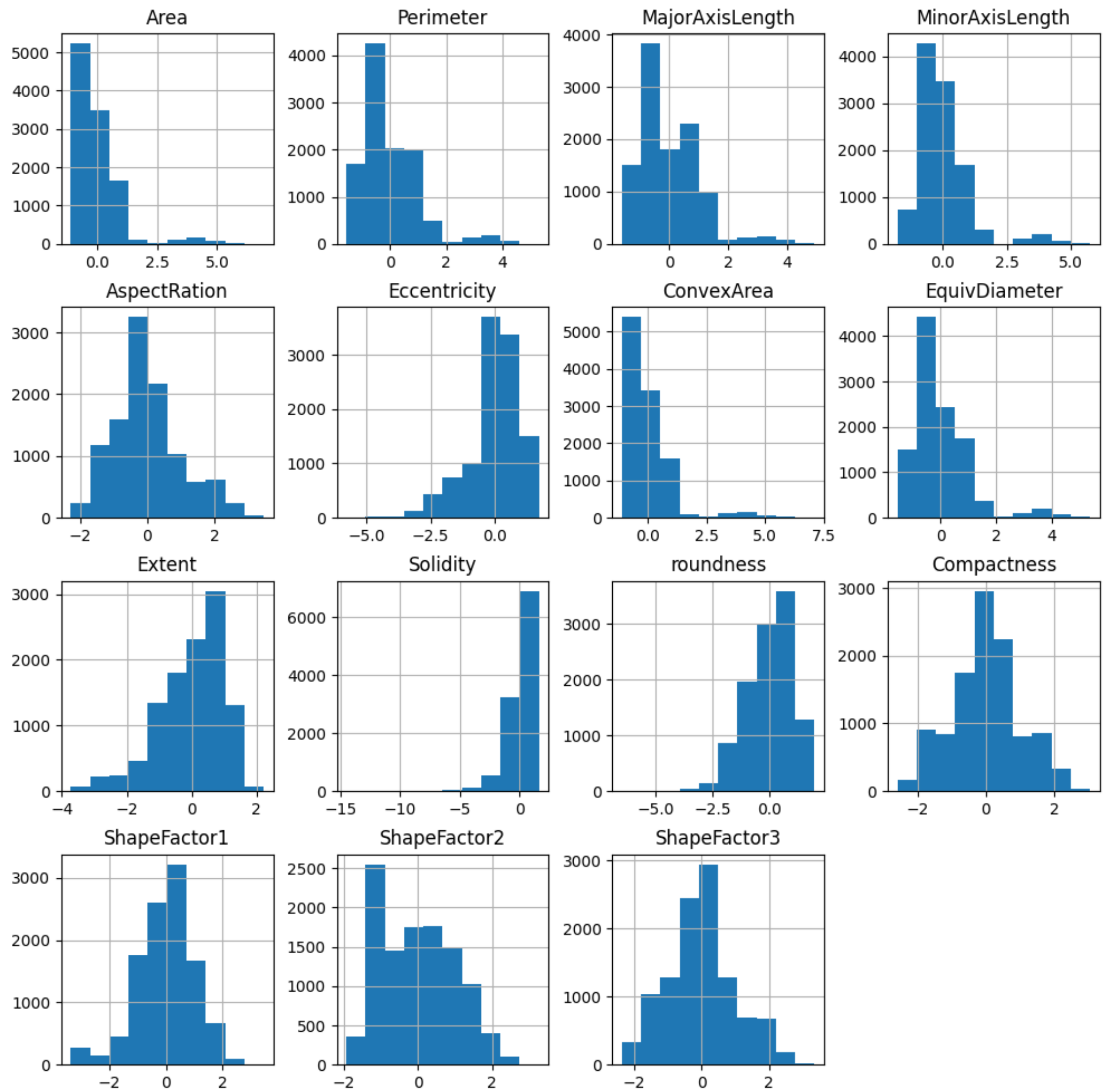
[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(
    df.iloc[:, :-2],
    df["Class_numerical"],
    test_size=0.2,
    random_state=42,
    shuffle=True,
    stratify=df["Class_numerical"],
)
```

[ ]:
```
scaler = StandardScaler()
scaled_features = scaler.fit_transform(X_train)
X_train = pd.DataFrame(scaled_features, columns=df.columns[:-2])
X_test = pd.DataFrame(scaler.transform(X_test), columns=df.columns[:-2])
X_train.head()
```

[ ]:
```
       Area  Perimeter  MajorAxisLength  MinorAxisLength  AspectRation  \
0 -0.094682   0.123942         0.534241        -0.594340      2.031821
1  0.878664   1.006483         1.163613         0.850757      0.655072
2 -0.692833  -0.941668        -1.172820        -0.268002     -1.735064
3 -0.224897  -0.212262        -0.184148        -0.158521     -0.095074
4 -0.211924  -0.156527        -0.107921        -0.217161      0.129117

   Eccentricity  ConvexArea  EquivDiameter    Extent  Solidity  roundness  \
0      1.374019   -0.099211      -0.001218 -0.114303  0.462137  -1.022080
1      0.743798    0.868142       1.074000  0.643785  0.584813  -0.178636
2     -2.703900   -0.696906      -0.823466 -0.069434  0.889031   1.534785
3      0.178891   -0.229717      -0.166248 -0.094418  0.579471   0.286425
4      0.372271   -0.210366      -0.149512 -0.200857 -0.266950  -0.025981

   Compactness  ShapeFactor1  ShapeFactor2  ShapeFactor3  ShapeFactor4
0    -1.756226      0.630856     -1.156022     -1.671171      0.369200
1    -0.738280     -1.089141     -1.092138     -0.753530     -0.541939
2     2.103855      0.121343      2.297855      2.228919      0.873688
3    -0.011595     -0.011651     -0.113542     -0.049872      0.228672
4    -0.231699      0.064534     -0.262659     -0.267274      0.350360
```

As you can see in below histograms, the range of features are normalized

[ ]:
```
_ = X_train.iloc[:, :-1].hist(figsize=(12, 12))
```



4 Model Training

[ ]:
```
results_comparison = pd.DataFrame({'RandomForest': {}, 'SVM': {}, 'KNN': {},␣
↪'Conv1D-NN': {}})
```

4.1 Random Forest

[ ]:
```
# params = {
#     "n_estimators": [100, 200, 300, 400],
#     "max_depth": [8, 10, 12],
#     "ccp_alpha": [5e-4, 1e-3],
# }
stratified_split = StratifiedShuffleSplit(n_splits=10, test_size=0.1,␣
↪random_state=42)
# gridsearch_rf = GridSearchCV(
#     RandomForestClassifier(n_jobs=-1),
#     params,
#     cv=stratified_split,
#     n_jobs=-1,
#     verbose=1,
#     scoring=custom_calculate_f1_score
# )
# gridsearch_rf.fit(X_train, y_train)
# gridsearch_rf.best_params_

# If you uncomment the commented code above, you will get the following output
# Output - {'ccp_alpha': 0.0005, 'max_depth': 12, 'n_estimators': 300}
```

