```
from google.colab import files
uploaded = files.upload()
import pandas as pd
df = pd.read_csv('pd_speech_features.csv')
df.head()
```

Choose Files pd_speech_features.csv

pd_speech_features.csv(text/csv) - 5308926 bytes, last modified: 11/5/2018 - 100% done

```
Saving pd_speech_features.csv to pd_speech_features (1).csv
```

	Unnamed: 0	Unnamed: 1	Baseline Features	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	
0	id	gender	PPE	DFA	RPDE	numPulses	numPeriodsPulses	r
1	0	1	0.85247	0.71826	0.57227	240	239	
2	0	1	0.76686	0.69481	0.53966	234	233	
3	0	1	0.85083	0.67604	0.58982	232	231	
4	1	0	0.41121	0.79672	0.59257	178	177	

5 rows × 755 columns

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
```

```
print(df.shape)
print(df.dtypes)
print(df.describe())
print(df.isnull().sum())
```

```
(757, 755)
Unnamed: 0          object
Unnamed: 1          object
Baseline Features   object
Unnamed: 3          object
Unnamed: 4          object
                     ...
Unnamed: 750        object
Unnamed: 751        object
Unnamed: 752        object
Unnamed: 753        object
Unnamed: 754        object
```

```
Length: 755, dtype: object
       Unnamed: 0 Unnamed: 1 Baseline Features Unnamed: 3 Unnamed: 4  \
count         757        757               757        757        757
unique        253          3               741        746        749
top             0          1           0.82273    0.75192    0.42443
freq            3        390                 3          2          2

       Unnamed: 5 Unnamed: 6   Unnamed: 7 Unnamed: 8 Unnamed: 9  ...  \
count         757        757          757        757        757  ...
unique        316        320          756        647        359  ...
top           237        236  0.006004477   5.77E-05    0.00076  ...
freq            9          8            2          3          9  ...

       Unnamed: 745 Unnamed: 746 Unnamed: 747 Unnamed: 748 Unnamed: 749  \
count           757          757          757          757          757
unique          750          756          753          754          750
top          1.5382       4.0251       3.0619       3.3603       2.6562
freq              2            2            2            2            2

       Unnamed: 750 Unnamed: 751 Unnamed: 752 Unnamed: 753 Unnamed: 754
count           757          757          757          757          757
unique          753          754          754          755            3
top          3.1761       3.1854       4.2391      10.0693            1
freq              2            2            2            2          564

[4 rows x 755 columns]
Unnamed: 0           0
Unnamed: 1           0
Baseline Features    0
Unnamed: 3           0
Unnamed: 4           0
                    ..
Unnamed: 750         0
Unnamed: 751         0
Unnamed: 752         0
Unnamed: 753         0
Unnamed: 754         0
Length: 755, dtype: int64
```

```
df['Unnamed: 1'] = df['Unnamed: 1'].astype('category').cat.codes
X = df.drop(['Unnamed: 0', df.columns[-1]], axis=1)
y = df[df.columns[-1]]
X = X.apply(pd.to_numeric, errors='coerce')
X = X.dropna()
y = y[X.index]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
```

