Implementation of Adaboost-M1 Classifier Based on Linear SVM

Sami Alperen Akgun
sami.alperen.akgun@gmail.com

A. *Visualization of Data*

The visualization of given dataset containing class A and class B data can be seen in Figure 1.

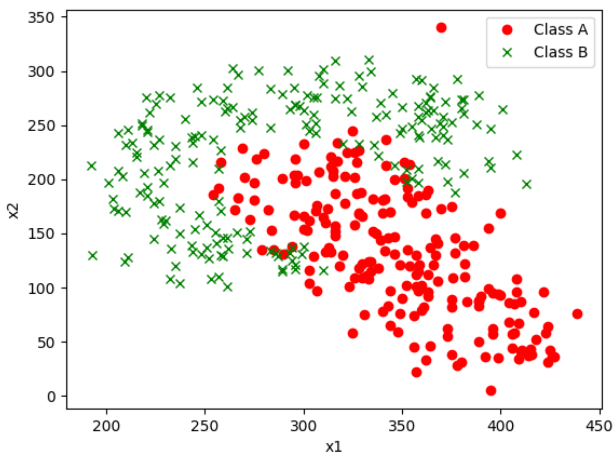


Fig. 1. Given data for Question 2

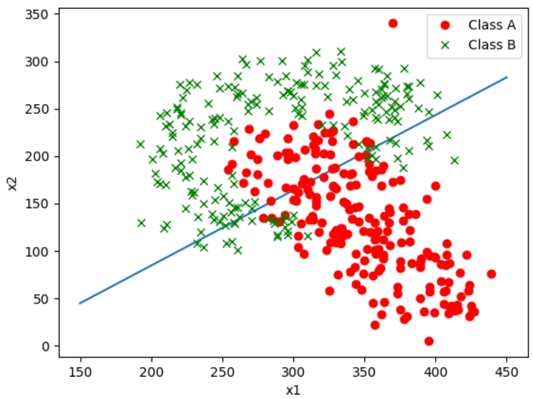
B. *Training SVMs and Check Accuracy via 10-times-10-fold Cross Validation*

Linear SVM classifiers were trained for different penalization factors (c values). Decision boundaries were plotted for these values as it can be seen in Figure 2. Decision boundaries with margins can be seen in Figure 3. 10-times-10-fold cross validation were also applied on these linear SVM classifiers with different c values. Obtained results can be seen below:

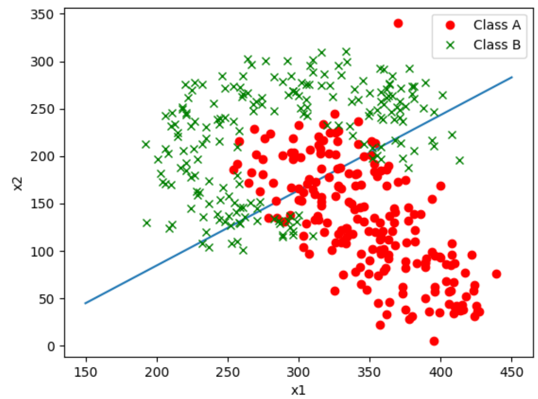
```
C value:  0.1
Mean of the accuracy:  0.7975152439024389 --> 79.75%
Var of the accuracy:  1.1149851435049755e-06
Best accuracy:  0.90625 --> 90.62%

C value:  1
Mean of the accuracy:  0.7980564024390244
Var of the accuracy:  4.437610043916163e-06
Best accuracy:  0.9375

C value:  10
Mean of the accuracy:  0.7964634146341464
Var of the accuracy:  2.014219261754062e-06
Best accuracy:  0.926829268292683

C value:  100
Mean of the accuracy:  0.7988948170731708
Var of the accuracy:  1.4532859742308036e-06
Best accuracy:  0.926829268292683
```

C. *Select SVM with best Penalization Factor (c) and Implement Adaboost-M1 based on that*

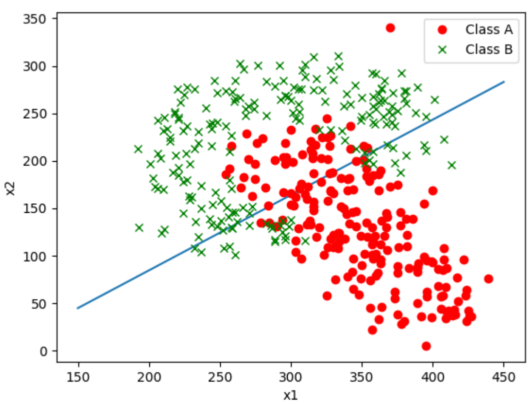
After analyzing obtained results in Part 2, one can clearly see that mean accuracy corresponding to different c values are almost the same for all cases. However, c=1 case has the best accuracy value (not mean accuracy) among them. Therefore, a linear SVM classifier with c value 1 was selected as a weak classifier for Adaboost approach. Nonetheless, selecting c value as 0.1 might be another good solution since actually we are interested in mean accuracies after 10-times-10-fold cross validation and they are almost the same for all cases. Higher values of c were not selected since they don't offer any better



