Artificial Intelligence Capstone Project1

Student name: 吳權祐 Student ID: 110550014

1 Introduction

The 2024 Paris Olympics are coming just around the corner, as one of the most renowned sporting events globally, the result of the competition is focused by the entire world. I want to see if we can predict the result of the game. Based on this motivation, I collected the historical records of men's 100 meters as my dataset and explored how machine learning and artificial intelligence can use the features to make predictions and improve sports.

Github repo with complete code:

https://github.com/ailuropodaWu/NYCU-AI-Capstone/tree/main/Project1

2 Dataset

2.1 Introduction

This is a dataset on the men's 100 meters events in the Olympics from 1948 to 2020 (without 1952 since there is no wind information from that year).

The attributes in this dataset correspond to the information about athletes and other relevant details about the events. The label of the dataset represents the performance of each athlete in a competition.

2.1.1 Compositions

Each row represents a record with following attributes about a competitor in a single game.

Attributes List

Name: string , name of the competitor, unuseful feature

Nation: int , code of the nation where the competitor from according to the dictionary (sorted by nation frequency in the dataset) (Appendix 1.2).

Weight *(kg)*: float , weight of the competitor

Height *(m)*: float , height of the competitor

BMI: float , calculated by the formula $\frac{weight(kg)}{(height(m))^2}$

Age: float , age of the competitor, calculated from the birthday to the first day the event start

Year: int , the year of the event

Round: int , round in the competition, range from 1 – 4 (4 means final)

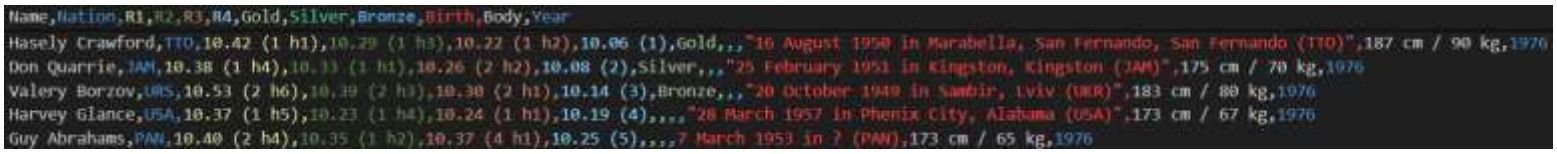
Wind *(m/s)*: float , wind information of the game

isHometown: bool , whether the venue of the event is the hometown of the competitor or not

Label *(s)*: float , performance of the competitor in the game

2.1.2 Data examples

- Raw data



```
Name,Nation,R1,R2,R3,R4,Gold,Silver,Bronze,Birth,Body,Year
Hasely Crawford,TTO,10.42 (1 h1),10.29 (1 h3),10.22 (1 h2),10.06 (1),Gold,,,"16 August 1950 in Marabella, San Fernando, San Fernando (TTO)",187 cm / 90 kg,1976
Don Quarrie,JAM,10.38 (1 h4),10.33 (1 h1),10.26 (2 h2),10.08 (2),Silver,,,"25 February 1951 in Kingston, Kingston (JAM)",175 cm / 70 kg,1976
Valery Borzov,URS,10.53 (2 h6),10.39 (2 h3),10.30 (2 h1),10.14 (3),Bronze,,,"20 October 1949 in Sambir, Lviv (UKR)",183 cm / 80 kg,1976
Harvey Glance,USA,10.37 (1 h5),10.23 (1 h4),10.24 (1 h1),10.19 (4),,,,"28 March 1957 in Phenix City, Alabama (USA)",173 cm / 67 kg,1976
Guy Abrahams,PAN,10.40 (2 h4),10.35 (1 h2),10.37 (4 h1),10.25 (5),,,,7 March 1953 in ? (PAN),173 cm / 65 kg,1976
```

- Corresponding data

```
Name,Age,Nation,Weight,Height,BMI,Year,Round,Wind,isHometown,Label
Hasely Crawford,25.935660506502394,6,90.0,1.87,25.73708141496754,1976,1,0.0,0.0,10.42
Hasely Crawford,25.935660506502394,6,90.0,1.87,25.73708141496754,1976,2,0.1,0.0,10.29
Hasely Crawford,25.935660506502394,6,90.0,1.87,25.73708141496754,1976,3,0.0,0.0,10.22
Hasely Crawford,25.935660506502394,6,90.0,1.87,25.73708141496754,1976,4,0.0,0.0,10.06
Don Quarrie,25.4072553045859,2,70.0,1.75,22.857142857142858,1976,1,0.0,0.0,10.38
Don Quarrie,25.4072553045859,2,70.0,1.75,22.857142857142858,1976,2,1.1,0.0,10.33
Don Quarrie,25.4072553045859,2,70.0,1.75,22.857142857142858,1976,3,0.0,0.0,10.26
Don Quarrie,25.4072553045859,2,70.0,1.75,22.857142857142858,1976,4,0.0,0.0,10.08
Valery Borzov,26.757015742642025,8,80.0,1.83,23.8884409806205,1976,1,0.0,0.0,10.53
Valery Borzov,26.757015742642025,8,80.0,1.83,23.8884409806205,1976,2,0.1,0.0,10.39
Valery Borzov,26.757015742642025,8,80.0,1.83,23.8884409806205,1976,3,0.0,0.0,10.3
Valery Borzov,26.757015742642025,8,80.0,1.83,23.8884409806205,1976,4,0.0,0.0,10.14
Harvey Glance,19.32101300479124,0,67.0,1.73,22.386314277122523,1976,1,0.0,0.0,10.37
Harvey Glance,19.32101300479124,0,67.0,1.73,22.386314277122523,1976,2,0.1,0.0,10.23
Harvey Glance,19.32101300479124,0,67.0,1.73,22.386314277122523,1976,3,0.0,0.0,10.24
Harvey Glance,19.32101300479124,0,67.0,1.73,22.386314277122523,1976,4,0.0,0.0,10.19
Guy Abrahams,23.378507871321013,36,65.0,1.73,21.718066089745733,1976,1,0.0,0.0,10.4
Guy Abrahams,23.378507871321013,36,65.0,1.73,21.718066089745733,1976,2,0.1,0.0,10.35
Guy Abrahams,23.378507871321013,36,65.0,1.73,21.718066089745733,1976,3,0.0,0.0,10.37
Guy Abrahams,23.378507871321013,36,65.0,1.73,21.718066089745733,1976,4,0.0,0.0,10.25
```

2.1.3 Amount and conditions

There are totally **2090** data in the dataset, and the label (performance) of data is conditioned to be under **11.5** to avoid outlier data. And the dataset consists of **921** athletes and **180** nations.

2.2 Collection

The stages of collection consist of data scraping and data processing. And all these processes are performed on my laptop using the Visual Studio Code IDE.