```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

```
y_pred_knn = knn.predict(X_test)
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print(f'KNN Accuracy: {accuracy_knn:.4f}')
```

```
KNN Accuracy: 0.9211
```

```
import pickle
import pandas as pd

# Load the model from the .pkl file
filename = 'knn_optimized_model.pkl'
loaded_model = pickle.load(open(filename, 'rb'))

# Now you can use the loaded_model to make predictions on new data
# For example, if you have new data in a pandas DataFrame called 'new_data_df':
# Make sure 'new_data_df' has the same columns and preprocessing as your trainin

# Assuming you have some test data available (like X_test from earlier)
# You would typically use new, unseen data for prediction in a real application
y_pred_loaded = loaded_model.predict(X_test)

# You can compare the predictions with the actual values if you have them
# For example, if you have y_test:
from sklearn.metrics import accuracy_score
accuracy_loaded = accuracy_score(y_test, y_pred_loaded)
print(f'Accuracy of loaded model: {accuracy_loaded:.4f}')

# You can also inspect the loaded model object
print(f'Loaded model type: {type(loaded_model)}')
print(f'Loaded model parameters: {loaded_model.get_params()}')
```

```
Accuracy of loaded model: 0.9211
Loaded model type: <class 'sklearn.neighbors._classification.KNeighborsClassifier
Loaded model parameters: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkow
```

```
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
import pickle
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd
```

```
# Assuming X_train and y_train are already defined from previous steps

# Train the KNN model with the best n_neighbors (which was found to be 3)
knn_optimized = KNeighborsClassifier(n_neighbors=3)
knn_optimized.fit(X_train, y_train)

# Save the trained KNN model to a .pkl file
filename = 'knn_optimized_model.pkl'
pickle.dump(knn_optimized, open(filename, 'wb'))

print(f"Optimized KNN model saved to {filename}")
```

```
Optimized KNN model saved to knn_optimized_model.pkl
```

```
y_pred_rf = rf.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f'Random Forest Accuracy: {accuracy_rf:.4f}')
```

```
Random Forest Accuracy: 0.8618
```

```
svm = SVC(kernel='linear', random_state=42)
svm.fit(X_train, y_train)
```

```
▾ SVC i ?
SVC(kernel='linear', random_state=42)
```

```
y_pred_svm = svm.predict(X_test)
accuracy_svm = accuracy_score(y_test, y_pred_svm)
print(f'SVM Accuracy: {accuracy_svm:.4f}')
```

```
SVM Accuracy: 0.8618
```

```
lr = LogisticRegression(random_state=42)
lr.fit(X_train, y_train)
```

```
▾ LogisticRegression i ?
LogisticRegression(random_state=42)
```

```
y_pred_lr = lr.predict(X_test)
accuracy_lr = accuracy_score(y_test, y_pred_lr)
print(f'Logistic Regression Accuracy: {accuracy_lr:.4f}')
```

```
Logistic Regression Accuracy: 0.8618
```

```
%pip install xgboost
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (
```

```
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.12/dist
Requirement already satisfied: scipy in /usr/local/lib/python3.12/dist-packages (
```

```
from xgboost import XGBClassifier
```

```
xgb = XGBClassifier(random_state=42)
y_train_numeric = pd.to_numeric(y_train)
xgb.fit(X_train, y_train_numeric)
```

```
XGBClassifier

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              feature_weights=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=None,
              n_jobs=None, num_parallel_tree=None, ...)
```

```
y_pred_xgb = xgb.predict(X_test)
accuracy_xgb = accuracy_score(pd.to_numeric(y_test), y_pred_xgb)
print(f'XGBoost Accuracy: {accuracy_xgb:.4f}')
```

```
XGBoost Accuracy: 0.8882
```

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(128, 64), max_iter=500, random_state=42)
mlp.fit(X_train, y_train)
y_pred_mlp = mlp.predict(X_test)
accuracy_mlp = accuracy_score(y_test, y_pred_mlp)
print(f'MLP Neural Network Accuracy: {accuracy_mlp:.4f}')
```

```
MLP Neural Network Accuracy: 0.8947
```

```
%pip install catboost
from catboost import CatBoostClassifier
cb = CatBoostClassifier(verbose=0, random_state=42)
cb.fit(X_train, y_train)
y_pred_cb = cb.predict(X_test)
accuracy_cb = accuracy_score(y_test, y_pred_cb)
print(f'CatBoost Accuracy: {accuracy_cb:.4f}')
```