```
Fitting 10 folds for each of 24 candidates, totalling 240 fits
```

[ ]:
```
{'ccp_alpha': 0.0005, 'max_depth': 12, 'n_estimators': 400}
```

[ ]:
```
random_forest_cv = {}
random_forest_cv_models = []

for i, (train_index, test_index) in enumerate(stratified_split.split(X_train,␣
↪y_train)):
    X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.
↪iloc[test_index]
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.
↪iloc[test_index]

    random_forest = RandomForestClassifier(n_estimators=300, max_depth=12,␣
↪ccp_alpha=5e-4, n_jobs=-1)
    random_forest.fit(X_train_fold, y_train_fold)

    y_pred = random_forest.predict(X_val_fold)
    y_pred_proba = random_forest.predict_proba(X_val_fold)[:, 1]
    random_forest_cv[i+1] = get_all_metrics(y_val_fold, y_pred)
    random_forest_cv[i+1]['Brier Score'] = np.mean((y_pred_proba -␣
↪y_val_fold)**2)
    random_forest_cv[i+1]['Brier Skill Score'] = random_forest_cv[i+1]['Brier␣
↪Score'] / (np.mean((y_val_fold - np.mean(y_pred_proba))**2))
```

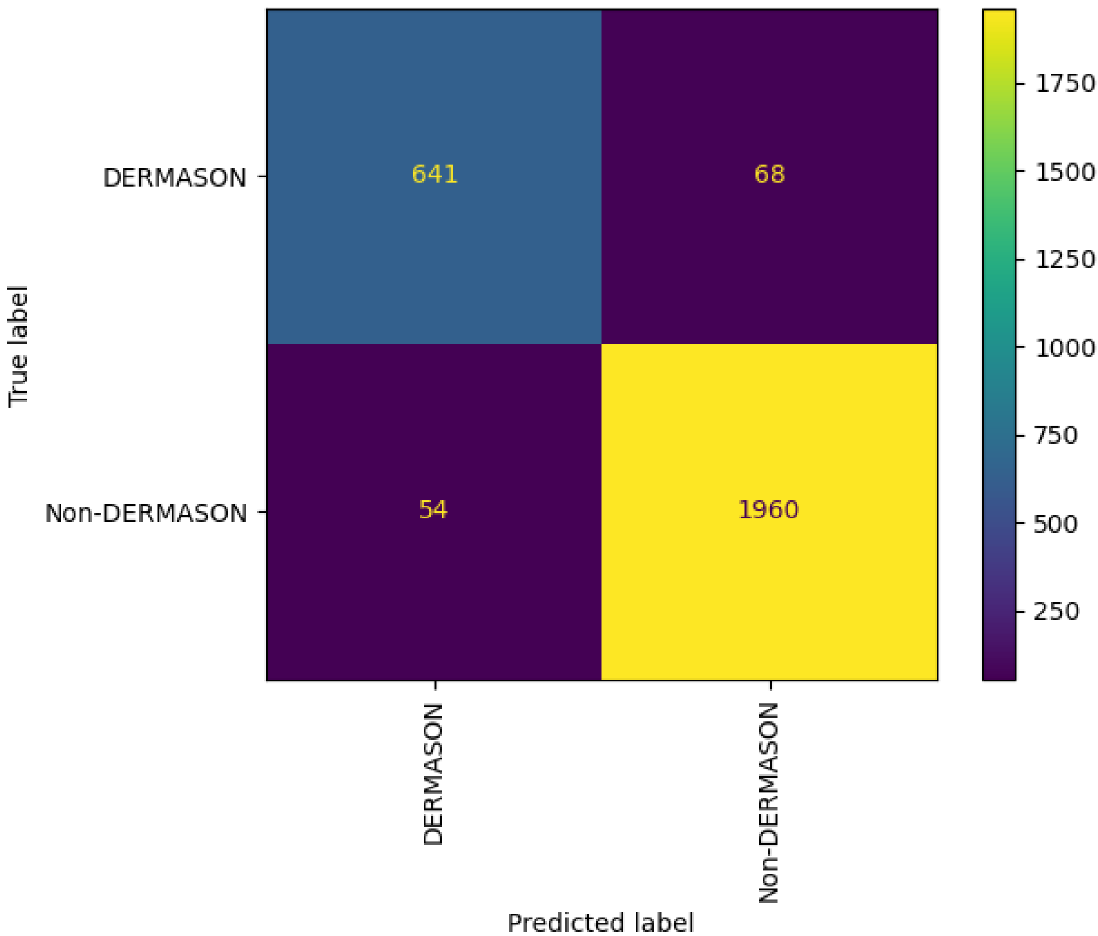
```python
    random_forest_cv_models.append(random_forest)

random_forest_cv['mean'] = pd.DataFrame(random_forest_cv).mean(axis=1)
pd.DataFrame(random_forest_cv).round(4)
```

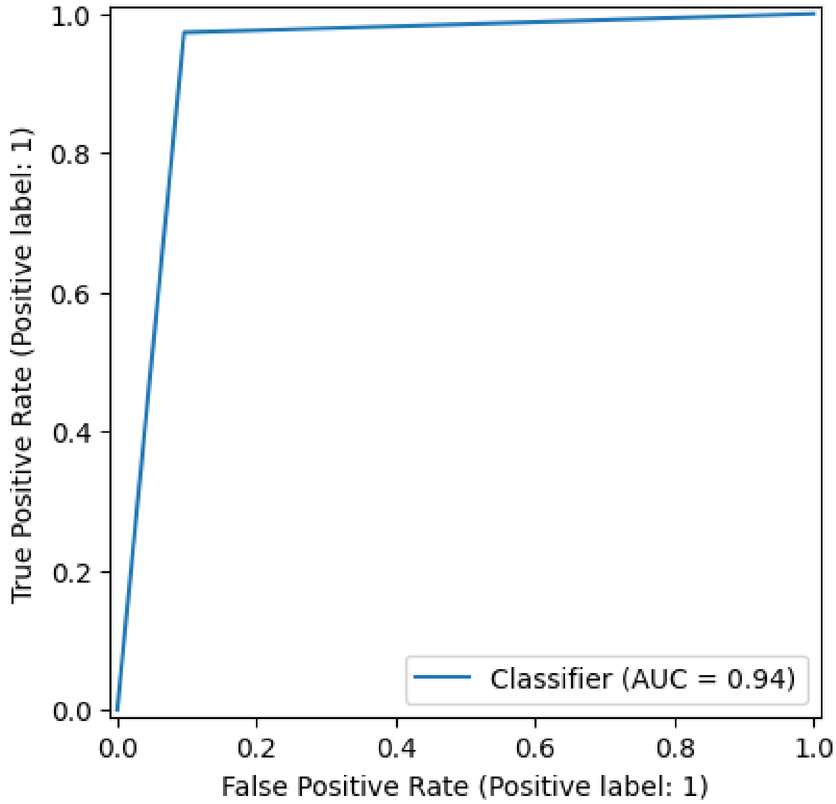
[ ]:

```
                              1         2         3         4         5  \
TP                     260.0000  263.0000  257.0000  258.0000  262.0000
TN                     776.0000  777.0000  783.0000  782.0000  784.0000
FP                      29.0000   28.0000   22.0000   23.0000   21.0000
FN                      24.0000   21.0000   27.0000   26.0000   22.0000
P                      284.0000  284.0000  284.0000  284.0000  284.0000
N                      805.0000  805.0000  805.0000  805.0000  805.0000
TPR                      0.9155    0.9261    0.9049    0.9085    0.9225
TNR                      0.9640    0.9652    0.9727    0.9714    0.9739
FPR                      0.0360    0.0348    0.0273    0.0286    0.0261
FNR                      0.0845    0.0739    0.0951    0.0915    0.0775
Recall                   0.9155    0.9261    0.9049    0.9085    0.9225
Precision                0.8997    0.9038    0.9211    0.9181    0.9258
F1 Score                 0.9075    0.9148    0.9130    0.9133    0.9242
Accuracy                 0.9513    0.9550    0.9550    0.9550    0.9605
Error Rate               0.0487    0.0450    0.0450    0.0450    0.0395
Balanced Accuracy        0.9397    0.9456    0.9388    0.9399    0.9482
True Skill Statistics    0.8795    0.8913    0.8776    0.8799    0.8964
Heidke Skill Score       0.8795    0.8913    0.8776    0.8799    0.8964
Brier Score              0.0335    0.0315    0.0293    0.0312    0.0275
Brier Skill Score        0.1735    0.1635    0.1520    0.1620    0.1424