2.2.1 Scrape

All data is scrapped from the internet, particularly from the ***Olympedia*** website. I wrote a ***sraper.py*** (Appendix 1.1) scripts using the *requests* and *BeautifulSoup* packages to extract the raw data of all competitors in each competition and the wind information of the competition separately.

2.2.2 Process

Then, I used a ***generate_train_data.ipynb*** (Appendix 1.2) to process the raw data and generate the processed data. The packages used in the files are *pandas, re, datetime* and *calendar*.

In this stage, it was necessary for me to preprocess the data to make the following steps easier. First, I transformed any data with null values to match the format of the other values in the same columns. Then, some of the features were extracted from the raw data, such as *Weight* and *Height* from the *Body* column. Lastly, I dropped some useless attributes, such as *Gold*, *Silver*, and *Bronze*, which are used to record who got the medal. I also processed some information in the data. For example, the original *Birth* data contained information about the birthplace, and I processed it to preserve only the pure birthday.

The next step is to process the data into a specific format. Since the format of the raw data contains all results for one athlete in an event in a specific year, I separately extracted and filtered the results of the games the athlete attended that year.

In the final step of this stage, I created three new attributes

extended from the original columns. *Age* was calculated based on the athlete's birthday and the start date of the event. *isHometown* was a Boolean value that determined by whether the *Nation* of the athlete matched the country of the venue. And *BMI* attribute was calculated using the formula mentioned previously. After completing these steps, I retained the desired columns to obtain my own dataset.

2.3 External source

2.3.1 Packages

- requests (https://requests.readthedocs.io/en/latest/#)
- BeautifulSoup (https://requests.readthedocs.io/en/latest/#)
- pandas (https://pandas.pydata.org/docs/index.html)
- re (https://docs.python.org/3/library/re.html)
- datetime (https://docs.python.org/3/library/datetime.html)
- calendar (https://docs.python.org/dev/library/calendar.html)

2.3.2 Websites

- Olympedia men's 100 meters events page (https://www.olympedia.org/event_names/40)

3 Methods

Since the label in this dataset consists of continuous floating numbers, it appears to be a regression problem for prediction. I performed three types of supervised linear regression algorithms and two unsupervised method. The three supervised method are respectively closed-form linear regression, support vector machine (SVM) and deep-learning based model, while the unsupervised methods are both based on the nearest neighbor search problem. All algorithms are implemented in the ***algorithm.ipynb*** (Appendix 1.3) using the ***scikit-learn*** library.

3.1 Supervised learning

3.1.1 Closed-form linear regression

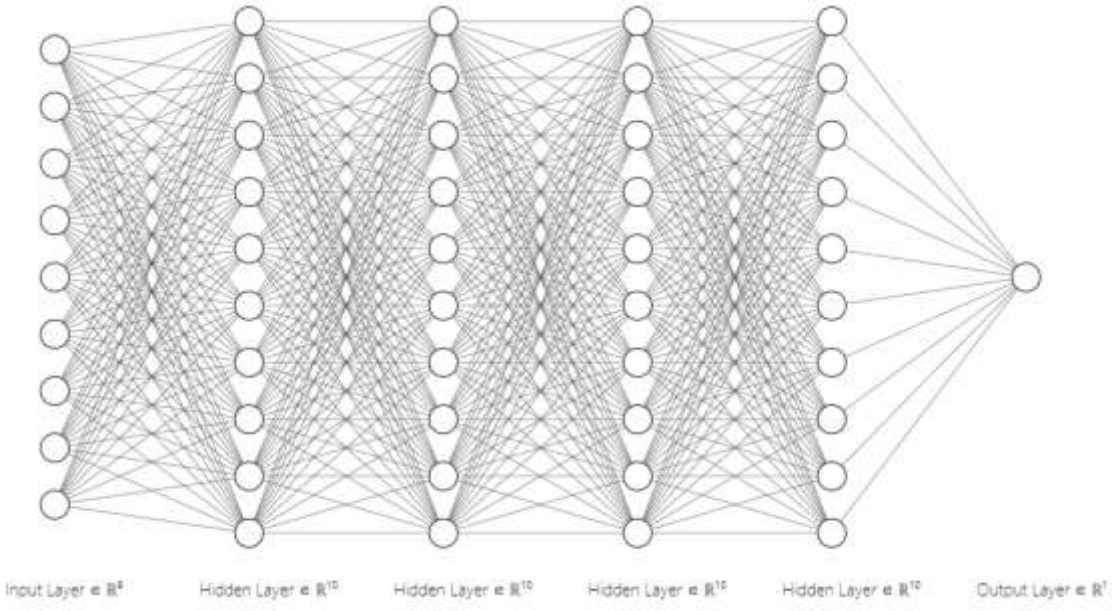
The algorithm uses Ordinary Least Squares (OLS) method to fit the line to the data, finding all coefficients and the intercept by minimizing errors.

3.1.2 Support Vector Machine

The core concept of Support Vector Machine (SVM) regressor is to find the best kernel function to transform data to a hyperplane that separates data points. I employed the SVM regressor with a polynomial kernel of degree 20 as one of my methods.

3.1.3 Deep-learning based

I used a Multi-layer Perceptron (MLP) as the regressor in my project. The network architecture includes 4 hidden layers, with each layer comprising 10 nodes, as illustrated below.



3.2 Unsupervised learning

The core concept of regression is to predict continuous value from the data. In unsupervised learning, I implemented algorithms based on the Nearest Neighbor Search (NNS) problem to find the nearest neighbor of the data and predict its label with the value from the neighbor. Specifically, I used KD-Tree algorithm for this purpose.

4 Experiments

4.1 Experiments detail

4.1.1 Data split

In the real-world, people focus on using historical data to predict the results of upcoming Olympic games in a specific year. To simulate this scenario, I divided the original dataset based on

the *Year* attributes. The training data includes information from 1948 to 2008, while the testing data with **151** rows comprises data from 2016 and 2020(2021).

4.1.2 Experiment setting

In the project, I conducted several experiments using different training datasets to explore the effects of data quantity and quality, and compare the performance of the algorithms. Here are the four types of training datasets.

- ***Complete***: *(1939 data)* the whole dataset
- ***Filtered***: *(1088 data)* each athlete from a specific year is recorded only once with their best performance in that year
- ***Thousand***: *(999 data)* call ***train_test_split*** from ***scikit-learn*** with *train_size* = 1000 / (numbers of rows in complete training data)
- ***Hundred***: *(100 data)* call ***train_test_split*** from ***scikit-learn*** with *train_size* = 100 / (numbers of rows in complete training data)

4.1.3 Evaluation metrics

Here are three types of metrics I used in the experiments to evaluate the performance.