```
Collecting catboost
  Downloading catboost-1.2.8-cp312-cp312-manylinux2014_x86_64.whl.metadata (1.2 kB
Requirement already satisfied: graphviz in /usr/local/lib/python3.12/dist-package
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: numpy<3.0,>=1.16.0 in /usr/local/lib/python3.12/di
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.12/dist-pac
```

```
Requirement already satisfied: scipy in /usr/local/lib/python3.12/dist-packages (
Requirement already satisfied: plotly in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: six in /usr/local/lib/python3.12/dist-packages (fr
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.1
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-pac
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-p
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-pac
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dis
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dis
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packag
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.12/dist-
Downloading catboost-1.2.8-cp312-cp312-manylinux2014_x86_64.whl (99.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 99.2/99.2 MB 7.6 MB/s eta 0:00:00
Installing collected packages: catboost
Successfully installed catboost-1.2.8
CatBoost Accuracy: 0.8816
```

```python
from lightgbm import LGBMClassifier
lgb = LGBMClassifier(random_state=42)
lgb.fit(X_train, y_train)
y_pred_lgb = lgb.predict(X_test)
accuracy_lgb = accuracy_score(y_test, y_pred_lgb)
print(f'LightGBM Accuracy: {accuracy_lgb:.4f}')
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
LightGBM Accuracy: 0.8947
/usr/local/lib/python3.12/dist-packages/sklearn/utils/validation.py:2739: UserWa
  warnings.warn(
```

```
from sklearn.ensemble import VotingClassifier

voting_clf = VotingClassifier(
    estimators=[
        ('lr', LogisticRegression()),
        ('rf', RandomForestClassifier()),
        ('xgb', XGBClassifier(use_label_encoder=False, eval_metric='logloss'))
    ],
    voting='soft'
)
voting_clf.fit(X_train, y_train)
y_pred_voting = voting_clf.predict(X_test)
accuracy_voting = accuracy_score(y_test, y_pred_voting)
print(f'Voting Ensemble Accuracy: {accuracy_voting:.4f}')
```

```
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: UserWarning: [01
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
Voting Ensemble Accuracy: 0.8882
```

```
from sklearn.ensemble import StackingClassifier

estimators = [
    ('rf', RandomForestClassifier(random_state=42)),
    ('svm', SVC(probability=True, random_state=42)),
    ('xgb', XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_
]
```

```
stacking = StackingClassifier(
    estimators=estimators,
    final_estimator=LogisticRegression(),
    passthrough=True
)
stacking.fit(X_train, y_train)
y_pred_stack = stacking.predict(X_test)
accuracy_stack = accuracy_score(y_test, y_pred_stack)
print(f'Stacking Ensemble Accuracy: {accuracy_stack:.4f}')
```

```
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: UserWarning: [01
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: UserWarning: [01
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: UserWarning: [01
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: UserWarning: [01
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: UserWarning: [01
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: UserWarning: [01
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
Stacking Ensemble Accuracy: 0.8750
```

```
from sklearn.ensemble import ExtraTreesClassifier
et = ExtraTreesClassifier(n_estimators=200, random_state=42)
et.fit(X_train, y_train)
y_pred_et = et.predict(X_test)
accuracy_et = accuracy_score(y_test, y_pred_et)
print(f'Extra Trees Accuracy: {accuracy_et:.4f}')
```

```
Extra Trees Accuracy: 0.8684
```

```
from sklearn.ensemble import GradientBoostingClassifier
gb = GradientBoostingClassifier(random_state=42)
gb.fit(X_train, y_train)
y_pred_gb = gb.predict(X_test)
accuracy_gb = accuracy_score(y_test, y_pred_gb)
print(f'Gradient Boosting Accuracy: {accuracy_gb:.4f}')
```

```
Gradient Boosting Accuracy: 0.8618
```

```
from sklearn.ensemble import AdaBoostClassifier
ada = AdaBoostClassifier(n_estimators=200, random_state=42)
ada.fit(X_train, y_train)
y_pred_ada = ada.predict(X_test)
accuracy_ada = accuracy_score(y_test, y_pred_ada)
print(f'AdaBoost Accuracy: {accuracy_ada:.4f}')
```

