BLM 4800 INTRODUCTION to DATA MINING

COURSE PROJECT REPORT

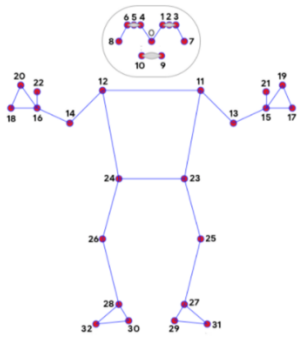
GÜLSÜM İREM BAŞ

19011502

2022

1. Introduction

In this project, classification of actions was applied using the given dataset. The given dataset contains 10 different exercise names, each with the starting and ending positions of the 5 exercise names. The dataset includes location information found using Mediapipe pose estimation landmarks, which detects the human skeleton of the person in the frame, based on the analysis of 500 videos. You can see the Mediapipe Pose Estimation Landmark below.



0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
17. left_pinky
18. right_pinky
19. left_index
20. right_index
21. left_thumb
22. right_thumb
23. left_hip
24. right_hip
25. left_knee
26. right_knee
27. left_ankle
28. right_ankle
29. left_heel
30. right_heel
31. left_foot_index
32. right_foot_index

The goal of this project is to train a model that classifies the poses in the given dataset using the pose positions or by adding different features.

2. Data Analysis

The exercise dataset was imported using the read_csv function and the general structure of the data was examined. The features and labels were analyzed to determine what they are and how many different values they take. Below you can see some parts, features and labels from the dataset.

➔ Dataset overview:

```
[314] exercise_data = pd.read_csv("/content/exercise.csv")
```

```
exercise_data.head()
```

	pose_id	pose	x_nose	y_nose	z_nose	x_left_eye_inner	y_left_eye_inner	z_left_eye_inner	x_left_eye	y_left_eye	...	z_left_heel	x_ri
0	0	squats_up	-0.382815	-48.231250	-54.405792	0.137189	-50.040543	-51.997875	0.502047	-50.058890	...	33.283375	
1	1	situp_down	54.146880	-12.822491	5.564175	56.762527	-11.221117	-0.363063	56.795986	-10.608183	...	-8.073100	
2	2	situp_down	9.891440	-54.147266	85.344970	12.784414	-55.229970	88.534775	14.006874	-54.291880	...	-132.024460	
3	3	jumping_jacks_up	0.904673	-51.350130	-33.606970	1.338871	-53.172337	-30.013737	1.743913	-53.050697	...	51.615970	
4	4	jumping_jacks_down	-3.153129	-55.255062	-17.745928	-2.046205	-57.477790	-18.198952	-1.506304	-57.428230	...	-2.849554	

5 rows × 101 columns

➔ Labels:

```
exercise_data['pose']
```

```
0                squats_up
1              situp_down
2              situp_down
3         jumping_jacks_up
4       jumping_jacks_down
              ...
1092             situp_up
1093      jumping_jacks_up
1094         pullups_down
1095            situp_down
1096      jumping_jacks_up
Name: pose, Length: 1097, dtype: object
```

➔ Features:

```
] exercise_data.columns
```

```
Index(['pose_id', 'pose', 'x_nose', 'y_nose', 'z_nose', 'x_left_eye_inner',
       'y_left_eye_inner', 'z_left_eye_inner', 'x_left_eye', 'y_left_eye',
       ...
       'z_left_heel', 'x_right_heel', 'y_right_heel', 'z_right_heel',
       'x_left_foot_index', 'y_left_foot_index', 'z_left_foot_index',
       'x_right_foot_index', 'y_right_foot_index', 'z_right_foot_index'],
      dtype='object', length=101)
```

New features were derived from the existing features in order to improve the training of the model. The distances between two points were used as new features by using the positions of some points. The Euclidean rule was used to calculate the distances. Below you can see the function used for distance calculation and the calculated distances.

```
def euclidean_distance(dataframe, first_point, second_point, distance_col):
    first_point = dataframe[first_point].to_numpy()
    other_points = dataframe[second_point].to_numpy()
    dataframe[distance_col] = np.linalg.norm(first_point - other_points, axis=1)
```

```
exercise_data.columns
```

```
Index(['pose_id', 'pose', 'x_nose', 'y_nose', 'z_nose', 'x_left_eye_inner',
       'y_left_eye_inner', 'z_left_eye_inner', 'x_left_eye', 'y_left_eye',
       ...
       'y_left_foot_index', 'z_left_foot_index', 'x_right_foot_index',
       'y_right_foot_index', 'z_right_foot_index', 'dist_left_wrist_shoulder',
       'dist_left_hip_elbow', 'dist_left_hip_ankle', 'dist_left_right_ankle',
       'dist_left_right_wrist'],
      dtype='object', length=106)
```

The data is ready for model training as it does not contain any null values or categorical values

3. Kaggle Competition Results

Two algorithms were used to train the prepared dataset and test the trained model with the test data used in the competition. One is the decision tree algorithm and the other is a neural network.