- $\boldsymbol{R^2}$ ***score***: a score calculated by the below formula, it is used to get the coefficient of determination of the prediction.

$$R^2 = \frac{SS_{\text{reg}}}{SS_{\text{tot}}} = \frac{\sum_i (\hat{y}_i - \bar{y})^2}{\sum_i (y_i - \bar{y})^2}$$

- ***RMSE***: calculated by the below formula, it represents the

(RMSD) average error of the prediction.

$$\mathrm{RMSD}(\hat{\theta}) = \sqrt{\mathrm{MSE}(\hat{\theta})} = \sqrt{\mathrm{E}((\hat{\theta} - \theta)^2)}.$$

- ***Accuracy***: the accuracy rate under a specific error.

4.2 Experiments result

The underlined numbers are the best performances from the same training data, while the bold ones are from the same algorithm.

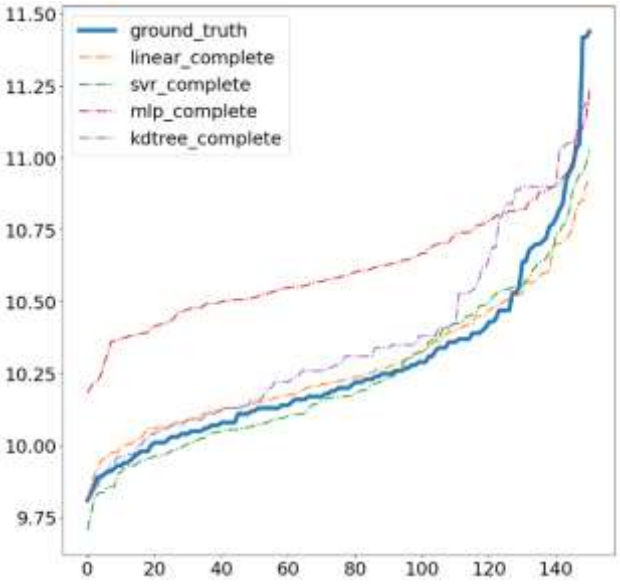
It appears that the closed-form linear regression algorithm consistently performs best in predicting results across different datasets related to the field, whereas deep learning-based methods show poor performance. Furthermore, regarding datasets, data size significantly affects training, while whether the data are filtered or not seems not as critical.

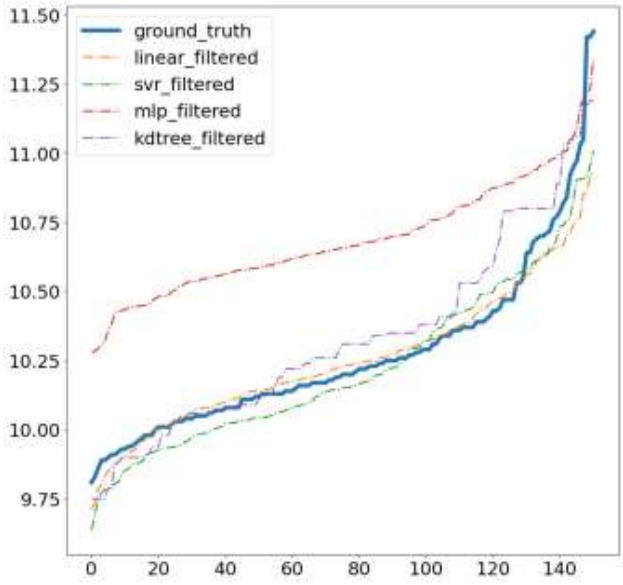
algorithms	***Linear***	***SVR***	***MLP***	***KD-Tree***
Complete				
R^2***score***	0.563	**0.531**	-1.306	-
RMSE	0.202	**0.210**	0.465	0.370
ACC (0.1)	0.430	**0.384**	0.060	**0.265**
ACC (0.01)	**0.073**	0.033	0.007	**0.046**
Filtered				
R^2***score***	**0.606**	0.507	-1.883	-
RMSE	**0.192**	0.215	0.520	**0.367**
ACC (0.1)	**0.464**	0.344	0.073	0.245
ACC (0.01)	0.046	0.013	0.007	0.040
Thousand				
R^2***score***	0.575	0.518	**-1.243**	-
RMSE	0.200	0.212	**0.459**	0.380
ACC (0.1)	0.450	0.338	**0.079**	0.232
ACC (0.01)	0.066	**0.053**	0.007	0.020
Hundred				
R^2***score***	0.525	0.269	-3.278	-
RMSE	0.202	0.262	0.634	0.445
ACC (0.1)	0.331	0.305	0.066	0.126
ACC (0.01)	0.026	0.046	**0.013**	0.

4.3 Visualized results

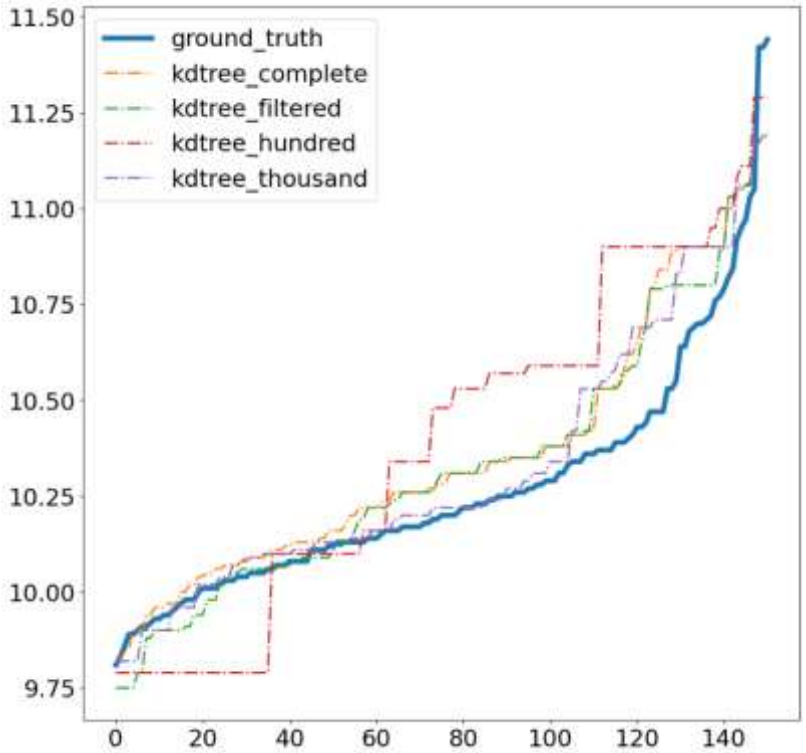
I visualized the experiments result using the ***matplotlib*** library for additional analysis. Complete visualized results could be found in Appendix 2.

Closed-form linear regression and SVM regressor are stable algorithms that closely approximate the ground truth.





The predictions from unsupervised learning become more discrete when the amount of data decreases. This is due to the reduced number of neighbors in the training data available for prediction.