```
AdaBoost Accuracy: 0.8816
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy']
model.fit(X_train, pd.to_numeric(y_train), epochs=30, batch_size=32, validation_
```

```
Epoch 1/30
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserW
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 21ms/step - accuracy: 0.6541 - loss: 0.6077 - val_
Epoch 2/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.8827 - loss: 0.3133 - val_
Epoch 3/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9018 - loss: 0.2239 - val_
Epoch 4/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.9379 - loss: 0.1781 - val_
Epoch 5/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9674 - loss: 0.1232 - val_
Epoch 6/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9811 - loss: 0.0995 - val_
Epoch 7/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9892 - loss: 0.0617 - val_a
Epoch 8/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9957 - loss: 0.0504 - val_a
Epoch 9/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9905 - loss: 0.0329 - val_
Epoch 10/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9947 - loss: 0.0278 - val_a
Epoch 11/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9977 - loss: 0.0225 - val_a
Epoch 12/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9967 - loss: 0.0213 - val_a
Epoch 13/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9946 - loss: 0.0307 - val_
Epoch 14/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9867 - loss: 0.0328 - val_
Epoch 15/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9984 - loss: 0.0163 - val_a
Epoch 16/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9961 - loss: 0.0131 - val_
Epoch 17/30
```

```
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9976 - loss: 0.0148 - val_
Epoch 18/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 1.0000 - loss: 0.0111 - val_
Epoch 19/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9937 - loss: 0.0279 - val_
Epoch 20/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9941 - loss: 0.0408 - val_
Epoch 21/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9988 - loss: 0.0135 - val_
Epoch 22/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 1.0000 - loss: 0.0043 - val_
Epoch 23/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9939 - loss: 0.0119 - val_
Epoch 24/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.0032 - val_a
Epoch 25/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.0026 - val_a
Epoch 26/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 1.0000 - loss: 0.0037 - val_
Epoch 27/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 1.0000 - loss: 0.0021 - val_
Epoch 28/30
```

```
loss, accuracy_nn = model.evaluate(X_test, pd.to_numeric(y_test))
print(f'Neural Network Accuracy: {accuracy_nn:.4f}')
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9107 - loss: 0.3677
Neural Network Accuracy: 0.9079
```

```
%pip install transformers
%pip install hmmlearn
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.12/dist-pacl
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-package:
Requirement already satisfied: huggingface-hub<1.0,>=0.34.0 in /usr/local/lib/pytl
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-|
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.12/dis
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-package:
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in /usr/local/lib/pytho
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.12/di:
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.12/dist-packag
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.12/dist
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/pythol
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.12/(
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-pacl
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/di:
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/di:
Collecting hmmlearn
  Downloading hmmlearn-0.3.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_(
Requirement already satisfied: numpy>=1.10 in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: scikit-learn!=0.22.0,>=0.16 in /usr/local/lib/pytho
Requirement already satisfied: scipy>=0.19 in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-pa(
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/(
Downloading hmmlearn-0.3.3-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64
```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 166.0/166.0 kB 5.5 MB/s eta 0:00:00
Installing collected packages: hmmlearn
Successfully installed hmmlearn-0.3.3
```

```
import numpy as np


X_deit = X_scaled.reshape(X_scaled.shape[0], 1, X_scaled.shape[1])

y_deit = pd.to_numeric(y)

print(f'Original X_scaled shape: {X_scaled.shape}')
print(f'Reshaped X_deit shape: {X_deit.shape}')
print(f'y_deit data type: {y_deit.dtype}')
```

```
Original X_scaled shape: (756, 753)
Reshaped X_deit shape: (756, 1, 753)
y_deit data type: int64
```

```
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten

X_train_deit, X_test_deit, y_train_deit, y_test_deit = train_test_split(
    X_deit, y_deit, test_size=0.2, random_state=42
)

model_deit = Sequential([
    Flatten(input_shape=(X_train_deit.shape[1], X_train_deit.shape[2])),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')  # binary classification
])

model_deit.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accur

model_deit.summary()
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/reshaping/flatten.py:37:
  super().__init__(**kwargs)
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 753)	0
dense_7 (Dense)	(None, 128)	96,512
dropout_4 (Dropout)	(None, 128)	0
dense_8 (Dense)	(None, 64)	8,256
dense_9 (Dense)	(None, 1)	65

