train_model

July 30, 2024

[1]:

```
import numpy as np
import sqlite3
import pandas as pd

from sklearn.linear_model import LinearRegression
from tqdm.notebook import tqdm

from utils.get_or_create_combined_database import␣
 ↪get_or_create_combined_database
from utils.create_sequences_in_batches import calculate_sequences_in_batches
from utils.compare_models import compare_models, shape_input_for_model
from utils.get_data import clear_cache, fetch_data_batches
from utils.recreate_cleaned_data import recreate_cleaned_data

from utils.create_sequences_in_batches import␣
 ↪create_sequences_from_database_rows
from utils.plot_prediction_on_plot import plot_prediction_on_plot
from utils.create_prediction_animation import create_prediction_animation
from matplotlib import pyplot as plt

from constants import DB_columns

import os
from dotenv import load_dotenv
load_dotenv(verbose=True, override=True)

RECREATE_CLEANED_DATA = False
TRAINING = False
CREATE_ANIMATIONS = False
CREATE_VISUALIZATIONS = True

zoom_range = ((75, 14350), (75, 14350))
normalized_zoom_range = ((0, 1), (0, 1))
```

1 Data

[2]:

```python
database_folder = os.getenv("DATABASE_FOLDER")

database_file = get_or_create_combined_database(database_folder)

table_name = "champs_cleaned"
```

```
Thumbs.db
combined2.db
Found 2 database files in the folder specified by DATABASE_FOLDER
Found combined database /u/23/tarpill1/unix/Documents/combined2.db
```

[3]:

```python
if RECREATE_CLEANED_DATA:
    from utils.clean_and_normalize_table import clean_and_normalize_table
    clean_and_normalize_table(database_file, table_name, "champs")
```

[4]:

```python
# Check values from the new table

conn = sqlite3.connect(database_file)
pd.set_option('display.max_columns', None)
pd.read_sql_query(f"SELECT * FROM {table_name} LIMIT 5", conn)
```

[4]:

```
      game_id      time         name     hp  max_hp     mana  max_mana  armor  \
0  2841236401  5.541945  Mordekaiser  645.0   645.0      0.0     100.0   61.0
1  2841236401  5.541945        Viego  630.0   630.0  10000.0   10000.0   46.0
2  2841236401  5.541945        Riven  745.0   745.0      0.0       0.0   33.0
3  2841236401  5.541945       Ezreal  600.0   600.0    375.0     375.0   36.0
4  2841236401  5.541945      Leblanc  598.0   598.0    400.0     400.0   34.0

     mr    ad    ap  level  atk_range  visible  team  pos_x  pos_z  \
0  32.0  61.0   0.0      1      240.0        1   100  604.0  612.0
1  32.0  62.4   0.0      1      265.0        1   100  786.0  436.0
2  32.0  84.8   0.0      1      190.0        1   100  364.0  136.0
3  30.0  67.4   0.0      1      615.0        1   100  132.0  402.0
4  30.0  55.0  18.0      1      590.0        1   100  298.0  676.0

           q_name      q_cd        w_name      w_cd        e_name      e_cd  \
0    MordekaiserQ -4.541945  MordekaiserW -4.541945  MordekaiserE -4.541945
1          ViegoQ -4.541945        ViegoW -4.541945        ViegoE -4.541945
2  RivenTriCleave -4.541945   RivenMartyr -4.541945    RivenFeint -4.541945
3         EzrealQ -4.541945       EzrealW -4.541945       EzrealE -4.541945
4        LeblancQ -4.541945      LeblancW -4.541945      LeblancE -4.541945

                r_name      r_cd         d_name       d_cd            f_name  \
0         MordekaiserR -4.541945  SummonerFlash  10.458055       SummonerDot
1               ViegoR -4.541945  SummonerFlash  10.458055     SummonerSmite
```

```
2  RivenFengShuiEngine -4.541945  SummonerFlash  10.458055  SummonerTeleport
3              EzrealR -4.541945  SummonerFlash  10.458055      SummonerHeal
4             LeblancR -4.541945  SummonerFlash  10.458055       SummonerDot

        f_cd  normalized_time  normalized_name  normalized_hp  \
0  10.458055         0.030789               82         0.0645
1  10.458055         0.030789              234         0.0630
2  10.458055         0.030789               92         0.0745
3  10.458055         0.030789               81         0.0600
4  10.458055         0.030789                7         0.0598

   normalized_max_hp  normalized_mana  normalized_max_mana  normalized_armor  \
0             0.0645           0.0000               0.0100            0.0061
1             0.0630           1.0000               1.0000            0.0046
2             0.0745           0.0000               0.0000            0.0033
3             0.0600           0.0375               0.0375            0.0036
4             0.0598           0.0400               0.0400            0.0034

   normalized_mr  normalized_ad  normalized_ap  normalized_atk_range  \
0         0.0032        0.00610         0.0000              0.023383
1         0.0032        0.00624         0.0000              0.025818
2         0.0032        0.00848         0.0000              0.018511
3         0.0030        0.00674         0.0000              0.059918
4         0.0030        0.00550         0.0018              0.057482

   normalized_pos_x  normalized_pos_z  normalized_q_name  normalized_q_cd  \
0          0.040267          0.040800                  1        -0.009084
1          0.052400          0.029067                  2        -0.009084
2          0.024267          0.009067                  3        -0.009084
3          0.008800          0.026800                  4        -0.009084
4          0.019867          0.045067                  5        -0.009084

   normalized_w_name  normalized_w_cd  normalized_e_name  normalized_e_cd  \
0                  1        -0.009084                  1        -0.009084
1                  2        -0.009084                  2        -0.009084
2                  3        -0.009084                  3        -0.009084
3                  4        -0.009084                  4        -0.009084
4                  5        -0.009084                  5        -0.009084

   normalized_r_name  normalized_r_cd  normalized_d_name  normalized_d_cd  \
0                  1        -0.009084                  1         0.020916
1                  2        -0.009084                  1         0.020916
2                  3        -0.009084                  1         0.020916
3                  4        -0.009084                  1         0.020916
4                  5        -0.009084                  1         0.020916

   normalized_f_name  normalized_f_cd                compound_key    role
```

```
0                   1         0.020916  2841236401_100_Mordekaiser     Top
1                   2         0.020916        2841236401_100_Viego  Jungle
2                   3         0.020916        2841236401_100_Riven     Mid
3                   4         0.020916       2841236401_100_Ezreal     Bot
4                   1         0.020916      2841236401_100_Leblanc     Bot
```

2 Models

[5]:

```
import torch
import torch.nn as nn
import torch.optim as optim


def train_model(model, X_train, y_train, epochs=50, batch_size=64,␣
 ↪learning_rate=0.001, cutoff_loss=None):
    device = model.device
    model.to(device)
    criterion = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    X_train_tensor = torch.tensor(X_train, dtype=torch.float32).to(device)
    y_train_tensor = torch.tensor(y_train, dtype=torch.float32).to(device)

    dataset = torch.utils.data.TensorDataset(X_train_tensor, y_train_tensor)
    train_loader = torch.utils.data.DataLoader(
        dataset, batch_size=batch_size, shuffle=True)

    model.train()
    for epoch in range(epochs):
        pbar = tqdm(
            train_loader, desc=f'Epoch {epoch+1}/{epochs}', leave=False)
        for X_batch, y_batch in pbar:
            optimizer.zero_grad()
            output = model(X_batch)
            # Only use the first two feature dimensions for loss calculation
            loss = criterion(output[:, :2], y_batch[:, :2])
            loss.backward()
            optimizer.step()
            pbar.set_postfix({'Loss': loss.item()})
        current_loss = loss.item()
        if cutoff_loss is not None and current_loss < cutoff_loss:
            print(
                f'Loss is below cutoff value of {cutoff_loss}. Stopping␣
 ↪training.')
            break
        pbar.close()
```

```python
# Function to predict with the PyTorch model

def predict_model(model, X, batch_size=64, no_progress=True):
    device = model.device
    model.to(device)
    model.eval()
    X_tensor = torch.tensor(X, dtype=torch.float32).to(device)
    dataset = torch.utils.data.TensorDataset(X_tensor)
    loader = torch.utils.data.DataLoader(dataset, batch_size=batch_size)
    predictions = []
    pbar = tqdm(loader, desc='Predicting') if not no_progress else loader
    with torch.no_grad():
        for X_batch, in pbar:
            output = model(X_batch)
            predictions.append(output.cpu().numpy())
    return np.vstack(predictions)


class TrajectoryPredictor(nn.Module):
    def __init__(self, input_shape, output_shape, lstm_units=128, device='cpu',␣
 ↪parameters=None):
        super(TrajectoryPredictor, self).__init__()
        if parameters is not None:
            self.epochs = parameters['epochs']
            self.batch_size = parameters['batch_size']
            self.learning_rate = parameters['learning_rate']
            self.dropout_rate = parameters['dropout_rate']
        else:
            self.epochs = 10
            self.batch_size = 640
            self.learning_rate = 0.001
            self.dropout_rate = 0.2

        self.lstm1 = nn.LSTM(input_shape[-1], lstm_units, batch_first=True)
        self.dropout1 = nn.Dropout(self.dropout_rate)
        self.lstm2 = nn.LSTM(lstm_units, lstm_units, batch_first=True)
        self.dropout2 = nn.Dropout(self.dropout_rate)
        self.fc = nn.Linear(lstm_units, output_shape)
        self.device = device

    def forward(self, x):
        x, _ = self.lstm1(x)
        x = self.dropout1(x)
        x, _ = self.lstm2(x)
        x = self.dropout2(x)
        x = self.fc(x[:, -1, :])  # taking the output of the last time step
        return x

    def fit(self, X, y, cutoff_loss=None):
        train_model(self, X, y, self.epochs,
                    self.batch_size, self.learning_rate, cutoff_loss)

    def predict(self, X):
        return predict_model(self, X, self.batch_size)
```

2.1 Training

[6]:

```
# Calculate training and test data sizes
from utils.get_data import get_unique_key_count
def get_data_set_sizes(training_and_validation_set_fraction,␣
 ↪target_test_set_fraction, database_file, table_name):
    conn = sqlite3.connect(database_file)
    cursor = conn.cursor()
    unique_keys = get_unique_key_count(cursor, table_name)

    training_and_validation_set_size = int( unique_keys *␣
 ↪training_and_validation_set_fraction )
    testing_set_size = min(unique_keys - training_and_validation_set_size, int(␣
 ↪unique_keys * target_test_set_fraction ))

    return training_and_validation_set_size, testing_set_size

def calculate_fraction_size_of_all_keys(count, database_file, table_name):
    conn = sqlite3.connect(database_file)
    cursor = conn.cursor()
    unique_keys = get_unique_key_count(cursor, table_name)

    return count / unique_keys
```

[7]:

```
# Training Parameters

import pandas as pd

from utils.get_data import get_table_columns


device = 'cuda' if torch.cuda.is_available() else 'cpu'

print(f'Using {device} device')
```

```
conn = sqlite3.connect(database_file)
cursor = conn.cursor()
table_columns = [column[1] for column in get_table_columns(cursor, table_name)]
print(table_columns)

# all_features = [ column.value for column in DB_columns.__members__.values()␣
 ↪if "normalized" in column.value ]

data_features =  [ DB_columns.NORMALIZED_POS_X.value, DB_columns.
 ↪NORMALIZED_POS_Z.value ]

labels = [ DB_columns.NORMALIZED_POS_X.value, DB_columns.NORMALIZED_POS_Z.value␣
 ↪]


# Specify a float to fetch a given fraction, int to fetch a specific amount of␣
 ↪keys
total_amount_of_data_to_use = 100
target_training_set_fraction = 0.6
target_validation_set_fraction = 0.2
target_training_and_validation_set_fraction =␣
 ↪target_training_set_fraction+target_validation_set_fraction
target_test_set_fraction = 0.2

training_and_validation_set_fraction =␣
 ↪target_training_and_validation_set_fraction * (total_amount_of_data_to_use␣
 ↪if total_amount_of_data_to_use < 1.0 else min(1.0,␣
 ↪calculate_fraction_size_of_all_keys(total_amount_of_data_to_use,␣
 ↪database_file, table_name)))
testing_set_fraction = target_test_set_fraction * (total_amount_of_data_to_use␣
 ↪if total_amount_of_data_to_use < 1.0 else min(1.0,␣
 ↪calculate_fraction_size_of_all_keys(total_amount_of_data_to_use,␣
 ↪database_file, table_name)))

training_and_validation_set_size, testing_set_size =␣
 ↪get_data_set_sizes(training_and_validation_set_fraction,␣
 ↪testing_set_fraction, database_file, table_name)

H_values = [80]
T_values = [20]

# Display values to be used in a table
pd.DataFrame({
    'H': H_values,
    'T': T_values,
    'Training and Validation Set Size': training_and_validation_set_size,
    'Testing Set Size': testing_set_size,

})
```

```
Using cpu device
['game_id', 'time', 'name', 'hp', 'max_hp', 'mana', 'max_mana', 'armor', 'mr',
'ad', 'ap', 'level', 'atk_range', 'visible', 'team', 'pos_x', 'pos_z', 'q_name',
'q_cd', 'w_name', 'w_cd', 'e_name', 'e_cd', 'r_name', 'r_cd', 'd_name', 'd_cd',
'f_name', 'f_cd', 'compound_key', 'role']
Counting keys…

Key count: 100580
Using database cache for key count
Using in-memory cache for keys
```

[7]:

```
    H   T  Training and Validation Set Size  Testing Set Size
0  80  20                                80                20
```

