Data Analysis Mini-Project: Learning with Explanations

Tal Marciano 315505644

Alon Yaakov Ben-Melech 313184178

Part A:

In our code we implemented several classes to model the behaviors of different parts of the algorithm:

The main part of the algorithm is written in a run function and given the data and an instance of some type of teacher

We implemented an abstract teacher to be implemented in different ways and a data class for the teacher answer.

Part A

The main loop:

$and\ c$ which is a list of Defines L as an empty list and fills it with tuples that hold $\hat{x}$, $\hat{y}$ conjunctions.

On each iteration we find an example which has a conjunction that is satisfied, predict by it using the given teacher.

If no such example is found, we predict using the default x and y.

Part A: Teacher abstract class:

```python
class Teacher(ABC):
    X: np.array
    y: np.array

    def __init__(self, X: np.array, y: np.array):
        self.X = X
        self.y = y

    @staticmethod
    def _discriminative_feature(X: np.array, X_hat: np.array) -> List[tuple]:
        d = len(X)
        res = []
        for i in range(d):
            if X[i] == 1 - X_hat[i]:
                res.append(tuple([i, X[i]]))
        return res

    @abstractmethod
    def teach(self, X: np.array, y: int, l: tuple) -> TeacherFeedBack:
        pass
```

Part A: SimpleTeacher

```python
class SimpleTeacher(Teacher):

    def teach(self, X: np.array, y: int, l: tuple) -> TeacherFeedBack:
        if y == l[1]:
            return TeacherFeedBack(True, None)
        discriminative_features: List[tuple] = self._discriminative_feature(X, l[0])
        feedback = FeedBack(_X, y, [random.choice(discriminative_features)])
        return TeacherFeedBack(False, feedback)
```

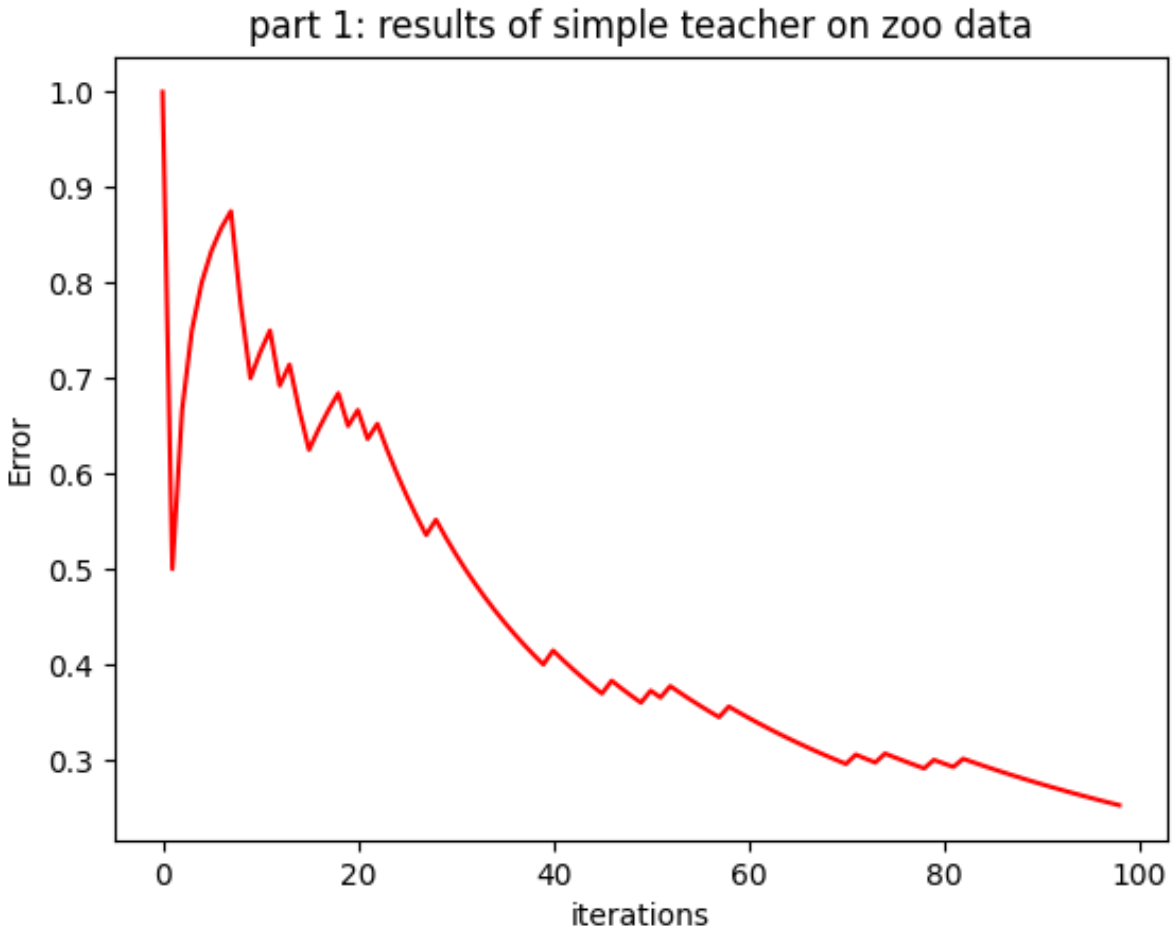
Part A: DiscriminativeTeacher

```python
class DiscriminativeTeacher(Teacher):

    def _getP(self, feature, label):
        label_indices = np.where(self.y == label + 1)[0]
        term1 = len([ind for ind in label_indices if self.X[ind][feature] == 1])

        p = (term1 / len(label_indices)) * 100
        return p

    def _build_Pmatrix(self):
        num_of_labels = len(np.unique(self.y))
        num_of_features = self.X[0].shape[0]
        Pmatrix = np.zeros((num_of_features, num_of_labels))
        for i in range(num_of_features):
            for j in range(num_of_labels):
                Pmatrix[i, j] = self._getP(i, j)
        return Pmatrix
```