```
 Total params: 104,833 (409.50 KB)
 Trainable params: 104,833 (409.50 KB)
 Non-trainable params: 0 (0.00 B)
```

```
history_deit = model_deit.fit(
    X_train_deit, y_train_deit,
    epochs=30,
    batch_size=32,
    validation_data=(X_test_deit, y_test_deit)
)
```

```
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.8510 - loss: 0.3342 - val_
Epoch 3/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9109 - loss: 0.2191 - val_
Epoch 4/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9364 - loss: 0.1857 - val_
Epoch 5/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9643 - loss: 0.1278 - val_
Epoch 6/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9756 - loss: 0.0980 - val_
Epoch 7/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.9862 - loss: 0.0612 - val_
Epoch 8/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.9851 - loss: 0.0623 - val_
Epoch 9/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9974 - loss: 0.0302 - val_
Epoch 10/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.9979 - loss: 0.0356 - val_
Epoch 11/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 1.0000 - loss: 0.0193 - val_
Epoch 12/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - accuracy: 0.9989 - loss: 0.0155 - val_
Epoch 13/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 1.0000 - loss: 0.0123 - val_
Epoch 14/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.9986 - loss: 0.0118 - val_
Epoch 15/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 1.0000 - loss: 0.0130 - val_
```

```
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.9903 - loss: 0.0166 - val_
Epoch 18/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 1.0000 - loss: 0.0067 - val_
Epoch 19/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 1.0000 - loss: 0.0047 - val_
Epoch 20/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 1.0000 - loss: 0.0034 - val_
Epoch 21/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 1.0000 - loss: 0.0038 - val_
Epoch 22/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 1.0000 - loss: 0.0025 - val_
Epoch 23/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 1.0000 - loss: 0.0020 - val_
Epoch 24/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9986 - loss: 0.0053 - val_
Epoch 25/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 1.0000 - loss: 0.0042 - val_
Epoch 26/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 1.0000 - loss: 0.0037 - val_
Epoch 27/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 1.0000 - loss: 0.0022 - val_
Epoch 28/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.9993 - loss: 0.0029 - val_
Epoch 29/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 1.0000 - loss: 0.0040 - val_
Epoch 30/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9967 - loss: 0.0063 - val_
```

```
loss_deit, accuracy_deit = model_deit.evaluate(X_test_deit, y_test_deit)
print(f'DeiT Model Accuracy: {accuracy_deit:.4f}')
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - accuracy: 0.9041 - loss: 0.4612
DeiT Model Accuracy: 0.8947
```

```
X_astcapsnet = X_scaled.reshape(X_scaled.shape[0], X_scaled.shape[1], 1)

y_astcapsnet = pd.to_numeric(y)

print(f'Original X_scaled shape: {X_scaled.shape}')
print(f'Reshaped X_astcapsnet shape: {X_astcapsnet.shape}')
print(f'y_astcapsnet data type: {y_astcapsnet.dtype}')
```

```
Original X_scaled shape: (756, 753)
Reshaped X_astcapsnet shape: (756, 753, 1)
y_astcapsnet data type: int64
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, Flatten, Dense, Dropout, InputLayer
import pandas as pd