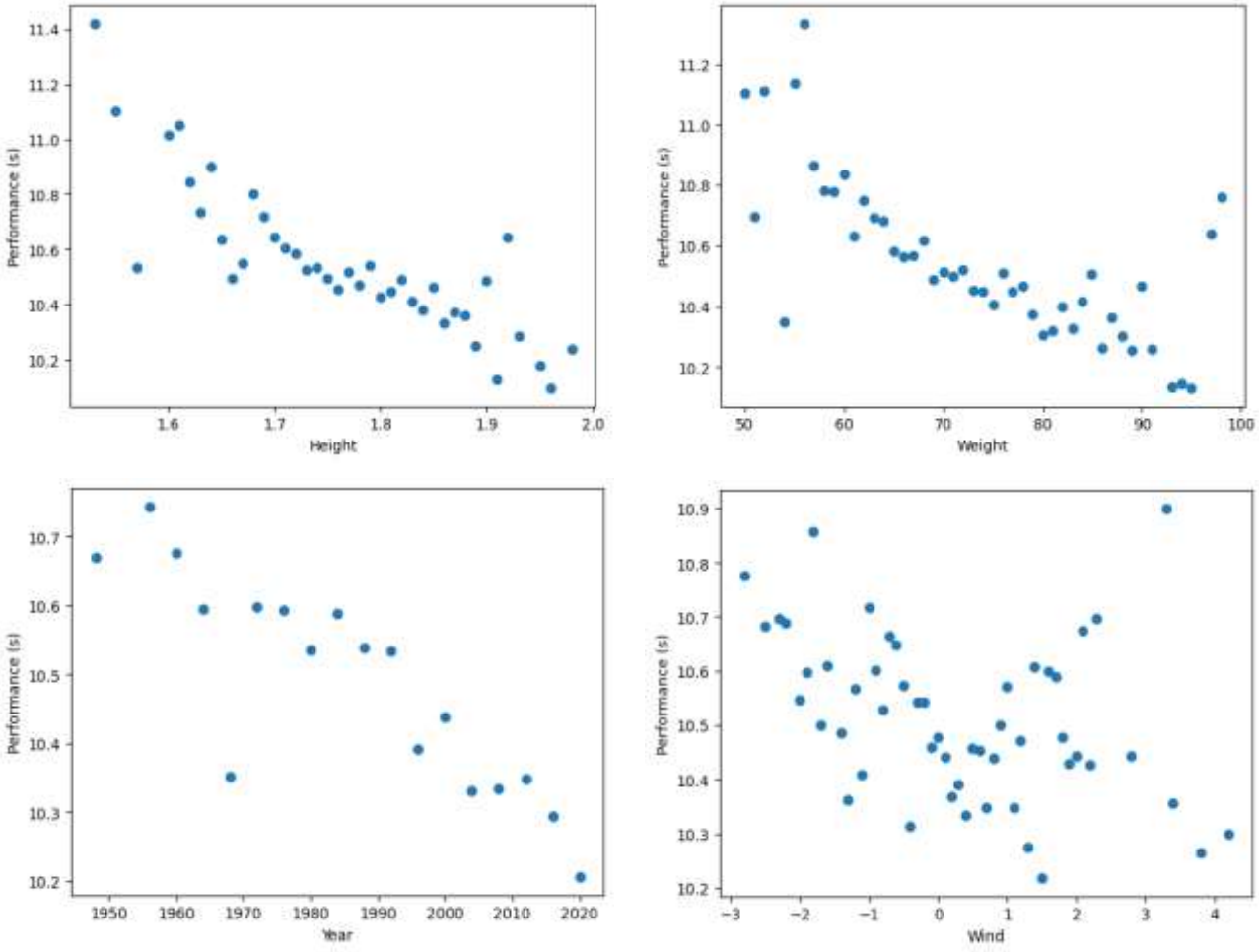


5 Discussion

5.1 Conclusion

Before the experiments, I expected deep-learning based methods to achieve the best performance, however, they actually showed the poorest performance. This led to the understanding that the issue might be related to the size of the data. With only about 2,000 data available, it seems insufficient for training a deep-learning model effectively. Something I didn't expect is the outperformance of closed-form linear regression. I believe this suggests that sports, especially events like men's 100 meters, are highly scientific and analyzable.

I have performed some feature engineering and discovered that most features in the dataset are highly correlated with the label. Specifically, features like *Height*, *Weight*, *Year*, and *Wind* exhibit high negative correlation with the average of performances, as shown in the figures below.



5.2 Future experiments

If there is more time available, I would like to conduct experiments related to hyperparameters and dimension reduction. I am curious about how hyperparameters impact the performance of deep-learning methods and what architecture would be suitable for models learning from limited data. Additionally, I am interested in exploring how dimension reduction affects performance and determining the optimal amount of feature reduction required.

5.3 Gained knowledge

Sports is an interesting and scientific field that is suitable for research. However, as seen in the results, the performance of the algorithms is still not optimal. In the future, we could explore other methods for data collection to create a stronger dataset or conduct further research on feature engineering to better understand the relationship between the data and the label.