                              6         7         8         9        10  \
TP                     263.0000  261.0000  254.0000  258.0000  268.0000
TN                     786.0000  778.0000  780.0000  785.0000  785.0000
FP                      19.0000   27.0000   25.0000   20.0000   20.0000
FN                      21.0000   23.0000   30.0000   26.0000   16.0000
P                      284.0000  284.0000  284.0000  284.0000  284.0000
N                      805.0000  805.0000  805.0000  805.0000  805.0000
TPR                      0.9261    0.9190    0.8944    0.9085    0.9437
TNR                      0.9764    0.9665    0.9689    0.9752    0.9752
FPR                      0.0236    0.0335    0.0311    0.0248    0.0248
FNR                      0.0739    0.0810    0.1056    0.0915    0.0563
Recall                   0.9261    0.9190    0.8944    0.9085    0.9437
Precision                0.9326    0.9062    0.9104    0.9281    0.9306
F1 Score                 0.9293    0.9126    0.9023    0.9181    0.9371
Accuracy                 0.9633    0.9541    0.9495    0.9578    0.9669
Error Rate               0.0367    0.0459    0.0505    0.0422    0.0331
Balanced Accuracy        0.9512    0.9427    0.9317    0.9418    0.9594
True Skill Statistics    0.9025    0.8855    0.8633    0.8836    0.9188
Heidke Skill Score       0.9025    0.8855    0.8633    0.8836    0.9188
Brier Score              0.0270    0.0316    0.0336    0.0329    0.0262
```

```
Brier Skill Score        0.1399    0.1637    0.1741    0.1709    0.1358

                          mean
TP                    260.4000
TN                    781.6000
FP                     23.4000
FN                     23.6000
P                     284.0000
N                     805.0000
TPR                     0.9169
TNR                     0.9709
FPR                     0.0291
FNR                     0.0831
Recall                  0.9169
Precision               0.9176
F1 Score                0.9172
Accuracy                0.9568
Error Rate              0.0432
Balanced Accuracy       0.9439
True Skill Statistics   0.8878
Heidke Skill Score      0.8878
Brier Score             0.0304
Brier Skill Score       0.1578
```

[ ]:
```
# Mode function returns the values that appear most frequently in the array
y_pred = stats.mode([model.predict(X_test) for model in␣
 ↪random_forest_cv_models]).mode

ConfusionMatrixDisplay.from_predictions(
    y_test,
    y_pred,
    display_labels=encoder.inverse_transform([0, 1]),
    xticks_rotation="vertical",
)

RocCurveDisplay.from_predictions(y_test, y_pred)

results_comparison['RandomForest'] = get_all_metrics(y_test, y_pred)
results_comparison.loc["Brier Score", "RandomForest"] = np.mean((y_pred -␣
 ↪y_test) ** 2)
results_comparison.loc["Brier Skill Score", "RandomForest"] =␣
 ↪results_comparison["RandomForest"][
    "Brier Score"
] / (np.mean((y_test - np.mean(y_pred)) ** 2))
```


DERMASON
Non-DERMASON
True label
641
68
54
1960
DERMASON
Non-DERMASON
Predicted label
1750
1500
1250
1000
750
500
250



4.2 Additional Algorithm - SVM

[ ]:
```
# params = {
#     "C": [10, 100, 1000],
#     "kernel": ["linear", "rbf", "sigmoid", "poly"],
#     'degree': [2, 4, 6]
# }
stratified_split = StratifiedShuffleSplit(n_splits=10, test_size=0.1,␣
↪random_state=42)
# gridsearch_svc = GridSearchCV(
#     SVC(random_state=42),
#     params,
#     cv=stratified_split,
#     n_jobs=-1,
#     verbose=1,
#     scoring=custom_calculate_f1_score
# )
# gridsearch_svc.fit(X_train, y_train)
# gridsearch_svc.best_params_
```

```
# If you uncomment the commented code above, you will get the following output
# Output - {'C': 100, 'degree': 2, 'kernel': 'rbf'}
```

```
Fitting 10 folds for each of 36 candidates, totalling 360 fits
```

[ ]:
```
{'C': 100, 'degree': 2, 'kernel': 'rbf'}
```

[ ]:
```python
svc_cv = {}
svc_cv_models = []

for i, (train_index, test_index) in enumerate(stratified_split.split(X_train, y_train)):
    X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc[test_index]
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[test_index]

    svc = SVC(C= 100, degree= 2, kernel= 'rbf', probability=True, random_state=42)
    svc.fit(X_train_fold, y_train_fold)

    y_pred = svc.predict(X_val_fold)
    y_pred_proba = svc.predict_proba(X_val_fold)[:, 1]
    svc_cv[i+1] = get_all_metrics(y_val_fold, y_pred)
    svc_cv[i+1]['Brier Score'] = np.mean((y_pred_proba - y_val_fold)**2)
    svc_cv[i+1]['Brier Skill Score'] = svc_cv[i+1]['Brier Score'] / (np.mean((y_val_fold - np.mean(y_pred_proba))**2))
    svc_cv_models.append(svc)

svc_cv['mean'] = pd.DataFrame(svc_cv).mean(axis=1)
pd.DataFrame(svc_cv).round(4)
```

[ ]:

	1	2	3	4	5 \
TP	263.0000	265.0000	259.0000	260.0000	263.0000
TN	780.0000	778.0000	785.0000	784.0000	787.0000
FP	25.0000	27.0000	20.0000	21.0000	18.0000
FN	21.0000	19.0000	25.0000	24.0000	21.0000
P	284.0000	284.0000	284.0000	284.0000	284.0000
N	805.0000	805.0000	805.0000	805.0000	805.0000
TPR	0.9261	0.9331	0.9120	0.9155	0.9261
TNR	0.9689	0.9665	0.9752	0.9739	0.9776
FPR	0.0311	0.0335	0.0248	0.0261	0.0224
FNR	0.0739	0.0669	0.0880	0.0845	0.0739
Recall	0.9261	0.9331	0.9120	0.9155	0.9261
Precision	0.9132	0.9075	0.9283	0.9253	0.9359
F1 Score	0.9196	0.9201	0.9201	0.9204	0.9310
Accuracy	0.9578	0.9578	0.9587	0.9587	0.9642

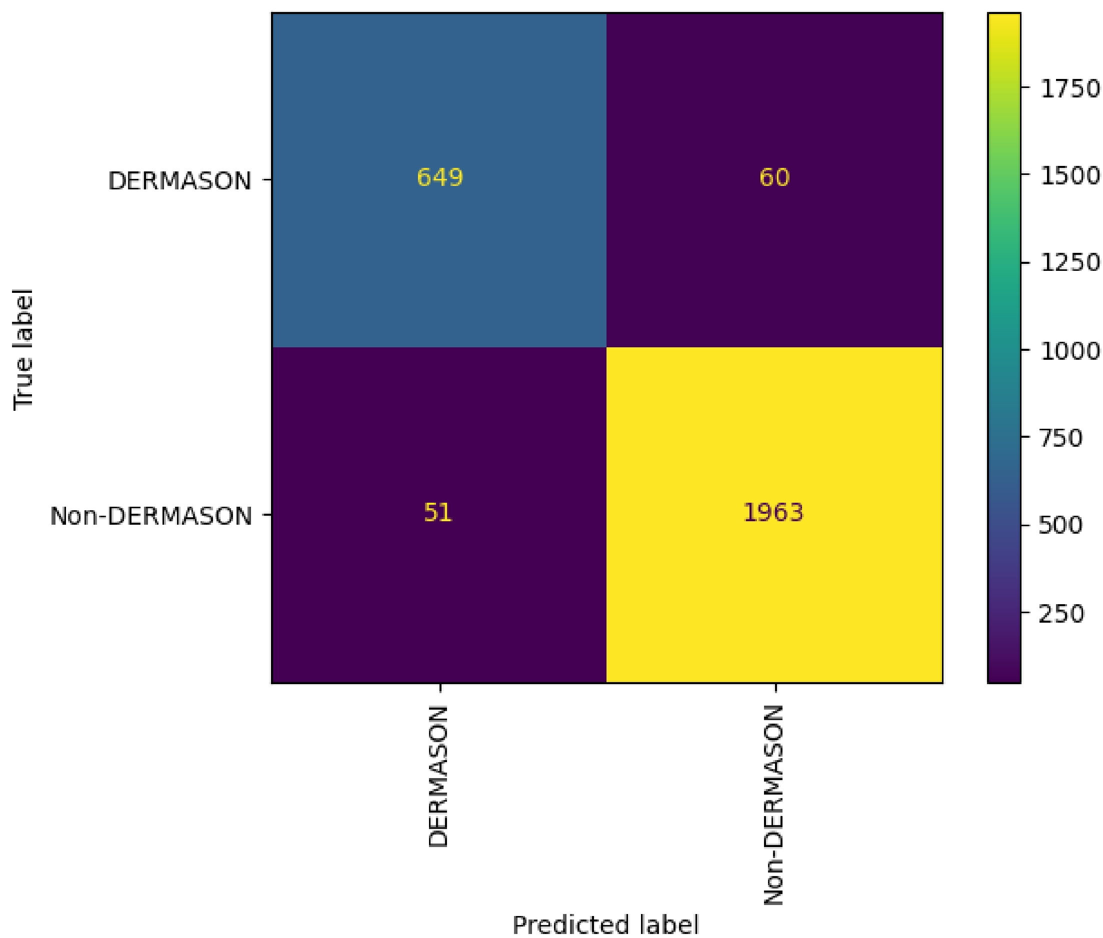
Error Rate	0.0422	0.0422	0.0413	0.0413	0.0358
Balanced Accuracy	0.9475	0.9498	0.9436	0.9447	0.9518
True Skill Statistics	0.8950	0.8996	0.8871	0.8894	0.9037
Heidke Skill Score	0.8950	0.8996	0.8871	0.8894	0.9037
Brier Score	0.0337	0.0310	0.0300	0.0297	0.0265
Brier Skill Score	0.1749	0.1608	0.1556	0.1543	0.1374

	6	7	8	9	10 \
TP	262.0000	261.0000	258.0000	262.0000	265.0000
TN	784.0000	781.0000	781.0000	783.0000	784.0000
FP	21.0000	24.0000	24.0000	22.0000	21.0000
FN	22.0000	23.0000	26.0000	22.0000	19.0000
P	284.0000	284.0000	284.0000	284.0000	284.0000
N	805.0000	805.0000	805.0000	805.0000	805.0000
TPR	0.9225	0.9190	0.9085	0.9225	0.9331
TNR	0.9739	0.9702	0.9702	0.9727	0.9739
FPR	0.0261	0.0298	0.0298	0.0273	0.0261
FNR	0.0775	0.0810	0.0915	0.0775	0.0669
Recall	0.9225	0.9190	0.9085	0.9225	0.9331
Precision	0.9258	0.9158	0.9149	0.9225	0.9266
F1 Score	0.9242	0.9174	0.9117	0.9225	0.9298
Accuracy	0.9605	0.9568	0.9541	0.9596	0.9633
Error Rate	0.0395	0.0432	0.0459	0.0404	0.0367
Balanced Accuracy	0.9482	0.9446	0.9393	0.9476	0.9535
True Skill Statistics	0.8964	0.8892	0.8786	0.8952	0.9070
Heidke Skill Score	0.8964	0.8892	0.8786	0.8952	0.9070
Brier Score	0.0281	0.0308	0.0342	0.0320	0.0264
Brier Skill Score	0.1460	0.1600	0.1774	0.1660	0.1367

	mean
TP	261.8000
TN	782.7000
FP	22.3000
FN	22.2000
P	284.0000
N	805.0000
TPR	0.9218
TNR	0.9723
FPR	0.0277
FNR	0.0782
Recall	0.9218
Precision	0.9216
F1 Score	0.9217
Accuracy	0.9591
Error Rate	0.0409
Balanced Accuracy	0.9471
True Skill Statistics	0.8941

