ExerciseDetection

October 3, 2024

1 Import Libraries

[1]:
```
import numpy as np
import pandas as pd
import zipfile
import os
import warnings
warnings.filterwarnings("ignore")
pd.options.mode.copy_on_write = True
```

[35]:
```
import plotly.express as px
import plotly.graph_objects as go
import plotly.io as pio
from plotly.subplots import make_subplots
import plotly.figure_factory as ff
pio.templates.default = 'plotly_dark'
```

[3]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder , MinMaxScaler
from sklearn.compose import ColumnTransformer
from sklearn.metrics import accuracy_score ,auc ,roc_auc_score ,␣
  ↪confusion_matrix
```

[4]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.linear_model import Perceptron
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
```

[5]:
```
from xgboost import XGBClassifier
from catboost import CatBoostClassifier
from lightgbm import LGBMClassifier
```

[37]:
```
import optuna
import logging
from optuna.samplers import TPESampler
from optuna.visualization import plot_optimization_history,␣
↪plot_param_importances ,plot_contour
optuna.logging.set_verbosity(optuna.logging.WARNING)
```

1.1 Read Data

[7]:
```
def unzip_file_to_same_location(zip_path):

    extract_to = os.path.dirname(zip_path)

    with zipfile.ZipFile(zip_path, 'r') as zip_ref:

        zip_ref.extractall(extract_to)
        print('Extraction Complete')
```

[8]:
```
zip_file_path = 'archive (2).zip'
unzip_file_to_same_location(zip_file_path)
```

```
Extraction Complete
```

[9]:
```
df = pd.read_csv('exercise_angles.csv')
```

[10]:
```
num_observations, num_features = df.shape
print(f"Number of observations: {num_observations}")
print(f"Number of features: {num_features}")
```

```
Number of observations: 31033
Number of features: 12
```

[11]:
```
side_counts = df['Side'].value_counts()
print("Value counts for 'Side' column:")
print(side_counts)
```

```
Value counts for 'Side' column:
