Model Selection

Predicting Bacterial Property

1. The given dataset is tabular in nature and has a lot of datapoints representing different fields, we are on a potato pc, not giving us a lot of compute power.
2. A neural network won't be very suitable in this case, however access to gpu's may change it as explored later in this answer.
3. Assuming our compute resources are lacking, this can be the approach:
    a. We start with PCA dimensionality reduction, since the dataset has a lot of columns (150) a lot of them will be quite similar or follow the same patterns as they may be related. By

    b. We can then use StandardScaler to scale the data. Scaling ensures that all features have similar ranges, preventing any particular feature from dominating the model's learning process

    c. Averaging: If there are multiple instances of the same bacterial property in the dataset, averaging them can reduce noise.

    d. Computing the correlation matrix. Then very highly correlated columns can be identified and removed.

    e. Now we have the data processed and ready to fit in a model. For tabular data XGBoost proves to be a very accurate model. We can also use linear regression. The benefits and trade-offs are:

        i. Linear Regression

        Benefits:

        - Computationally efficient and can handle high dimensional data and works well for linear relations.
        - Gives weights of each parameter highlighting its importance

        Cons:

        - Doesn't capture non-linear relations accurately

        ii. XGBoosting:

        Benefits:

        - Can handle non linear relations
        - Highly accurate predictions

        Cons:

- Requires more compute power
- Require careful hyperparameter tuning and regularisation to prevent overfitting

f. Now we can calculate accuracy score.

In case we have GPU we can a neural network. It can be useful the data is extremely non linear and has complex relations. We can use smart weight initialisation strategies that can lead to faster training. Like assigning positive weights intitially for ReLU. [1]

Refer:
[1]https://papers.nips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[2]Efficient backprop

Yann A. LeCun, Léon Bottou, Genevieve B. Orr, Klaus Robert Müller

Predicting number of people on the beach

1. Data pre-processing- The given dataset is likely to have words instead of numbers in some fields, we can use one hot encoding for this. After each entry is numerical we will need to drop the fields which are empty.
2. Since most of the weather conditions are related we will use PCA.
3. Then we can choose to use Linear Regression or XGBoosting model.

Matrix Multiplication

1. Training a neural network for matrix multiplication can work but would not guarantee a cent percent accuracy, as the problem statement said, neural networks are very good at finding local minima’s quickly and this may give us a good accuracy for matrix multiplication too but it wont be exact.
2. In essence the difference between convention programming is as follows:
   a. Conventional programming structure:

      Input ——> Set of rules ——> Output

   b. Machine Learning:

      Input(Data) and Output ——> Set of rules ——> Output based on new data
3. Given that we are to train a neural network for it
   a. Speed:
      i. We can reduce the number of layers in the neural network.

        ii. Increase Batch size if infrastructure supports

    b. Accuracy:
        i. Hyperparameter Tuning: Changing parameters like batch size
        ii. Increasing the number of layers and width of each layer: Increasing the complexity gives the network more parameters and can aid in learning more complex patterns.
        iii. Ensemble Learning: We can train multiple neural networks with different architecture and then take the output as weighted mean or any other method. This will require much more compute and higher training time.
        iv. Add BatchNormalisation layers

In short it may not practical to use a nn for matrix multiplication which can simply be done by common libraries like numpy, or torch’s matmul as it won’t be able to capture the intricacies especially when we increase the dimensions. Although there is a chance that we may be able to predict accurately if we train it on a very large database, and I feel it needs to be tested out.

About the training speed, it should not take very long to train as the matrices are of low dimension and the resulting matrix multiplications(not the ones we are aiming for in the target) will be less. Presence of a gpu cluster can reduce the training time.