```
Heidke Skill Score       0.8941
Brier Score              0.0303
Brier Skill Score        0.1569
```

[ ]:
```python
# Mode function returns the values that appear most frequently in the array
y_pred = stats.mode([model.predict(X_test) for model in svc_cv_models]).mode
matrix = confusion_matrix(y_test, y_pred)
ConfusionMatrixDisplay.from_predictions(
    y_test,
    y_pred,
    display_labels=encoder.inverse_transform([0, 1]),
    xticks_rotation="vertical",
)

results_comparison['SVM'] = get_all_metrics(y_test, y_pred)
results_comparison.loc["Brier Score", "SVM"] = np.mean((y_pred - y_test) ** 2)
results_comparison.loc["Brier Skill Score", "SVM"] = results_comparison["SVM"][
    "Brier Score"
] / (np.mean((y_test - np.mean(y_pred)) ** 2))
```



4.3 Additional Algorithm - KNN

[ ]:
```
# params = {
#     "n_neighbors": [3, 5, 7],
#     "weights": ["uniform", "distance"],
#     "algorithm": ["ball_tree", "kd_tree", "brute"],
#     "leaf_size": [10, 30, 50],
#     "p": [1, 2, 3],
# }
stratified_split = StratifiedShuffleSplit(n_splits=6, test_size=0.1,␣
↪random_state=42)
# gridsearch_knn = GridSearchCV(
#     KNeighborsClassifier(n_jobs=-1),
#     params,
#     cv=stratified_split,
#     n_jobs=-1,
#     verbose=1,
#     scoring=custom_calculate_f1_score
# )
# gridsearch_knn.fit(X_train, y_train)
# gridsearch_knn.best_params_

# If you uncomment the commented code above, you will get the following output
# Output - {'algorithm': 'ball_tree',
#  'leaf_size': 10,
#  'n_neighbors': 5,
#  'p': 2,
#  'weights': 'distance'}
```

```
Fitting 6 folds for each of 162 candidates, totalling 972 fits
```

[ ]:
```
{'algorithm': 'ball_tree',
 'leaf_size': 10,
 'n_neighbors': 5,
 'p': 2,
 'weights': 'distance'}
```

[ ]:
```
knn_cv = {}
knn_cv_models = []

for i, (train_index, test_index) in enumerate(stratified_split.split(X_train,␣
↪y_train)):
    X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.
↪iloc[test_index]
```

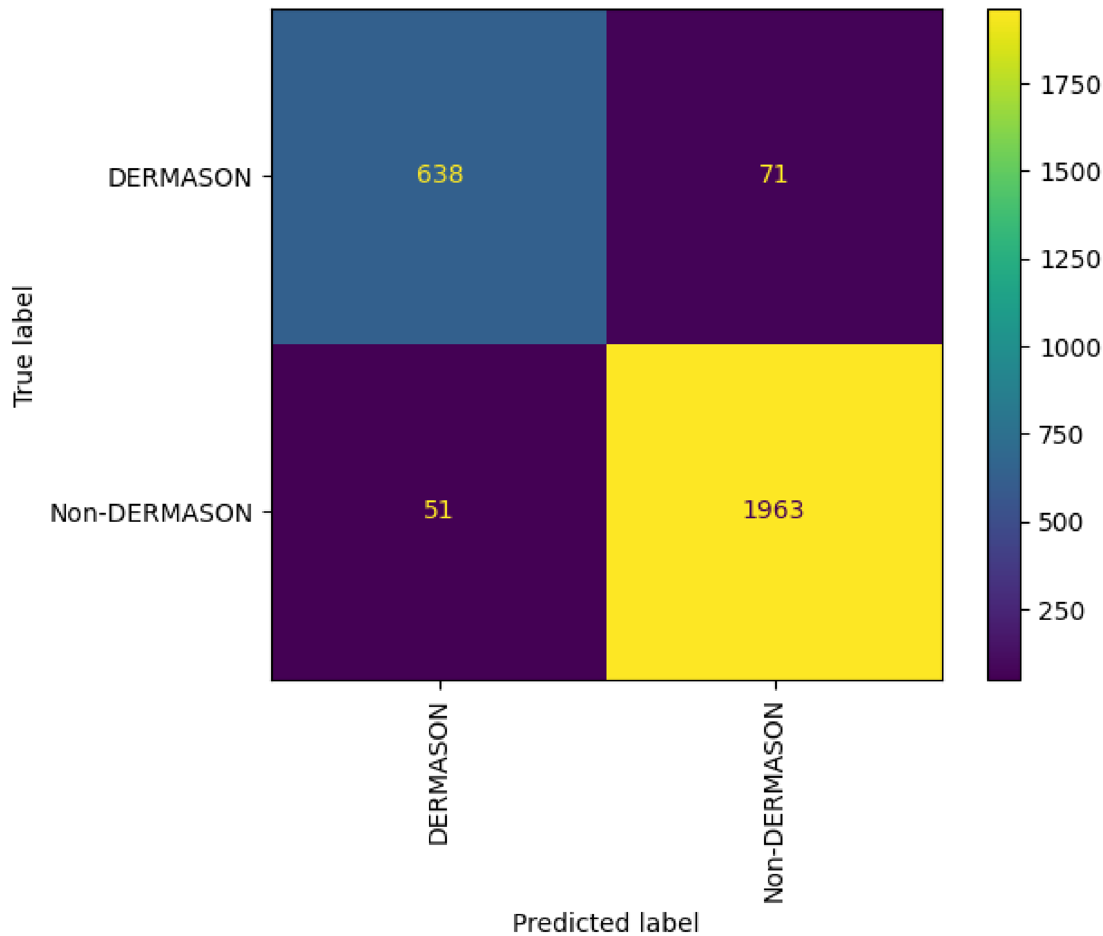
```
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.
↪iloc[test_index]