Side
left    31033
Name: count, dtype: int64
```

[12]:
```
df.drop('Side',axis=1,inplace=True)
```

[13]:
```
df.describe()
```

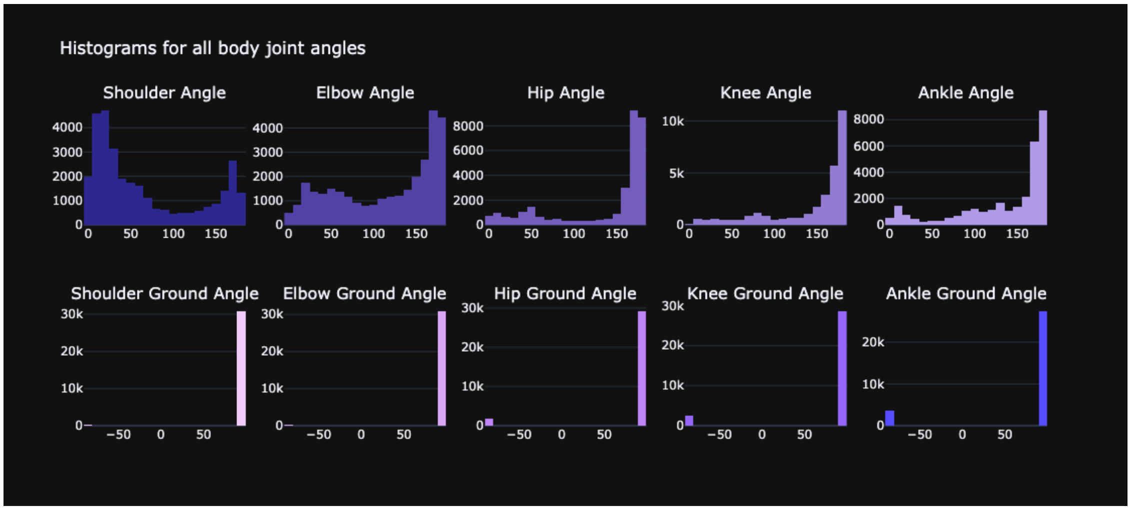
[13]:
```
       Shoulder_Angle   Elbow_Angle     Hip_Angle    Knee_Angle   Ankle_Angle  \
count    31033.000000  31033.000000  31033.000000  31033.000000  31033.000000
mean        66.522206    114.303010    137.466151    143.273623    135.211957
std         60.226756     57.906279     57.048278     48.041715     53.304068
```

```
min           0.002748      0.000974      0.006850      0.116036      0.031297
25%          17.852184     58.900491    111.556724    123.646144    106.740814
50%          40.585632    132.999090    168.374922    168.227063    162.926184
75%         121.209005    168.769517    175.656498    177.225089    175.735039
max         179.991577    179.998861    179.999848    179.999277    179.999942

       Shoulder_Ground_Angle  Elbow_Ground_Angle  Hip_Ground_Angle  \
count           31033.000000        31033.000000      31033.000000
mean               88.816743           88.926949         79.408694
std                14.546233           13.856550         42.359381
min               -90.000000          -90.000000        -90.000000
25%                90.000000           90.000000         90.000000
50%                90.000000           90.000000         90.000000
75%                90.000000           90.000000         90.000000
max                90.000000           90.000000         90.000000

       Knee_Ground_Angle  Ankle_Ground_Angle
count       31033.000000        31033.000000
mean           75.795121           68.985596
std            48.530150           57.802208
min           -90.000000          -90.000000
25%            90.000000           90.000000
50%            90.000000           90.000000
75%            90.000000           90.000000
max            90.000000           90.000000
```