6 References

https://requests.readthedocs.io/en/latest/#

https://requests.readthedocs.io/en/latest/#

https://requests.readthedocs.io/en/latest/#

https://pandas.pydata.org/docs/index.html

https://docs.python.org/3/library/re.html

https://docs.python.org/3/library/datetime.html

https://docs.python.org/dev/library/calendar.html

https://www.olympedia.org/event_names/40

https://scikit-learn.org/stable/index.html

https://matplotlib.org/

https://zhuanlan.zhihu.com/p/67706712

https://en.wikipedia.org/wiki/Root-mean-square_deviation

Appendix

1 Source Code

1.1 Scrape

```
# scrape the url of the men's 100m of 1948 - 2020
url_root = "https://www.olympedia.org/"
data = []
wind_list = []
header = []

url_men100 = url_root + "/event_names/40"
url_list_men100 = []
html = rq.get(url_men100).text
soup = BeautifulSoup(html, 'html.parser')
table = soup.find('table', {"class": "table table-striped"})
for row in table.find_all('tr'):
    cols = row.find_all('td')
    if len(cols) == 0 or int(cols[0].text) < 1948:
        continue
    url_list_men100.append(cols[1].a.get('href'))
```

```
# scrape the data from the url of the men's 100m on each years
for url_idx in tqdm(range(len(url_list_men100))):
    url = url_list_men100[url_idx]
    html = rq.get(url_root + url).text
    soup = BeautifulSoup(html, 'html.parser')
    wind_info = []

    while True: # Get the information of wind
        t_wind = soup.find('table', {'class': 'biodata'})
        if t_wind is None:
            break
        for row in t_wind.find_all('tr'):
            head = row.find('th').text
            if head != "Wind":
                continue
            wind = row.find('td').text.replace('m/s', '').replace('<', '')
            wind = float(wind)
            wind_info.append(wind)
            break
        t_wind.decompose()
```

```python
# get the record of all athletes
    table = soup.find('table', {"class": "table table-striped"})
    for row in table.find_all('tr'):
        cols = row.find_all('td')
        if len(cols) == 0:
            continue
        if url_idx < 16 and cols[4].text.strip()[0] ==  '-':
            # since 2012 the rule of Preliminary Round is changed
            break
        cols = cols[2:] # since the first 2 columns are Pos & Nr
```

```python
# scrape data of athlete
        url_athlete = cols[0].a.get('href')
        html_athlete = rq.get(url_root + url_athlete).text
        soup_athlete = BeautifulSoup(html_athlete, 'html.parser')
        t_athlete = soup_athlete.find('table', {'class': 'biodata'})
        athlete_info = [0] * 2 # to record birthday and weight&height
        for r in t_athlete.find_all('tr'):
            if r.find('th').text.strip() == 'Born':
                athlete_info[0] = r.find('td')
            if r.find('th').text.strip() == 'Measurements':
                athlete_info[1] = r.find('td')

        cols += athlete_info
        for i in range(len(cols)):
            try:
                cols[i] = cols[i].text.strip()
            except:
                cols[i] = None
        cols.append(url_idx * 4 + 1948)
        data.append(cols)
    wind_list.append(wind_info)
```

```python
# manually add header and save the data as .csv
header = ['Name', 'Nation', 'R1', 'R2', 'R3', 'R4', 'Gold', 'Silver', 'Bronze', 'Birth', 'Body', 'Year']
df = pd.DataFrame(data, columns=header)
pd.to_pickle(wind_list, './dataset/wind_list.pkl')
df.to_csv('./dataset/raw_data.csv', index=False)
```

1.2 Process

```
### Preprocess the data
###     1. process birthday attribute with dropping the place of the birth in the string
###     2. process record of the performance with replacing '-' into '-1 (h0)'
###     3. extract weight and height attribute from body and drop the rows with no body information
###     4. drop some unuseful columns

raw_data = pd.read_csv(raw_data_path)
raw_data.dropna(subset=['Birth', 'Body', 'Year'], inplace=True)
raw_data['Birthday'] = raw_data['Birth'].apply(lambda x: x.split("in")[0])
raw_data.loc[raw_data['R1'] == '-', 'R1'] = '-1 (h0)'
raw_data.loc[raw_data['R2'] == '-', 'R2'] = '-1 (h0)'
raw_data.loc[raw_data['R3'] == '-', 'R3'] = '-1 (h0)'
drop_index = []
for idx in raw_data.index:
    try:
        raw_data.loc[idx, 'Height'] = float(raw_data['Body'][idx].split()[0])  / 100
        raw_data.loc[idx, 'Weight'] = float(raw_data['Body'][idx].split()[3])
    except:
        drop_index.append(idx)
raw_data.drop(index=drop_index, inplace=True)
raw_data.drop(columns=['Gold', 'Silver', 'Bronze', 'Birth', 'Body'], inplace=True)

### Process the information of wind

wind_list = pd.read_pickle(wind_list_path)
wind_dict = {}
for idx in range(len(wind_list)):
    wind_dict[1948 + idx * 4] = wind_list[idx]
print(wind_dict)
```

```
### Process the data of performance record
###     1. to make sure the data could be convert to float type
###     2. extract the heat information

raw_data['R1'] = raw_data['R1'].apply(lambda x: re.sub(r'[\[\]w]', '', x))
raw_data['R2'] = raw_data['R2'].apply(lambda x: re.sub(r'[\[\]w]', '', x))
raw_data['R3'] = raw_data['R3'].apply(lambda x: re.sub(r'[\[\]w]', '', x))

raw_data['R1'] = raw_data['R1'].apply(lambda x: re.sub('–', '-1', x))
raw_data['R2'] = raw_data['R2'].apply(lambda x: re.sub('–', '-1', x))
raw_data['R3'] = raw_data['R3'].apply(lambda x: re.sub('–', '-1', x))
raw_data['R4'] = raw_data['R4'].apply(lambda x: re.sub('–', '-1', x))
for i in raw_data.index:
    p1 = raw_data['R1'][i].find('h')
    p2 = raw_data['R2'][i].find('h')
    p3 = raw_data['R3'][i].find('h')
    raw_data.loc[i, 'heat_r1'] = int(raw_data['R1'][i][p1+1:-1])
    raw_data.loc[i, 'heat_r2'] = int(raw_data['R2'][i][p2+1:-1])
    raw_data.loc[i, 'heat_r3'] = int(raw_data['R3'][i][p3+1:-1])
    p1 = raw_data['R1'][i].find('(')
    p2 = raw_data['R2'][i].find('(')
    p3 = raw_data['R3'][i].find('(')
    p4 = raw_data['R4'][i].find('(')
    raw_data.loc[i, 'R1'] = float(raw_data['R1'][i][:p1])
    raw_data.loc[i, 'R2'] = float(raw_data['R2'][i][:p2])
    raw_data.loc[i, 'R3'] = float(raw_data['R3'][i][:p3])
    try:
        raw_data.loc[i, 'R4'] = float(raw_data['R4'][i][:p4])
    except:
        raw_data.loc[i, 'R4'] = -1
```

```python
dataset = pd.DataFrame(columns=['Name', 'NOC', 'Weight', 'Height', 'Birthday' , 'Year', 'Round', 'Wind', 'Label'])
for _, row in raw_data.iterrows():
    for round in range(1, 5):
        col = f'R{round}'
        if row[col] == -1:
            continue
        try:
            wind_idx = 0 if round == 1 else num_heat[round - 2][row['Year']]
            wind_idx = wind_idx + row[f'heat_r{round}'] - 1 if round != 4 else -1
            wind = wind_dict[row['Year']][int(wind_idx)]
        except:
            break
        new_row = list(row[['Name', 'Nation', 'Weight', 'Height', 'Birthday' , 'Year']])
        new_row += [round, wind, row[col]]
        dataset.loc[len(dataset)] = new_row
dataset.dropna(inplace=True)
dataset = dataset.loc[dataset['Label'] <= 11.5]
print(dataset)
```

```python
drop_index = []
for idx, row in dataset.iterrows():
    birthday = row['Birthday']
    try:
        month = birthday.split()[1]
    except:
        drop_index.append(idx)
        continue
    birthday = birthday.replace(month, str(list(calendar.month_name).index(month)))
    while birthday[-1] == ' ':
        birthday = birthday[:-1]
    birthday = datetime.strptime(birthday, "%d %m %Y")
    dataset.loc[idx, "Age"] = (datetime.strptime(olympic_list[row['Year']][1], "%Y%m%d") - birthday).days / 365.25
    dataset.loc[idx, "isHometown"] = int(row['NOC'] == olympic_list[row['Year']][0])
    dataset.loc[idx, "BMI"] = row['Weight'] / row['Height'] / row['Height']
dataset.drop(index=drop_index, inplace=True)
dataset.dropna(inplace=True)
nation_cnt = dict(dataset.groupby(by=['NOC']).count()['Label'].sort_values(ascending=False))
nation_list = {}
nation_index = 0
for k, i in nation_cnt.items():
    nation_list[k] = nation_index
    nation_index += 1
print(nation_list)
dataset['Nation'] = dataset['NOC'].apply(lambda x: nation_list[x])

dataset_path = './dataset/dataset.csv'
cols = ['Name', 'Age', 'Nation', 'Weight', 'Height', 'BMI', 'Year', 'Round', 'Wind', 'isHometown', 'Label']
dataset = dataset[cols]
dataset.to_csv(dataset_path, index=False)
```

```python
test_data = dataset[dataset['Year'] >= 2016]
train_data = dataset[dataset['Year'] < 2016]
train_data.to_csv('./dataset/train.csv', index=False)
test_data.to_csv('./dataset/test.csv', index=False)

filter_data_path = './dataset/filter_train_data.csv'
filter_data = train_data.sort_values(by=['Label'])
filter_data.drop_duplicates(['Name', 'Year'], keep='first', ignore_index=True, inplace=True)
filter_data.sort_values(by=['Year'], inplace=True, ignore_index=True)
filter_data.to_csv(filter_data_path, index=False)
filter_data
```