[8]:

```python
linear_regression_features = [
    DB_columns.NORMALIZED_POS_X.value, DB_columns.NORMALIZED_POS_Z.value]

lstm_parameters = {'epochs': 10, 'batch_size': 256,
                   'learning_rate': 0.0005}
learning_rates = [0.0001, 0.001, 0.01]
batch_sizes = [64, 128, 256]
dropout_rates = [0.2, 0.4, 0.6]

lstm_parameter_sets = [
    {'epochs': 10,
        'batch_size': bs,
        'learning_rate': lr,
        'dropout_rate': dr
    } for bs in batch_sizes for lr in learning_rates for dr in dropout_rates
]

def get_lstm_name(params):
    return␣
  ↪f"lstm_lr_{params['learning_rate']}_bs_{params['batch_size']}_dr_{params['dropout_rate']}"

lstm_models = [ (get_lstm_name(params), data_features, params) for params in␣
  ↪lstm_parameter_sets ]
lstm_getters = dict(map(lambda x: (x[0], lambda H, T: (TrajectoryPredictor(
    input_shape=(H, len(x[1])),
    output_shape=2,
    device=device,
    parameters=x[2],
), x[1], (-1, H, len(x[1])))), lstm_models))

model_getters = {
    'linear_regression': lambda H, T: (LinearRegression(),␣
 ↪linear_regression_features, (-1, H*len(linear_regression_features))),
    **lstm_getters
}

# Display model getters and their values in a table
pd.DataFrame({
    'Model': [ (key, H, T) for key in model_getters.keys() for H in H_values␣
 ↪for T in T_values],
    'Features': [ len(x(H, T)[1]) for x in model_getters.values() for H in␣
 ↪H_values for T in T_values],
    'Shape': [ x(H, T)[2] for x in model_getters.values() for H in H_values for␣
 ↪T in T_values],
    'Parameters': [ x(H, T)[0].parameters if hasattr(x(H, T)[0], 'parameters')␣
 ↪else None for x in model_getters.values() for H in H_values for T in␣
 ↪T_values]
})
```

[8]:

```
                                     Model  Features         Shape  \
0                (linear_regression, 80, 20)         2     (-1, 160)
1     (lstm_lr_0.0001_bs_64_dr_0.2, 80, 20)         2  (-1, 80, 2)
2     (lstm_lr_0.0001_bs_64_dr_0.4, 80, 20)         2  (-1, 80, 2)
3     (lstm_lr_0.0001_bs_64_dr_0.6, 80, 20)         2  (-1, 80, 2)
4      (lstm_lr_0.001_bs_64_dr_0.2, 80, 20)         2  (-1, 80, 2)
5      (lstm_lr_0.001_bs_64_dr_0.4, 80, 20)         2  (-1, 80, 2)
6      (lstm_lr_0.001_bs_64_dr_0.6, 80, 20)         2  (-1, 80, 2)
7       (lstm_lr_0.01_bs_64_dr_0.2, 80, 20)         2  (-1, 80, 2)
8       (lstm_lr_0.01_bs_64_dr_0.4, 80, 20)         2  (-1, 80, 2)
9       (lstm_lr_0.01_bs_64_dr_0.6, 80, 20)         2  (-1, 80, 2)
10   (lstm_lr_0.0001_bs_128_dr_0.2, 80, 20)         2  (-1, 80, 2)
11   (lstm_lr_0.0001_bs_128_dr_0.4, 80, 20)         2  (-1, 80, 2)
12   (lstm_lr_0.0001_bs_128_dr_0.6, 80, 20)         2  (-1, 80, 2)
13    (lstm_lr_0.001_bs_128_dr_0.2, 80, 20)         2  (-1, 80, 2)
14    (lstm_lr_0.001_bs_128_dr_0.4, 80, 20)         2  (-1, 80, 2)
15    (lstm_lr_0.001_bs_128_dr_0.6, 80, 20)         2  (-1, 80, 2)
16     (lstm_lr_0.01_bs_128_dr_0.2, 80, 20)         2  (-1, 80, 2)
17     (lstm_lr_0.01_bs_128_dr_0.4, 80, 20)         2  (-1, 80, 2)
18     (lstm_lr_0.01_bs_128_dr_0.6, 80, 20)         2  (-1, 80, 2)
19   (lstm_lr_0.0001_bs_256_dr_0.2, 80, 20)         2  (-1, 80, 2)
20   (lstm_lr_0.0001_bs_256_dr_0.4, 80, 20)         2  (-1, 80, 2)
21   (lstm_lr_0.0001_bs_256_dr_0.6, 80, 20)         2  (-1, 80, 2)
22    (lstm_lr_0.001_bs_256_dr_0.2, 80, 20)         2  (-1, 80, 2)
23    (lstm_lr_0.001_bs_256_dr_0.4, 80, 20)         2  (-1, 80, 2)
24    (lstm_lr_0.001_bs_256_dr_0.6, 80, 20)         2  (-1, 80, 2)
```

```
25    (lstm_lr_0.01_bs_256_dr_0.2, 80, 20)        2  (-1, 80, 2)
26    (lstm_lr_0.01_bs_256_dr_0.4, 80, 20)        2  (-1, 80, 2)
27    (lstm_lr_0.01_bs_256_dr_0.6, 80, 20)        2  (-1, 80, 2)

                                           Parameters
0                                                None
1   <bound method Module.parameters of TrajectoryP…
2   <bound method Module.parameters of TrajectoryP…
3   <bound method Module.parameters of TrajectoryP…
4   <bound method Module.parameters of TrajectoryP…
5   <bound method Module.parameters of TrajectoryP…
6   <bound method Module.parameters of TrajectoryP…
7   <bound method Module.parameters of TrajectoryP…
8   <bound method Module.parameters of TrajectoryP…
9   <bound method Module.parameters of TrajectoryP…
10  <bound method Module.parameters of TrajectoryP…
11  <bound method Module.parameters of TrajectoryP…
12  <bound method Module.parameters of TrajectoryP…
13  <bound method Module.parameters of TrajectoryP…
14  <bound method Module.parameters of TrajectoryP…
15  <bound method Module.parameters of TrajectoryP…
16  <bound method Module.parameters of TrajectoryP…
17  <bound method Module.parameters of TrajectoryP…
18  <bound method Module.parameters of TrajectoryP…
19  <bound method Module.parameters of TrajectoryP…
20  <bound method Module.parameters of TrajectoryP…
21  <bound method Module.parameters of TrajectoryP…
22  <bound method Module.parameters of TrajectoryP…
23  <bound method Module.parameters of TrajectoryP…
24  <bound method Module.parameters of TrajectoryP…
25  <bound method Module.parameters of TrajectoryP…
26  <bound method Module.parameters of TrajectoryP…
27  <bound method Module.parameters of TrajectoryP…
```

[9]:

```
if TRAINING:
    trained_models, training_errors, validation_errors = compare_models(
        database_file, table_name, H_values, T_values, model_getters,␣
  ↪data_features=data_features, labels=labels,␣
  ↪total_keys_to_fetch=training_and_validation_set_size,␣
  ↪batch_size=training_and_validation_set_size, train=True)

    print(training_errors)
```

[10]:

```
# Print rmse results
if TRAINING:
    print("Training Error (MSE)")
    for model_name, mse in training_errors.items():
        print(f"{model_name}: {mse}")
    print("Validation Error (MSE)")
    for model_name, mse in validation_errors.items():
        print(f"{model_name}: {mse}")
```

[11]:

```
if TRAINING:
    # Generate test error by predicting on unseen data (offset with the␣
 ↪training data amount)
    conn = sqlite3.connect(database_file)
    cursor = conn.cursor()
    data = fetch_data_batches(cursor, table_name, "1=1", 1, round(0.
 ↪1*training_and_validation_set_size), data_features)
    conn.close()

    test_errors = {}
    for model_name, model in trained_models.items():
        H, T, model_type_name = model_name
        model_getter = model_getters[model_type_name](H, T)
        model_instance, features, input_shape = model_getter
        X_test, y_test = create_sequences_from_database_rows(
            data, H, T, H, T)
        X_test_reshaped = shape_input_for_model(X_test, data_features,␣
 ↪features, input_shape)
        y_pred = model.predict(X_test_reshaped)
        # Use L2 distance for error calculation
        test_errors[model_name] = np.linalg.norm(
            y_test - y_pred, axis=1)

    print("Test Error (L2 distance)")
    for model_name, test_error in test_errors.items():
        print(f"{model_name}: {np.mean(test_error)}")
```

[12]:

```
import json
import datetime

# Save models
folder = 'models'
if TRAINING:
    for model_name, model in trained_models.items():
        file_name = f'{"_".join([str(part) for part in model_name])}.pt'
        file_name = os.path.join(folder, file_name)
        torch.save(model, file_name)
        # Save training information into a separate file
        H, T, model_type_name = model_name
        model_getter = model_getters[model_type_name](H, T)
        model_instance, features, input_shape = model_getter
        training_info = {
            'model_name': model_name,
            'model_type': model_type_name,
            'model_instance': str(model_instance),
            'H': H,
            'T': T,
            'features': features,
            'input_shape': input_shape,
            'training_error': [float(error) for error in␣
 ↪training_errors[model_name]],
            'validation_error': [float(error) for error in␣
 ↪validation_errors[model_name]],
            'test_error': [float(error) for error in test_errors[model_name]],
            'training_size': training_and_validation_set_size,
            'training_date': datetime.datetime.now().isoformat(),
        }
        training_info_file_name = file_name.replace('.pt', '.json')
        with open(training_info_file_name, 'w') as f:
            json.dump(training_info, f)
```

[13]:

```
# Load models
folder = 'models'
if not TRAINING:
    trained_models = {}
    training_errors = {}
    validation_errors = {}
    test_errors = {}
    model_names = model_getters.keys()
    model_file_names = [ f'{H}_{T}_{model_name}.pt' for H in H_values for T in␣
 ↪T_values for model_name in model_names]
    for file_name in model_file_names:
        model_name_parts = file_name.split('.')
        model_name_parts = ".".join(model_name_parts[:-1]).split('_') if␣
 ↪len(model_name_parts) > 2 else model_name_parts[0].split('_')
        model_name = (int(model_name_parts[0]), int(model_name_parts[1]), '_'.
 ↪join(model_name_parts[2:]))
        print(model_name)
        trained_models[model_name] = torch.load(os.path.join(folder,␣
 ↪file_name), map_location=torch.device(device))
        trained_models[model_name].device = device
        try:
            training_info_file_name = file_name.replace('.pt', '.json')
            with open(os.path.join(folder, training_info_file_name), 'r') as f:
                training_info = json.load(f)
            training_errors[model_name] = training_info['training_error']
            validation_errors[model_name] = training_info['validation_error']
            test_errors[model_name] = training_info['test_error']
        except FileNotFoundError:
            training_errors[model_name] = None
            validation_errors[model_name] = None
            print(f'Error loading training information for␣
 ↪{training_info_file_name}')
        except e:
            print(f"Uncatched error: {e}")

# trained_models, training_errors, validation_errors
```

```
(80, 20, 'linear_regression')
(80, 20, 'lstm_lr_0.0001_bs_64_dr_0.2')
```

```
/u/23/tarpill1/unix/.local/lib/python3.8/site-packages/sklearn/base.py:329:
UserWarning: Trying to unpickle estimator LinearRegression from version 1.4.2
when using version 1.1.2. This might lead to breaking code or invalid results.
Use at your own risk. For more info please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-
limitations
  warnings.warn(
```

```
(80, 20, 'lstm_lr_0.0001_bs_64_dr_0.4')
(80, 20, 'lstm_lr_0.0001_bs_64_dr_0.6')
```

```
(80, 20, 'lstm_lr_0.001_bs_64_dr_0.2')
(80, 20, 'lstm_lr_0.001_bs_64_dr_0.4')
(80, 20, 'lstm_lr_0.001_bs_64_dr_0.6')
(80, 20, 'lstm_lr_0.01_bs_64_dr_0.2')
(80, 20, 'lstm_lr_0.01_bs_64_dr_0.4')
(80, 20, 'lstm_lr_0.01_bs_64_dr_0.6')
(80, 20, 'lstm_lr_0.0001_bs_128_dr_0.2')
(80, 20, 'lstm_lr_0.0001_bs_128_dr_0.4')
(80, 20, 'lstm_lr_0.0001_bs_128_dr_0.6')
(80, 20, 'lstm_lr_0.001_bs_128_dr_0.2')
(80, 20, 'lstm_lr_0.001_bs_128_dr_0.4')
(80, 20, 'lstm_lr_0.001_bs_128_dr_0.6')
(80, 20, 'lstm_lr_0.01_bs_128_dr_0.2')
(80, 20, 'lstm_lr_0.01_bs_128_dr_0.4')
(80, 20, 'lstm_lr_0.01_bs_128_dr_0.6')
(80, 20, 'lstm_lr_0.0001_bs_256_dr_0.2')
(80, 20, 'lstm_lr_0.0001_bs_256_dr_0.4')
(80, 20, 'lstm_lr_0.0001_bs_256_dr_0.6')
(80, 20, 'lstm_lr_0.001_bs_256_dr_0.2')
(80, 20, 'lstm_lr_0.001_bs_256_dr_0.4')
(80, 20, 'lstm_lr_0.001_bs_256_dr_0.6')
(80, 20, 'lstm_lr_0.01_bs_256_dr_0.2')
(80, 20, 'lstm_lr_0.01_bs_256_dr_0.4')
(80, 20, 'lstm_lr_0.01_bs_256_dr_0.6')
```