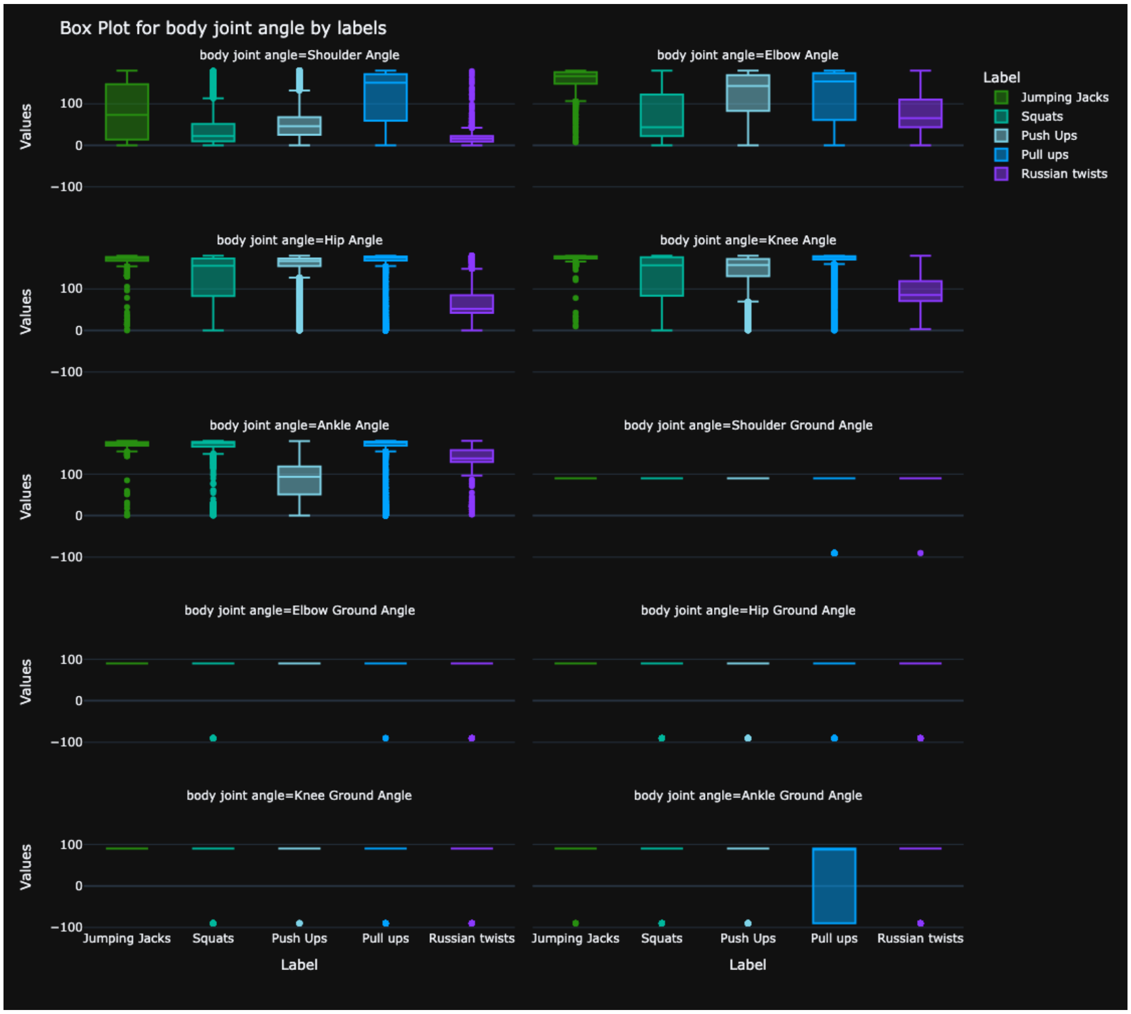
[15]:

```python
numeric_columns = df.select_dtypes(include=['float64']).columns



colors = ['#2d288f', '#5342a5', '#735ebb', '#927bd1', '#b199e8',
          '#f1cbf9', '#dca7f7', '#bf85f8', '#9666fa', '#554cff']

fig = make_subplots(rows=2, cols=5, subplot_titles=[col.replace('_', ' ') for␣
 ↪col in numeric_columns])

for i, col in enumerate(numeric_columns):
    row = (i // 5) + 1
    col_pos = (i % 5) + 1
    fig.add_trace(
        go.Histogram(x=df[col], marker_color=colors[i], name=col.replace('_', '␣
 ↪') , nbinsx=20, showlegend=False),
        row=row, col=col_pos
    )

fig.update_layout(height=500, width=1100, title_text="Histograms for all body␣
 ↪joint angles")
```

```python
fig.show()
```




```python
[16]: df_melted = df.melt(id_vars='Label', var_name='body joint angle',␣
       ↪value_name='Values')
      colors = ['#288f11', '#00b79d', '#7fd3e8', '#00a3ff', '#8c3aff']


      df_melted['body joint angle'] = df_melted['body joint angle'].str.replace('_',␣
       ↪' ')
      fig = px.box(df_melted, x='Label', y='Values', color='Label',
                   facet_col='body joint angle', facet_col_wrap=2,
                   color_discrete_sequence=colors)

      fig.update_layout(height=1000, width=1200, title="Box Plot for body joint angle␣
       ↪by labels")


      fig.show()
```