Nation Code

'USA': 0, 'GBR': 1, 'JAM': 2, 'FRA': 3, 'CAN': 4, 'NGR': 5, 'TTO': 6,

'BRA': 7, 'URS': 8, 'POL': 9, 'JPN': 10, 'GHA': 11, 'CUB': 12, 'AUS':

13, 'BAH': 14, 'ITA': 15, 'GDR': 16, 'GER': 17, 'CIV': 18, 'FRG': 19,

'CHN': 20, 'SKN': 21, 'BAR': 22, 'HUN': 23, 'SEN': 24, 'ESP': 25,

'INA': 26, 'VEN': 27, 'KEN': 28, 'GRE': 29, 'RSA': 30, 'BEL': 31,

'POR': 32, 'CMR': 33, 'NAM': 34, 'BUL': 35, 'PAN': 36, 'UGA': 37,
'PAK': 38, 'QAT': 39, 'CGO': 40, 'SUR': 41, 'ANT': 42, 'TPE': 43,
'GAM': 44, 'THA': 45, 'ISV': 46, 'CYP': 47, 'DOM': 48, 'SUI': 49,
'AHO': 50, 'SGP': 51, 'UKR': 52, 'MDV': 53, 'BER': 54, 'IRI': 55,
'MAS': 56, 'PUR': 57, 'BUR': 58, 'NOR': 59, 'MAD': 60, 'LBR': 61,
'SLE': 62, 'KOR': 63, 'KSA': 64, 'MLI': 65, 'TGA': 66, 'ZAM': 67,
'OMA': 68, 'RUS': 69, 'NZL': 70, 'BAN': 71, 'CHI': 72, 'GAB': 73,
'NED': 74, 'BEN': 75, 'ARG': 76, 'AUT': 77, 'ISL': 78, 'CAY': 79,
'MRI': 80, 'SWE': 81, 'MEX': 82, 'FIJ': 83, 'GUI': 84, 'TCH': 85,
'KAZ': 86, 'LES': 87, 'PHI': 88, 'EUN': 89, 'COL': 90, 'GBS': 91,
'GUY': 92, 'HAI': 93, 'HKG': 94, 'TUR': 95, 'URU': 96, 'BOL': 97,
'PLW': 98, 'MLT': 99, 'SEY': 100, 'ESA': 101, 'BRN': 102, 'FIN': 103,
'FSM': 104, 'AZE': 105, 'ZIM': 106, 'TAN': 107, 'SLO': 108, 'KUW':
109, 'MTN': 110, 'PNG': 111, 'ANG': 112, 'ASA': 113, 'ISR': 114,
'VAN': 115, 'IND': 116, 'CAF': 117, 'COK': 118, 'LAO': 119, 'COM':
120, 'SRI': 121, 'SOL': 122, 'SMR': 123, 'TOG': 124, 'IVB': 125,
'SWZ': 126, 'ROU': 127, 'STP': 128, 'VIE': 129, 'WIF': 130, 'SUD':
131, 'AFG': 132, 'HON': 133, 'BOT': 134, 'GEQ': 135, 'GUA': 136,
'PER': 137, 'IRL': 138, 'IRQ': 139, 'LBN': 140, 'ETH': 141, 'BRU':
142, 'LIE': 143, 'BIZ': 144, 'MKD': 145, 'NCA': 146, 'MON': 147,
'MAW': 148, 'CAM': 149, 'ARU': 150, 'CRC': 151, 'CRO': 152, 'CZE':
153, 'UZB': 154, 'VIN': 155, 'TUV': 156, 'ECU': 157, 'ALG': 158,
'EGY': 159, 'ALB': 160, 'EST': 161, 'PLE': 162, 'GEO': 163, 'LUX':
164, 'PAR': 165, 'NEP': 166, 'MOZ': 167, 'MGL': 168, 'MAR': 169,
'MAL': 170, 'LTU': 171, 'GRN': 172, 'LCA': 173, 'LAT': 174, 'KIR':
175, 'JOR': 176, 'SVK': 177, 'GUM': 178, 'KGZ': 179

1.3 Algorithm

```python
def algorithm(X_train, y_train, alg, train_data_type):
    global all_result
    global X_test, y_test
    prediction = None
    if alg != 'kdtree':
        model_list = {
            'linear': LinearRegression(),
            'mlp': MLPRegressor(hidden_layer_sizes=(10, 10, 10, 10),
                                learning_rate_init=5e-6,
                                batch_size=len(X_train) // 10,
                                random_state=random_seed,
                                max_iter=5000),
            'svr': SVR(kernel='poly', degree=20, C=8.5),
        }
        supervised_model = model_list[alg]
        supervised_model.fit(X_train, y_train)
        prediction = supervised_model.predict(X_test)
        print(f'Score (training): {supervised_model.score(X_train, y_train) * 100}')
        print(f'Score (testing): {supervised_model.score(X_test, y_test) * 100}')
        print(f'Cross-validation: {cross_val_score(supervised_model, X_train, y_train, cv=5)}')
    else:
        unsupervised_model = KDTree(X_train)
        neighbor = list(int(ele[0]) for ele in unsupervised_model.query(X_test, return_distance=False))
        prediction = y_train[neighbor]

    result_label = alg + '_' + train_data_type
    all_result[result_label] = prediction
    result(y_test, prediction)
```

```python
def result(truth, prediction):
    thresholds = [0.1, 0.05, 0.01]
    error = prediction - truth
    MSE = np.sum(np.power(error, 2)) / error.shape[0]
    for threshold in thresholds:
        print(f'Accuracy under the threshold {threshold}: {len(truth[abs(truth - prediction) <= threshold]) / len(truth) * 100}')
    print(f'RMSE: {np.sqrt(MSE)}')
    compare = [truth, prediction]
    compare = np.array(compare)
    compare = compare.transpose()
    compare.sort(axis=0)
    x = list(range(len(compare)))
    Tplot = plt.scatter(x, compare[:, 0], s=2)
    Pplot = plt.scatter(x, compare[:, 1], s=2)
    plt.legend(handles=[Pplot, Tplot], labels=['y_pred', 'y_truth'])
    plt.show()
```