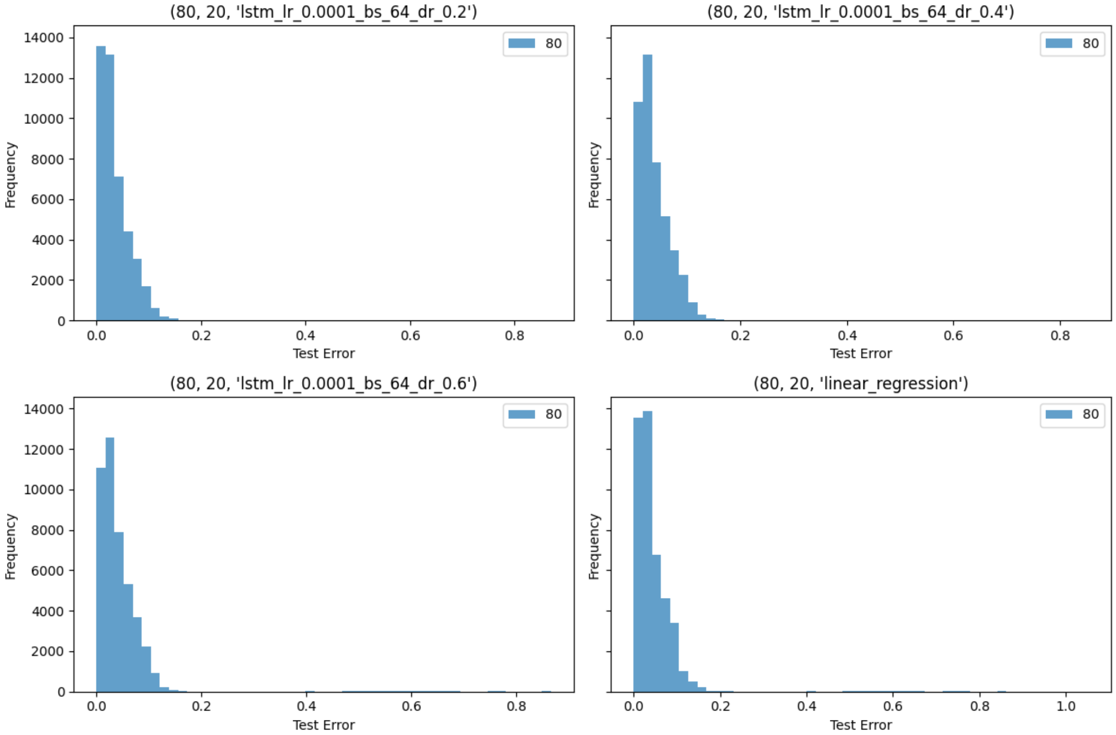
[14]:

```
# Plot the test error distribution for each model
for model_name, test_error in test_errors.items():
        print(f"{model_name}: {np.mean(test_error)}")
```

```
(80, 20, 'linear_regression'): 0.04674459874519098
(80, 20, 'lstm_lr_0.0001_bs_64_dr_0.2'): 0.04051125417176376
(80, 20, 'lstm_lr_0.0001_bs_64_dr_0.4'): 0.043687712039264866
(80, 20, 'lstm_lr_0.0001_bs_64_dr_0.6'): 0.0444811922559739
(80, 20, 'lstm_lr_0.001_bs_64_dr_0.2'): 0.04506473910335845
(80, 20, 'lstm_lr_0.001_bs_64_dr_0.4'): 0.04016022727556758
(80, 20, 'lstm_lr_0.001_bs_64_dr_0.6'): 0.04306400144749778
(80, 20, 'lstm_lr_0.01_bs_64_dr_0.2'): 0.08747022366791957
(80, 20, 'lstm_lr_0.01_bs_64_dr_0.4'): 0.3078988100469179
(80, 20, 'lstm_lr_0.01_bs_64_dr_0.6'): 0.3073832372448833
(80, 20, 'lstm_lr_0.0001_bs_128_dr_0.2'): 0.04169421252565093
(80, 20, 'lstm_lr_0.0001_bs_128_dr_0.4'): 0.04221150204215978
(80, 20, 'lstm_lr_0.0001_bs_128_dr_0.6'): 0.043795009725085085
(80, 20, 'lstm_lr_0.001_bs_128_dr_0.2'): 0.03999634306199614
(80, 20, 'lstm_lr_0.001_bs_128_dr_0.4'): 0.04108552916883758
(80, 20, 'lstm_lr_0.001_bs_128_dr_0.6'): 0.04311481982587056
(80, 20, 'lstm_lr_0.01_bs_128_dr_0.2'): 0.21376861616090226
(80, 20, 'lstm_lr_0.01_bs_128_dr_0.4'): 0.2373791350520932
(80, 20, 'lstm_lr_0.01_bs_128_dr_0.6'): 0.3092209167099088
(80, 20, 'lstm_lr_0.0001_bs_256_dr_0.2'): 0.042449087085216955
(80, 20, 'lstm_lr_0.0001_bs_256_dr_0.4'): 0.04273869553394
(80, 20, 'lstm_lr_0.0001_bs_256_dr_0.6'): 0.044478978858552745
(80, 20, 'lstm_lr_0.001_bs_256_dr_0.2'): 0.03964649651667043
(80, 20, 'lstm_lr_0.001_bs_256_dr_0.4'): 0.0414429749581
(80, 20, 'lstm_lr_0.001_bs_256_dr_0.6'): 0.04243898708160208
(80, 20, 'lstm_lr_0.01_bs_256_dr_0.2'): 0.06994693865932186
(80, 20, 'lstm_lr_0.01_bs_256_dr_0.4'): 0.44197044483774034
(80, 20, 'lstm_lr_0.01_bs_256_dr_0.6'): 0.46477672985011315
```

[15]:

```
# Plot the test error distribution for Linear Regression and three best LSTMs␣
  ↪in a compact subplot
fig, axes = plt.subplots(2, 2, figsize=(12, 8), sharey=True)
axes = axes.flatten()

lstm_test_errors = [e for e in test_errors.items() if "lstm" in e[0][2]]
best_three_lstm_errors = lstm_test_errors[:3]
linear_regression_test_error = [e for e in test_errors.items() if␣
  ↪"linear_regression" in e[0]][0]
chosen_test_errors = best_three_lstm_errors + [linear_regression_test_error]

for ax, (model_name, test_error) in zip(axes, chosen_test_errors):
    ax.hist(test_error, bins=50, label=model_name, alpha=0.7)
    ax.legend()
    ax.set_title(model_name)
    ax.set_xlabel('Test Error')
    ax.set_ylabel('Frequency')

plt.tight_layout()
plt.savefig("best_distributions.png")
plt.show()
```


(80, 20, 'lstm_lr_0.0001_bs_64_dr_0.2')

(80, 20, 'lstm_lr_0.0001_bs_64_dr_0.4')

(80, 20, 'lstm_lr_0.0001_bs_64_dr_0.6')

(80, 20, 'linear_regression')

[18]:

```
import sqlite3
import numpy as np

def connect_to_database(database_file):
    return sqlite3.connect(database_file)

def fetch_data(cursor, table_name, query, offset, limit, features):
    return fetch_data_batches(cursor, table_name, query, offset, limit,␣
  ↪features)

def get_model_input(model_name, H, T):
    return model_getters[model_name](H, T)

def create_test_features(data, features, data_features, input_shape):
    X_test_features = data[:, :, [data_features.index(feature) for feature in features]]
    return X_test_features.reshape(X_test_features.shape[0], *input_shape)

def predict_sequences(model, X_test_features):
    return np.array([model.predict(X_test_features[i]) for i in range(X_test_features.shape[0])], dtype=np.float32)

def prepare_plotting_features(data, data_features, plotting_features, H):
    plotting_input_shape = [H, len(plotting_features)]
    X_test_plotting_features = data[:, :, [data_features.index(feature) for feature in plotting_features]]
    return X_test_plotting_features.reshape(-1, *plotting_input_shape)

def reshape_predictions(predictions, shape, dims):
    return predictions.reshape(-1, shape[-1])[:, dims]

# Prediction part
conn = connect_to_database(database_file)
cursor = conn.cursor()

clear_cache(cursor)

plotting_features = [DB_columns.NORMALIZED_POS_X.value, DB_columns.NORMALIZED_POS_Z.value, DB_columns.NORMALIZED_NAME.value]
additional_features = [DB_columns.TIME.value, DB_columns.HP.value, DB_columns.NORMALIZED_NAME.value]

fetched_features = list(np.unique(data_features + plotting_features + additional_features))
fetched_features.sort(key=lambda feature: data_features.index(feature) if feature in data_features else len(data_features))

data = fetch_data(cursor, table_name, "1=1", training_and_validation_set_size, testing_set_size, fetched_features)

max_H = max(H_values)
max_T = max(T_values)

if CREATE_ANIMATIONS:


    animation_options = {
        "speed": 500
    }
    for H in H_values:
        for T in T_values:
            for model_name in model_getters.keys():
                X, y = create_sequences_from_database_rows(data, H, T, max_H,␣
↪max_T)
                ground_truths = y.reshape(-1, y.shape[-1])[:, :2]
                input_shape = get_model_input(model_name, H, T)[2]
                features = get_model_input(model_name, H, T)[1]
                fetched_features_to_model_features = [fetched_features.
↪index(feature) for feature in features]

                X_test_features = create_test_features(X, features,␣
↪fetched_features, input_shape)
                y_pred = predict_sequences(trained_models[(H, T, model_name)],␣
↪X_test_features)
                predictions = reshape_predictions(y_pred, y_pred.shape,␣
↪slice(0, 2))

                additional_data = X[:, -1, [fetched_features.index(feature) for␣
↪feature in additional_features]]
                additional_data_strings = [[f"{additional_features[i]}:␣
↪{additional_data[j, i]:.2f}" for i in range(len(additional_features))] for j␣
↪in range(additional_data.shape[0])]

                plotting_options = [{
                    "padding": 0.1,
                    "truthPointsSize": 10,
                    "predictionPointsSize": 10,
                    "title": f"Model {model_name} - Predictions \n(H={H},␣
↪T={T}, {', '.join(additional_data_strings[i])})",
                } for i in range(X.shape[0])]

                X_plotting_features = prepare_plotting_features(X,␣
↪fetched_features, plotting_features, H)

                ani = create_prediction_animation(X_plotting_features,␣
↪predictions, ground_truths, "assets/2x_2dlevelminimap.png",␣
↪normalized_zoom_range, plotting_options, animation_options)
                display(ani)

    conn.close()
```

```
Counting rows…
Counts: [(656, '1458747983_100_Ezreal'), (656, '1458747983_100_Graves'), (656,
'1458747983_100_KSante'), (656, '1458747983_100_Lissandra'), (656,
'1458747983_100_Sylas'), (656, '1458747983_200_Brand'), (656,
'1458747983_200_Kayle'), (656, '1458747983_200_Nilah'), (656,
'1458747983_200_Orianna'), (656, '1458747983_200_Skarner'), (656,
'1458766628_100_Akshan'), (656, '1458766628_100_Ezreal'), (656,
'1458766628_100_Shaco'), (656, '1458766628_100_Swain'), (656,
'1458766628_100_XinZhao'), (656, '1458766628_200_Anivia'), (656,
'1458766628_200_Diana'), (656, '1458766628_200_Janna'), (656,
'1458766628_200_Jax'), (656, '1458766628_200_Kaisa'), (656,
'1458767921_100_Aatrox'), (656, '1458767921_100_Ezreal'), (656,
'1458767921_100_Maokai'), (656, '1458767921_100_Nunu'), (656,
'1458767921_100_Veigar'), (656, '1458767921_200_Jinx'), (656,
'1458767921_200_Kassadin'), (656, '1458767921_200_LeeSin'), (656,
'1458767921_200_Thresh'), (656, '1458767921_200_Yone'), (656,
'1458770016_100_Ezreal'), (656, '1458770016_100_Jayce'), (656,
'1458770016_100_Karma'), (656, '1458770016_100_Kayn'), (656,
'1458770016_100_Orianna'), (656, '1458770016_200_Briar'), (656,
'1458770016_200_Darius'), (656, '1458770016_200_Kaisa'), (656,
'1458770016_200_Malzahar'), (656, '1458770016_200_Renata'), (656,
'1458782453_100_Amumu'), (656, '1458782453_100_Ezreal'), (656,
'1458782453_100_Jayce'), (656, '1458782453_100_Maokai'), (656,
'1458782453_100_Neeko'), (656, '1458782453_200_Ahri'), (656,
'1458782453_200_Belveth'), (656, '1458782453_200_Milio'), (656,
'1458782453_200_Ornn'), (656, '1458782453_200_Vayne'), (657,
'1458785011_100_Aatrox'), (657, '1458785011_100_Bard'), (657,
'1458785011_100_Ezreal'), (657, '1458785011_100_JarvanIV'), (657,
'1458785011_100_Zoe'), (657, '1458785011_200_Akali'), (657,
'1458785011_200_Amumu'), (657, '1458785011_200_Caitlyn'), (657,
'1458785011_200_Gangplank'), (657, '1458785011_200_Senna'), (655,
'1458816665_100_Aatrox'), (655, '1458816665_100_Lulu'), (655,
'1458816665_100_Nidalee'), (655, '1458816665_100_TwistedFate'), (655,
'1458816665_100_Xayah'), (655, '1458816665_200_Ezreal'), (655,
'1458816665_200_Gwen'), (655, '1458816665_200_Jayce'), (655,
'1458816665_200_Nautilus'), (655, '1458816665_200_Yone'), (655,
'1458823265_100_Ezreal'), (655, '1458823265_100_Irelia'), (655,
'1458823265_100_Nocturne'), (655, '1458823265_100_Sylas'), (655,
'1458823265_100_Thresh'), (655, '1458823265_200_LeeSin'), (655,
'1458823265_200_Pyke'), (655, '1458823265_200_Syndra'), (655,
'1458823265_200_Vayne'), (655, '1458823265_200_Yone'), (657,
'1458842893_100_Ezreal'), (657, '1458842893_100_Lux'), (657,
'1458842893_100_Senna'), (657, '1458842893_100_Sett'), (657,
'1458842893_100_Taliyah'), (657, '1458842893_200_Rell'), (657,
'1458842893_200_Varus'), (657, '1458842893_200_Xerath'), (657,
'1458842893_200_Yone'), (657, '1458842893_200_Zac'), (656,
'1458844288_100_Ahri'), (656, '1458844288_100_Belveth'), (656,
'1458844288_100_Ezreal'), (656, '1458844288_100_KSante'), (656,
'1458844288_100_Singed'), (656, '1458844288_200_Fiora'), (656,
'1458844288_200_Kaisa'), (656, '1458844288_200_Karma'), (656,
'1458844288_200_Rammus'), (656, '1458844288_200_Xerath')]
Fetched 20 keys for offset: 80, limit: 20
```