[17]:
```
label_encoder = LabelEncoder()
df['Label'] = label_encoder.fit_transform(df['Label'])
```

[20]:
```
def preprocess_data(df, label_column):


    X = df.drop(columns=[label_column])
    y = df[label_column]

    first_four_columns = X.columns[:5]
    last_four_columns = X.columns[5:10]

    transformers = [
        ('standard_scaler', StandardScaler(), first_four_columns),
        ('minmax_scaler', MinMaxScaler(), last_four_columns),
```

```
    ]

    column_transformer = ColumnTransformer(transformers)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

    X_train_scaled = column_transformer.fit_transform(X_train)
    X_test_scaled = column_transformer.transform(X_test)

    X_train_scaled = pd.DataFrame(X_train_scaled, columns=np.
↪concatenate([first_four_columns, last_four_columns]))
    X_test_scaled = pd.DataFrame(X_test_scaled, columns=np.
↪concatenate([first_four_columns, last_four_columns]))

    return X_train_scaled,y_train, X_test_scaled,  y_test
```

[21]:
```
X_train, y_train, X_test, y_test = preprocess_data(df, 'Label')
```

[22]:
```
def train_classifiers(X_train, y_train, X_test, y_test, random_state=42):

    classifiers = {
        'Logistic Regression': LogisticRegression(max_iter=1000,␣
↪random_state=random_state),
        'Decision Tree': DecisionTreeClassifier(random_state=random_state),
        'Random Forest': RandomForestClassifier(random_state=random_state),
        'Support Vector Machine': SVC(random_state=random_state),
        'K-Nearest Neighbors': KNeighborsClassifier(),
        'Naive Bayes': GaussianNB(),
        'Gradient Boosting':␣
↪GradientBoostingClassifier(random_state=random_state),
        'Perceptron': Perceptron(random_state=random_state),
        'Quadratic Discriminant Analysis': QuadraticDiscriminantAnalysis(),
        'XGBoost': XGBClassifier(use_label_encoder=False,␣
↪eval_metric='mlogloss', random_state=random_state),
        'LightGBM': LGBMClassifier(random_state=random_state,verbosity=-1),
        'CatBoost': CatBoostClassifier(silent=True, random_state=random_state)
    }

    results = []

    for name, clf in classifiers.items():
        print(f"Training {name}...")
        clf.fit(X_train, y_train)

        train_acc = accuracy_score(y_train, clf.predict(X_train))
        test_acc = accuracy_score(y_test, clf.predict(X_test))
```

```
        results.append({'Model': name, 'Train Accuracy': train_acc, 'Test␣
↪Accuracy': test_acc})

    results_df = pd.DataFrame(results).sort_values('Test␣
↪Accuracy',ascending=False).reset_index(drop=True)
    return results_df
```

[23]:
```
results_df = train_classifiers(X_train, y_train, X_test, y_test)
```

```
Training Logistic Regression…
Training Decision Tree…
Training Random Forest…
Training Support Vector Machine…
Training K-Nearest Neighbors…
Training Naive Bayes…
Training Gradient Boosting…
Training Perceptron…
Training Quadratic Discriminant Analysis…
Training XGBoost…
Training LightGBM…
Training CatBoost…
```

[24]:
```
results_df
```

[24]:

	Model	Train Accuracy	Test Accuracy
0	LightGBM	0.992427	0.968745
1	Random Forest	1.000000	0.967617
2	XGBoost	0.998187	0.966812
3	K-Nearest Neighbors	0.973294	0.966006
4	CatBoost	0.982438	0.965684
5	Decision Tree	1.000000	0.938940
6	Gradient Boosting	0.930718	0.920735
7	Support Vector Machine	0.889914	0.891091
8	Logistic Regression	0.693547	0.697761
9	Perceptron	0.555426	0.549863
10	Naive Bayes	0.479981	0.483809
11	Quadratic Discriminant Analysis	0.168372	0.165781

[27]:
```
def optimize_lightgbm(X_train_scaled, y_train,X_test_scaled, y_test):

    def objective(trial):
        param = {
            'objective': 'multiclass',
            'num_class': len(set(y_train)),
            'boosting_type': trial.suggest_categorical('boosting_type',␣
↪['gbdt', 'dart', 'goss']),
```