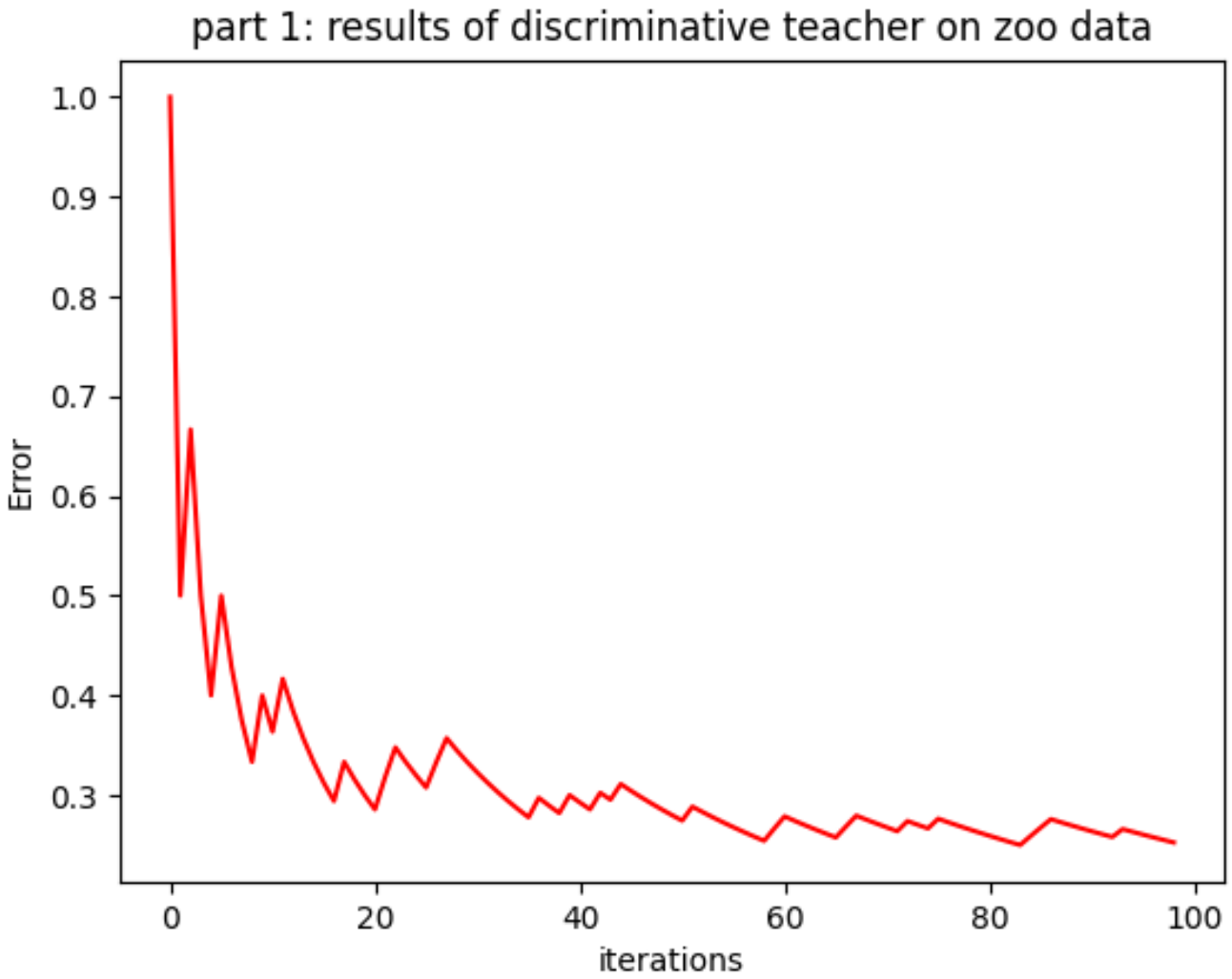
Part A: TeacherFeedBack

```python
@dataclass
class FeedBack:
    X: np.array
    y: int
    c: List[tuple]

    def get_phi(self) -> tuple:
        return self.c[0][0], self.c[0][1]

    def get_not_phi(self) -> tuple:
        return self.c[0][0], 1-self.c[0][1]

@dataclass
class TeacherFeedBack:
    student_answer: bool
    feed_back: FeedBack
```