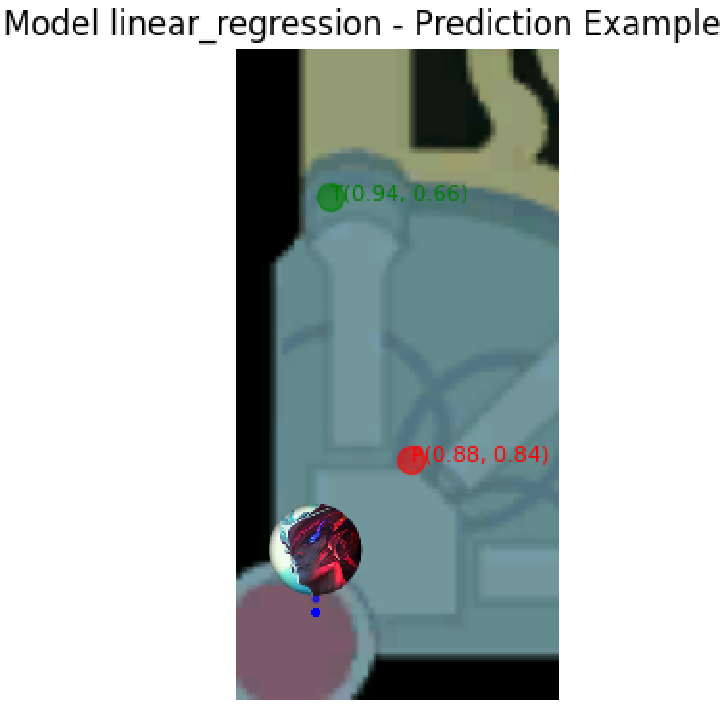
[27]:

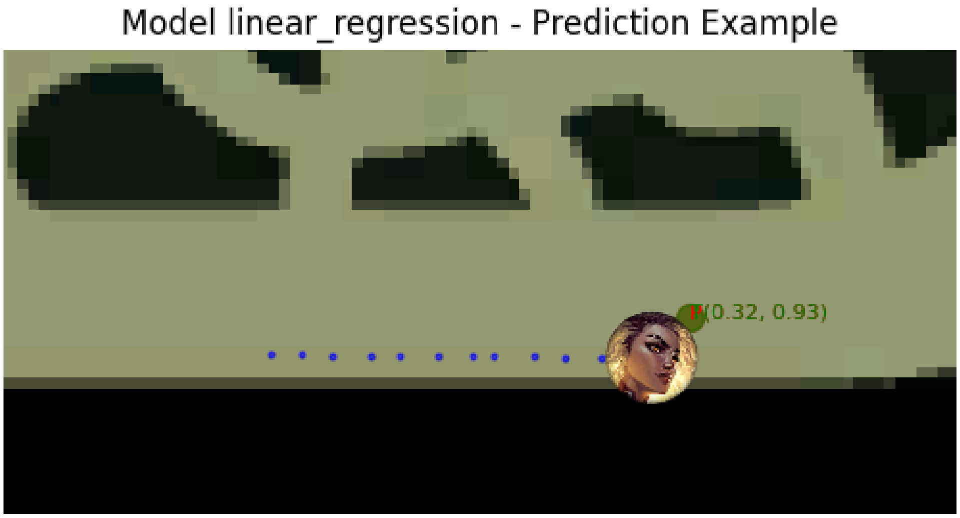
```
# Output the best and worst predictions
if CREATE_VISUALIZATIONS:
    for (H, T, model_name), model in list(trained_models.items())[:2]:
        input_shape = model_getters[model_name](H, T)[2]
        features = model_getters[model_name](H, T)[1]
        print(f"Predicting with model {model_name}")
        sequences = create_sequences_from_database_rows(data, H, T, max_H,␣
 ↪max_T)
        X, y = sequences
        X_test_features = X[:, :, [
            data_features.index(feature) for feature in features]]
        X_test_features = X_test_features.reshape(
            X_test_features.shape[0], *input_shape)
        # Run the prediction on all the sequences
        y_pred = [ model.predict(X_test_features[i]) for i in␣
 ↪range(X_test_features.shape[0]) ]
        y_pred = np.array(y_pred, dtype=np.float32).reshape(-1, 2)
        # Visualize the best and worst predictions
        absolute_errors = np.linalg.norm(y - y_pred, axis=1)

        number_of_best_sequences = 5

        best_sequence_indices = np.argpartition(absolute_errors,␣
 ↪number_of_best_sequences)[:number_of_best_sequences]
        worst_sequence_indices = np.argpartition(absolute_errors,␣
 ↪-number_of_best_sequences)[-number_of_best_sequences:]
        print(f"Best sequence index: {best_sequence_indices}")
        print(f"Worst sequence index: {worst_sequence_indices}")
        print(f"Lowest errors: {absolute_errors[best_sequence_indices]}")
        print(f"Highest errors: {absolute_errors[worst_sequence_indices]}")

        plotting_features = [DB_columns.NORMALIZED_POS_X.value, DB_columns.
 ↪NORMALIZED_POS_Z.value, DB_columns.NORMALIZED_NAME.value]

        X_test_plotting_features = prepare_plotting_features(X,␣
 ↪fetched_features, plotting_features, H)
        best_sequences = X_test_plotting_features[best_sequence_indices]
        worst_sequence = X_test_plotting_features[worst_sequence_indices]



        best_predictions = y_pred[best_sequence_indices].reshape(-1, y_pred.
 ↪shape[-1])[: , :2]
        worst_predictions = y_pred[worst_sequence_indices].reshape(-1, y_pred.
 ↪shape[-1])[ : , :2]
```

```
        best_truths = y[best_sequence_indices].reshape(-1, len(y[0]))[: , :2]
        worst_truths = y[worst_sequence_indices].reshape(-1, len(y[0]))[ : , :2]

        # Plot the worst prediction
        plotting_options = {
            "padding": 0.1,
            "truthPointsSize": 10,
            "predictionPointsSize": 10,
            "pointsSize": 5,
            "title": f"Model {model_name} - Prediction Example"
        }

        plot_prediction_on_plot(plt, worst_sequence[0:1], worst_predictions[0:
↪1], worst_truths[0:1], "assets/2x_2dlevelminimap.png",␣
↪normalized_zoom_range, plotting_options)

        # Save the worst prediction

        folder = f"output/{model_name}"
        os.makedirs(folder, exist_ok=True)
        file_name = f"{folder}/worst_prediction.png"
        plt.savefig(file_name)

        # Plot the best prediction

        plotting_options = {
            "padding": 0.1,
            "truthPointsSize": 10,
            "predictionPointsSize": 10,
            "title": f"Model {model_name} - Prediction Example"
        }

        plot_prediction_on_plot(plt, best_sequences[0:1], best_predictions[0:
↪1], best_truths[0:1], "assets/2x_2dlevelminimap.png", normalized_zoom_range,␣
↪plotting_options)

        # Save the best prediction

        folder = f"output/{model_name}"
        os.makedirs(folder, exist_ok=True)
        file_name = f"{folder}/best_prediction.png"
        plt.savefig(file_name)

        random_index = np.random.randint(len(X))
        chosen_sequence = X_test_plotting_features[random_index, ]
        chosen_prediction = y_pred[random_index].reshape(-1, y_pred.shape[-1])[:
↪ , :2]
```

```
        chosen_truth = y[random_index].reshape(-1, len(y[0]))[: , :2]


        plot_prediction_on_plot(plt, [chosen_sequence], [chosen_prediction], [chosen_truth], "assets/2x_2dlevelminimap.png", normalized_zoom_range, plotting_options)
```

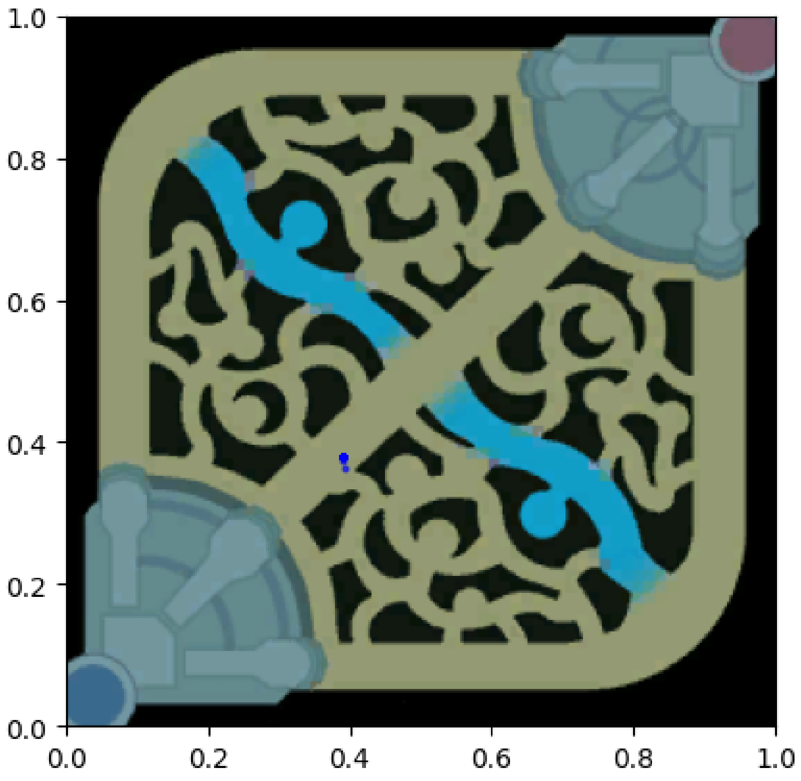
```
Predicting with model linear_regression
Best sequence index: [ 2946  5599  5601 10639  5600]
Worst sequence index: [ 4462  1184  4463 10007 10006]
Lowest errors: [0.00023185 0.00030266 0.00036314 0.00043836 0.0004564 ]
Highest errors: [0.18666112 0.19120559 0.19825432 0.8525179  0.852944  ]
```





```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In [27], line 84
     80 chosen_prediction = y_pred[random_index].reshape(-1, y_pred.shape[-1])[ ␣
  ↪, :2]
     81 chosen_truth = y[random_index].reshape(-1, len(y[0]))[: , :2]
---> 84␣
  ↪plot_prediction_on_plot(plt, [chosen_sequence], [chosen_prediction], [chosen_truth], "asse

File /m/home/home2/23/tarpill1/data/Documents/masters-thesis/utils/
  ↪plot_prediction_on_plot.py:67, in plot_prediction_on_plot(plot, points,␣
  ↪prediction, truth, map_image_path, zoom_range, options)
     64 for player_sequence in points:
     65     plot_positions(player_sequence[:, :2], input_points_size,␣
  ↪input_points_color)
---> 67␣
  ↪plot_positions(prediction, prediction_points_size, prediction_points_color, 'P')
     68 plot_positions(truth, truth_points_size, truth_points_color, 'T')
     70 denormalization_data_path = "denormalization_data.json"

File /m/home/home2/23/tarpill1/data/Documents/masters-thesis/utils/
  ↪plot_prediction_on_plot.py:59, in plot_prediction_on_plot.<locals>.
  ↪plot_positions(positions, size, color, label)
     57 def plot_positions(positions, size, color, label=False):
     58     for player in positions:
---> 59         plot.plot(player[1], player[0], markersize=size, alpha=0.6,␣
  ↪color=color, marker='o')
```

```
     60     if label:
     61         plot.text(positions[-1][1], positions[-1][0],␣
 ↪f'{label}({positions[-1][1]:.2f}, {positions[-1][0]:.2f})',
     62                       fontsize=8, color=color)

IndexError: index 1 is out of bounds for axis 0 with size 1
```



[ ]:

```
[ ]: # # Plot the mse results
     # from matplotlib import pyplot as plt

     # for H in H_values:
     #     for T in T_values:
     #         model_names = model_getters.keys()
     #         mse_values = [mse_results[(H, T, model_name)] for model_name in␣
      ↪model_names]
     #         plt.plot(model_names, mse_values, label=f'H={H}, T={T}, {model_name}')
     # plt.legend()

     # # Save plot
# plt.savefig("output/mses.png")
```

[ ]:

```
import sys
import platform
import os
import subprocess
import json
import psutil

# Function to get GPU details if available
def get_gpu_info():
    try:
        import torch
        if torch.cuda.is_available():
            return torch.cuda.get_device_name(0)
        else:
            return "No GPU available"
    except ImportError:
        return "PyTorch not installed"

# Get Python version
python_version = sys.version

# Get system platform
system_platform = platform.platform()

# Get installed packages
installed_packages = subprocess.check_output([sys.executable, '-m', 'pip',␣
 ↪'freeze']).decode('utf-8')

# Get environment variables
environment_variables = {k: v for k, v in os.environ.items()}

# Get GPU info
gpu_info = get_gpu_info()

# Get available RAM
available_ram = psutil.virtual_memory().available / (1024 ** 3)  # Convert␣
 ↪bytes to GB

# Collect all information in a dictionary
env_info = {
    "python_version": python_version,
    "system_platform": system_platform,
    "gpu_info": gpu_info,
    "available_ram": f"{available_ram:.2f} GB"
}
```

```python
# Print environment information
print(json.dumps(env_info, indent=4))
```

```
WARNING: Ignoring invalid distribution -vidia-cuda-runtime-cu12
(/m/home/home2/23/tarpill1/unix/.local/lib/python3.10/site-packages)
```

```
{
    "python_version": "3.10.12 (main, Mar 22 2024, 16:50:05) [GCC 11.4.0]",
    "system_platform": "Linux-5.15.0-112-generic-x86_64-with-glibc2.35",
    "gpu_info": "NVIDIA GeForce RTX 3080",
    "available_ram": "17.33 GB"
}
```

data_features

July 30, 2024

```
[2]: import pandas as pd
     import sqlite3

     import os
     from dotenv import load_dotenv
     from utils.get_or_create_combined_database import␣
       ↪get_or_create_combined_database
     load_dotenv(verbose=True, override=True)

     database_folder = os.getenv("DATABASE_FOLDER")

     database_file = get_or_create_combined_database(database_folder)

     table_name = "champs_cleaned"

     conn = sqlite3.connect(database_file)
     query = 'SELECT * FROM champs_cleaned'
     data = pd.read_sql_query(query, conn)
     conn.close()

     # Display the first few rows of the dataframe
     data.head()
```

```
Found 101 database files in the folder specified by DATABASE_FOLDER
Found combined database D:\league-ezreal-dataset\ml_project\combined2.db
```