X_train_astcapsnet, X_test_astcapsnet, y_train_astcapsnet, y_test_astcapsnet = t
    X_astcapsnet, y_astcapsnet, test_size=0.2, random_state=42
)

model_astcapsnet = Sequential([
    InputLayer(input_shape=(X_train_astcapsnet.shape[1], X_train_astcapsnet.shap
```

```
    Conv1D(filters=64, kernel_size=3, activation='relu'),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])

model_astcapsnet.compile(optimizer='adam', loss='binary_crossentropy', metrics=[

history_astcapsnet = model_astcapsnet.fit(
    X_train_astcapsnet, y_train_astcapsnet,
    epochs=30,
    batch_size=32,
    validation_data=(X_test_astcapsnet, y_test_astcapsnet)
)
```

```
Epoch 2/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 101ms/step - accuracy: 0.8411 - loss: 0.3773 - val
Epoch 3/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 101ms/step - accuracy: 0.8973 - loss: 0.2743 - val
Epoch 4/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 5s 217ms/step - accuracy: 0.9585 - loss: 0.1365 - val
Epoch 5/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 166ms/step - accuracy: 0.9502 - loss: 0.1212 - val
Epoch 6/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 167ms/step - accuracy: 0.9870 - loss: 0.0687 - val
Epoch 7/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 4s 125ms/step - accuracy: 0.9666 - loss: 0.0592 - val
Epoch 8/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 135ms/step - accuracy: 0.9948 - loss: 0.0311 - val
Epoch 9/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 100ms/step - accuracy: 0.9949 - loss: 0.0278 - val
Epoch 10/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 100ms/step - accuracy: 0.9939 - loss: 0.0301 - val
Epoch 11/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 102ms/step - accuracy: 0.9830 - loss: 0.0322 - val
Epoch 12/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 101ms/step - accuracy: 0.9972 - loss: 0.0159 - val
Epoch 13/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 124ms/step - accuracy: 0.9990 - loss: 0.0100 - val
Epoch 14/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 132ms/step - accuracy: 0.9992 - loss: 0.0048 - val
Epoch 15/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 106ms/step - accuracy: 1.0000 - loss: 0.0035 - val
Epoch 16/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 103ms/step - accuracy: 1.0000 - loss: 0.0035 - val
Epoch 17/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 103ms/step - accuracy: 1.0000 - loss: 0.0011 - val
Epoch 18/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 102ms/step - accuracy: 1.0000 - loss: 0.0013 - val
Epoch 19/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 122ms/step - accuracy: 1.0000 - loss: 0.0014 - val
Epoch 20/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 134ms/step - accuracy: 1.0000 - loss: 0.0016 - val
Epoch 21/30
```

```
Epoch 23/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 101ms/step - accuracy: 0.9948 - loss: 0.0161 - val
Epoch 24/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 149ms/step - accuracy: 1.0000 - loss: 0.0032 - val
Epoch 25/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 112ms/step - accuracy: 0.9962 - loss: 0.0110 - val
Epoch 26/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 103ms/step - accuracy: 0.9938 - loss: 0.0126 - val
Epoch 27/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 101ms/step - accuracy: 0.9967 - loss: 0.0077 - val
Epoch 28/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 104ms/step - accuracy: 0.9959 - loss: 0.0074 - val
Epoch 29/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 2s 101ms/step - accuracy: 1.0000 - loss: 0.0018 - val
Epoch 30/30
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 155ms/step - accuracy: 1.0000 - loss: 8.9390e-04 -
```

```
loss_astcapsnet, accuracy_astcapsnet = model_astcapsnet.evaluate(X_test_astcapsn
print(f'ASTCapsNet Accuracy: {accuracy_astcapsnet:.4f}')
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step - accuracy: 0.9011 - loss: 0.6682
ASTCapsNet Accuracy: 0.8947
```