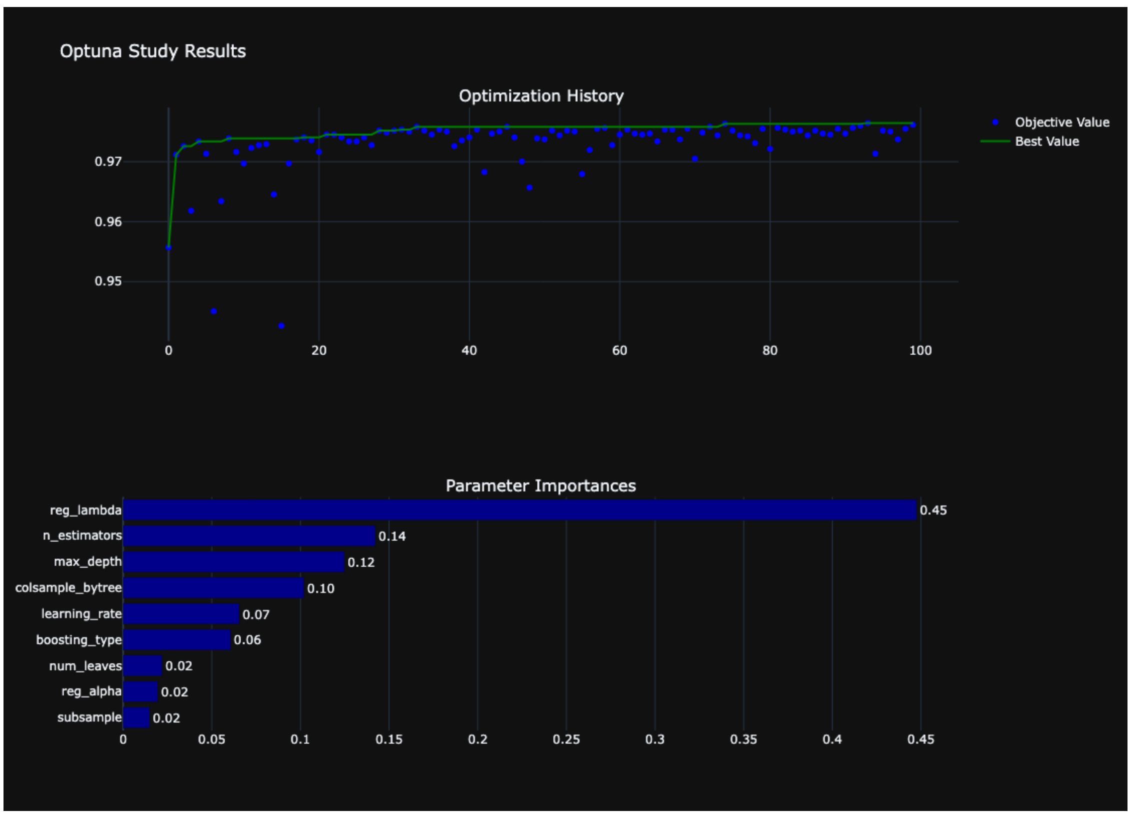
```
            'verbosity': -1,
            'num_leaves': trial.suggest_int('num_leaves', 70, 200),
            'max_depth': trial.suggest_int('max_depth', 6, 20),  # -1 means no␣
 ↪limit
            'learning_rate': trial.suggest_loguniform('learning_rate', 0.1, 1.
 ↪0),
            'n_estimators': trial.suggest_int('n_estimators', 20, 400),
            'subsample': trial.suggest_float('subsample', 0.2, 1.0),
            'colsample_bytree': trial.suggest_float('colsample_bytree', 0.6, 1.
 ↪0),
            'reg_alpha': trial.suggest_loguniform('reg_alpha', 1e-3, 0.5),
            'reg_lambda': trial.suggest_loguniform('reg_lambda', 0.01, 100),

        }

        model = LGBMClassifier(**param, random_state=42)
        model.fit(X_train_scaled, y_train)

        y_pred = model.predict(X_test_scaled)
        accuracy = accuracy_score(y_test, y_pred)

        return accuracy

    sampler = TPESampler(seed=42)
    study = optuna.create_study(direction='maximize',sampler=sampler)
    study.optimize(objective, n_trials=100)

    return study
```

[28]:
```
study = optimize_lightgbm(X_train,y_train, X_test, y_test)
print("Best hyperparameters: ", study.best_params)
```

```
Best hyperparameters:  {'boosting_type': 'gbdt', 'num_leaves': 155, 'max_depth':
14, 'learning_rate': 0.1652995026469861, 'n_estimators': 266, 'subsample':
0.636504703516293, 'colsample_bytree': 0.969280540948789, 'reg_alpha':
0.0025343248745577033, 'reg_lambda': 0.018495999829183322}
```

[48]:
```
fig = make_subplots(rows=2, cols=1, subplot_titles=("Optimization History",␣
 ↪"Parameter Importances"))

scatter_color = 'blue'
line_color = 'green'
fig1 = plot_optimization_history(study)


for trace in fig1.data:
    if isinstance(trace, go.Scatter) and 'markers' in trace.mode:
```