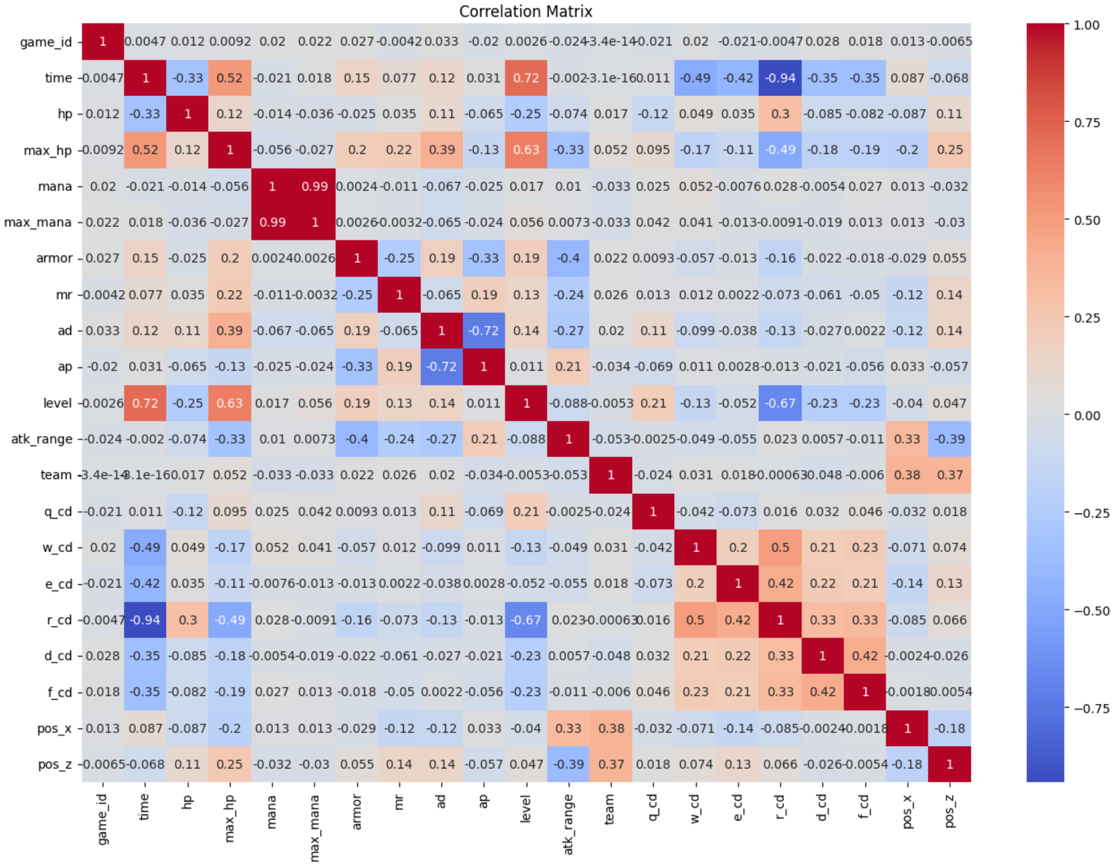
```
[2]:       game_id      time    name     hp  max_hp   mana  max_mana  armor    mr  \
     0  4848459903  5.028642  KSante  570.0   570.0  290.0     290.0   57.0  30.0
     1  4848459903  5.028642    Ekko  655.0   655.0  280.0     280.0   44.0  32.0
     2  4848459903  5.028642   Swain  610.0   610.0  468.0     468.0   26.0  46.0
     3  4848459903  5.028642  Ezreal  600.0   600.0  375.0     375.0   36.0  30.0
     4  4848459903  5.028642  Rumble  650.0   650.0    0.0     150.0   48.0  28.0

          ad  …         d_name       d_cd            f_name       f_cd  \
     0  64.0  …  SummonerFlash  10.971358  SummonerTeleport  10.971358
     1  58.0  …  SummonerFlash  10.971358      SummonerSmite  10.971358
     2  58.0  …  SummonerFlash  10.971358      SummonerHaste  10.971358
     3  67.4  …  SummonerHaste  10.971358      SummonerFlash  10.971358
```

```
4  61.0  …   SummonerDot  10.971358    SummonerFlash  10.971358

   normalized_pos_x  normalized_pos_z  normalized_time normalized_hp  \
0          0.040267          0.040800         0.002794         0.114
1          0.044267          0.019067         0.002794         0.131
2          0.024267          0.009067         0.002794         0.122
3          0.008800          0.026800         0.002794         0.120
4          0.019867          0.045067         0.002794         0.130

   normalized_name           compound_key
0              897  4848459903_100_KSante
1              245    4848459903_100_Ekko
2               50   4848459903_100_Swain
3               81  4848459903_100_Ezreal
4               68  4848459903_100_Rumble

[5 rows x 35 columns]
```

1 Correlation Matrix

[3]:

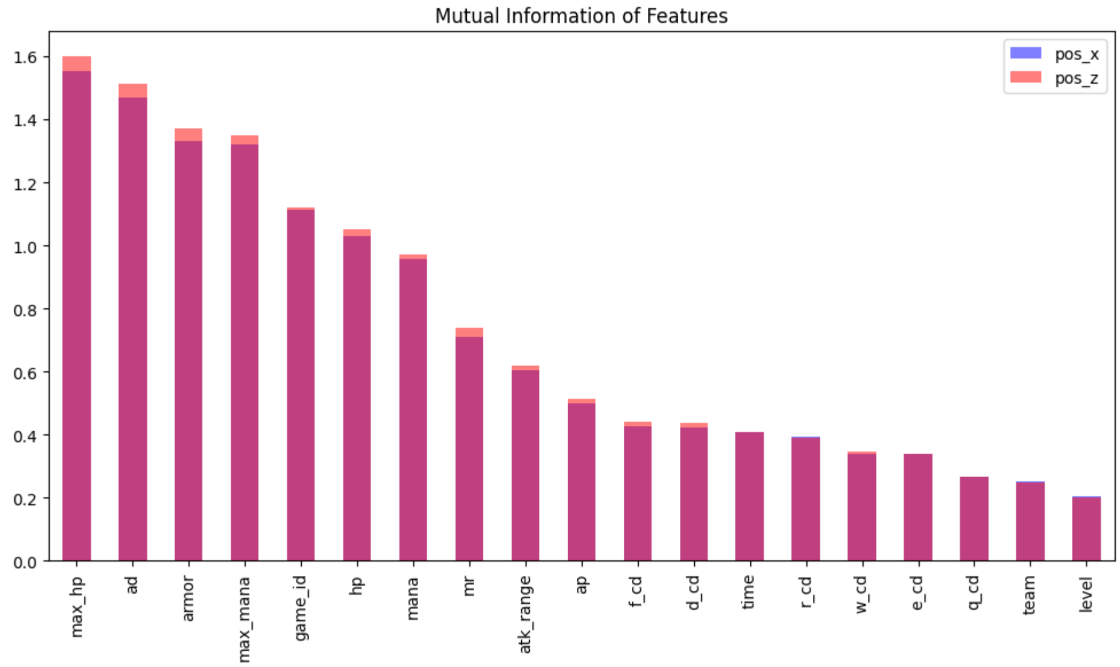
```
import matplotlib.pyplot as plt
import seaborn as sns

# Define the target variables and features
target = ['pos_x', 'pos_z']
features = [col for col in data.columns if col not in target and "normalized"␣
 ↪not in col and col not in [
    'name', 'q_name', 'w_name', 'e_name', 'r_name', 'd_name', 'f_name',␣
 ↪'compound_key', 'visible', ]]

# Correlation matrix
corr_matrix = data[features + target].corr()
plt.figure(figsize=(16, 11))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



2 Mutual Information

[4]:
```
from sklearn.feature_selection import mutual_info_regression

# Mutual Information for pos_x
X = data[features]
y_pos_x = data['pos_x']
mi_pos_x = mutual_info_regression(X, y_pos_x)
mi_pos_x_series = pd.Series(mi_pos_x, index=features)

# Mutual Information for pos_z
y_pos_z = data['pos_z']
mi_pos_z = mutual_info_regression(X, y_pos_z)
mi_pos_z_series = pd.Series(mi_pos_z, index=features)

# Plot Mutual Information
plt.figure(figsize=(12, 6))
mi_pos_x_series.sort_values(ascending=False).plot.bar(
    color='blue', alpha=0.5, label='pos_x')
```

```
mi_pos_z_series.sort_values(ascending=False).plot.bar(
    color='red', alpha=0.5, label='pos_z')
plt.title('Mutual Information of Features')
plt.legend()
plt.show()
```

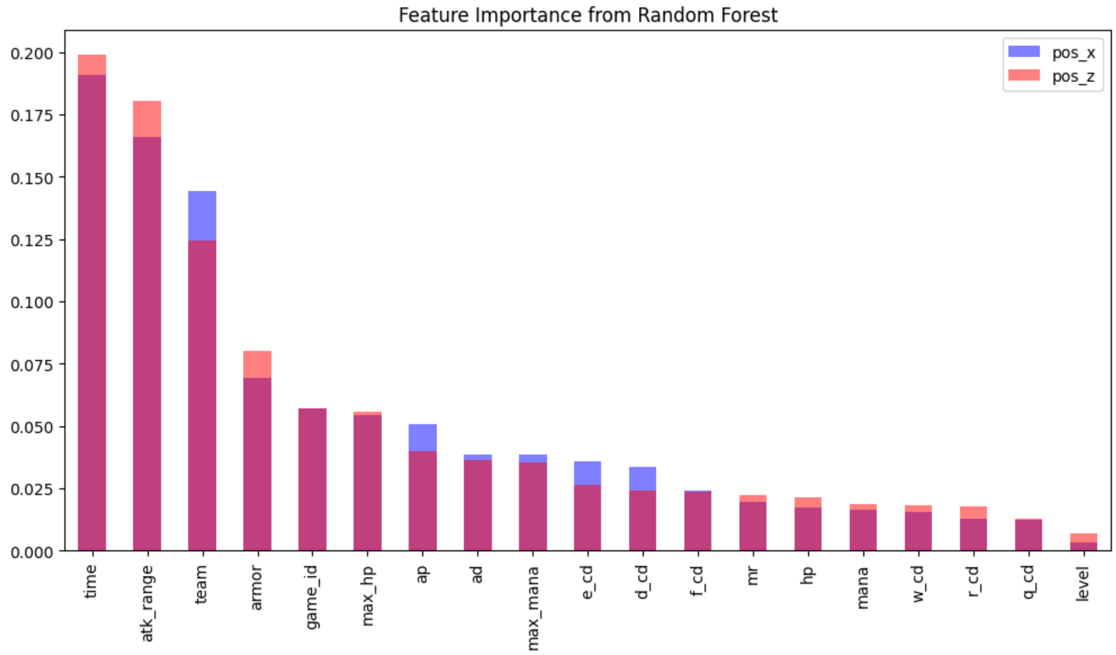


3 Feature Importance

[5]:

```
from sklearn.ensemble import RandomForestRegressor

# Feature importance for pos_x
rf_pos_x = RandomForestRegressor(n_estimators=100, random_state=42)
rf_pos_x.fit(X, y_pos_x)
importances_pos_x = rf_pos_x.feature_importances_
importance_pos_x_series = pd.Series(importances_pos_x, index=features)

# Feature importance for pos_z
rf_pos_z = RandomForestRegressor(n_estimators=100, random_state=42)
rf_pos_z.fit(X, y_pos_z)
importances_pos_z = rf_pos_z.feature_importances_
importance_pos_z_series = pd.Series(importances_pos_z, index=features)

# Plot Feature Importance
plt.figure(figsize=(12, 6))
```

```
importance_pos_x_series.sort_values(ascending=False).plot.bar(
    color='blue', alpha=0.5, label='pos_x', logy=True)
importance_pos_z_series.sort_values(ascending=False).plot.bar(
    color='red', alpha=0.5, label='pos_z', logy=True)
plt.title('Feature Importance from Random Forest')
plt.legend()
plt.show()
```



4 Select Important Features and Create DataFrame

[13]:
```
# Select important features based on correlation and feature importance
correlation_threshold = 0.6
importance_treshold = 0.01

important_features_x = mi_pos_x_series[mi_pos_x_series >
                                       correlation_threshold].index.
 ↪intersection(importance_pos_x_series[importance_pos_x_series >␣
 ↪importance_treshold].index).tolist()

important_features_z = mi_pos_z_series[mi_pos_z_series >
                                       correlation_threshold].index.
 ↪intersection(importance_pos_z_series[importance_pos_z_series >␣
 ↪importance_treshold].index).tolist()
```

```
important_features = list(set(important_features_x + important_features_z))

# Create a dataframe with important features
selected_data = data[important_features + target]
print("Selected features:")
print(important_features)
selected_data.head()
```

```
Selected features:
['armor', 'max_mana', 'max_hp', 'mr', 'ad', 'mana', 'atk_range', 'hp',
'game_id']
```

```
[13]:    armor  max_mana  max_hp    mr    ad   mana  atk_range     hp     game_id  \
      0   57.0     290.0   570.0  30.0  64.0  290.0      240.0  570.0  4848459903
      1   44.0     280.0   655.0  32.0  58.0  280.0      190.0  655.0  4848459903
      2   26.0     468.0   610.0  46.0  58.0  468.0      590.0  610.0  4848459903
      3   36.0     375.0   600.0  30.0  67.4  375.0      615.0  600.0  4848459903
      4   48.0     150.0   650.0  28.0  61.0    0.0      190.0  650.0  4848459903

         pos_x  pos_z
      0  604.0  612.0
      1  664.0  286.0
      2  364.0  136.0
      3  132.0  402.0
      4  298.0  676.0
```

```
[ ]:
```

temporal_features

July 30, 2024

[3]:

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import sqlite3
import pandas as pd
import sqlite3

import os
from dotenv import load_dotenv
from utils.get_or_create_combined_database import␣
  ↪get_or_create_combined_database
load_dotenv(verbose=True, override=True)

database_folder = os.getenv("DATABASE_FOLDER")

database_file = get_or_create_combined_database(database_folder)

table_name = "champs_cleaned"

conn = sqlite3.connect(database_file)
query = 'SELECT * FROM champs_cleaned'
data = pd.read_sql_query(query, conn)
conn.close()

# Display the first few rows of the dataframe
data.head()
```

```
Found 101 database files in the folder specified by DATABASE_FOLDER
Found combined database D:\league-ezreal-dataset\ml_project\combined2.db
```

