ANN Final

Fatih Er - 040200004

I tried 2 different methods for this final. One with manual recording and the other one with image classification.

First Method

For this method I dropped a coin 250 times from a height of 1 meter. I recorded the values to an excel file.

```
#Reading csv
df = pd.read_csv("Final.csv",delimiter = ';')
df
```

	Initial Side	Initial Velocity	Material	r distance (cm)	Distance to Origin	Coin Flip	Direction	Final Side
0	Heads	Free fall	Carpet	13	1	1	South	Tails
1	Heads	Free fall	Carpet	26	3	2	South	Tails
2	Heads	Free fall	Carpet	18	1	1	North	Tails
3	Heads	Free fall	Carpet	22	2	1	East	Heads
4	Heads	Free fall	Carpet	15	1	2	East	Tails
...	...	...	...	...	...	...	...	...
245	Tails	Velocity	Carpet	40	6	2	North West	Heads
246	Tails	Velocity	Carpet	29	7	2	South	Tails
247	Tails	Velocity	Carpet	35	5	2	East	Heads
248	Tails	Velocity	Carpet	26	6	2	North	Heads
249	Tails	Velocity	Carpet	46	11	2	North	Heads

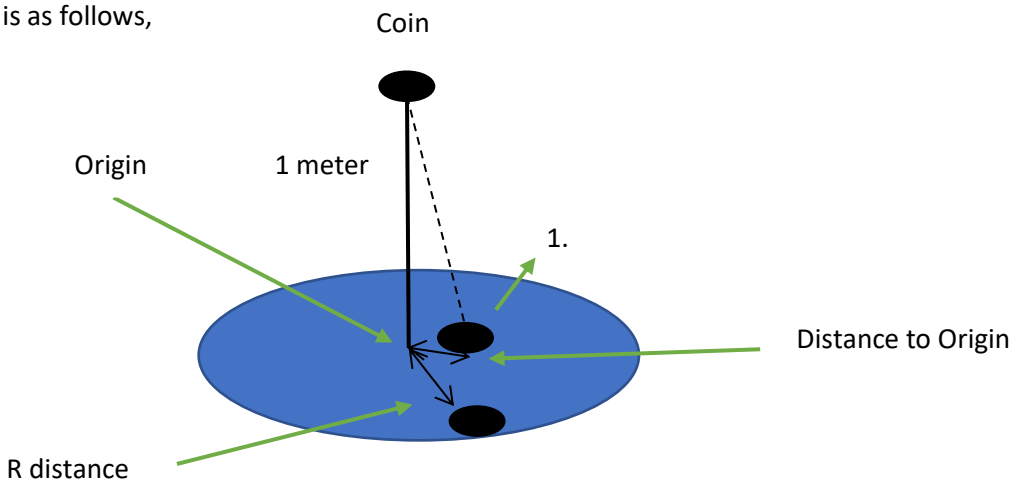
250 rows × 8 columns

Final form of the table is like this figure. The column information is as follows,

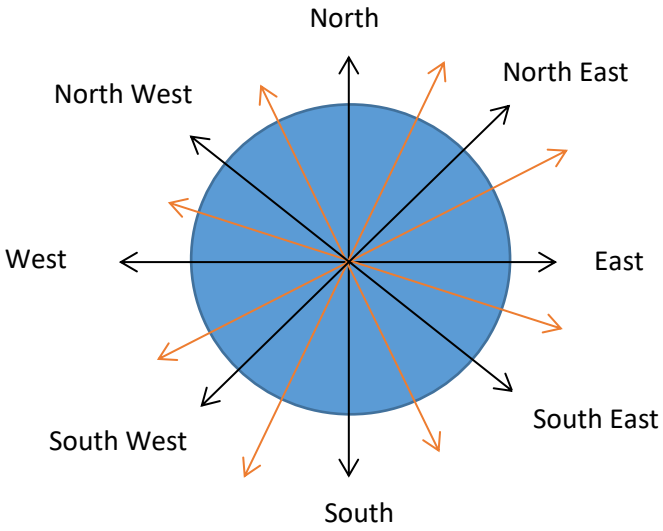
- Initial side: Before the dropping action the coin's side: Heads / Tails
- Initial Velocity: I make the dropping action in two ways: Free fall and with velocity. In with velocity category, I dropped it with the same velocity every time.
- Material: I dropped the the coin to two different categories: Carpet and Marble
- R distance: R distance is the final position of the coin to the origin.
- Distance to Origin: This parameter is the point where the coin first hits.
- Coin Flip: This parameter is the total amount of flips that coin did.
- Direction: Direction is based on where I faced.