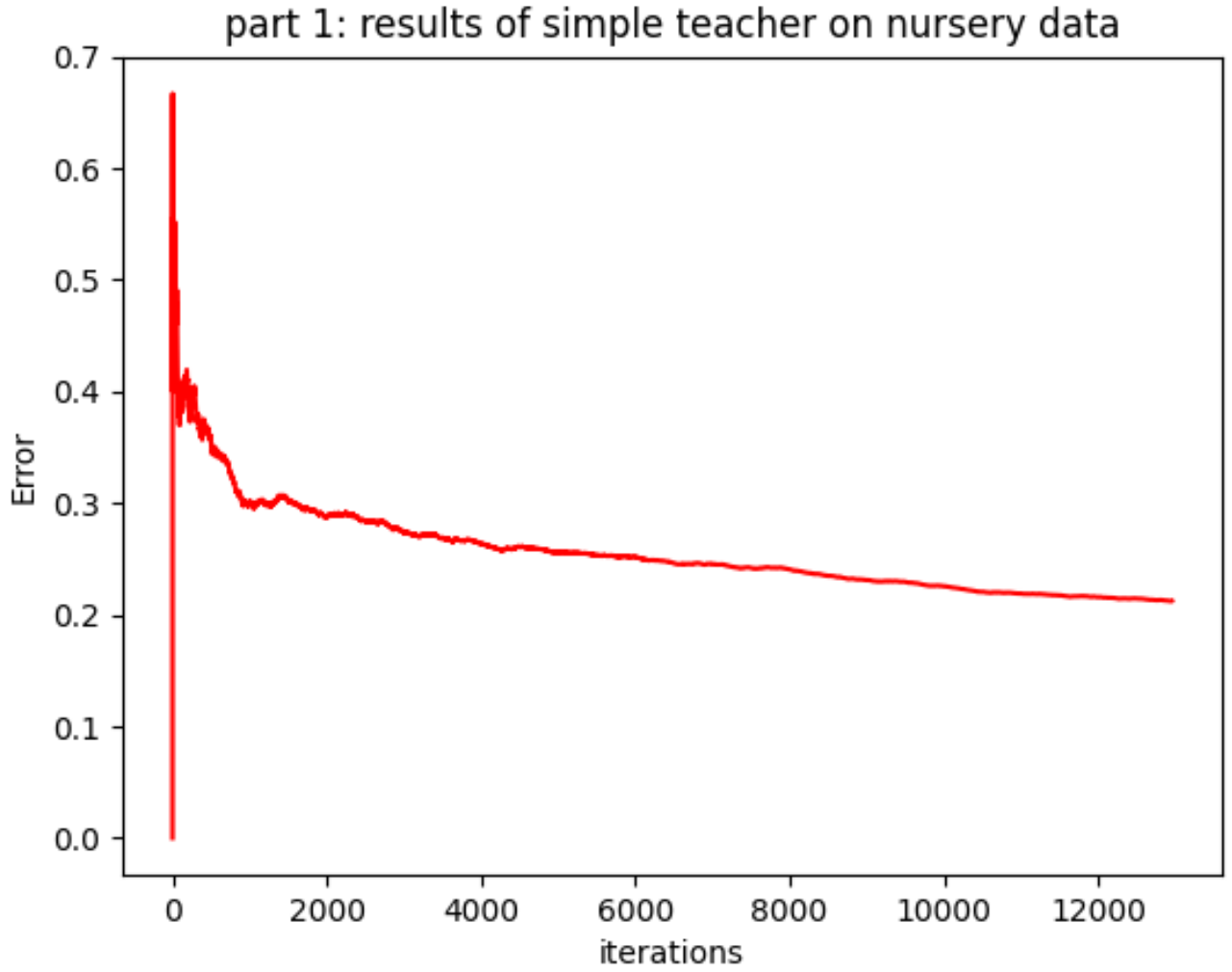
Part A: Graphs



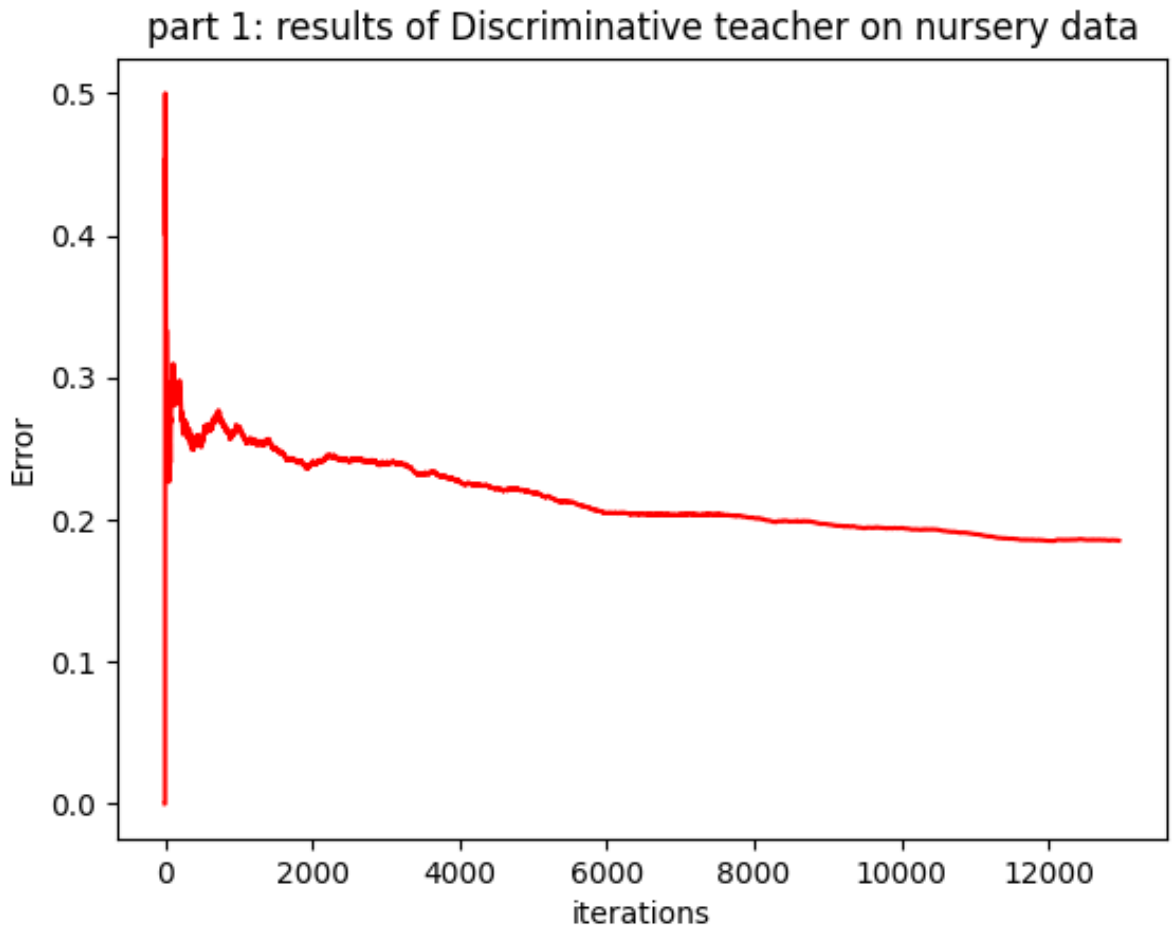
Part A: Graphs



Part A: Graphs



Part A: Graphs



Part A: Decision List for simple teacher on zoo data

decision list

does X satisfies [(20, 0), (9, 0), (14, 1)] ? --> label = [4]