[3]:

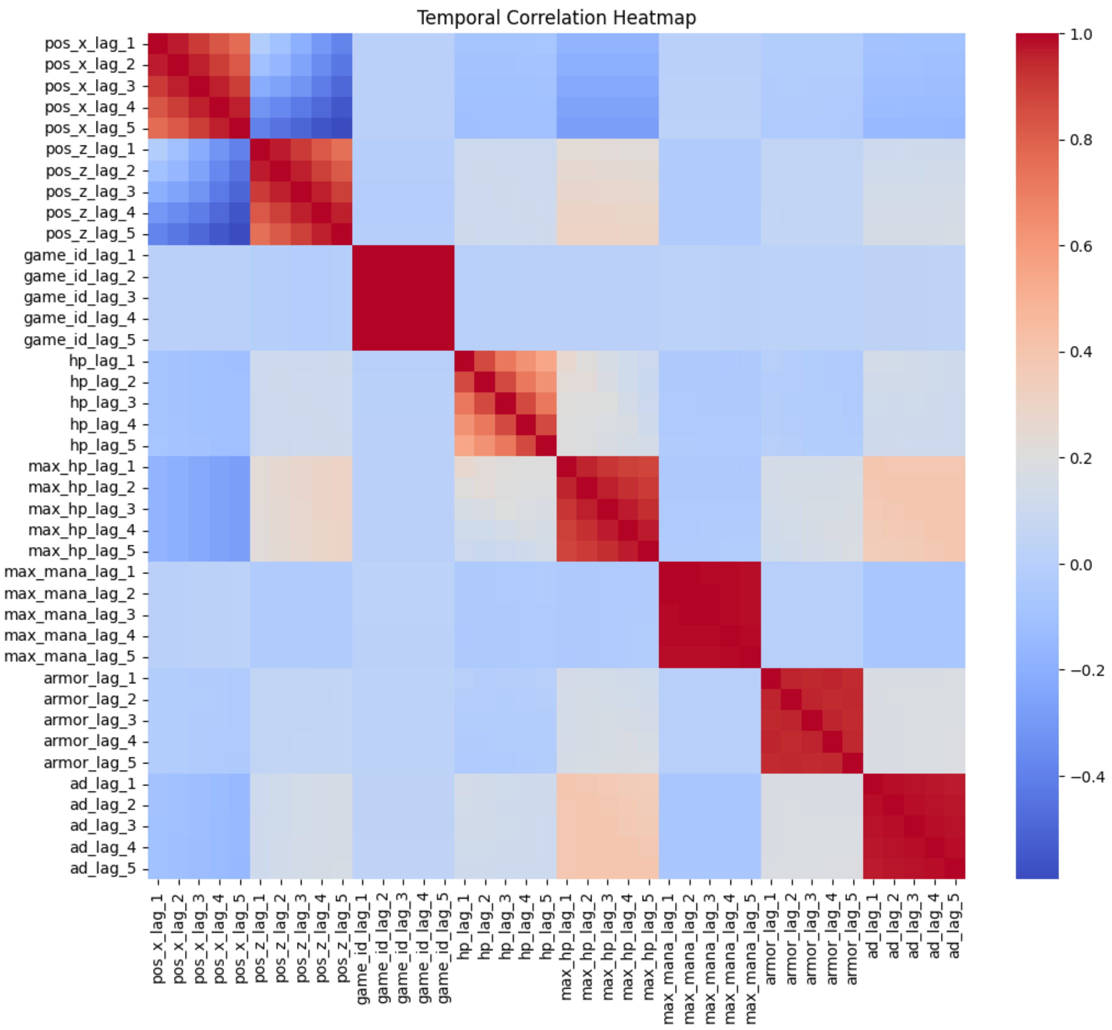
```
      game_id      time    name     hp  max_hp   mana  max_mana  armor    mr  \
0  4848459903  5.028642  KSante  570.0   570.0  290.0     290.0   57.0  30.0
1  4848459903  5.028642    Ekko  655.0   655.0  280.0     280.0   44.0  32.0
2  4848459903  5.028642   Swain  610.0   610.0  468.0     468.0   26.0  46.0
3  4848459903  5.028642  Ezreal  600.0   600.0  375.0     375.0   36.0  30.0
4  4848459903  5.028642  Rumble  650.0   650.0    0.0     150.0   48.0  28.0
```

```
     ad  …         d_name       d_cd            f_name       f_cd  \
0  64.0  …  SummonerFlash  10.971358  SummonerTeleport  10.971358
1  58.0  …  SummonerFlash  10.971358     SummonerSmite  10.971358
2  58.0  …  SummonerFlash  10.971358     SummonerHaste  10.971358
3  67.4  …  SummonerHaste  10.971358     SummonerFlash  10.971358
4  61.0  …    SummonerDot  10.971358     SummonerFlash  10.971358

   normalized_pos_x  normalized_pos_z  normalized_time normalized_hp  \
0          0.040267          0.040800         0.002794         0.114
1          0.044267          0.019067         0.002794         0.131
2          0.024267          0.009067         0.002794         0.122
3          0.008800          0.026800         0.002794         0.120
4          0.019867          0.045067         0.002794         0.130

   normalized_name           compound_key
0              897  4848459903_100_KSante
1              245    4848459903_100_Ekko
2               50   4848459903_100_Swain
3               81  4848459903_100_Ezreal
4               68  4848459903_100_Rumble

[5 rows x 35 columns]
```

1 Correlation Matrix

[4]:

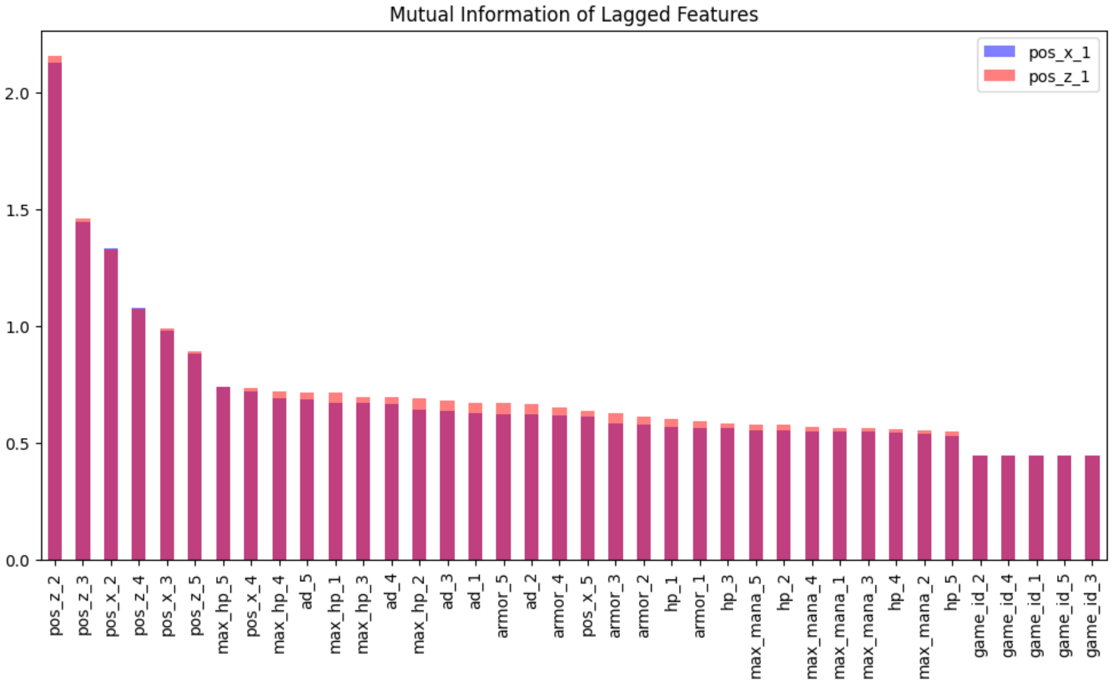
```python
# Group data by compound_key
grouped_data = data.groupby('compound_key')

# Parameters for sliding windows
H = 5  # Window size (number of steps)
step_size = 20  # Step size (time steps apart)

columns_to_use = ['pos_x', 'pos_z', 'game_id', 'hp', 'max_hp',
                  'max_mana', 'armor', 'ad']  # Columns to create sliding␣
 ↪windows for


def create_sliding_windows(group, columns, window_size, step_size):
    data_windows = []
    for i in range(window_size * step_size, len(group), step_size):
        window = group[columns].iloc[i-window_size *
                                     step_size:i:step_size].
 ↪reset_index(drop=True)
        data_windows.append(window.values.flatten())
    columns_expanded = [f'{col}_{j}' for j in range(
        1, window_size+1) for col in columns]
    return pd.DataFrame(data_windows, columns=columns_expanded)


# Create sliding windows for each group
windows = [create_sliding_windows(
    group, columns_to_use, H, step_size) for name, group in grouped_data]

# Combine the windows into a single DataFrame
sliding_windows_data = pd.concat(windows, ignore_index=True)

# Function to reshape the sliding windows data


def reshape_sliding_windows(sliding_windows_data, columns, window_size):
    reshaped_data = {}
    for col in columns:
        for lag in range(window_size):
            reshaped_data[f'{col}_lag_{lag+1}'] =␣
  ↪sliding_windows_data[f'{col}_{lag+1}']
    return pd.DataFrame(reshaped_data)


# Reshape the sliding windows data
reshaped_data = reshape_sliding_windows(
    sliding_windows_data, columns_to_use, H)

# Compute the correlation matrix
correlation_matrix = reshaped_data.corr()

# Save the heatmap as an image file
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=False, cmap='coolwarm')
plt.title('Temporal Correlation Heatmap')
plt.show()
```

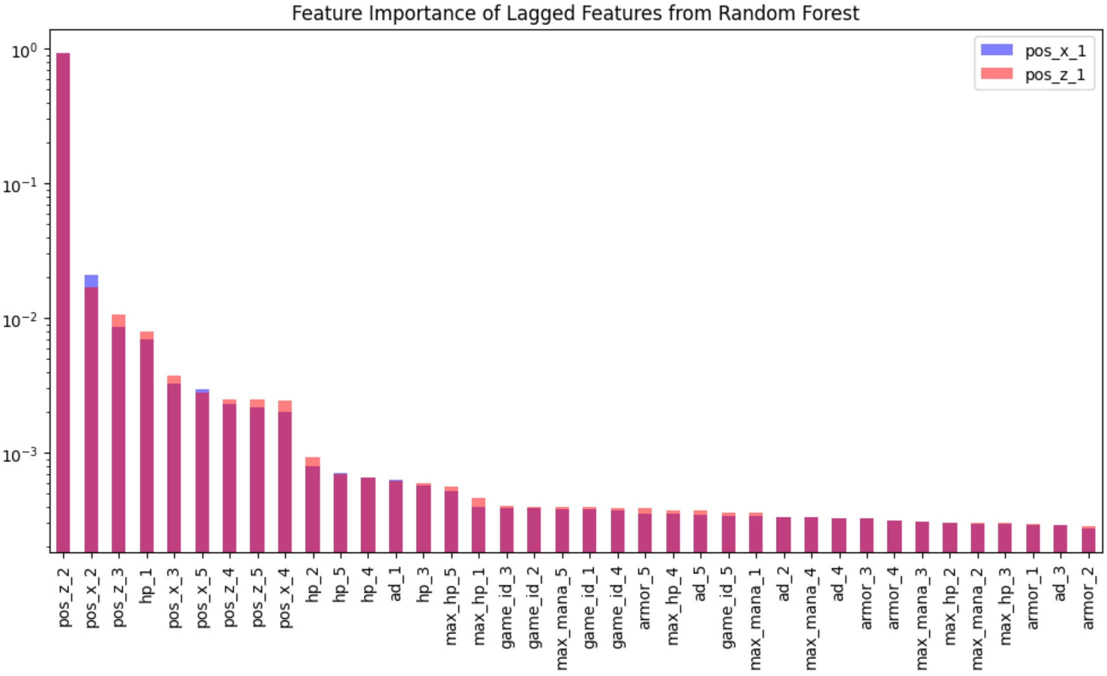


2 Mutual Information and Feature Importance

[14]:
```
from sklearn.feature_selection import mutual_info_regression
from sklearn.ensemble import RandomForestRegressor

# Prepare data for mutual information and feature importance
target_columns = ['pos_x_1', 'pos_z_1']

lagged_features = [
    f'{col}_{lag+1}' for col in columns_to_use for lag in range(1, H)]

X = sliding_windows_data[lagged_features]
y_pos_x = sliding_windows_data["pos_x_1"]
y_pos_z = sliding_windows_data["pos_z_1"]
```

```python
# Mutual Information for pos_x
mi_pos_x = mutual_info_regression(X, y_pos_x)
mi_pos_x_series = pd.Series(mi_pos_x, index=lagged_features)

# Mutual Information for pos_z
mi_pos_z = mutual_info_regression(X, y_pos_z)
mi_pos_z_series = pd.Series(mi_pos_z, index=lagged_features)

# Plot Mutual Information
plt.figure(figsize=(12, 6))
mi_pos_x_series.sort_values(ascending=False).plot.bar(
    color='blue', alpha=0.5, label=target_columns[0])
mi_pos_z_series.sort_values(ascending=False).plot.bar(
    color='red', alpha=0.5, label=target_columns[1])
plt.title('Mutual Information of Lagged Features')
plt.legend()
plt.show()

# Feature importance for normalized_pos_x
rf_pos_x = RandomForestRegressor(n_estimators=100, random_state=42)
rf_pos_x.fit(X, y_pos_x)
importances_pos_x = rf_pos_x.feature_importances_
importance_pos_x_series = pd.Series(importances_pos_x, index=lagged_features)

# Feature importance for normalized_pos_z
rf_pos_z = RandomForestRegressor(n_estimators=100, random_state=42)
rf_pos_z.fit(X, y_pos_z)
importances_pos_z = rf_pos_z.feature_importances_
importance_pos_z_series = pd.Series(importances_pos_z, index=lagged_features)

# Plot Feature Importance
plt.figure(figsize=(12, 6))
importance_pos_x_series.sort_values(ascending=False).plot.bar(
    color='blue', alpha=0.5, label=target_columns[0], logy=True)
importance_pos_z_series.sort_values(ascending=False).plot.bar(
    color='red', alpha=0.5, label=target_columns[1], logy=True)
plt.title('Feature Importance of Lagged Features from Random Forest')
plt.legend()
plt.show()
```