I can show these values in a figure.

The figure is as follows,



Directions:



Every direction is divided by 45 degrees.

Importance of data collection:

- The most important thing to consider when collecting data is that the same environment must be provided for each coin drop. The only things that can change is the independent variables.
- In addition, measurements must be made very cautiously.
- Measuring many parameters also helps diversify the experiment and helps machine learning algorithms produce more accurate results.

Starting to code:

1. Imported the needed libraries

```
#Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import random
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.neural_network import MLPClassifier
```

2. Imported the data set

```
#Reading csv
df = pd.read_csv("Final.csv",delimiter = ';')
df
```

	Initial Side	Initial Velocity	Material	r distance (cm)	Distance to Origin	Coin Flip	Direction	Final Side
0	Heads	Free fall	Carpet	13	1	1	South	Tails
1	Heads	Free fall	Carpet	26	3	2	South	Tails
2	Heads	Free fall	Carpet	18	1	1	North	Tails
3	Heads	Free fall	Carpet	22	2	1	East	Heads
4	Heads	Free fall	Carpet	15	1	2	East	Tails
...	...	...	...	...	...	...	...	...
245	Tails	Velocity	Carpet	40	6	2	North West	Heads
246	Tails	Velocity	Carpet	29	7	2	South	Tails
247	Tails	Velocity	Carpet	35	5	2	East	Heads
248	Tails	Velocity	Carpet	26	6	2	North	Heads
249	Tails	Velocity	Carpet	46	11	2	North	Heads

250 rows × 8 columns

3. EDA

EDA steps are crucial for understanding the data we are using.

```
#Time for EDA
df.info
```

```
<bound method DataFrame.info of     Initial Side Initial Velocity Material  r distance (cm)  \
0          Heads        Free fall   Carpet               13
1          Heads        Free fall   Carpet               26
2          Heads        Free fall   Carpet               18
3          Heads        Free fall   Carpet               22
4          Heads        Free fall   Carpet               15
..           ...              ...      ...              ...
245        Tails         Velocity   Carpet               40
246        Tails         Velocity   Carpet               29
247        Tails         Velocity   Carpet               35
248        Tails         Velocity   Carpet               26
249        Tails         Velocity   Carpet               46

     Distance to Origin  Coin Flip    Direction Final Side
0                     1          1       South      Tails
1                     3          2       South      Tails
2                     1          1      North       Tails
3                     2          1        East      Heads
4                     1          2        East      Tails
..                  ...        ...         ...        ...
245                   6          2  North West      Heads
246                   7          2       South      Tails
247                   5          2        East      Heads
248                   6          2       North      Heads
249                  11          2       North      Heads

[250 rows x 8 columns]>
```

```
df.isna().sum()
```

```
Initial Side          0
Initial Velocity      0
Material              0
r distance (cm)       0
Distance to Origin    0
Coin Flip             0
Direction             0
Final Side            0
dtype: int64
```

```
df.dtypes
```

```
Initial Side          object
Initial Velocity      object
Material              object
r distance (cm)        int64
Distance to Origin     int64
Coin Flip              int64
Direction             object
Final Side            object
dtype: object
```

Checking the types of the column types. I will encode the columns with object type.

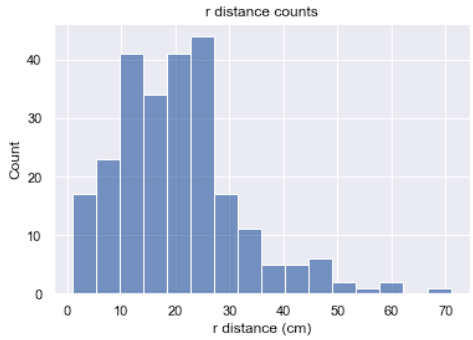
```
n = df.nunique(axis=0)

print("No.of.unique values in each column :\n",
      n)
```

```
No.of.unique values in each column :
 Initial Side           2
Initial Velocity        2
Material                2
r distance (cm)        52
Distance to Origin     22
Coin Flip               4
Direction               9
Final Side              2
dtype: int64
```

4. Data Visualization

```
#Visualization
sns.set()
sns.histplot(df["r distance (cm)"])
plt.title("r distance counts")
```

```
Text(0.5, 1.0, 'r distance counts')
```



As we can see from this graph, the coin usually dropped between 10 and 30 cm interval.