|

|

V

does X satisfies [(5, 1), (4, 0), (12, 1), (11, 0), (17, 1), (3, 0)] ? --> label = [5]

|

|

V

does X satisfies [(14, 1), (3, 1)] ? --> label = [1]

|

|

V

does X satisfies [(1, 1)] ? --> label = [2]

|

|

V

does X satisfies [(6, 0), (12, 0), (0, 0), (8, 0), (15, 0)] ? --> label = [6]

|

|

V

does X satisfies [(10, 0), (0, 1)] ? --> label = [1]

|

|

V

does X satisfies [(8, 1), (17, 1), (5, 1)] ? --> label = [5]

decision list

does X satisfies [(9, 0), (15, 1), (12, 1)] ? --> label = [4]

|

|

V

does X satisfies [(8, 1)] ? --> label = [3]

|

|

V

does X satisfies [(19, 1), (5, 0)] ? --> label = [6]

|

|

V

unknown label

Part A: Decision List for discriminative teacher on zoo data

does X satisfies [(9, 1), (1, 0), (3, 0), (2, 1), (15, 0), (8, 0)] ? --> label = [6]

|

|

V

does X satisfies [(1, 1)] ? --> label = [2]

|

|

V

does X satisfies [(3, 1)] ? --> label = [1]

|

|

V

does X satisfies [(8, 1), (2, 0)] ? --> label = [3]

|

|

V

does X satisfies [(7, 1), (11, 0), (15, 0), (5, 1)] ? --> label = [5]

|

|

V

does X satisfies [(7, 1), (9, 0)] ? --> label = [4]

|

|

V

unknown label

Part A: part of the Decision List for Simple teacher on nursery data

does X satisfies [(15, 0), (17, 0), (0, 1), (10, 0), (21, 1), (26, 1), (14, 0), (11, 0)] ? --> label = [2]

|

|

v

does X satisfies [(0, 0), (1, 0), (7, 1), (15, 0), (20, 0), (12, 0), (16, 1), (21, 0), (22, 0), (10, 0), (24, 0), (13, 1), (26, 1)] ? --> label = [2]

|

|

v

does X satisfies [(22, 0), (25, 1), (6, 0), (15, 1), (17, 1), (20, 0), (23, 0), (0, 1)] ? --> label = [3]

|

|

v

does X satisfies [(5, 1), (13, 1), (21, 1), (2, 0), (19, 1), (8, 0), (26, 1), (16, 0), (9, 0)] ? --> label = [2]

|

|

v

does X satisfies [(15, 0), (24, 0), (19, 0), (3, 0), (2, 1), (13, 1), (7, 0), (18, 0), (23, 1), (8, 0), (16, 0)] ? --> label = [2]

|

|

v

does X satisfies [(13, 1), (22, 0), (26, 1), (20, 0), (9, 1), (6, 1), (23, 0), (17, 0)] ? --> label = [4]

|

|

v

does X satisfies [(9, 0), (1, 0), (8, 0), (24, 0), (23, 0), (21, 0), (15, 0), (16, 0), (19, 1), (17, 0), (25, 0), (11, 0), (13, 0), (14, 0), (2, 0)] ? --> label = [3]

.....

.....

does X satisfies [(24, 0), (9, 1), (22, 1), (4, 1)] ? --> label = [2]

|

|

V

does X satisfies [(3, 0), (24, 0), (0, 1)] ? --> label = [3]

|

|

V

does X satisfies [(21, 1), (13, 1), (4, 0), (5, 0), (19, 0)] ? --> label = [3]

|

|

V

does X satisfies [(3, 0), (16, 0), (0, 0), (13, 0), (10, 1), (17, 0)] ? --> label = [2]

|

|

V

does X satisfies [(26, 1), (1, 0)] ? --> label = [2]

|

|

V

does X satisfies [(2, 1)] ? --> label = [3]

|

|

V

unknown label

Part A: A part of the decision List for discriminative teacher on nursery data

does X satisfies [(26, 1), (0, 0), (17, 1), (22, 0), (20, 1), (12, 0), (7, 0), (3, 0)] ? --> label = [2]

|

|

V

does X satisfies [(2, 0), (20, 1), (24, 0), (0, 0), (22, 1), (25, 1), (7, 0), (17, 0), (18, 1), (3, 0), (10, 0), (14, 0), (11, 1), (4, 0)] ? --> label = [2]

|

|

V

does X satisfies [(25, 1), (19, 0), (2, 0), (16, 1), (5, 1), (21, 1), (12, 0), (1, 0), (8, 0), (9, 0)] ? --> label = [3]

|

|

V

does X satisfies [(25, 1), (21, 1), (2, 0), (5, 0), (19, 1), (16, 0), (1, 1), (18, 0), (6, 0), (12, 0)] ? --> label = [3]

|

|

V

does X satisfies [(26, 1), (20, 1), (22, 0), (0, 1), (17, 0), (14, 0), (7, 0), (18, 0), (3, 0), (21, 0), (4, 0)] ? --> label = [2]

|

|

V

does X satisfies [(26, 1), (7, 0), (0, 1), (20, 0), (22, 0), (10, 1), (3, 0), (17, 0), (21, 0), (13, 0), (14, 0), (4, 0)] ? --> label = [2]

• • •

...