    knn = KNeighborsClassifier(
        algorithm="ball_tree",
        leaf_size=10,
        n_neighbors=5,
        p=2,
        weights="distance",
        n_jobs=-1,
    )
    knn.fit(X_train_fold, y_train_fold)

    y_pred = knn.predict(X_val_fold)
    y_pred_proba = knn.predict_proba(X_val_fold)[:, 1]
    knn_cv[i + 1] = get_all_metrics(y_val_fold, y_pred)
    knn_cv[i+1]['Brier Score'] = np.mean((y_pred_proba - y_val_fold)**2)
    knn_cv[i+1]['Brier Skill Score'] = knn_cv[i+1]['Brier Score'] / (np.
↪mean((y_val_fold - np.mean(y_pred_proba))**2))
    knn_cv_models.append(knn)

knn_cv["mean"] = pd.DataFrame(knn_cv).mean(axis=1)
pd.DataFrame(knn_cv).round(4)
```

```
[ ]:                            1         2         3         4         5  \
TP                       261.0000  264.0000  257.0000  255.0000  260.0000
TN                       780.0000  775.0000  783.0000  785.0000  779.0000
FP                        25.0000   30.0000   22.0000   20.0000   26.0000
FN                        23.0000   20.0000   27.0000   29.0000   24.0000
P                        284.0000  284.0000  284.0000  284.0000  284.0000
N                        805.0000  805.0000  805.0000  805.0000  805.0000
TPR                        0.9190    0.9296    0.9049    0.8979    0.9155
TNR                        0.9689    0.9627    0.9727    0.9752    0.9677
FPR                        0.0311    0.0373    0.0273    0.0248    0.0323
FNR                        0.0810    0.0704    0.0951    0.1021    0.0845
Recall                     0.9190    0.9296    0.9049    0.8979    0.9155
Precision                  0.9126    0.8980    0.9211    0.9273    0.9091
F1 Score                   0.9158    0.9135    0.9130    0.9123    0.9123
Accuracy                   0.9559    0.9541    0.9550    0.9550    0.9541
Error Rate                 0.0441    0.0459    0.0450    0.0450    0.0459
Balanced Accuracy          0.9440    0.9462    0.9388    0.9365    0.9416
True Skill Statistics      0.8880    0.8923    0.8776    0.8730    0.8832
Heidke Skill Score         0.8880    0.8923    0.8776    0.8730    0.8832
Brier Score                0.0379    0.0343    0.0335    0.0356    0.0326
Brier Skill Score          0.1965    0.1780    0.1736    0.1844    0.1690

                                6      mean
```

```
TP                        265.0000  260.3333
TN                        786.0000  781.3333
FP                         19.0000   23.6667
FN                         19.0000   23.6667
P                         284.0000  284.0000
N                         805.0000  805.0000
TPR                         0.9331    0.9167
TNR                         0.9764    0.9706
FPR                         0.0236    0.0294
FNR                         0.0669    0.0833
Recall                      0.9331    0.9167
Precision                   0.9331    0.9169
F1 Score                    0.9331    0.9167
Accuracy                    0.9651    0.9565
Error Rate                  0.0349    0.0435
Balanced Accuracy           0.9547    0.9436
True Skill Statistics       0.9095    0.8873
Heidke Skill Score          0.9095    0.8873
Brier Score                 0.0278    0.0336
Brier Skill Score           0.1443    0.1743
```

[ ]:

```
# Mode function returns the values that appear most frequently in the array
y_pred = stats.mode([model.predict(X_test) for model in knn_cv_models]).mode
matrix = confusion_matrix(y_test, y_pred)
ConfusionMatrixDisplay.from_predictions(
    y_test,
    y_pred,
    display_labels=encoder.inverse_transform([0, 1]),
    xticks_rotation="vertical",
)

results_comparison['KNN'] = get_all_metrics(y_test, y_pred)
results_comparison.loc["Brier Score", "KNN"] = np.mean((y_pred - y_test) ** 2)
results_comparison.loc["Brier Skill Score", "KNN"] = results_comparison["KNN"][
    "Brier Score"
] / (np.mean((y_test - np.mean(y_pred)) ** 2))
```



4.4 Additional Deep Learning Algorithm - Conv1D

[ ]:
```
device = (
    "cuda"
    if torch.cuda.is_available()
    else "mps" if torch.backends.mps.is_available() else "cpu"
)
print(f"Using {device} device for torch models")
```

```
Using mps device for torch models
```

[ ]:
```
class Conv1DNNModel(nn.Module):
    def __init__(self, *args, **kwargs) -> None:
        super().__init__(*args, **kwargs)
        self.conv1d_relu_stack = nn.Sequential(
            nn.Conv1d(in_channels=1, out_channels=128, kernel_size=3),
            nn.ReLU(),
```

```
            nn.BatchNorm1d(128),
            nn.Conv1d(in_channels=128, out_channels=64, kernel_size=3),
            nn.ReLU(),
            nn.BatchNorm1d(64),
            nn.Conv1d(in_channels=64, out_channels=32, kernel_size=3),
            nn.ReLU(),
            nn.BatchNorm1d(32),
            nn.Flatten(),
            nn.Linear(32 * 10, 64),
            nn.ReLU(),
            nn.BatchNorm1d(64),
            nn.Linear(64, 2),
            nn.Sigmoid(),
        )

    def forward(self, x):
        return self.conv1d_relu_stack(x)
```

[ ]:
```
learning_rate = 1e-2
batch_size = 64
epochs = 20

conv1d_model = Conv1DNNModel()
loss_fn = nn.BCEWithLogitsLoss()
summary(conv1d_model, input_size=(batch_size, 1, 16))
```

[ ]:
```
================================================================================
==========
Layer (type:depth-idx)                   Output Shape              Param #
================================================================================
==========
Conv1DNNModel                            [64, 2]                   --
 Sequential: 1-1                         [64, 2]                   --
     Conv1d: 2-1                         [64, 128, 14]             512
     ReLU: 2-2                           [64, 128, 14]             --
     BatchNorm1d: 2-3                    [64, 128, 14]             256
     Conv1d: 2-4                         [64, 64, 12]              24,640
     ReLU: 2-5                           [64, 64, 12]              --
     BatchNorm1d: 2-6                    [64, 64, 12]              128
     Conv1d: 2-7                         [64, 32, 10]              6,176
     ReLU: 2-8                           [64, 32, 10]              --
     BatchNorm1d: 2-9                    [64, 32, 10]              64
     Flatten: 2-10                       [64, 320]                 --
     Linear: 2-11                        [64, 64]                  20,544
     ReLU: 2-12                          [64, 64]                  --
     BatchNorm1d: 2-13                   [64, 64]                  128
     Linear: 2-14                        [64, 2]                   130
```

```
     Sigmoid: 2-15                       [64, 2]                   --
================================================================================
==========
Total params: 52,578
Trainable params: 52,578
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 24.69
================================================================================
==========
Input size (MB): 0.00
Forward/backward pass size (MB): 3.02
Params size (MB): 0.21
Estimated Total Size (MB): 3.23
================================================================================
==========
```

[ ]:
```
y_train_one_hot = pd.get_dummies(y_train.values, dtype=np.float32)
y_test_one_hot = pd.get_dummies(y_test.values, dtype=np.float32)

test_dataset = TensorDataset(
    torch.tensor(X_test.values.reshape((-1, 1, 16)), dtype=torch.float32),
    torch.tensor(y_test.values, dtype=torch.float32),
)
```

[ ]:
```
stratified_cv_split = StratifiedShuffleSplit(n_splits=10, test_size=0.1,␣
 ↪random_state=42)

conv1d_cv = TorchKFoldCrossValidation(
    model_class=Conv1DNNModel,
    loss_fn=loss_fn,
    learning_rate=learning_rate,
    batch_size=batch_size,
    epochs=epochs,
    cv=stratified_cv_split,
    device=device
)
conv1d_cv.fit(X_train.values.reshape(-1,1,16), y_train_one_hot.values)