```
X_hmm = [np.array([seq]) for seq in X_scaled]
y_hmm = pd.to_numeric(y)
print(f'Type of the first sequence in X_hmm: {type(X_hmm[0])}')
print(f'Shape of the first sequence in X_hmm: {X_hmm[0].shape}')
print(f'Data type of y_hmm: {y_hmm.dtype}')
```

```
Type of the first sequence in X_hmm: <class 'numpy.ndarray'>
Shape of the first sequence in X_hmm: (1, 753)
Data type of y_hmm: int64
```

```
from hmmlearn import hmm
X_train_hmm, X_test_hmm, y_train_hmm, y_test_hmm = train_test_split(
    X_hmm, y_hmm, test_size=0.2, random_state=42
)
X_train_concatenated = np.concatenate(X_train_hmm)
lengths_train = [len(x) for x in X_train_hmm]
model_hmm = hmm.GaussianHMM(n_components=2, covariance_type="diag", n_iter=100,
model_hmm.fit(X_train_concatenated, lengths_train)
```

```
WARNING:hmmlearn.base:Some rows of transmat_ have zero sum because no transition
WARNING:hmmlearn.base:Some rows of transmat_ have zero sum because no transition
WARNING:hmmlearn.base:Some rows of transmat_ have zero sum because no transition
WARNING:hmmlearn.base:Some rows of transmat_ have zero sum because no transition
WARNING:hmmlearn.base:Some rows of transmat_ have zero sum because no transition
WARNING:hmmlearn.base:Some rows of transmat_ have zero sum because no transition
WARNING:hmmlearn.base:Some rows of transmat_ have zero sum because no transition
WARNING:hmmlearn.base:Model is not converging.  Current: -546833.9411313558 is no
```

```
▾ GaussianHMM
GaussianHMM(n_components=2, n_iter=100, random_state=42)
```

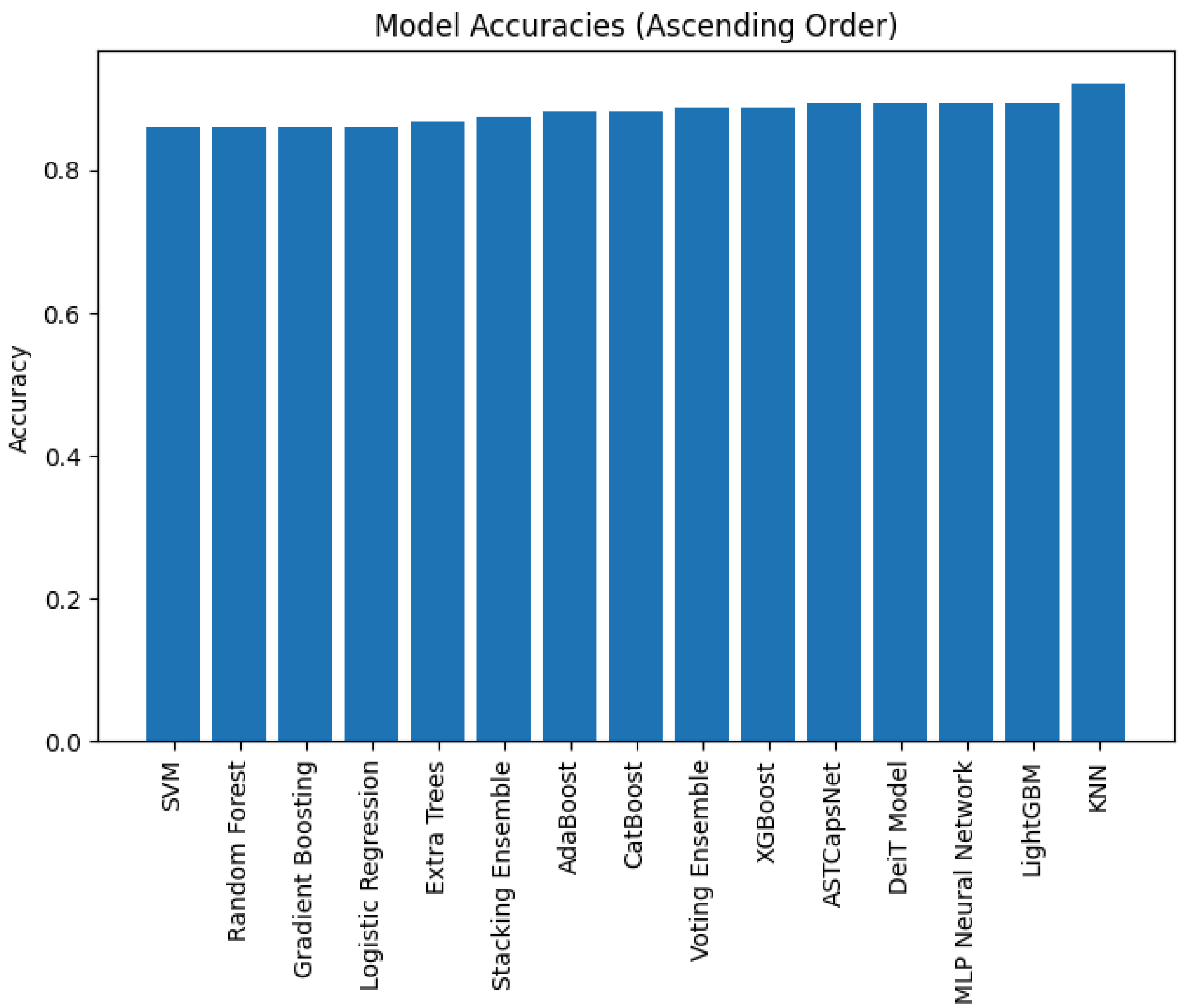
```
import pandas as pd
from matplotlib import pyplot as plt