does X satisfies [(22, 1), (24, 0), (16, 0), (19, 0)] ? --> label = [3]

|

|

V

does X satisfies [(26, 1), (0, 0), (22, 1)] ? --> label = [2]

|

|

V

does X satisfies [(26, 1), (16, 1), (20, 1), (7, 0)] ? --> label = [2]

|

|

V

does X satisfies [(17, 1), (24, 0), (19, 0)] ? --> label = [3]

|

|

V

does X satisfies [(26, 1), (20, 1)] ? --> label = [2]

|

|

V

does X satisfies [(25, 1)] ? --> label = [3]

|

|

V

does X satisfies [(26, 1), (16, 1)] ? --> label = [4]

|

|

V

unknown label

Part A: comparison between teachers

After running every option 10 times and calculating the average error:

error average of simple teacher on zoo data: 26.868686868686872%

error average of discriminative teacher on zoo data: 21.212121212121215%

simple teacher on nursery data: 24.659669702114524%

discriminative teacher on nursery data: 19.48448346967126%

we can confirm what we previously predicted:

The discriminative teacher is in fact, better.

Its error is smaller in the vast majority of the runs on both data sets, and intuitively it make choices that are not completely random like the simple teacher.

Part B: Chosen Data Set

We chose to work with a mushroom data set

The data set details 22 mushroom properties:

```
mushrooms_names = ["label", "cap-Shape", "cap_surface", "cap_color", "bruises?", "odor", "gill-attachment",
                   "gill-spacing",
                   "gill-size", "gill-color", "stalk-shape", "stalk-root", "stalk-surface-above-ring",
                   "stalk-surface-below-ring", "stalk-color-above-ring", "stalk-color-below-ring",
                   "veil-type", "veil-color", "ring-number", "ring-type", "spore-print-color", "population",
                   "habitat"]
```

And classifies each mushroom as edible or poisonous

This dataset is suitable for classification and is Multivariate.

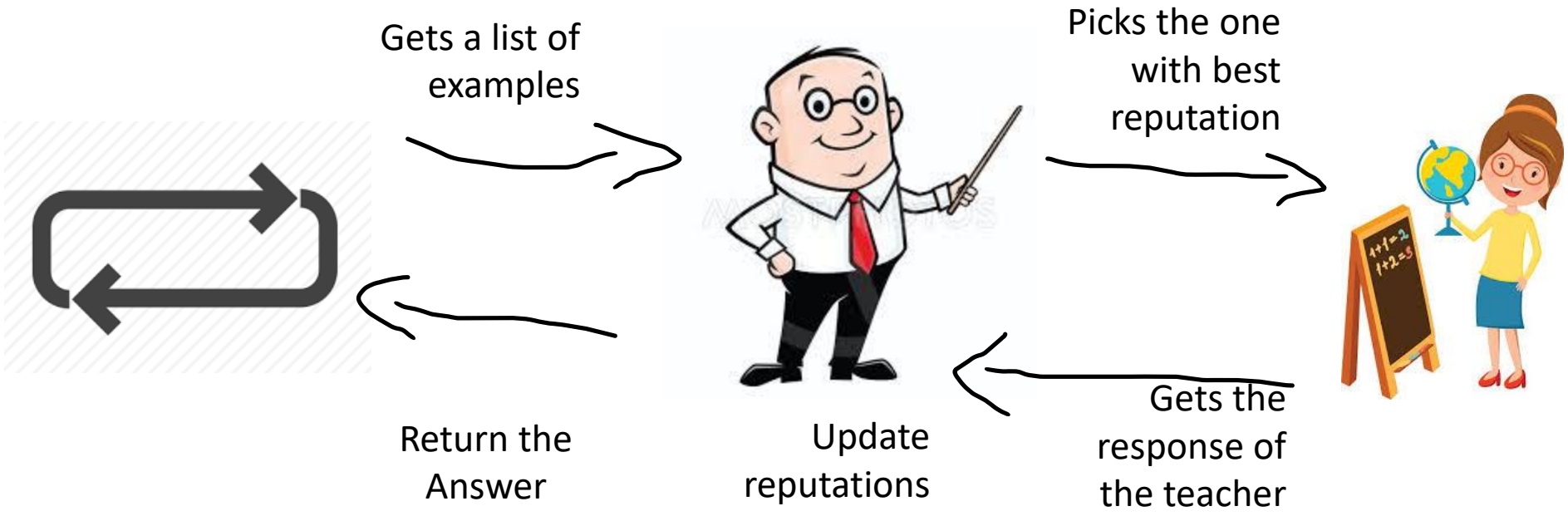
Our intuition




We think that some examples are better than others.
The octopus has a lot less mutual features with other animals than the horse,
So we believe that choosing the horse to predict will do better.

New class: Teacher assistant

Acts as a middle man between the teacher and the main loop.



The Teacher assistance is also an abstract class, meant to be implemented in several ways.

Part B: The way we tried to improve the algorithm

```python
class TeacherAssistance(ABC):
    X: np.array
    y: np.array
    teacher: Teacher
    reputation_map: np.array
    alpha: float

    def __init__(self, X: np.array, y: np.array, teacher: Teacher, alpha: int):
        m = len(X)
        self.X = X
        self.y = y
        self.teacher = teacher
        self.alpha = alpha
        self.reputation_map = np.full(m, 1 / m)

    @abstractmethod
    def _get_probability_vec(self, X_indices) -> List[float]:
        pass

    @abstractmethod
    def _set_reputation(self, response, chosen_x_ind):
        pass

    def assist(self, X: np.array, y: int, L_l: List[tuple]) -> TAFeedBack:
        X_indices = [np.flatnonzero((self.X == l[0]).all(1))[0] for l in L_l]
        P = self._get_probability_vec(X_indices)
        chosen_l_ind = np.random.choice(len(L_l), p=P)
        response = self.teacher.teach(X, y, L_l[chosen_l_ind])
        self._set_reputation(response, chosen_l_ind)
        return TAFeedBack(response, chosen_l_ind)
```

```python
@dataclass
class TAFeedBack:
    teacher_feed_back: TeacherFeedBack
    ind_of_x_hat: int
```

Maintaining the reputation

In our example TA implementation:

The T.A holds an array that holds the reputations of each example by index.