## Load models, it is faster than to train the model again
## If you want to train the model then comment out the below line and uncomment␣
 ↪the above code line
# conv1d_cv.load_models()
```

```
Cross validation step 1

Epoch 1
-------------------------------
Model saved with F1 Score: 0.9195804195804196
```

```
Epoch 1 completed

Epoch 2
-------------------------------
Epoch 2 completed

Epoch 3
-------------------------------
Epoch 3 completed

Epoch 4
-------------------------------
Model saved with F1 Score: 0.9206349206349206
Epoch 4 completed

Epoch 5
-------------------------------
Model saved with F1 Score: 0.9228070175438596
Epoch 5 completed

Epoch 6
-------------------------------
Model saved with F1 Score: 0.9233511586452763
Epoch 6 completed

Epoch 7
-------------------------------
Epoch 7 completed

Epoch 8
-------------------------------
Model saved with F1 Score: 0.927797833935018
Epoch 8 completed

Epoch 9
-------------------------------
Epoch 9 completed

Epoch 10
-------------------------------
Epoch 10 completed

Epoch 11
-------------------------------
Epoch 11 completed

Epoch 12
-------------------------------
```

```
Epoch 12 completed

Epoch 13
-------------------------------
Epoch 13 completed

Epoch 14
-------------------------------
Epoch 14 completed

Epoch 15
-------------------------------
Epoch 15 completed

Epoch 16
-------------------------------
Epoch 16 completed

Epoch 17
-------------------------------
Epoch 17 completed

Epoch 18
-------------------------------
Epoch 18 completed

Epoch 19
-------------------------------
Epoch 19 completed

Epoch 20
-------------------------------
Epoch 20 completed

Cross validation step 2

Epoch 1
-------------------------------
Model saved with F1 Score: 0.9194139194139195
Epoch 1 completed

Epoch 2
-------------------------------
Model saved with F1 Score: 0.9298245614035088
Epoch 2 completed

Epoch 3
-------------------------------
```

```
Model saved with F1 Score: 0.9330985915492958
Epoch 3 completed

Epoch 4
-------------------------------
Epoch 4 completed

Epoch 5
-------------------------------
Epoch 5 completed

Epoch 6
-------------------------------
Epoch 6 completed

Epoch 7
-------------------------------
Model saved with F1 Score: 0.9342806394316163
Epoch 7 completed

Epoch 8
-------------------------------
Epoch 8 completed

Epoch 9
-------------------------------
Model saved with F1 Score: 0.9377162629757785
Epoch 9 completed

Epoch 10
-------------------------------
Epoch 10 completed

Epoch 11
-------------------------------
Epoch 11 completed

Epoch 12
-------------------------------
Model saved with F1 Score: 0.9428076256499133
Epoch 12 completed

Epoch 13
-------------------------------
Epoch 13 completed

Epoch 14
-------------------------------
```

```
Epoch 14 completed

Epoch 15
-------------------------------
Epoch 15 completed

Epoch 16
-------------------------------
Epoch 16 completed

Epoch 17
-------------------------------
Epoch 17 completed

Epoch 18
-------------------------------
Epoch 18 completed

Epoch 19
-------------------------------
Epoch 19 completed

Epoch 20
-------------------------------
Epoch 20 completed

Cross validation step 3

Epoch 1
-------------------------------
Model saved with F1 Score: 0.9264214046822743
Epoch 1 completed

Epoch 2
-------------------------------
Epoch 2 completed

Epoch 3
-------------------------------
Epoch 3 completed

Epoch 4
-------------------------------
Epoch 4 completed

Epoch 5
-------------------------------
Epoch 5 completed
```

```
Epoch 6
--------------------------------
Epoch 6 completed

Epoch 7
--------------------------------
Model saved with F1 Score: 0.9342560553633218
Epoch 7 completed

Epoch 8
--------------------------------
Epoch 8 completed

Epoch 9
--------------------------------
Model saved with F1 Score: 0.9375
Epoch 9 completed

Epoch 10
--------------------------------
Epoch 10 completed

Epoch 11
--------------------------------
Model saved with F1 Score: 0.9389179755671903
Epoch 11 completed

Epoch 12
--------------------------------
Epoch 12 completed

Epoch 13
--------------------------------
Model saved with F1 Score: 0.9391304347826086
Epoch 13 completed

Epoch 14
--------------------------------
Epoch 14 completed

Epoch 15
--------------------------------
Epoch 15 completed

Epoch 16
--------------------------------
Epoch 16 completed
```

```
Epoch 17
-------------------------------
Epoch 17 completed

Epoch 18
-------------------------------
Epoch 18 completed

Epoch 19
-------------------------------
Epoch 19 completed

Epoch 20
-------------------------------
Epoch 20 completed

Cross validation step 4

Epoch 1
-------------------------------
Model saved with F1 Score: 0.910420475319927
Epoch 1 completed

Epoch 2
-------------------------------
Epoch 2 completed

Epoch 3
-------------------------------
Epoch 3 completed

Epoch 4
-------------------------------
Model saved with F1 Score: 0.9137614678899083
Epoch 4 completed

Epoch 5
-------------------------------
Model saved with F1 Score: 0.9222614840989399
Epoch 5 completed

Epoch 6
-------------------------------
Epoch 6 completed

Epoch 7
-------------------------------
```

```
Epoch 7 completed

Epoch 8
-------------------------------
Epoch 8 completed

Epoch 9
-------------------------------
Model saved with F1 Score: 0.9276895943562611
Epoch 9 completed

Epoch 10
-------------------------------
Epoch 10 completed

Epoch 11
-------------------------------
Epoch 11 completed

Epoch 12
-------------------------------
Epoch 12 completed

Epoch 13
-------------------------------
Epoch 13 completed

Epoch 14
-------------------------------
Epoch 14 completed

Epoch 15
-------------------------------
Epoch 15 completed

Epoch 16
-------------------------------
Epoch 16 completed

Epoch 17
-------------------------------
Epoch 17 completed

Epoch 18
-------------------------------
Epoch 18 completed

Epoch 19
```

```
-------------------------------
Epoch 19 completed

Epoch 20
-------------------------------
Epoch 20 completed

Cross validation step 5

Epoch 1
-------------------------------
Model saved with F1 Score: 0.9222614840989399
Epoch 1 completed

Epoch 2
-------------------------------
Epoch 2 completed

Epoch 3
-------------------------------
Epoch 3 completed

Epoch 4
-------------------------------
Model saved with F1 Score: 0.9270833333333333
Epoch 4 completed

Epoch 5
-------------------------------
Epoch 5 completed

Epoch 6
-------------------------------
Model saved with F1 Score: 0.9271758436944938
Epoch 6 completed

Epoch 7
-------------------------------
Epoch 7 completed

Epoch 8
-------------------------------
Model saved with F1 Score: 0.9300699300699301
Epoch 8 completed

Epoch 9
-------------------------------
Epoch 9 completed
```

```
Epoch 10
-------------------------------
Epoch 10 completed

Epoch 11
-------------------------------
Model saved with F1 Score: 0.9354275741710296
Epoch 11 completed

Epoch 12
-------------------------------
Epoch 12 completed

Epoch 13
-------------------------------
Epoch 13 completed

Epoch 14
-------------------------------
Epoch 14 completed

Epoch 15
-------------------------------
Epoch 15 completed

Epoch 16
-------------------------------
Epoch 16 completed

Epoch 17
-------------------------------
Epoch 17 completed

Epoch 18
-------------------------------
Epoch 18 completed

Epoch 19
-------------------------------
Epoch 19 completed

Epoch 20
-------------------------------
Epoch 20 completed

Cross validation step 6
```

```
Epoch 1
-------------------------------
Model saved with F1 Score: 0.8884688090737239
Epoch 1 completed

