Invariant

Regardless of how the predictive system works, it will require a predictive function that is calibrated by minimizing an error function.

Goal

The job of the predictive system is to provide an estimation function \( f \) for what future data \( D_f \) will be based on the set of past and current data \( D \). Each element of this data \( d \) is a point in a multi-dimensional space where each dimension is a different feature \( p \in P \) such that \( d = (p_{time}, p_{latitude}, ...). \)

\[
f(p_{time}, p_{location}, ...) = d \in D
\]

Error

To get this function \( f \), we will train over sets of data by finding trying to predict known values at time \( t \) by only looking at data that is \( \Delta t \) older than it. We will form a \( f_{t-\Delta t} \) by only using past data \( D_{t-\Delta t} \) where \( p_{time} \leq t - \Delta t \) \( \forall p_{time} \in D \). Then we will try to minimize the error (root-mean-square error) of \( f_{t-\Delta t} \) across all data points in \( D_{t-\Delta t} \). Let \( p_{known} \) be the values that aren’t predicted such as time and location that are for \( f \) for one specific data point.

\[
Error(f_{t-\Delta t}(p_{known})) = \sqrt{\frac{1}{|P|} \sum_{p \in P} (d_{estimated} - d_{known})^2}
\]

Variant

Calculating \( d_{estimated} \) takes two parts, finding data points that are closely related to a given set of features and using those data points to estimate the future data point.

Nearest Data

Finding the set of \( k \) nearest data points to another data point \( d_n \) is a multi-dimensional spacial query. To find the nearest points to another data, we need to define what it means to be near.
Distance Function

Given two data points, \(a\) and \(b\), there are many ways of calculating the distance. The most basic type of distance function is the Euclidean distance between the two points.

\[
D_{\text{Euclid}}(a, b) = \sqrt{\sum_{p \in P} (a_p - b_p)^2}
\]

To calibrate this distance function, we can introduce a scalar vector \(m\) with a value for every feature where every \(m_{\text{feature}}\) is independent of one another. With this, we can get a weighted distance.

\[
D_{\text{Weighted}}(a, b) = D_{\text{Euclid}}(a_p \times m_p \forall p \in P, b_p \times m_p \forall p \in P)
\]

By tweaking each \(m_{\text{feature}}\), we change what weight of certain features are in the data. One of the main disadvantages of this distance algorithm is that it doesn’t allow for multiple features to be incorporated together as each feature is independently measured. While there are many more different and more intricate ways of calculating the distance function, but we will start with this for now.

Query Item

Because of the way the distance function works, to query for a data point, our query has to have the same features represented as the expected output(s). Due to the nature of predictions, however, we will only know a subset of the values for the features in advance such as time and location. This constraint leads to two solutions, restricting data’s feature set or sampling the query.

By restricting the features, the distances between data points will be based exclusively off features that are always known. Because of this, measured features won’t factor in the distance function. However, measured features will still be taken into account for the error function, and so they factor in while calibrating \(m\).

Alternatively, the query can take the other features in account by setting random values for each of the measured features by sampling from the corresponding distribution of values, querying multiple times, and taking the most often occurring data points from the queries. While method allows for the distance function between data points to factor in all of the features, it
will be heavily dependent on the way the random sampling works, assuming
the measured features are significant.

The functionality for the former algorithm is required for the latter, but
not the other way around, so we will start with the restricted feature algo-

**Combining**

Once the distance query finds the nearest data points, they need to be com-
bined to calculate the final data point. One major advantage of the combining
function is that it can take into account data that is more complicated to
quantitatively account for such as direction. The first, a simple way of cal-
culating $d_{t+\Delta t}$ given its k-nearest neighbors, $D_k$ is to take a weighted average
of the data points where the weight is dependent on how close the point is
to the query point. A more sophisticated method can incorporate a vector
of scalars, $n$, which behaves much like $m$ does for the distance. Like $m$, $n$
can also be calibrated with the feedback of the error function. Neither
of these two combining methods are guaranteed to be particularly effective,
however because most combining functions will require the functionality from
the weighted average algorithm, that is the first one we will implement.