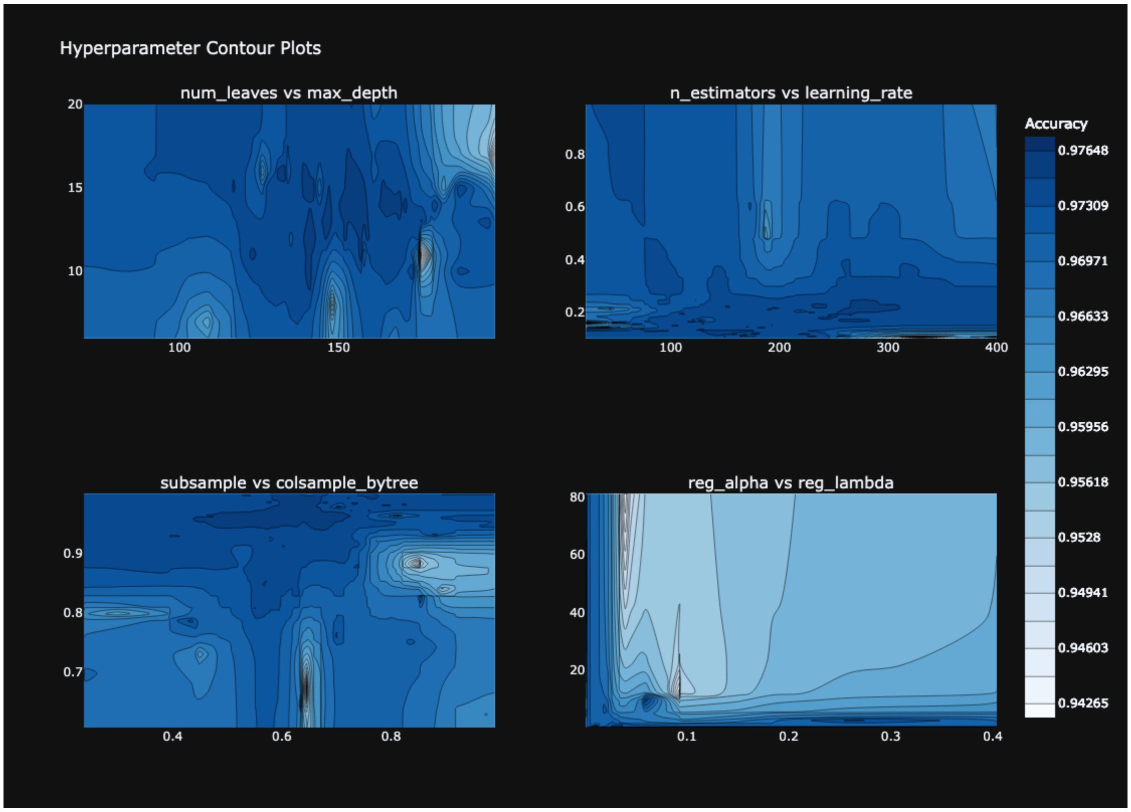
```python
        trace.update(marker=dict(color=scatter_color))
    elif isinstance(trace, go.Scatter) and 'lines' in trace.mode:
        trace.update(line=dict(color=line_color))

    fig.add_trace(trace, row=1, col=1)

fig2 = plot_param_importances(study)

bar_color = 'darkblue'

for trace in fig2.data:
    if isinstance(trace, go.Bar):
        trace.update(marker=dict(color=bar_color))
    trace.showlegend = False
    fig.add_trace(trace, row=2, col=1)

fig.update_layout(height=800, title_text="Optuna Study Results")
fig.show()
```



[49]:
```python
def plot_hyperparameter_contour(study, param1, param2, row, col):
    x = np.array([trial.params[param1] for trial in study.trials])
```

```
    y = np.array([trial.params[param2] for trial in study.trials])
    z = np.array([trial.value for trial in study.trials])

    contour = go.Contour(
        x=x,
        y=y,
        z=z,
        colorscale='Blues',
        colorbar=dict(title="Accuracy"),
        contours=dict(
            start=np.min(z),
            end=np.max(z),
            size=(np.max(z) - np.min(z)) / 20,
            showlabels=False
        ),
        name=f"{param1} vs {param2}"
    )

    return contour

def plot_all_hyperparameter_contours(study):
    fig = make_subplots(rows=2, cols=2, subplot_titles=(
        "num_leaves vs max_depth",
        "n_estimators vs learning_rate",
        "subsample vs colsample_bytree",
        "reg_alpha vs reg_lambda"
    ))

    hyperparams = [
        ('num_leaves', 'max_depth', 1, 1),
        ('n_estimators', 'learning_rate', 1, 2),
        ('subsample', 'colsample_bytree', 2, 1),
        ('reg_alpha', 'reg_lambda', 2, 2)
    ]

    for param1, param2, row, col in hyperparams:
        contour = plot_hyperparameter_contour(study, param1, param2, row, col)
        fig.add_trace(contour, row=row, col=col)

    fig.update_layout(title="Hyperparameter Contour Plots", height=800,␣
  ↪width=1000)
    fig.show()
```