3.1) Decision Tree Algorithm

The decision tree algorithm is a simple and powerful method for making predictions based on multiple features. It is fast and easy to understand, and can handle both numerical and categorical data. However, it can also be prone to overfitting if the tree becomes too complex.

The code is performing a grid search with cross-validation to tune the hyperparameters of a decision tree classifier on the exercise dataset. The hyperparameters being tuned are the **max_depth** and **min_samples_split** parameters, which control the complexity of the decision tree model.

The dataset is first split into training and testing sets using the **train_test_split** function. The decision tree classifier is then trained on the training set.The grid search object is created using the decision tree classifier and a grid of hyperparameter values to try. The grid search object performs cross-validation on the training set and tunes the hyperparameters to find the best combination that maximizes the model's performance.Once the best hyperparameters are found, a new decision tree classifier is trained using those hyperparameters and then used to make predictions on the test set. The accuracy of the predictions is then calculated and printed.

In conclusion, a model training was carried out with the hyperparameters that maximized the performance that can be obtained from the decision tree algorithm.

Below are the train and test accuracies obtained from model training with the decision tree algorithms are shown. In these accuracy calculations, the test data set aside from our own data was used.

```
train_accuracy = round(cart_tuned.score(X_train, y_train)*100)
test_accuracy = round(accuracy_score(y_pred, y_test)*100)

print("Decision Tree Train Accuracy Score : {}% ".format(train_accuracy))
print("Decision Tree Test Accuracy Score  : {}% ".format(test_accuracy))
```

```
Decision Tree Train Accuracy Score : 93%
Decision Tree Test Accuracy Score  : 72%
```

3.2) Neural Network

The code is defining and training a neural network using the TensorFlow library. The network consists of an input layer, three hidden layers, and an output layer. The hidden layers use the ReLU activation function and the output layer uses the softmax activation function. The network also includes dropout layers, which randomly drop out a fraction of the neurons during training to prevent overfitting.The network is compiled using the Adam optimization algorithm and the categorical cross-entropy loss function. The accuracy metric is also specified as a measure of the network's performance.The network is then trained using the **fit** method, which trains the network on the training data and evaluates it on the test data at the end of each epoch. The number of epochs is set to 100, and the number of steps per epoch and the number of validation steps are also specified.

Below are the train and test accuracies obtained from model training with the neural network algorithm are shown. In these accuracy calculations, the test data set aside from our own data was used.

```
y_pred = My_model.predict(X_train).argmax(1)
print(classification_report(y_train.argmax(1), y_pred))
```

```
26/26 [==============================] - 0s 2ms/step
              precision    recall  f1-score   support

           0       0.97      0.91      0.94        92
           1       1.00      1.00      1.00        64
           2       0.95      0.94      0.94       110
           3       0.97      0.99      0.98       114
           4       0.94      0.97      0.95        96
           5       1.00      1.00      1.00        55
           6       1.00      0.99      0.99        87
           7       1.00      1.00      1.00        73
           8       0.98      1.00      0.99        79
           9       1.00      1.00      1.00        52

    accuracy                           0.98       822
   macro avg       0.98      0.98      0.98       822
weighted avg       0.98      0.98      0.98       822
```

```
y_pred = My_model.predict(X_test).argmax(1)
print(classification_report(y_test.argmax(1), y_pred))
```

```
9/9 [==============================] - 0s 3ms/step
              precision    recall  f1-score   support

           0       0.48      0.53      0.50        19
           1       0.85      0.94      0.89        18
           2       0.70      0.80      0.75        35
           3       0.85      0.76      0.80        37
           4       0.67      0.67      0.67        27
           5       0.89      0.93      0.91        27
           6       0.96      0.86      0.91        28
           7       0.76      0.79      0.77        28
           8       0.88      0.79      0.84        29
           9       0.77      0.74      0.75        27

    accuracy                           0.78       275
   macro avg       0.78      0.78      0.78       275
weighted avg       0.79      0.78      0.78       275
```

3.3) The accuracy of the decision tree algorithm was slightly lower than that of the neural network, and the neural network also had better handling of overfitting. Therefore, the neural network was chosen to use in the competition. A few other algorithms were also tried, but they encountered overfitting, so it was decided that the neural network was the best performing algorithm for this data.

4. Results on Given Test Set

After a few trials, I decided to use a neural network for the Kaggle competition. Below is the accuracy obtained with the test data in the competition.