Epoch 2
-------------------------------
Model saved with F1 Score: 0.9365351629502573
Epoch 2 completed

Epoch 3
-------------------------------
Epoch 3 completed

Epoch 4
-------------------------------
Epoch 4 completed

Epoch 5
-------------------------------
Model saved with F1 Score: 0.9397590361445785
Epoch 5 completed

Epoch 6
-------------------------------
Model saved with F1 Score: 0.9446366782006922
Epoch 6 completed

Epoch 7
-------------------------------
Epoch 7 completed

Epoch 8
-------------------------------
Epoch 8 completed

Epoch 9
-------------------------------
Epoch 9 completed

Epoch 10
-------------------------------
Epoch 10 completed

Epoch 11
-------------------------------
Epoch 11 completed
```

```
Epoch 12
-------------------------------
Epoch 12 completed

Epoch 13
-------------------------------
Epoch 13 completed

Epoch 14
-------------------------------
Epoch 14 completed

Epoch 15
-------------------------------
Epoch 15 completed

Epoch 16
-------------------------------
Epoch 16 completed

Epoch 17
-------------------------------
Epoch 17 completed

Epoch 18
-------------------------------
Epoch 18 completed

Epoch 19
-------------------------------
Epoch 19 completed

Epoch 20
-------------------------------
Epoch 20 completed

Cross validation step 7

Epoch 1
-------------------------------
Model saved with F1 Score: 0.8956834532374102
Epoch 1 completed

Epoch 2
-------------------------------
Model saved with F1 Score: 0.9020979020979022
Epoch 2 completed
```

```
Epoch 3
-------------------------------
Model saved with F1 Score: 0.9033391915641477
Epoch 3 completed

Epoch 4
-------------------------------
Model saved with F1 Score: 0.9078260869565217
Epoch 4 completed

Epoch 5
-------------------------------
Model saved with F1 Score: 0.9084628670120899
Epoch 5 completed

Epoch 6
-------------------------------
Model saved with F1 Score: 0.9150779896013864
Epoch 6 completed

Epoch 7
-------------------------------
Epoch 7 completed

Epoch 8
-------------------------------
Epoch 8 completed

Epoch 9
-------------------------------
Epoch 9 completed

Epoch 10
-------------------------------
Epoch 10 completed

Epoch 11
-------------------------------
Epoch 11 completed

Epoch 12
-------------------------------
Epoch 12 completed

Epoch 13
-------------------------------
Epoch 13 completed
```

```
Epoch 14
-------------------------------
Epoch 14 completed

Epoch 15
-------------------------------
Epoch 15 completed

Epoch 16
-------------------------------
Epoch 16 completed

Epoch 17
-------------------------------
Epoch 17 completed

Epoch 18
-------------------------------
Epoch 18 completed

Epoch 19
-------------------------------
Epoch 19 completed

Epoch 20
-------------------------------
Epoch 20 completed

Cross validation step 8

Epoch 1
-------------------------------
Model saved with F1 Score: 0.9078260869565217
Epoch 1 completed

Epoch 2
-------------------------------
Model saved with F1 Score: 0.9129662522202486
Epoch 2 completed

Epoch 3
-------------------------------
Epoch 3 completed

Epoch 4
-------------------------------
Model saved with F1 Score: 0.9138840070298769
Epoch 4 completed
```

```
Epoch 5
-------------------------------
Epoch 5 completed

Epoch 6
-------------------------------
Epoch 6 completed

Epoch 7
-------------------------------
Model saved with F1 Score: 0.9150090415913201
Epoch 7 completed

Epoch 8
-------------------------------
Epoch 8 completed

Epoch 9
-------------------------------
Epoch 9 completed

Epoch 10
-------------------------------
Model saved with F1 Score: 0.91651865008881
Epoch 10 completed

Epoch 11
-------------------------------
Epoch 11 completed

Epoch 12
-------------------------------
Epoch 12 completed

Epoch 13
-------------------------------
Epoch 13 completed

Epoch 14
-------------------------------
Epoch 14 completed

Epoch 15
-------------------------------
Epoch 15 completed

Epoch 16
```

```
-------------------------------
Epoch 16 completed

Epoch 17
-------------------------------
Epoch 17 completed

Epoch 18
-------------------------------
Epoch 18 completed

Epoch 19
-------------------------------
Epoch 19 completed

Epoch 20
-------------------------------
Epoch 20 completed

Cross validation step 9

Epoch 1
-------------------------------
Model saved with F1 Score: 0.9163763066202091
Epoch 1 completed

Epoch 2
-------------------------------
Model saved with F1 Score: 0.9235993208828521
Epoch 2 completed

Epoch 3
-------------------------------
Epoch 3 completed

Epoch 4
-------------------------------
Epoch 4 completed

Epoch 5
-------------------------------
Epoch 5 completed

Epoch 6
-------------------------------
Epoch 6 completed

Epoch 7
```

```
--------------------------------
Epoch 7 completed

Epoch 8
--------------------------------
Epoch 8 completed

Epoch 9
--------------------------------
Epoch 9 completed

Epoch 10
--------------------------------
Epoch 10 completed

Epoch 11
--------------------------------
Epoch 11 completed

Epoch 12
--------------------------------
Epoch 12 completed

Epoch 13
--------------------------------
Epoch 13 completed

Epoch 14
--------------------------------
Model saved with F1 Score: 0.9265734265734266
Epoch 14 completed

Epoch 15
--------------------------------
Epoch 15 completed

Epoch 16
--------------------------------
Epoch 16 completed

Epoch 17
--------------------------------
Epoch 17 completed

Epoch 18
--------------------------------
Epoch 18 completed
```

```
Epoch 19
-------------------------------
Epoch 19 completed

Epoch 20
-------------------------------
Epoch 20 completed

Cross validation step 10

Epoch 1
-------------------------------
Model saved with F1 Score: 0.9109947643979057
Epoch 1 completed

Epoch 2
-------------------------------
Model saved with F1 Score: 0.9217687074829931
Epoch 2 completed

Epoch 3
-------------------------------
Epoch 3 completed

Epoch 4
-------------------------------
Model saved with F1 Score: 0.9235993208828521
Epoch 4 completed

Epoch 5
-------------------------------
Epoch 5 completed

Epoch 6
-------------------------------
Epoch 6 completed

Epoch 7
-------------------------------
Model saved with F1 Score: 0.9251700680272109
Epoch 7 completed

Epoch 8
-------------------------------
Epoch 8 completed

Epoch 9
-------------------------------
```

```
Model saved with F1 Score: 0.9312714776632303
Epoch 9 completed

Epoch 10
-------------------------------
Epoch 10 completed

Epoch 11
-------------------------------
Model saved with F1 Score: 0.9333333333333333
Epoch 11 completed