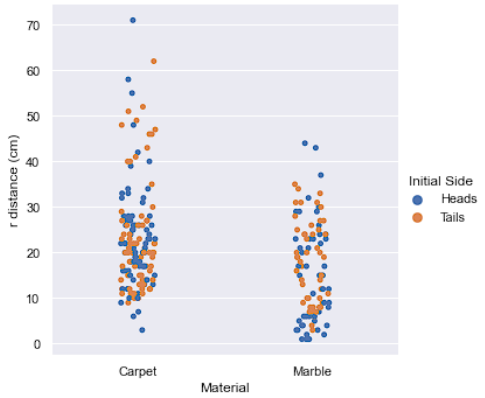
```
sns.set()
sns.catplot(data = df, x= "Initial Velocity", y= "r distance (cm)", hue = "Initial Side")
```

```
<seaborn.axisgrid.FacetGrid at 0x1da7808da00>
```

As can be seen from this graph, when I release the coin at a certain speed, it hoes farther than in free fall.

```
sns.catplot(data = df, x= "Material", y= "r distance (cm)", hue = "Initial Side")
```

```
<seaborn.axisgrid.FacetGrid at 0x1da78e6d910>
```



Some inferences:

- Mostly coins drop between 10 and 30 cm interval

- If coin drops with velocity it can go further than a free fall.
- If the coin dropped to carpet it can go further than marble.
- The coin's final drop side is a random event. It can be Heads or Tails by luck.

5. Preprocess Data

I have to encode the object typed columns. I used LabelEncoder to do this.

```
#Preprocess Data
le = LabelEncoder()
df["Initial Side"] = le.fit_transform(df["Initial Side"])
df["Final Side"] = le.fit_transform(df["Final Side"])
df["Initial Velocity"] = le.fit_transform(df["Initial Velocity"])
df["Material"] = le.fit_transform(df["Material"])
df["Direction"] = le.fit_transform(df["Direction"])
df
```

	Initial Side	Initial Velocity	Material	r distance (cm)	Distance to Origin	Coin Flip	Direction	Final Side
0	0	0	0	13	1	1	5	1
1	0	0	0	26	3	2	5	1
2	0	0	0	18	1	1	2	1
3	0	0	0	22	2	1	0	0
4	0	0	0	15	1	2	0	1
...	...	...	...	...	...	...	...	...
245	1	1	0	40	6	2	4	0
246	1	1	0	29	7	2	5	1
247	1	1	0	35	5	2	0	0
248	1	1	0	26	6	2	1	0
249	1	1	0	46	11	2	1	0

250 rows × 8 columns

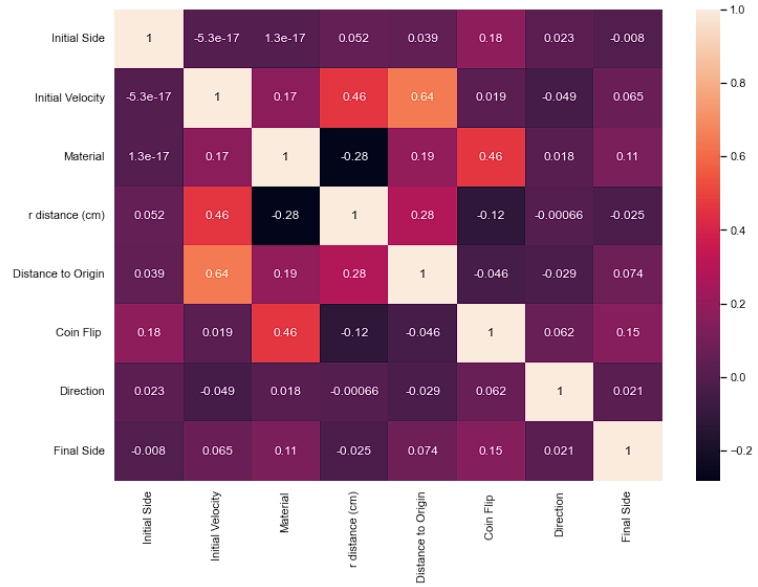
Encoded column values:

- Initial Side -> Heads: 0, Tails: 1
- Initial Velocity -> Free Fall: 0, Velocity: 1
- Material -> Carpet: 0, Marble: 1
- Direction -> East:0, North: 1-2, North East: 3 North West: 4, South: 5, South East:6, South West: 7, West: 8

Correlation between columns:

```
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.heatmap(df.corr(),annot = True)
```

```
<AxesSubplot:>
```



6. Split and Train Data

```
# Ready to Predict
X = df.drop(["r distance (cm)", "Distance to Origin", "Final Side","Direction"], axis=1)
y_coin_position = df["r distance (cm)"]
y_distance_to_origin = df["Distance to Origin"]
y_final_side = df["Final Side"]
y_direction = df["Direction"]
```

```
# Train-test split
X_train, X_test, y_train_coin_position, y_test_coin_position = train_test_split(X, y_coin_position, test_size=0.2, random_state=4
X_train, X_test, y_train_distance_to_origin, y_test_distance_to_origin = train_test_split(X, y_distance_to_origin, test_size=0.2,
X_train, X_test, y_train_final_side, y_test_final_side = train_test_split(X, y_final_side, test_size=0.2, random_state=42)
X_train, X_test, y_direction, y_direction = train_test_split(X, y_direction, test_size=0.2, random_state=42)
```

I created three variables that I will predict. I want to predict the r distance, Distance to Origin and Final Side. After that I splitted the data to train and test.

```
# Build the MLPClassifier model for each target variable
model_coin_position = MLPClassifier(hidden_layer_sizes=(64, 32), activation='relu', max_iter=1000)
model_distance_to_origin = MLPClassifier(hidden_layer_sizes=(64, 32), activation='relu', max_iter=1000)
model_final_side = MLPClassifier(hidden_layer_sizes=(64, 32), activation='relu', max_iter=1000)
model_direction = MLPClassifier(hidden_layer_sizes=(64, 32), activation='relu', max_iter=1000)

# Train each model
model_coin_position.fit(X_train, y_train_coin_position)
model_distance_to_origin.fit(X_train, y_train_coin_flip)
model_final_side.fit(X_train, y_train_final_side)
model_direction.fit(X_test, y_direction)
```

I used MLPClassifier Neural Network for predicting the data.

```
#Accuracy
accuracy_final_side = model_final_side.score(X_test, y_test_final_side)
print(f'Final Side Accuracy: {accuracy_final_side}')
```

```
Final Side Accuracy: 0.6
```

Accuracy result is given.

7. Predicted Values

```
#Randomized events
Initial_side = []
Initial_velocity = []
Material = []
for i in range(500):
    Initial_side.append(random.randint(0,1))
for i in range(500):
    Initial_velocity.append(random.randint(0,1))
for i in range(500):
    Material.append(random.randint(0,1))


input_data = pd.DataFrame({
    'Initial Side': Initial_side,
    'Initial Velocity': Initial_velocity,
    'Material': Material,
})
```

I made randomized events for different cases in order to predict more values.

```
# Predictions for randomized events
for i in range(500):
    prediction_coin_position = model_coin_position.predict(input_data)
    prediction_distance_to_origin = model_distance_to_origin.predict(input_data)
    prediction_final_side = model_final_side.predict(input_data)
    prediction_direction = model_direction.predict(input_data)

input_data["r distance"] = prediction_coin_position
input_data["Distance to Origin"] = prediction_distance_to_origin
input_data["Direction"] = prediction_direction
input_data["Final Side"] = prediction_final_side
predicted_data = input_data.drop_duplicates(inplace=False)
display(predicted_data)
```