[19]:

```
# Select important features based on mutual information and feature importance
correlation_threshold = 0.6
importance_treshold = 0.01
```

```
important_features_x = mi_pos_x_series[mi_pos_x_series >
                                       correlation_threshold].index.
 ↪intersection(importance_pos_x_series[importance_pos_x_series >␣
 ↪importance_treshold].index).tolist()

important_features_z = mi_pos_z_series[mi_pos_z_series >
                                       correlation_threshold].index.
 ↪intersection(importance_pos_z_series[importance_pos_z_series >␣
 ↪importance_treshold].index).tolist()

important_features = list(set(important_features_x) &
                          set(important_features_z))

# Create a dataframe with important features
selected_data = sliding_windows_data[important_features]
print("Selected features:")
print(important_features)
print(selected_data.head())
```

```
Selected features:
['pos_x_2', 'pos_z_2']
     pos_x_2    pos_z_2
0   664.0000   286.0000
1   664.0000   286.0000
2   664.0000   286.0000
3  2344.5452  1573.4022
4  4362.9473  4192.7010
```

visualize_errors

July 30, 2024

[41]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import sqlite3
import pandas as pd
import sqlite3
import json
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.linear_model import LinearRegression

import os
from dotenv import load_dotenv
from utils.get_or_create_combined_database import␣
  ↪get_or_create_combined_database
from utils.get_data import fetch_data_batches
from utils.create_sequences_in_batches import␣
  ↪create_sequences_from_database_rows
load_dotenv(verbose=True, override=True)


from constants import DB_columns

database_folder = os.getenv("DATABASE_FOLDER")

database_file = get_or_create_combined_database(database_folder)

table_name = "champs_cleaned"

conn = sqlite3.connect(database_file)
query = 'SELECT * FROM champs_cleaned LIMIT 5'
data = pd.read_sql_query(query, conn)
conn.close()

# Display the first few rows of the dataframe
```

```
data.head()
```

```
Thumbs.db
combined2.db
Found 2 database files in the folder specified by DATABASE_FOLDER
Found combined database /u/23/tarpill1/unix/Documents/combined2.db
```

[41]:

```
      game_id      time         name     hp  max_hp     mana  max_mana  armor  \
0  2841236401  5.541945  Mordekaiser  645.0   645.0      0.0     100.0   61.0
1  2841236401  5.541945        Viego  630.0   630.0  10000.0   10000.0   46.0
2  2841236401  5.541945        Riven  745.0   745.0      0.0       0.0   33.0
3  2841236401  5.541945       Ezreal  600.0   600.0    375.0     375.0   36.0
4  2841236401  5.541945      Leblanc  598.0   598.0    400.0     400.0   34.0

     mr    ad  …  normalized_e_name  normalized_e_cd  normalized_r_name  \
0  32.0  61.0  …                  1        -0.009084                  1
1  32.0  62.4  …                  2        -0.009084                  2
2  32.0  84.8  …                  3        -0.009084                  3
3  30.0  67.4  …                  4        -0.009084                  4
4  30.0  55.0  …                  5        -0.009084                  5

   normalized_r_cd  normalized_d_name  normalized_d_cd  normalized_f_name  \
0        -0.009084                  1         0.020916                  1
1        -0.009084                  1         0.020916                  2
2        -0.009084                  1         0.020916                  3
3        -0.009084                  1         0.020916                  4
4        -0.009084                  1         0.020916                  1

  normalized_f_cd               compound_key    role
0        0.020916  2841236401_100_Mordekaiser     Top
1        0.020916        2841236401_100_Viego  Jungle
2        0.020916        2841236401_100_Riven     Mid
3        0.020916       2841236401_100_Ezreal     Bot
4        0.020916      2841236401_100_Leblanc     Bot

 [5 rows x 56 columns]
```

[42]:

```
data_features =  [ DB_columns.NORMALIZED_POS_X.value, DB_columns.
 ↪NORMALIZED_POS_Z.value ]

labels = [ DB_columns.NORMALIZED_POS_X.value, DB_columns.NORMALIZED_POS_Z.value␣
 ↪]
H_values = [80]
T_values = [20]

training_and_validation_set_size = 800
testing_set_size = 200
```

```
max_H = max(H_values)
max_T = max(T_values)
```

[43]:

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'

print(f'Using {device} device')
```

```
Using cpu device
```

[44]:

```
# Models

def train_model(model, X_train, y_train, epochs=50, batch_size=64,␣
 ↪learning_rate=0.001, cutoff_loss=None):
    device = model.device
    model.to(device)
    criterion = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    X_train_tensor = torch.tensor(X_train, dtype=torch.float32).to(device)
    y_train_tensor = torch.tensor(y_train, dtype=torch.float32).to(device)

    dataset = torch.utils.data.TensorDataset(X_train_tensor, y_train_tensor)
    train_loader = torch.utils.data.DataLoader(
        dataset, batch_size=batch_size, shuffle=True)

    model.train()
    for epoch in range(epochs):
        pbar = tqdm(
            train_loader, desc=f'Epoch {epoch+1}/{epochs}', leave=False)
        for X_batch, y_batch in pbar:
            optimizer.zero_grad()
            output = model(X_batch)
            # Only use the first two feature dimensions for loss calculation
            loss = criterion(output[:, :2], y_batch[:, :2])
            loss.backward()
            optimizer.step()
            pbar.set_postfix({'Loss': loss.item()})
        current_loss = loss.item()
        if cutoff_loss is not None and current_loss < cutoff_loss:
            print(
                f'Loss is below cutoff value of {cutoff_loss}. Stopping␣
 ↪training.')
            break
        pbar.close()
```

```python
# Function to predict with the PyTorch model


def predict_model(model, X, batch_size=64, no_progress=True):
    device = model.device
    model.to(device)
    model.eval()
    X_tensor = torch.tensor(X, dtype=torch.float32).to(device)
    dataset = torch.utils.data.TensorDataset(X_tensor)
    loader = torch.utils.data.DataLoader(dataset, batch_size=batch_size)
    predictions = []
    pbar = tqdm(loader, desc='Predicting') if not no_progress else loader
    with torch.no_grad():
        for X_batch, in pbar:
            output = model(X_batch)
            predictions.append(output.cpu().numpy())
    return np.vstack(predictions)


class TrajectoryPredictor(nn.Module):
    def __init__(self, input_shape, output_shape, lstm_units=128, device='cpu', parameters=None):
        super(TrajectoryPredictor, self).__init__()
        if parameters is not None:
            self.epochs = parameters['epochs']
            self.batch_size = parameters['batch_size']
            self.learning_rate = parameters['learning_rate']
            self.dropout_rate = parameters['dropout_rate']
        else:
            self.epochs = 10
            self.batch_size = 640
            self.learning_rate = 0.001
            self.dropout_rate = 0.2

        self.lstm1 = nn.LSTM(input_shape[-1], lstm_units, batch_first=True)
        self.dropout1 = nn.Dropout(self.dropout_rate)
        self.lstm2 = nn.LSTM(lstm_units, lstm_units, batch_first=True)
        self.dropout2 = nn.Dropout(self.dropout_rate)
        self.fc = nn.Linear(lstm_units, output_shape)
        self.device = device

    def forward(self, x):
        x, _ = self.lstm1(x)
        x = self.dropout1(x)
        x, _ = self.lstm2(x)
        x = self.dropout2(x)
        x = self.fc(x[:, -1, :])  # taking the output of the last time step
        return x

    def fit(self, X, y, cutoff_loss=None):
        train_model(self, X, y, self.epochs,
                    self.batch_size, self.learning_rate, cutoff_loss)

    def predict(self, X):
        return predict_model(self, X, self.batch_size)
```

[45]:

```
linear_regression_features = [
    DB_columns.NORMALIZED_POS_X.value, DB_columns.NORMALIZED_POS_Z.value]

lstm_parameters = {'epochs': 10, 'batch_size': 256,
                'learning_rate': 0.0005}
learning_rates = [0.0001, 0.001, 0.01]
batch_sizes = [64, 128, 256]
dropout_rates = [0.2, 0.4, 0.6]

lstm_parameter_sets = [
    {'epochs': 10,
        'batch_size': bs,
        'learning_rate': lr,
        'dropout_rate': dr
    } for bs in batch_sizes for lr in learning_rates for dr in dropout_rates
]

def get_lstm_name(params):
    return␣
 ↪f"lstm_lr_{params['learning_rate']}_bs_{params['batch_size']}_dr_{params['dropout_rate']}"

lstm_models = [ (get_lstm_name(params), data_features, params) for params in␣
 ↪lstm_parameter_sets ]
lstm_getters = dict(map(lambda x: (x[0], lambda H, T: (TrajectoryPredictor(
    input_shape=(H, len(x[1])),
    output_shape=2,
    device=device,
    parameters=x[2],
), x[1], (-1, H, len(x[1])))), lstm_models))

model_getters = {
    'linear_regression': lambda H, T: (LinearRegression(),␣
 ↪linear_regression_features, (-1, H*len(linear_regression_features))),
    **lstm_getters
}

# Display model getters and their values in a table
pd.DataFrame({
    'Model': [ (key, H, T) for key in model_getters.keys() for H in H_values␣
↪for T in T_values],
    'Features': [ len(x(H, T)[1]) for x in model_getters.values() for H in␣
↪H_values for T in T_values],
    'Shape': [ x(H, T)[2] for x in model_getters.values() for H in H_values for␣
↪T in T_values],
    'Parameters': [ x(H, T)[0].parameters if hasattr(x(H, T)[0], 'parameters')␣
↪else None for x in model_getters.values() for H in H_values for T in␣
↪T_values]
})
```

[45]:

```
                                        Model  Features        Shape  \
0              (linear_regression, 80, 20)         2    (-1, 160)
1     (lstm_lr_0.0001_bs_64_dr_0.2, 80, 20)         2  (-1, 80, 2)
2     (lstm_lr_0.0001_bs_64_dr_0.4, 80, 20)         2  (-1, 80, 2)
3     (lstm_lr_0.0001_bs_64_dr_0.6, 80, 20)         2  (-1, 80, 2)
4      (lstm_lr_0.001_bs_64_dr_0.2, 80, 20)         2  (-1, 80, 2)
5      (lstm_lr_0.001_bs_64_dr_0.4, 80, 20)         2  (-1, 80, 2)
6      (lstm_lr_0.001_bs_64_dr_0.6, 80, 20)         2  (-1, 80, 2)
7       (lstm_lr_0.01_bs_64_dr_0.2, 80, 20)         2  (-1, 80, 2)
8       (lstm_lr_0.01_bs_64_dr_0.4, 80, 20)         2  (-1, 80, 2)
9       (lstm_lr_0.01_bs_64_dr_0.6, 80, 20)         2  (-1, 80, 2)
10   (lstm_lr_0.0001_bs_128_dr_0.2, 80, 20)         2  (-1, 80, 2)
11   (lstm_lr_0.0001_bs_128_dr_0.4, 80, 20)         2  (-1, 80, 2)
12   (lstm_lr_0.0001_bs_128_dr_0.6, 80, 20)         2  (-1, 80, 2)
13    (lstm_lr_0.001_bs_128_dr_0.2, 80, 20)         2  (-1, 80, 2)
14    (lstm_lr_0.001_bs_128_dr_0.4, 80, 20)         2  (-1, 80, 2)
15    (lstm_lr_0.001_bs_128_dr_0.6, 80, 20)         2  (-1, 80, 2)
16     (lstm_lr_0.01_bs_128_dr_0.2, 80, 20)         2  (-1, 80, 2)
17     (lstm_lr_0.01_bs_128_dr_0.4, 80, 20)         2  (-1, 80, 2)
18     (lstm_lr_0.01_bs_128_dr_0.6, 80, 20)         2  (-1, 80, 2)
19   (lstm_lr_0.0001_bs_256_dr_0.2, 80, 20)         2  (-1, 80, 2)
20   (lstm_lr_0.0001_bs_256_dr_0.4, 80, 20)         2  (-1, 80, 2)
21   (lstm_lr_0.0001_bs_256_dr_0.6, 80, 20)         2  (-1, 80, 2)
22    (lstm_lr_0.001_bs_256_dr_0.2, 80, 20)         2  (-1, 80, 2)
23    (lstm_lr_0.001_bs_256_dr_0.4, 80, 20)         2  (-1, 80, 2)
24    (lstm_lr_0.001_bs_256_dr_0.6, 80, 20)         2  (-1, 80, 2)
25     (lstm_lr_0.01_bs_256_dr_0.2, 80, 20)         2  (-1, 80, 2)
26     (lstm_lr_0.01_bs_256_dr_0.4, 80, 20)         2  (-1, 80, 2)
27     (lstm_lr_0.01_bs_256_dr_0.6, 80, 20)         2  (-1, 80, 2)

                                           Parameters
0                                                None
1   <bound method Module.parameters of TrajectoryP…
2   <bound method Module.parameters of TrajectoryP…
3   <bound method Module.parameters of TrajectoryP…
4   <bound method Module.parameters of TrajectoryP…
```

```
5   <bound method Module.parameters of TrajectoryP…
6   <bound method Module.parameters of TrajectoryP…
7   <bound method Module.parameters of TrajectoryP…
8   <bound method Module.parameters of TrajectoryP…
9   <bound method Module.parameters of TrajectoryP…
10  <bound method Module.parameters of TrajectoryP…
11  <bound method Module.parameters of TrajectoryP…
12  <bound method Module.parameters of TrajectoryP…
13  <bound method Module.parameters of TrajectoryP…
14  <bound method Module.parameters of TrajectoryP…
15  <bound method Module.parameters of TrajectoryP…
16  <bound method Module.parameters of TrajectoryP…
17  <bound method Module.parameters of TrajectoryP…
18  <bound method Module.parameters of TrajectoryP…
19  <bound method Module.parameters of TrajectoryP…
20  <bound method Module.parameters of TrajectoryP…
21  <bound method Module.parameters of TrajectoryP…
22  <bound method Module.parameters of TrajectoryP…
23  <bound method Module.parameters of TrajectoryP…
24  <bound method Module.parameters of TrajectoryP…
25  <bound method Module.parameters of TrajectoryP…
26  <bound method Module.parameters of TrajectoryP…
27  <bound method Module.parameters of TrajectoryP…
```