Epoch 12
-------------------------------
Epoch 12 completed

Epoch 13
-------------------------------
Epoch 13 completed

Epoch 14
-------------------------------
Epoch 14 completed

Epoch 15
-------------------------------
Epoch 15 completed

Epoch 16
-------------------------------
Epoch 16 completed

Epoch 17
-------------------------------
Epoch 17 completed

Epoch 18
-------------------------------
Epoch 18 completed

Epoch 19
-------------------------------
Epoch 19 completed

Epoch 20
-------------------------------
Epoch 20 completed
```

[ ]:

	1	2	3	4
TP	258	259	269	264
TN	788	793	784	779
FP	17	12	21	26
FN	26	25	15	20
P	284	284	284	284
N	805	805	805	805
TPR	0.908451	0.911972	0.947183	0.929577
TNR	0.978882	0.985093	0.973913	0.967702
FPR	0.021118	0.014907	0.026087	0.032298
FNR	0.091549	0.088028	0.052817	0.070423
Recall	0.908451	0.911972	0.947183	0.929577
Precision	0.938182	0.95572	0.927586	0.910345
F1 Score	0.923077	0.933333	0.937282	0.919861
Accuracy	0.960514	0.966024	0.966942	0.957759
Error Rate	0.039486	0.033976	0.033058	0.042241
Balanced Accuracy	0.943666	0.948532	0.960548	0.94864
True Skill Statistics	0.887333	0.897065	0.921096	0.897279
Heidke Skill Score	0.887333	0.897065	0.921096	0.897279
Brier Score	0.036512285	0.026696311	0.034292948	0.03741867
Brier Skill Score	0.18939786	0.13846803	0.17721239	0.19384459

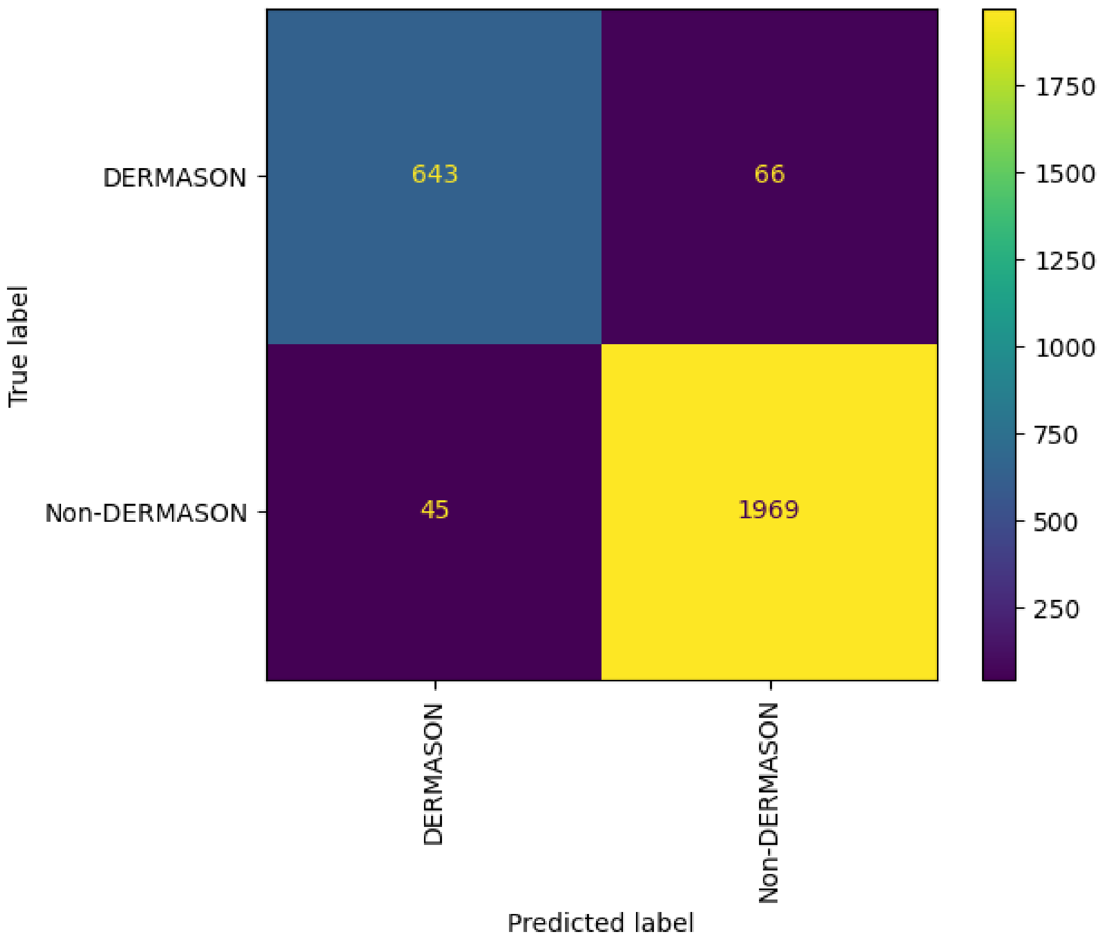
	5	6	7	8
TP	264	272	261	257
TN	786	783	774	783
FP	19	22	31	22
FN	20	12	23	27
P	284	284	284	284
N	805	805	805	805
TPR	0.929577	0.957746	0.919014	0.90493
TNR	0.976398	0.972671	0.961491	0.972671
FPR	0.023602	0.027329	0.038509	0.027329
FNR	0.070423	0.042254	0.080986	0.09507
Recall	0.929577	0.957746	0.919014	0.90493
Precision	0.932862	0.92517	0.893836	0.921147
F1 Score	0.931217	0.941176	0.90625	0.912966
Accuracy	0.964187	0.968779	0.950413	0.955005
Error Rate	0.035813	0.031221	0.049587	0.044995
Balanced Accuracy	0.952987	0.965209	0.940252	0.9388
True Skill Statistics	0.905975	0.930417	0.880505	0.8776
Heidke Skill Score	0.905975	0.930417	0.880505	0.8776
Brier Score	0.029044146	0.034414813	0.044193845	0.03766667
Brier Skill Score	0.15056041	0.17801704	0.22893846	0.19533204

	9	10	mean
TP	258	266	262.8
TN	788	779	783.7

```
FP                           17           26       21.3
FN                           26           18       21.2
P                           284          284      284.0
N                           805          805      805.0
TPR                    0.908451      0.93662   0.925352
TNR                    0.978882     0.967702    0.97354
FPR                    0.021118     0.032298    0.02646
FNR                    0.091549      0.06338   0.074648
Recall                 0.908451      0.93662   0.925352
Precision              0.938182     0.910959   0.925399
F1 Score               0.923077     0.923611   0.925185
Accuracy               0.960514     0.959596   0.960973
Error Rate             0.039486     0.040404   0.039027
Balanced Accuracy      0.943666     0.952161   0.949446
True Skill Statistics  0.887333     0.904322   0.898892
Heidke Skill Score     0.887333     0.904322   0.898892
Brier Score          0.03508866  0.038193576   0.035352
Brier Skill Score    0.18200251   0.19773223   0.183151
```

[ ]:

```
y_pred = conv1d_cv.predict(test_dataset.tensors[0]).argmax(axis=1)
matrix = confusion_matrix(y_test, y_pred)
ConfusionMatrixDisplay.from_predictions(
    y_test,
    y_pred,
    display_labels=encoder.inverse_transform([0, 1]),
    xticks_rotation="vertical",
)

results_comparison["Conv1D-NN"] = get_all_metrics(y_test, y_pred)
results_comparison.loc["Brier Score", "Conv1D-NN"] = np.mean((y_pred - y_test)␣
  ↪** 2)
results_comparison.loc["Brier Skill Score", "Conv1D-NN"] =␣
  ↪results_comparison["Conv1D-NN"][
    "Brier Score"
] / (np.mean((y_test - np.mean(y_pred)) ** 2))
```



5 Results

[ ]:
```python
# Result comparison on the test dataset for all the models
results_comparison.round(4)
```

[ ]:

	RandomForest	SVM	KNN	Conv1D-NN
TP	641.0000	649.0000	638.0000	643.0000
TN	1960.0000	1963.0000	1963.0000	1969.0000
FP	54.0000	51.0000	51.0000	45.0000
FN	68.0000	60.0000	71.0000	66.0000
P	709.0000	709.0000	709.0000	709.0000
N	2014.0000	2014.0000	2014.0000	2014.0000
TPR	0.9041	0.9154	0.8999	0.9069
TNR	0.9732	0.9747	0.9747	0.9777
FPR	0.0268	0.0253	0.0253	0.0223
FNR	0.0959	0.0846	0.1001	0.0931

Recall	0.9041	0.9154	0.8999	0.9069
Precision	0.9223	0.9271	0.9260	0.9346
F1 Score	0.9131	0.9212	0.9127	0.9205
Accuracy	0.9552	0.9592	0.9552	0.9592
Error Rate	0.0448	0.0408	0.0448	0.0408
Balanced Accuracy	0.9386	0.9450	0.9373	0.9423
True Skill Statistics	0.8773	0.8901	0.8745	0.8846
Heidke Skill Score	0.8773	0.8901	0.8745	0.8846
Brier Score	0.0448	0.0408	0.0448	0.0408
Brier Skill Score	0.2326	0.2117	0.2326	0.2116