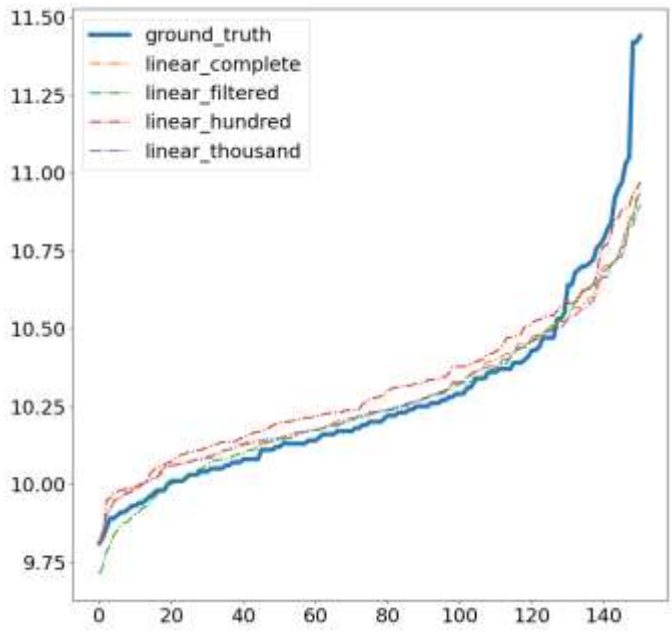
2 Result Visualization

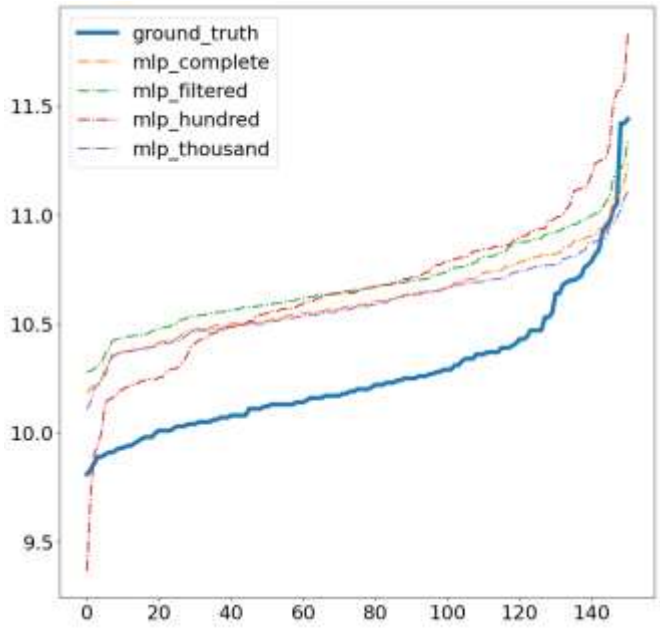
2.1 Source code

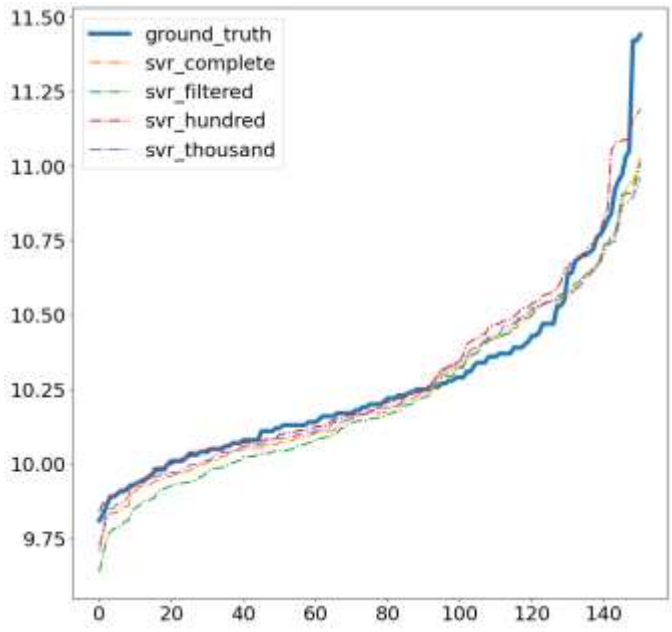
```python
def visualize_experiment(train_data_type=None, algorithm_type=None):
    global all_result
    if train_data_type:
        result_labels = ['ground_truth', 'linear_', 'svr_', 'mlp_', 'kdtree_']
        for i in range(1, len(result_labels)):
            result_labels[i] += train_data_type
    else:
        result_labels = ['ground_truth', '_complete', '_filtered', '_hundred', '_thousand']
        for i in range(1, len(result_labels)):
            result_labels[i] = algorithm_type + result_labels[i]

    handle = [all_result[label] for label in result_labels]
    handle = np.array(handle)
    handle.sort(axis=1)
    x = list(range(handle.shape[1]))

    plt.figure(figsize=(12, 12))
    for i in range(handle.shape[0]):
        if result_labels[i] == 'ground_truth':
            plt.plot(x, handle[i], label=result_labels[i], linestyle='-', linewidth=5)
        else:
            plt.plot(x, handle[i], label=result_labels[i], linestyle='-.', linewidth=2)

    plt.legend()
    plt.show()
```