[50]:
```
plot_all_hyperparameter_contours(study)
```



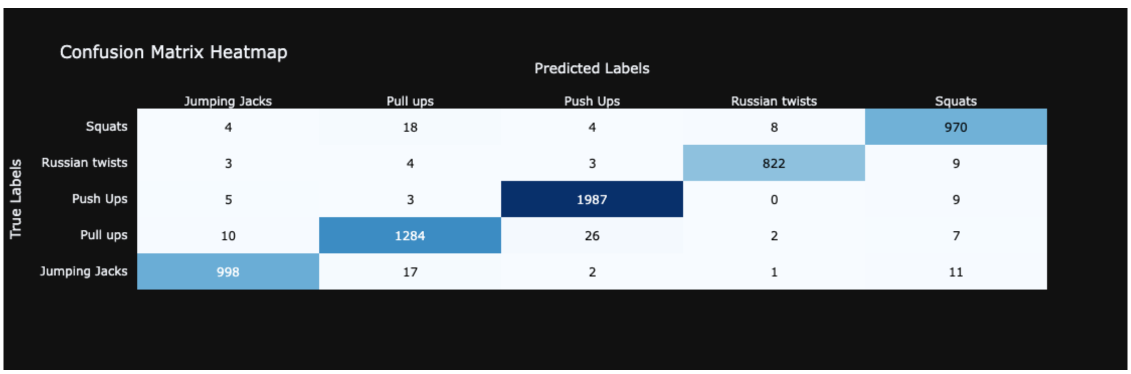
[32]:

```
best_params = study.best_params

print(f"Re-training LightGBM...")
model = LGBMClassifier(**best_params, random_state=42)

model.fit(X_train, y_train)


y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)


train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
confusion_matrix_test = confusion_matrix(y_test, y_test_pred)


print(f"Training Accuracy: {train_accuracy:.5f}")
print(f"Test Accuracy: {test_accuracy:.5f}")
```

```
Re-training LightGBM…
Training Accuracy: 1.00000
```

```
Test Accuracy: 0.97648
```

[36]:
```python
cm_df = pd.DataFrame(confusion_matrix_test,
                     index=label_encoder.
 ↪inverse_transform(range(confusion_matrix_test.shape[0])),
                     columns=label_encoder.
 ↪inverse_transform(range(confusion_matrix_test.shape[1])))

fig = ff.create_annotated_heatmap(
    z=cm_df.values,
    x=cm_df.columns.tolist(),
    y=cm_df.index.tolist(),
    colorscale='Blues',

)

fig.update_layout(
    title='Confusion Matrix Heatmap',
    xaxis_title='Predicted Labels',
    yaxis_title='True Labels',
)

fig.show()
```



[51]:
```python
feature_importances = model.feature_importances_
features = X_train.columns

importance_df = pd.DataFrame({
    'Feature': features,
    'Importance': feature_importances
})

importance_df['Normalized Importance'] = importance_df['Importance'] /␣
 ↪importance_df['Importance'].sum()
```

```python
importance_df = importance_df.sort_values(by='Normalized Importance',␣
↪ascending=True)

fig = go.Figure()

fig.add_trace(go.Bar(
    x=importance_df['Normalized Importance'],
    y=importance_df['Feature'],
    orientation='h',
    marker=dict(color='darkblue')
))

fig.update_layout(
    title='Normalized Feature Importance',
    xaxis_title='Normalized Importance Score',
    yaxis_title='Features',
    showlegend=False
)

fig.show()
```