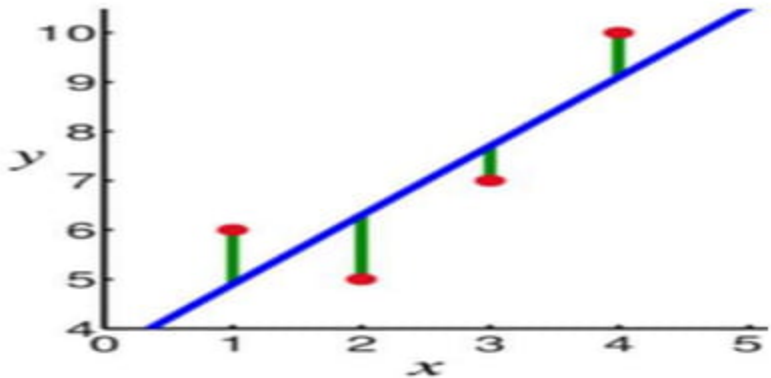
Linear models

- Linear models are relatively simple. In this case, the function is represented as a linear combination of its inputs.
- Thus, if x_1 and x_2 are two scalars or vectors of the same dimension and a and b are arbitrary scalars, then $ax_1 + bx_2$ represents a linear combination of x_1 and x_2.
- In the simplest case where $f(x)$ represents a straight line, we have an equation of the form $f(x) = mx + c$ where c represents the intercept and m represents the slope.



- Linear models are **parametric**, which means that they have a fixed form with a small number of numeric parameters that need to be learned from data.
- For example, in $f(x) = mx + c$, m and c are the parameters that we are trying to learn from the data.
- This technique is different from tree or rule models, where the structure of the model (e.g., which features to use in the tree, and where) is not fixed in advance.
- Linear models are **stable**, i.e., small variations in the training data have only a limited impact on the learned model.

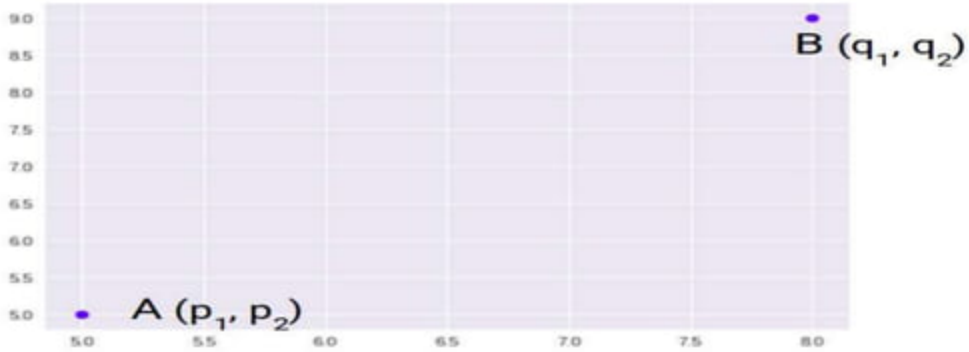
- In contrast, **tree models tend to vary more with the training data**, as the choice of a different split at the root of the tree typically means that the rest of the tree is different as well.
- As a result of having relatively few parameters, Linear models have **low variance and high bias**.
  - Data **bias in machine learning** is a type of error in which certain elements of a dataset are more heavily weighted and/or represented than others. A **biased** dataset does not accurately represent a model's use case, resulting in skewed outcomes, low accuracy levels, and analytical errors.
  - **Low Variance**: Suggests small changes to the estimate of the target function with changes to the training dataset. High **Variance**: Suggests large changes to the estimate of the target function with changes to the training dataset.

- This implies that **Linear models are less likely to overfit the training data** than some other models.
- However, they are more likely to underfit. For example, if we want to learn the boundaries between countries based on labelled data, then linear models are not likely to give a good approximation.