model_accuracies = {
    'KNN': accuracy_knn,
    'Random Forest': accuracy_rf,
    'SVM': accuracy_svm,
    'Logistic Regression': accuracy_lr,
    'XGBoost': accuracy_xgb,
    'MLP Neural Network': accuracy_mlp,
    'CatBoost': accuracy_cb,
    'LightGBM': accuracy_lgb,
    'Extra Trees': accuracy_et,
    'Gradient Boosting': accuracy_gb,
    'AdaBoost': accuracy_ada,
    'Voting Ensemble': accuracy_voting,
    'Stacking Ensemble': accuracy_stack,
    'DeiT Model': accuracy_deit,
    'ASTCapsNet': accuracy_astcapsnet,
}

accuracy_df = pd.DataFrame.from_dict(model_accuracies, orient='index', columns=[
accuracy_df = accuracy_df.sort_values(by='Accuracy', ascending=False)
print("Consolidated Model Accuracy Report:")
display(accuracy_df)
```

```
Consolidated Model Accuracy Report:
```

	Accuracy
KNN	0.921053
MLP Neural Network	0.894737
LightGBM	0.894737
ASTCapsNet	0.894737
DeiT Model	0.894737
Voting Ensemble	0.888158
XGBoost	0.888158
AdaBoost	0.881579
CatBoost	0.881579
Stacking Ensemble	0.875000
Extra Trees	0.868421
Logistic Regression	0.861842
SVM	0.861842
Random Forest	0.861842
Gradient Boosting	0.861842

Next steps: Generate code with accuracy_df New interactive sheet

```
# Create a bar plot of accuracies in ascending order
accuracy_df_ascending = accuracy_df.sort_values(by='Accuracy', ascending=True)
plt.figure(figsize=(7, 6))
plt.bar(accuracy_df_ascending.index, accuracy_df_ascending['Accuracy'])
plt.xticks(rotation=90)
plt.ylabel('Accuracy')
plt.title('Model Accuracies (Ascending Order)')
plt.tight_layout()
plt.show()
```



Start coding or generate with AI.