We have a constant p (which is chosen as an hyperparameter)

Initialization: all examples have reputation $\frac{1}{\#examples}$

If the example got a good prediction: we multiply its reputation by the constant.

else- we divide it by the constant.

Than we normalize it to be a probability vector of which we sample.

Part B: The way we tried to improve the algorithm

```python
class ExampleTA(TeacherAssistance):

    def _get_probability_vec(self, X_indices) -> List[float]:
        P = [self.reputation_map[ind] for ind in X_indices]
        divisor = sum(P)
        P = [p / divisor for p in P]
        return P

    def _set_reputation(self, response, chosen_x_ind):
        self.reputation_map[chosen_x_ind] *= 1 / self.alpha if response is not None else self.alpha
```

Part B: The way we tried to improve the algorithm

The change we made in Our implementation for part b included several changes:

First of all, we removed the “break” command that happens after finding the first time we find a good example that our x_t satisfies

We created a List called X_{hats} and we append all of the items from L that is satisfied by x_t.

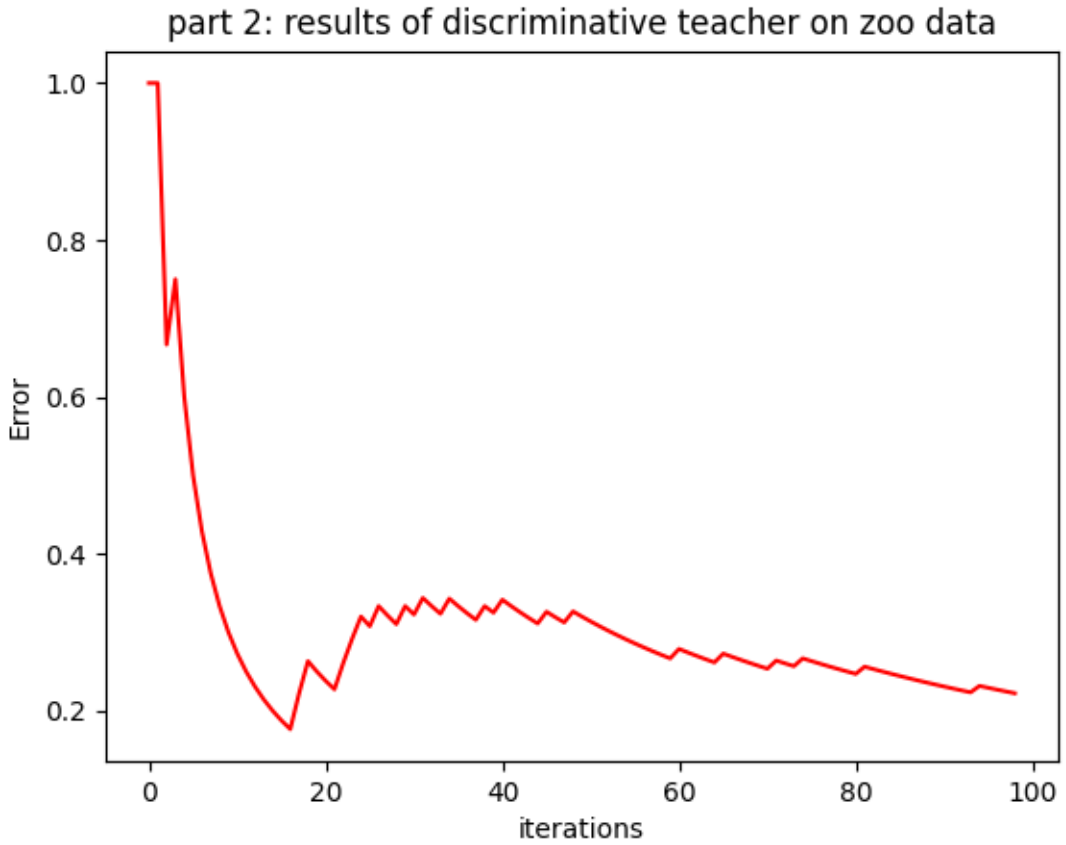
After looping over L and filling up x_{hats}, we sent it to the Teacher assistance.

The TA will pick one of the examples and will send it to the given teacher to make prediction, then will forward the teachers feedback back to us.