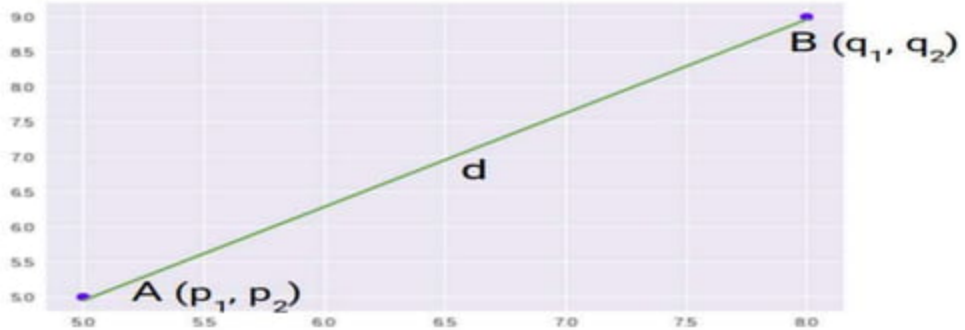
Distance-based models

- **Distance-based models** are the second class of Geometric models. Like Linear models, distance-based models are based on the geometry of data.
- As the name implies, distance-based models work on the concept of distance.
- In the context of Machine learning, the concept of distance is not based on merely the physical distance between two points. Instead, we could think of the distance between two points considering the **mode of transport** between two points.
- Travelling between two cities by plane covers less distance physically than by train because a plane is unrestricted. Similarly, in chess, the concept of distance depends on the piece used – for example, a Bishop can move diagonally.
- Thus, depending on the entity and the mode of travel, the concept of distance can be experienced differently. The distance metrics commonly used are **Euclidean**, **Manhattan etc..**

- **1. Euclidean Distance**
- *Euclidean Distance represents the shortest distance between two points.*
- Let's say we have two points as shown below:



- So, the Euclidean Distance between these two points A and B will be:



- Here's the formula for Euclidean Distance:

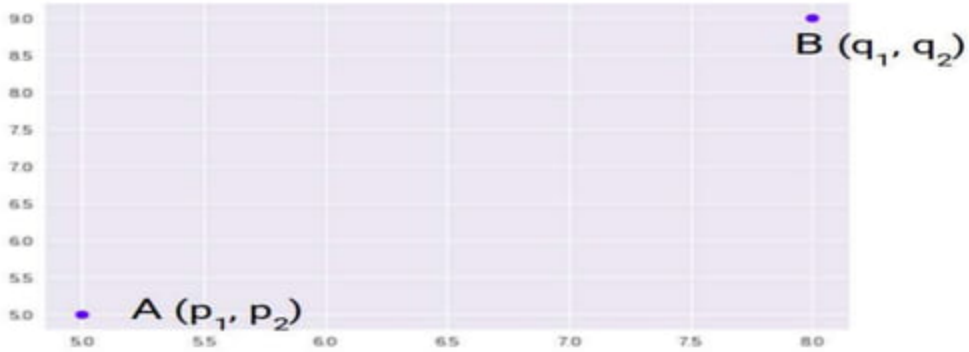
$$d = ((p_1 - q_1)^2 + (p_2 - q_2)^2)^{1/2}$$

- We use this formula when we are dealing with 2 dimensions. We can generalize this for an n-dimensional space as:

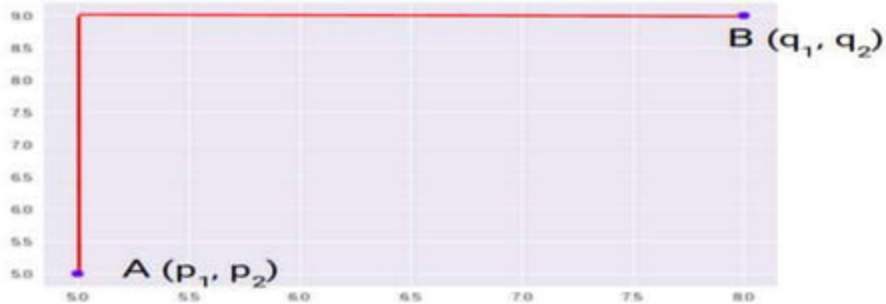
$$D_e = \left(\sum_{i=1}^{n} (p_i - q_i)^2 \right)^{1/2}$$

- Where,
  - n = number of dimensions
  - pi, qi = data points

- **2. Manhattan Distance**
- *Manhattan Distance is the sum of absolute differences between points across all the dimensions*
- Let's say we have two points as shown below:



- We can represent Manhattan Distance as:



- Since the above representation is 2 dimensional, to calculate Manhattan Distance, we will take the sum of absolute distances in both the x and y directions. So, the Manhattan distance in a 2-dimensional space is given as:

$$d = |p_1 - q_1| + |p_2 - q_2|$$

- And the generalized formula for an n-dimensional space is given as:

$$D_m = \sum_{i=1}^{n} |p_i - q_i|$$

- Notes:
  - The **centroid** represents the geometric centre of a plane figure, i.e., the arithmetic mean position of all the points in the figure from the centroid point. This definition extends to any object in *n*-dimensional space: its centroid is the mean position of all the points.
  - **Medoids** are similar in concept to means or centroids. Medoids are most commonly used on data when a mean or centroid cannot be defined. They are used in contexts where the centroid is not representative of the dataset, such as in image data.
- Examples of distance-based models include the **nearest-neighbour** models, which use the training data as exemplars – for example, in classification. The **K-means clustering** algorithm also uses exemplars to create clusters of similar data points.

Probabilistic models

- The third family of machine learning algorithms is the probabilistic models.
- The k-nearest neighbour algorithm (in Unit 2) uses the idea of distance (e.g., Euclidian distance) to classify entities, and logical models use a logical expression to partition the instance space.
- In this section, we see how the **probabilistic models use the idea of probability to classify new entities.**
- Probabilistic models see features and target variables as random variables. The process of modelling represents and **manipulates the level of uncertainty** with respect to these variables.
- There are two types of probabilistic models: **Predictive and Generative**.

- Predictive probability models use the idea of a **conditional probability** distribution $P(Y|X)$ from which Y can be predicted from X.
- Generative models estimate the **joint distribution** $P(Y, X)$. Once we know the joint distribution for the generative models, we can derive any conditional or marginal distribution involving the same variables.
- Thus, the generative model is capable of creating new data points and their labels, knowing the joint probability distribution.
- The joint distribution looks for a relationship between two variables. Once this relationship is inferred, it is possible to infer new data points.

- **Naïve Bayes** is an example of a probabilistic classifier.
- We can do this using the **Bayes rule** defined as

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

- The Naïve Bayes algorithm is based on the idea of **Conditional Probability. Conditional probability is based on finding the** probability that something will happen, *given that something else* has already happened. The task of the algorithm then is to look at the evidence and to determine the likelihood of a specific class and assign a label accordingly to each entity.

Some broad categories of models

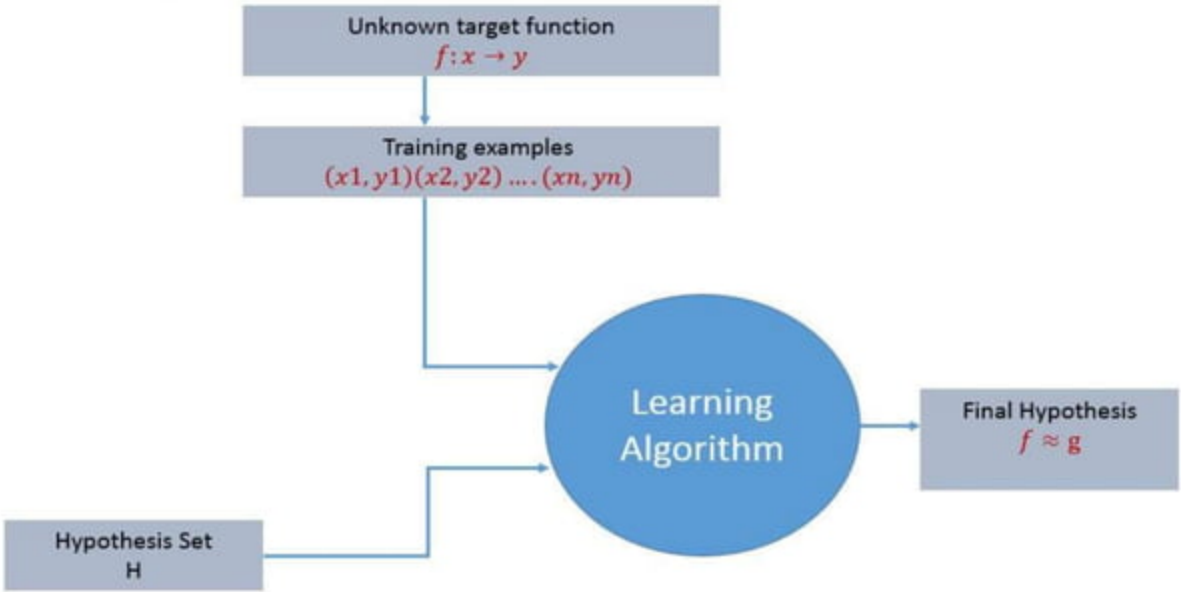
Geometric models	Probabilistic models	Logical models
E.g. K-nearest neighbors, linear regression, support vector machine, logistic regression, …	Naïve Bayes, Gaussian process regression, conditional random field, …	Decision tree, random forest, …