Gülsüm İrem Baş		0.72363

- Details of Neural Network Algoritm

Here is the model training with parameters

```
My_model = tf.keras.Sequential([
    tf.keras.layers.Reshape(target_shape=(105,), input_shape=(105,)),
    tf.keras.layers.Dense(units=256, activation='relu'),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(units=192, activation='relu'),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(units=10, activation='softmax')
])
```

The parameters used in the code are:

- **target_shape**: This parameter specifies the shape of the output tensor from the input layer. In this case, the output tensor has shape **(105,)**, which means it has 105 elements.
- **input_shape**: This parameter specifies the shape of the input tensor to the network. In this case, the input tensor has shape **(105,)**, which means it has 105 elements.
- **units**: This parameter specifies the number of units or neurons in a layer. In this case, the hidden layers have 256 and 192 units, respectively, and the output layer has 10 units, one for each class.
- **activation**: This parameter specifies the activation function to use in a layer. In this case, the hidden layers use the ReLU activation function and the output layer uses the softmax activation function.

- Results

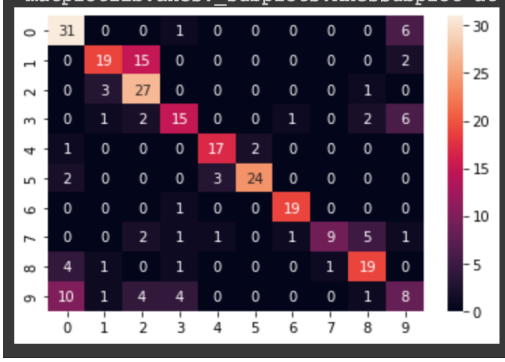
Below is the confusion matrix and metric calculations found with the test data

```
print(metrics.classification_report(result['pose'], test_df_solution['pose']))
```

	precision	recall	f1-score	support
jumping_jacks_down	0.74	0.76	0.75	37
jumping_jacks_up	0.69	0.74	0.71	34
pullups_down	0.87	0.60	0.71	45
pullups_up	0.44	0.86	0.59	14
pushups_down	0.80	0.94	0.86	17
pushups_up	0.93	0.93	0.93	29
situp_down	0.95	0.86	0.90	22
situp_up	0.55	0.69	0.61	16
squats_down	0.73	0.63	0.68	30
squats_up	0.50	0.45	0.47	31
accuracy			0.72	275
macro avg	0.72	0.75	0.72	275
weighted avg	0.74	0.72	0.72	275

```
import seaborn as sns
confusion_mat = confusion_matrix(test_df_solution['pose'], result['pose'])
sns.heatmap(confusion_mat, annot=True, fmt="d")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa8cbb9b2b0>
```



5. COMPARING RESULTS

The following machine learning algorithms were used with in the scope of the project.

- Decision Tree Algorithm
- Random Forest Algorithm
- KNN Algorithm
- Neural Network Algorithm

The following table shows the train and test accuracies of the algorithms used.

	Train Accuracy	Test Accuracy
Decision Tree	93%	74%
Random Forest	100%	80%
KNN	100%	68%
Neural Network	97%	78%

From this table, it can be seen that both models, random forest and knn algorithms, have very high accuracy on the training data and lower accuracy on the test data. This suggests that the model has overfitted. Overfitting occurs when a model is too complex and has too many parameters relative to the amount of data it is trained on. As a result, the model becomes highly sensitive to the specific details of the training data, and is not able to generalize well to new, unseen data.

To improve the performance of the model on the test data, it may be necessary to simplify the model by using fewer parameters or by using regularization techniques to prevent the model from overfitting. It may also be helpful to increase the amount of training data to give the model more examples to learn from and improve its ability to generalize to new data.

- To compare the Decision Tree and Neural Network Algorithm

The results for the decision tree model show that the model has achieved a 93% accuracy on the training data and a 74% accuracy on the test data. This indicates that the model is able

to capture the underlying patterns in the data to some extent, but is still making a relatively large number of mistakes on the test data.

The neural network algorithm was chosen because it gave higher accuracy on the test data. However, the number of layers or units can still be changed in order to increase accuracy further. In addition, the dropout values that were included to prevent overfitting can also be changed.

There are several techniques that can be used to prevent overfitting in neural networks:

1. Early stopping: This involves stopping the training process at an early stage, before the model has had a chance to fully fit the training data. This can be done by monitoring the performance of the model on a validation set and stopping the training when the performance on the validation set begins to degrade.
2. Regularization: This involves adding a penalty to the loss function that discourages the model from fitting the training data too closely. There are several types of regularization, including L1 and L2 regularization, which add penalties based on the magnitude of the weights, and dropout, which randomly sets a fraction of the units in the network to zero during training.
3. Simplifying the model: This can be done by reducing the number of layers or units in the network, or by using simpler network architectures such as linear models or shallow neural networks.