[46]:

```
# Load models
folder = 'models'
trained_models = {}
training_errors = {}
validation_errors = {}
test_errors = {}
model_names = model_getters.keys()
model_file_names = [ f'{H}_{T}_{model_name}.pt' for H in H_values for T in␣
 ↪T_values for model_name in model_names]
for file_name in model_file_names:
    model_name_parts = file_name.split('.')
    model_name_parts = ".".join(model_name_parts[:-1]).split('_') if␣
 ↪len(model_name_parts) > 2 else model_name_parts[0].split('_')
    model_name = (int(model_name_parts[0]), int(model_name_parts[1]), '_'.
 ↪join(model_name_parts[2:]))
    trained_models[model_name] = torch.load(os.path.join(folder, file_name),␣
 ↪map_location=torch.device(device))
    trained_models[model_name].device = device
    try:
        training_info_file_name = file_name.replace('.pt', '.json')
        with open(os.path.join(folder, training_info_file_name), 'r') as f:
            training_info = json.load(f)
        training_errors[model_name] = training_info['training_error']
        validation_errors[model_name] = training_info['validation_error']
        test_errors[model_name] = training_info['test_error']

    except FileNotFoundError:
        training_errors[model_name] = None
        validation_errors[model_name] = None
        print(f'Error loading training information for␣
 ↪{training_info_file_name}')
    except Exception as e:
        print(f"Uncatched error: {e}")

# trained_models, training_errors, validation_errors
print("Models loaded")
```

```
/u/23/tarpill1/unix/.local/lib/python3.8/site-packages/sklearn/base.py:329:
UserWarning: Trying to unpickle estimator LinearRegression from version 1.4.2
when using version 1.1.2. This might lead to breaking code or invalid results.
Use at your own risk. For more info please refer to:
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-
limitations
  warnings.warn(
```

```
Models loaded
```

[47]:
```
# Find the linear regression model name, and the best LSTM model name
linear_regression_model_name = [name for name in model_names if␣
 ↪"linear_regression" in name][0]
print(test_errors.keys())
best_lstm_model_name = min([key for key in test_errors.keys() if␣
 ↪linear_regression_model_name not in key], key=test_errors.get)[2]

models_to_visualize = list([ linear_regression_model_name,␣
 ↪best_lstm_model_name])
models_to_visualize
```

```
dict_keys([(80, 20, 'linear_regression'), (80, 20,
'lstm_lr_0.0001_bs_64_dr_0.2'), (80, 20, 'lstm_lr_0.0001_bs_64_dr_0.4'), (80,
20, 'lstm_lr_0.0001_bs_64_dr_0.6'), (80, 20, 'lstm_lr_0.001_bs_64_dr_0.2'), (80,
20, 'lstm_lr_0.001_bs_64_dr_0.4'), (80, 20, 'lstm_lr_0.001_bs_64_dr_0.6'), (80,
20, 'lstm_lr_0.01_bs_64_dr_0.2'), (80, 20, 'lstm_lr_0.01_bs_64_dr_0.4'), (80,
20, 'lstm_lr_0.01_bs_64_dr_0.6'), (80, 20, 'lstm_lr_0.0001_bs_128_dr_0.2'), (80,
20, 'lstm_lr_0.0001_bs_128_dr_0.4'), (80, 20, 'lstm_lr_0.0001_bs_128_dr_0.6'),
(80, 20, 'lstm_lr_0.001_bs_128_dr_0.2'), (80, 20,
'lstm_lr_0.001_bs_128_dr_0.4'), (80, 20, 'lstm_lr_0.001_bs_128_dr_0.6'), (80,
20, 'lstm_lr_0.01_bs_128_dr_0.2'), (80, 20, 'lstm_lr_0.01_bs_128_dr_0.4'), (80,
20, 'lstm_lr_0.01_bs_128_dr_0.6'), (80, 20, 'lstm_lr_0.0001_bs_256_dr_0.2'),
(80, 20, 'lstm_lr_0.0001_bs_256_dr_0.4'), (80, 20,
'lstm_lr_0.0001_bs_256_dr_0.6'), (80, 20, 'lstm_lr_0.001_bs_256_dr_0.2'), (80,
20, 'lstm_lr_0.001_bs_256_dr_0.4'), (80, 20, 'lstm_lr_0.001_bs_256_dr_0.6'),
(80, 20, 'lstm_lr_0.01_bs_256_dr_0.2'), (80, 20, 'lstm_lr_0.01_bs_256_dr_0.4'),
(80, 20, 'lstm_lr_0.01_bs_256_dr_0.6')])
```

[47]:

```
['linear_regression', 'lstm_lr_0.001_bs_128_dr_0.4']
```

[48]:

```
# Fetch data

conn = sqlite3.connect(database_file)
cursor = conn.cursor()

plotting_features = [DB_columns.NORMALIZED_POS_X.value, DB_columns.
 ↪NORMALIZED_POS_Z.value, DB_columns.NORMALIZED_NAME.value]
additional_features = [DB_columns.TIME.value, DB_columns.HP.value, DB_columns.
 ↪NORMALIZED_NAME.value]

fetched_features = list(np.unique(data_features + plotting_features +␣
 ↪additional_features))
fetched_features.sort(key=lambda feature: data_features.index(feature) if␣
 ↪feature in data_features else len(data_features))

data = fetch_data_batches(cursor, table_name, "1=1",␣
 ↪training_and_validation_set_size, testing_set_size, fetched_features)
```

```
Using in-memory cache for counts

Fetched 200 keys for offset: 800, limit: 200
```

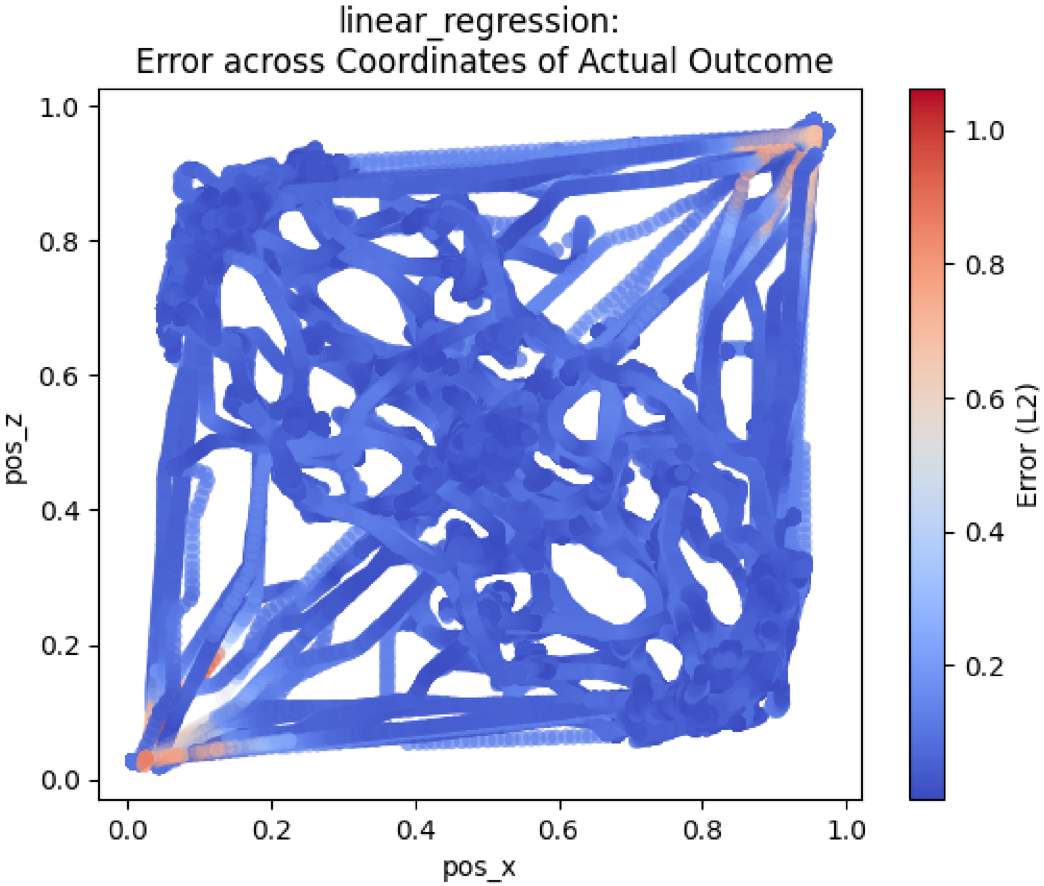
[49]:

```
predictions = {}
truths = {}

test_errors = {}
```

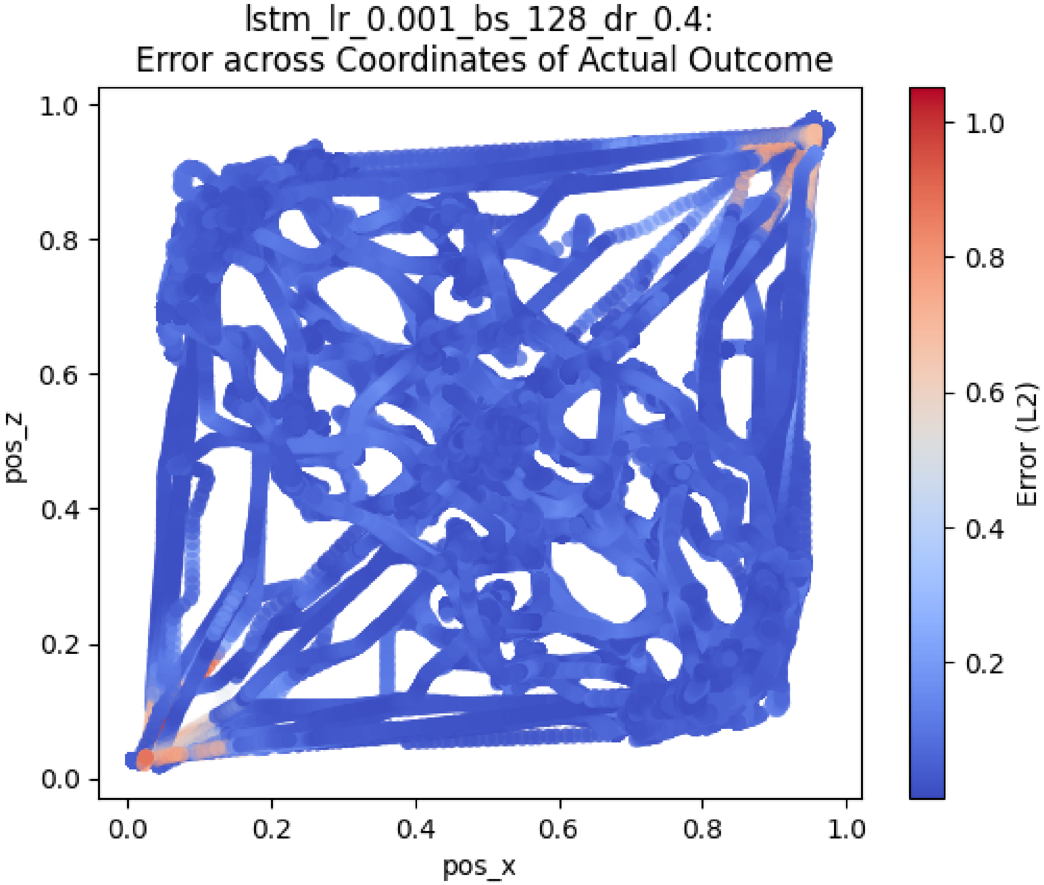
[50]:

```
for (H, T, model_name), model in trained_models.items():
    if model_name not in models_to_visualize:
        continue
    input_shape = model_getters[model_name](H, T)[2]
    features = model_getters[model_name](H, T)[1]
    sequences = create_sequences_from_database_rows(data, H, T, max_H, max_T)
    X, y = sequences
    truths[model_name] = y
    y_data_features = y[:, [labels.index(feature) for feature in features]]
    print(f"Predicting with model {model_name}")
    X_test_features = X[:, :, [
        data_features.index(feature) for feature in features]]
    X_test_features = X_test_features.reshape(
        X_test_features.shape[0], *input_shape)
    # Run the prediction on all the sequences
    y_pred = [ model.predict(X_test_features[i]) for i in range(X_test_features.
↪shape[0]) ]
    y_pred = np.array(y_pred, dtype=np.float32).reshape(-1, len(labels))
    predictions[model_name] = y_pred
    # Visualize the best and worst predictions
    absolute_errors = np.linalg.norm(y_data_features - y_pred, axis=1)
    test_errors[model_name] = absolute_errors

    absolute_errors_normalized = absolute_errors / absolute_errors.max()
    absolute_errors_normalized = absolute_errors_normalized - 0.5
    absolute_errors_normalized = abs(absolute_errors_normalized) /␣
↪abs(absolute_errors_normalized).max()
    plt.scatter(y_data_features[:, 0], y_data_features[:, 1],␣
↪c=absolute_errors, cmap='coolwarm', edgecolors='black', linewidth=0,␣
↪marker='o', alpha=absolute_errors_normalized)
    plt.colorbar( label='Error (L2)')
    plt.title(f'{model_name}:\n Error across Coordinates of Actual Outcome')
    plt.xlabel('pos_x')
    plt.ylabel('pos_z')
    plt.show()
```

```
Predicting with model linear_regression
```



```
Predicting with model lstm_lr_0.001_bs_128_dr_0.4
```


lstm_lr_0.001_bs_128_dr_0.4:
Error across Coordinates of Actual Outcome
pos_z
pos_x
Error (L2)
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
0.2
0.4
0.6
0.8
1.0