Grouping and Grading

- Grading vs grouping is an orthogonal categorization to geometric-probabilistic-logical-compositional.
- Grouping models break the instance space up into groups or segments and in each segment apply a very simple method (such as majority class).
  - E.g. decision tree, KNN.
- Grading models form one global model over the instance space.
  - E.g. Linear classifiers – Neural networks

1.4 Designing a Learning System

- For any learning system, we must be knowing the three elements — **T (Task)**, **P (Performance Measure)**, and **E (Training Experience)**. At a high level, the process of learning system looks as below.



- The learning process starts with task T, performance measure P and training experience E and objective are to find an unknown target function.
- The target function is an exact knowledge to be learned from the training experience and its unknown. For example, in a case of credit approval, the learning system will have customer application records as experience and task would be to classify whether the given customer application is eligible for a loan.
- So in this case, the training examples can be represented as (x1,y1)(x2,y2)..(xn,yn) where X represents customer application details and y represents the status of credit approval.
- the target function to be learned in the credit approval learning system is a mapping function **f:X →y.**

Components of Designing a Learning System

- The design choices will be to decide the following key components:
  - Type of training experience
  - Choosing the Target Function
  - Choosing a representation for the Target Function
  - Choosing an approximation algorithm for the Target Function
  - The final Design

Components of Designing a Learning System

- We will look into the game - checkers learning problem and apply the above design choices.
- For a checkers learning problem, the three elements will be,

1. ***Task T**: To play checkers*
2. ***Performance measure P**: Total percentage of the game won in the tournament.*
3. ***Training experience E**: A set of games played against itself*

1. Type of training experience

- During the design of the checker's learning system, the type of training experience available for a learning system will have a significant effect on the success or failure of the learning.
- Three attributes are there for choosing the type of training experience, They are
  - **Direct or Indirect training experience**
  - **Teacher or Not**
  - **Is the training experience good**

- **Direct or Indirect training experience —** In the case of direct training experience, an individual board states and correct move for each board state are given. In case of indirect training experience, the move sequences for a game and the final result (win, loss or draw) are given for a number of games.

Teacher or Not —

- Supervised — The training experience will be labeled, which means, all the board states will be labeled with the correct move.
- Unsupervised — The training experience will be unlabeled, which means, all the board states will not have the moves.
- Semi-supervised — Learner generates game states and asks the teacher for help in finding the correct move if the board state is confusing.

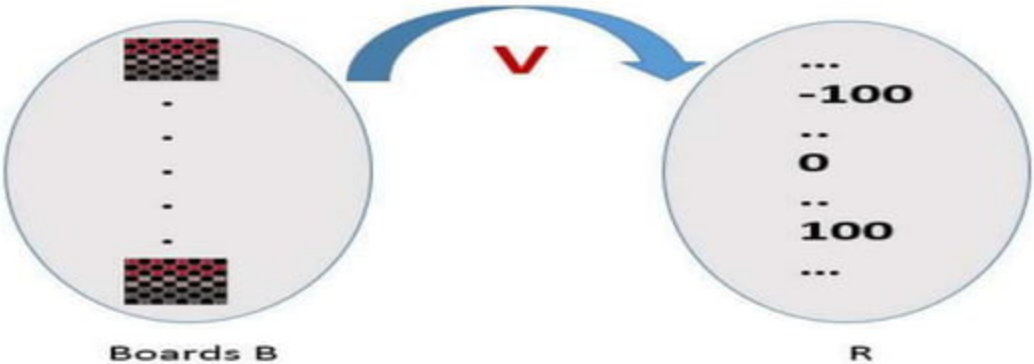
- **Is the training experience good —** Do the training examples represent the distribution of examples over which the final system performance will be measured? Performance is best when training examples and test examples are from the same/a similar distribution.