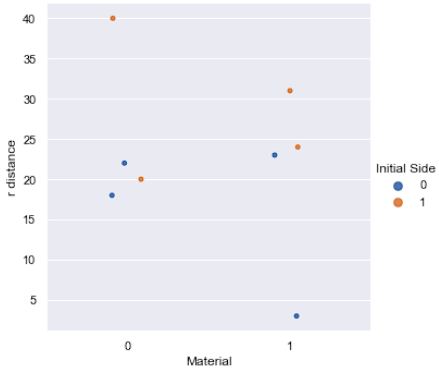
	Initial Side	Initial Velocity	Material	r distance	Distance to Origin	Direction	Final Side
0	1	0	0	20	4	8	0
1	0	0	1	3	3	1	1
2	0	1	1	23	5	1	1
3	1	0	1	24	4	5	1
6	0	1	0	18	4	1	1
10	1	1	0	40	5	0	0
11	1	1	1	31	9	1	1
14	0	0	0	22	1	8	0

After that I dropped the duplicates but I only had 8 results because of there are 8 possible ways with Initial Side, Initial Velocity and Material. The algorithm always predicts the same distance, direction and side for that specific case. I can say that this is a challenge because the coin can go different spots in specific case.

8. Predicted Values Visualization

```
sns.catplot(data = predicted_data, x= "Material", y= "r distance", hue = "Initial Side")
```

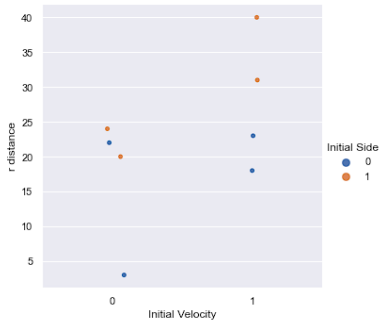
```
<seaborn.axisgrid.FacetGrid at 0x1da01dbf880>
```



We can make the same visualization as before. 0 is Carpet, 1 is Marble for Material. 0 is Heads, 1 is Tales for Initial Side.

```
sns.catplot(data = predicted_data, x= "Initial Velocity", y= "r distance", hue = "Initial Side")
```

```
<seaborn.axisgrid.FacetGrid at 0x1da0120b760>
```



0 is Free Fall, 1 is Velocity for Initial Velocity. 0 is Heads, 1 is Tales for Initial Side.

9. Outcomes and Challenges

- The predicted values are close to the overall average of r distance. So, this is a plus. But as I said before the algorithm always predicts the same distance, direction and side for that specific case. I can say that this is a challenge because the coin can go different spots in specific case.
- Model selection is a very crucial part in this prediction and accuracy. I selected this algorithm because of it trains iteratively with partial derivatives and this model prevents overfitting. In my opinion there can be more accurate models.
- As I said before data collection part is so important. All the coding and algorithm relies on to the data collection part.
- This final project developed me in the importance of collecting data, making inferences from this data, visualizing this data and choosing the right algorithm.
- Coin Flip column had problems that I couldn't solve so I had to remove it later on.

Second Method

This method is not accurate and well-prepared as the first method. For this method I took the photos of the coins that I dropped and labeled them as Heads or Tails. I put them into a folder.

Imported Libraries:

```
import os
import pickle

from skimage.io import imread
from skimage.transform import resize
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

Preparing the data:

```
# prepare data
input_dir = r'C:\Users\erfat\Desktop\Data'
categories = ['Heads', 'Tails']

data = []
labels = []
for category_idx, category in enumerate(categories):
    for file in os.listdir(os.path.join(input_dir, category)):
        img_path = os.path.join(input_dir, category, file)
        img = imread(img_path)
        img = resize(img, (15, 15))
        data.append(img.flatten())
        labels.append(category_idx)

data = np.asarray(data)
labels = np.asarray(labels)
```

This prepares the data by reading from the file and categorize it.

Split the data and Train it for classification

```
# train / test split
x_train, x_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, shuffle=True, stratify=labels)

# train classifier
classifier = SVC()

parameters = [{'gamma': [0.01, 0.001, 0.0001], 'C': [1, 10, 100, 1000]}]

grid_search = GridSearchCV(classifier, parameters)

grid_search.fit(x_train, y_train)
```

Performance of Classification

```
# test performance
best_estimator = grid_search.best_estimator_

y_prediction = best_estimator.predict(x_test)

score = accuracy_score(y_prediction, y_test)

print('{}% of samples were correctly classified'.format(str(score * 100)))

pickle.dump(best_estimator, open('./model.p', 'wb'))
```

```
55.00000000000001% of samples were correctly classified
```

This accuracy is close to the first method’s accuracy but we don’t have any data about the positions. I cannot get the values from the image so I get the data manually.