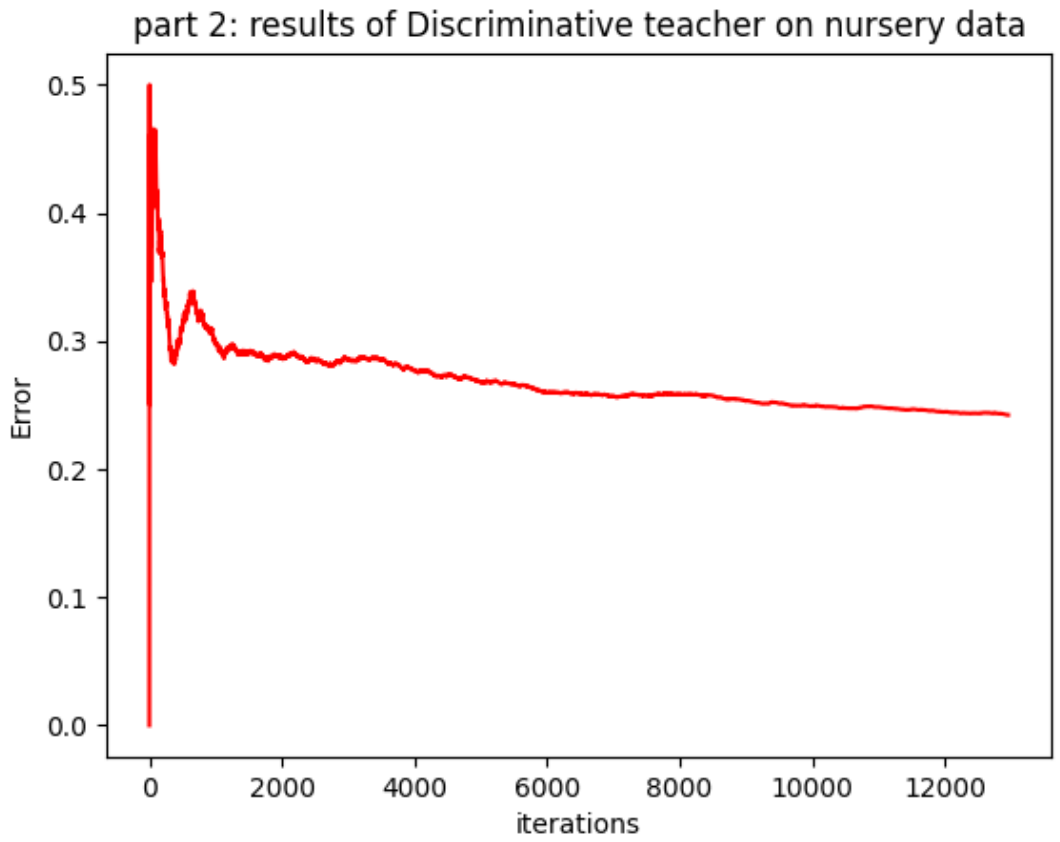
After we picked one example, we will fake predict using each of the other examples from x_hats.

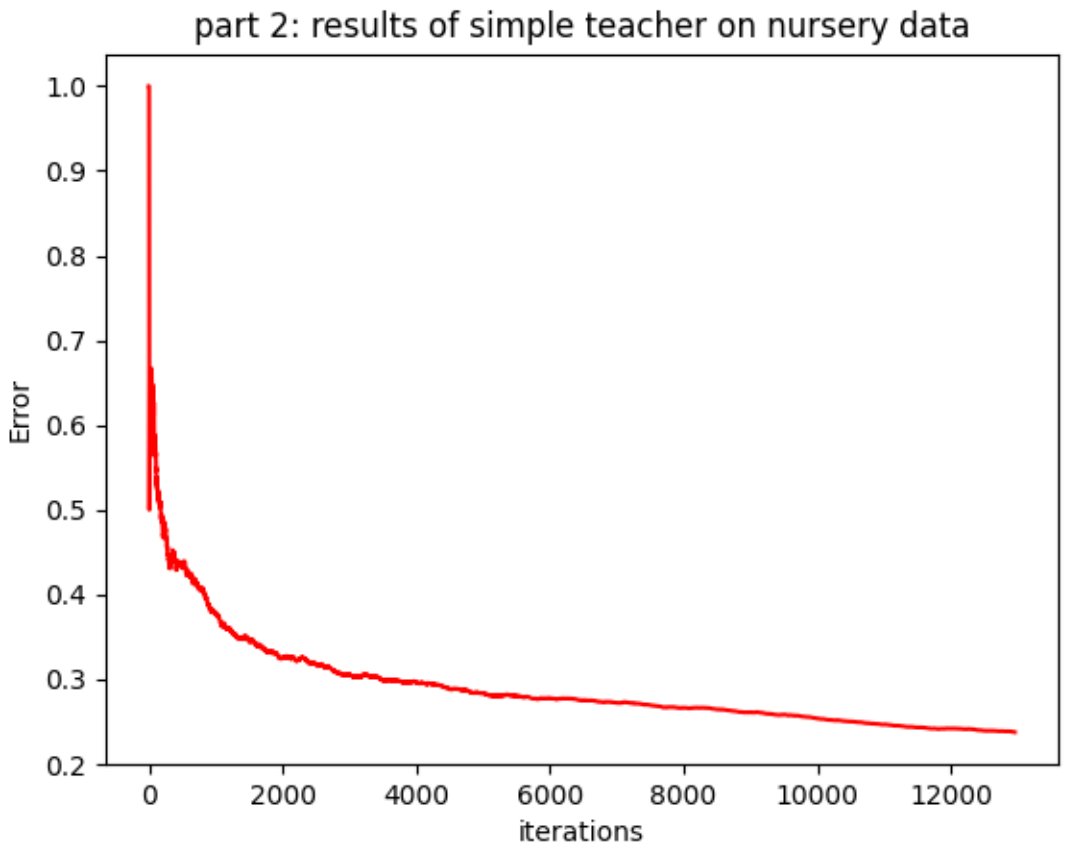
We are fake predicting only to get discriminative features to add to each of the example’s conjunctions. In this way the algorithm learn more for each x_t.