2. Choosing the Target Function

- When you are playing the checkers game, at any moment of time, you make a decision on choosing the best move from different legal possibilities move.
- You think and apply the learning that you have gained from the experience.
- Here the learning is, for a specific board, you move a checker such that your board state tends towards the winning situation.
- Now the same learning has to be defined in terms of the target function.
- Here there are 2 considerations
  - direct and indirect experience.

- **During the direct experience**, the checkers learning system, it needs only to learn how to choose the best move among some large search space.
- We need to find a target function that will help us choose the best move among alternatives.
- Let us call this function ChooseMove and use the notation **ChooseMove : B →M** to indicate that this function accepts as input any board from the set of legal board states B and produces as output some move from the set of legal moves M.
- **When there is an indirect experience**, it becomes difficult to learn such function. How about assigning a real score to the board state.

- So an alternate target function is evaluation function that assign numerical score to any given board state. Let target function V and notation be **V : B →R** indicating that this accepts as input any board from the set of legal board states B and produces an output a real score. This function assigns the higher scores to better board states.



- If the system can successfully learn such a target function V, then it can easily use it to select the best move from any board position.

- Let us therefore define the target value V(b) for an arbitrary board state b in B, as follows:
  - 1. **if b is a final board state that is won, then V(b) = 100**
  - **2. if b is a final board state that is lost, then V(b) = -100**
  - **3. if b is a final board state that is drawn, then V(b) = 0**
  - **4. if b is a not a final state in the game, then V (b) = V (b'),**

where b' is the best final board state that can be achieved starting from b and playing optimally until the end of the game.

3. Choosing a representation for the Target Function

- The more training data the program will require in order to choose among the alternative hypotheses it can represent.
- let us choose a simple representation:
- for any given board state, the function ^V will be calculated as a linear combination of the following board features:
  - x1(b) — number of black pieces on board b
  - x2(b) — number of red pieces on b
  - x3(b) — number of black kings on b
  - x4(b) — number of red kings on b
  - x5(b) — number of red pieces threatened by black (i.e., which can be taken on black's next turn)
  - x6(b) — number of black pieces threatened by red

3. Choosing a representation for the Target Function

- $\hat{V} = w_0 + w_1 \cdot x_1(b) + w_2 \cdot x_2(b) + w_3 \cdot x_3(b) + w_4 \cdot x_4(b) + w_5 \cdot x_5(b) + w_6 \cdot x_6(b)$

- Where w0 through w6 are numerical coefficients or weights to be obtained by a learning algorithm. Weights w1 to w6 will determine the relative importance of different board features.

3. Choosing a representation for the Target Function

- The checkers learning task can be summarized as below.
  - **Task T : Play Checkers**
  - **Performance Measure P : % of games won in world tournament**
  - **Training Experience E : opportunity to play against itself**
  - **Target Function : V : Board → R**
  - **Target Function Representation :**
    - $\hat{V} = w0 + w1 \cdot x1(b) + w2 \cdot x2(b) + w3 \cdot x3(b) + w4 \cdot x4(b) + w5 \cdot x5(b) + w6 \cdot x6(b)$
- The first three items above correspond to the specification of the learning task,whereas the final two items constitute design choices for the implementation of the learning program.

4. Choosing an approximation algorithm for the Target Function

- To train our learning program, we need a set of training data, each describing a specific board state b and the training value V_train (b) for b.
- Each training example is an ordered pair (b,V_train(b)).
- For example, a training example may be
- (x1 = 3, x2 = 0, x3 = 1, x4 = 0, x5 = 0, x6 = 0), +100).
- This is an example where black has won the game since x2 = 0 or red has no remaining pieces. However, such clean values of V_train (b) can be obtained only for board value b that are clear win, loss or draw.

4. Choosing an approximation algorithm for the Target Function

- Function Approximation Procedure
  - Derive training examples from the indirect training experience available to the learner.
  - Adjust the weight w_i to best fit these training example.