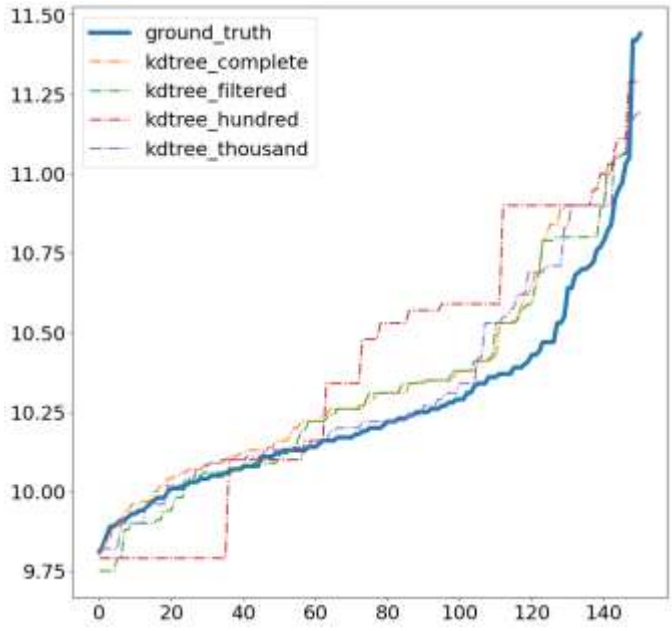
2.2 Complete results

2.2.1 Algorithms

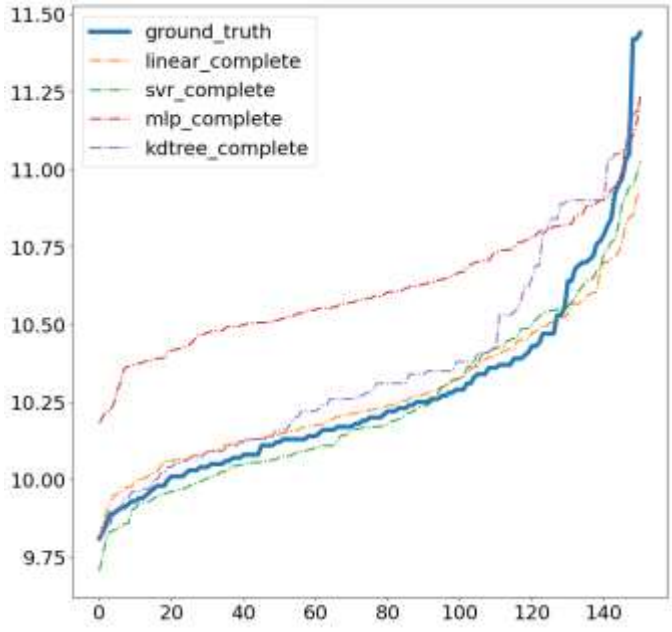


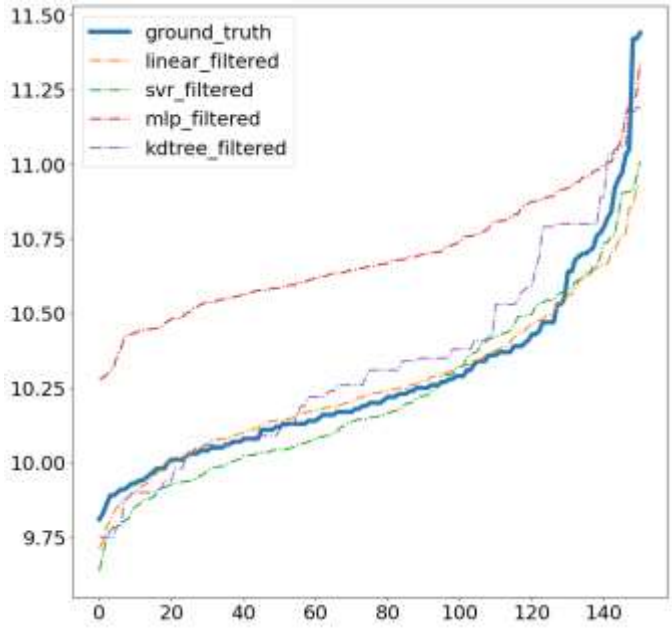

ground_truth
mlp_complete
mlp_filtered
mlp_hundred
mlp_thousand
11.5
11.0
10.5
10.0
9.5
0
20
40
60
80
100
120
140

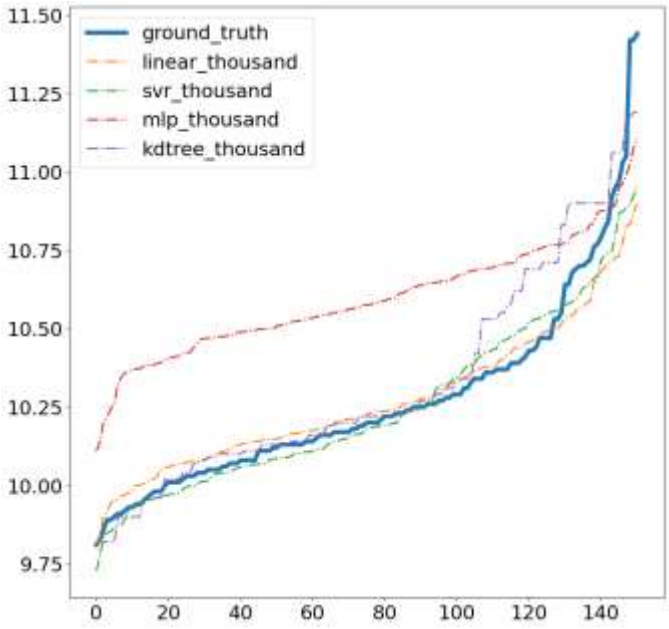

ground_truth
svr_complete
svr_filtered
svr_hundred
svr_thousand
11.50
11.25
11.00
10.75
10.50
10.25
10.00
9.75
0
20
40
60
80
100
120
140

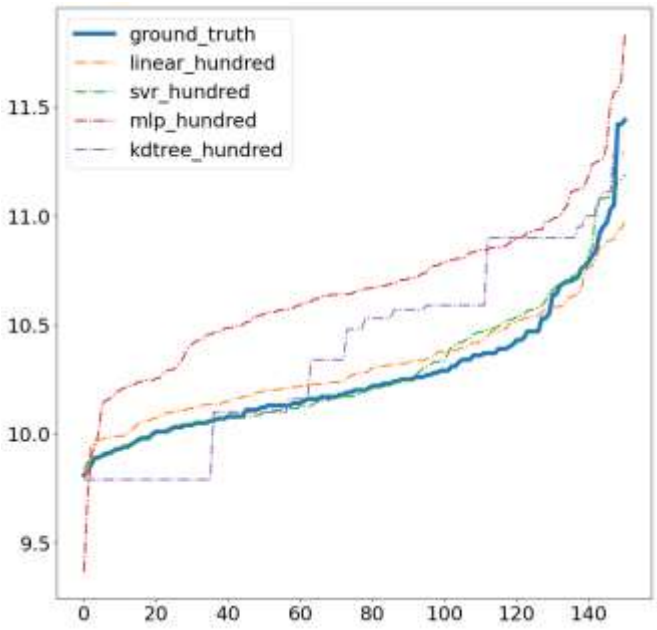

ground_truth
kdtree_complete
kdtree_filtered
kdtree_hundred
kdtree_thousand
11.50
11.25
11.00
10.75
10.50
10.25
10.00
9.75
0
20
40
60
80
100
120
140

2.2.2 Training data






ground_truth
linear_thousand
svr_thousand
mlp_thousand
kdtree_thousand
11.50
11.25
11.00
10.75
10.50
10.25
10.00
9.75
0
20
40
60
80
100
120
140


ground_truth
linear_hundred
svr_hundred
mlp_hundred
kdtree_hundred
11.5
11.0
10.5
10.0
9.5
0
20
40
60
80
100
120
140