Part B: Graphs of the “improved” algorithm

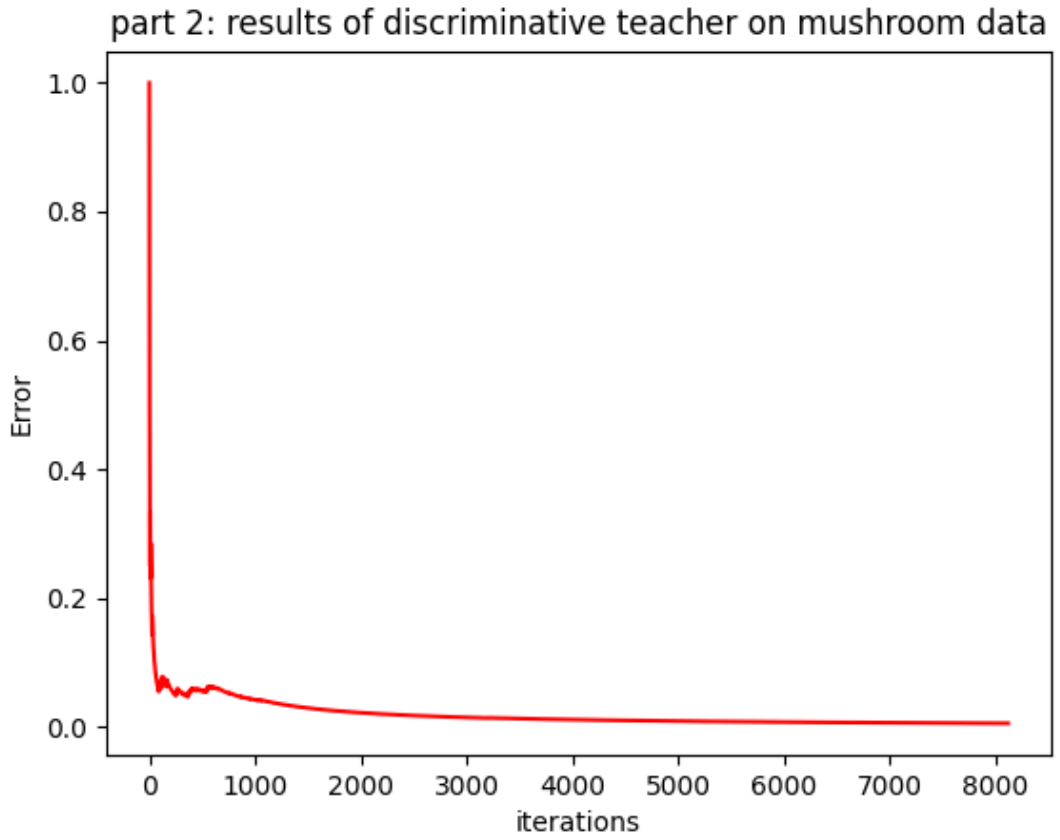


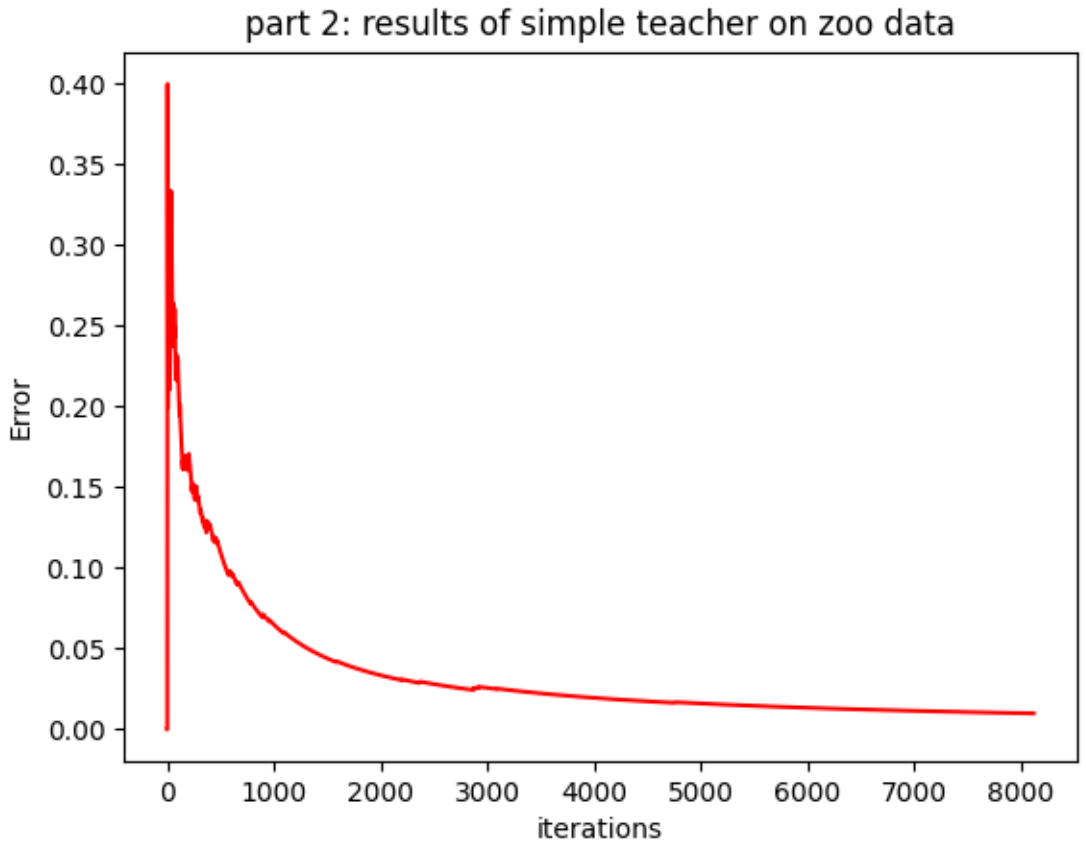
Part B: Graphs of the improved algorithm





Part B: Graphs of the “improved” algorithm





Part B: comparison between part A and part B

After running every option 10 times and calculating the averages:

error average of simple teacher on zoo data: 29.696969696969695%

error average of discriminative teacher on zoo data: 22.727272727273%

error average of simple teacher on nursery data: 26.875289396511807%

error average of discriminative teacher on nursery data: 22.364562432474 15%

error average of simple teacher on mushroom data: 1.1216449150455554%

error average of discriminative teacher on mushroom data: 0.6266929327751785%

Conclusion:

We can see that our changes didn’t improve the algorithm at all.

It could be that out base assumption is wrong-

We thought that there will be some examples which are significantly better than others.

It may be true for some problems, but not for the ones we tried.

It could also be that for a much bigger dataset we will start to see an improvement as the reputations will be more accurate.