4. Choosing an approximation algorithm for the Target Function

- **Estimating Training Values**:
  - Let Successor(b) denotes the next board state following b for which it is again the program's turn to move. ^V is the learner's current approximation to V. Using these information, assign the training value of V_train(b) for any intermediate board state b as below
  - **V_train(b) ← ^V(Successor(b))**

4. Choosing an approximation algorithm for the Target Function

- **Adjusting the weights**:
  - Now its time to define the learning algorithm for choosing the weights and best fit the set of training examples. {(b, V_{train} (b))}.
  - One common approach is to define the best hypothesis as that which minimizes the squared error E between the training values and the values predicted by the hypothesis ^V.

$$E \equiv \sum_{\langle b, V_{train}(b) \rangle \in training\ examples} (V_{train}(b) - \hat{V}(b))^2$$

4. Choosing an approximation algorithm for the Target Function

- The learning algorithm should incrementally refine weights as more training examples become available and it needs to be robust to errors in training data Least Mean Square (LMS) training rule is the one training algorithm that will adjust weights a small amount in the direction that reduces the error.
- The LMS algorithm is defined as follows:

For each training example b

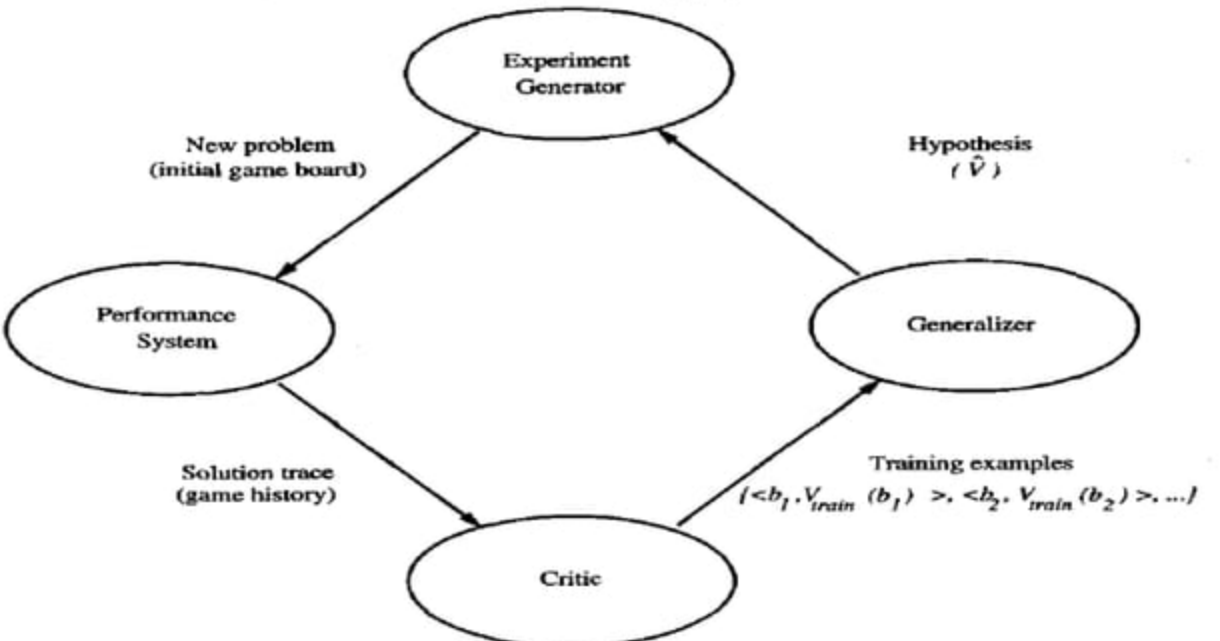
1. Compute $error(b) = V_{train}(b) - \hat{V}(b)$
2. for each board feature x_i, update weight w_i

$$w_i \leftarrow w_i + \mu . error(b) . x_i$$

Here μ is a small constant (e.g., 0.1) that moderates the size of the weight update

5. Final Design for Checkers Learning system

- The final design of our checkers learning system can be naturally described by four distinct program modules that represent the central components in many learning systems.



- The performance System — Takes a new board as input and outputs a trace of the game it played against itself.
- The Critic — Takes the trace of a game as an input and outputs a set of training examples of the target function.
- The Generalizer — Takes training examples as input and outputs a hypothesis that estimates the target function. Good generalization to new cases is crucial.
- The Experiment Generator — Takes the current hypothesis (currently learned function) as input and outputs a new problem (an initial board state) for the performance system to explore.