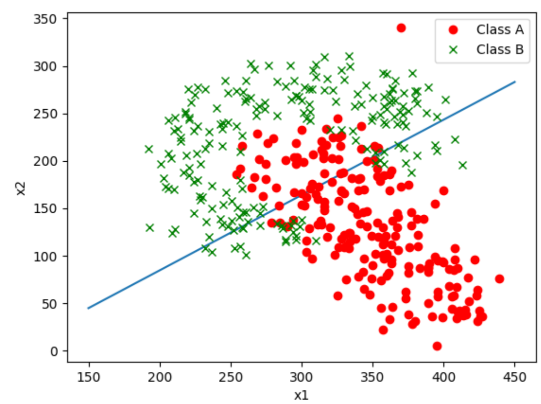
(a) c=0.1



(b) c=1



(c) c=10

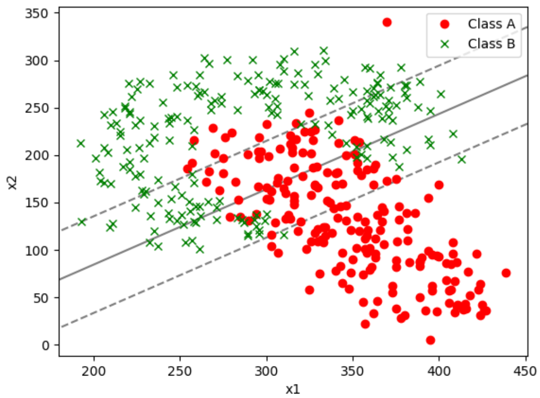


(d) c=100

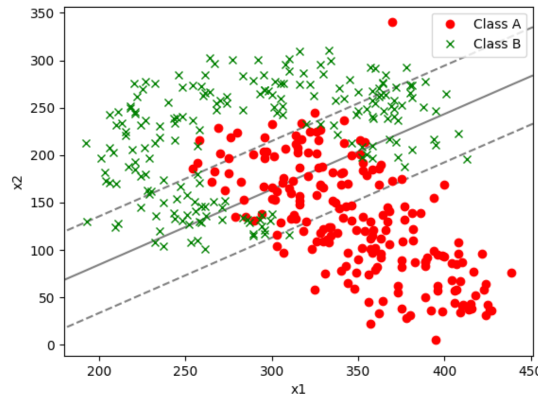
Fig. 2. Different decision boundaries of linear SVM classifiers

classification performance, but their convergence time is way larger than lower values, i.e. they are computationally expensive. The implemented function for Adaboost-M1 approach can be seen below:

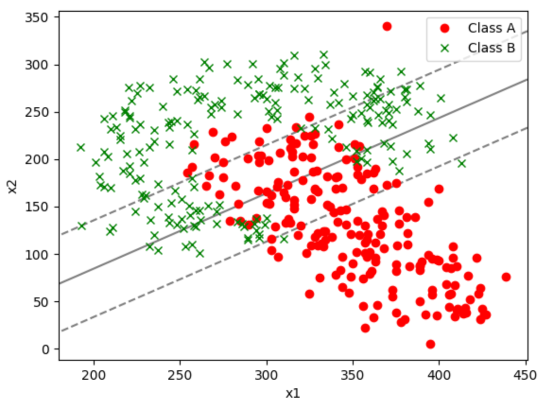
```python
def adaboost_M1(X_data,Y_data,c_value,T):
    """
        This function applies 10 fold Cross validation on the given input data
        Inputs: X_data --> input data contains features and their values
                Y_data --> input data contains output labels
                c_value --> Penalization factor for linear SVM classifier
                T --> Max number of weak learners, i.e. max number of iterations
                for adaboost mainloop
        Output: h --> fitted linear SVM model
    """

    m = X_data.shape[0] # Total number of samples
    n = X_data.shape[1] # Total number of features
    h_list = [] # The list to hold all hyptohesis, i.e. learned models
    D = [] # List to hold all weights for each hypothesis
    beta_list = [] # List to hold all beta values
    # Initialize weight list with 1/m for all samples
    D.append( np.ones(m) * (1/m) )# --> shape = (m,)

    t = 0
    while t<T:
        # Pick 100 examples as a training set out of m data with probability
        # distribution D[t]

        training_indices = np.random.choice(D[t].shape[0],100,replace=False,p=D[t])
        training_X = X_data[training_indices,:]
        training_Y = Y_data[training_indices]
        h = svm.SVC(kernel='linear', C = c_value) # current hypothesis
```



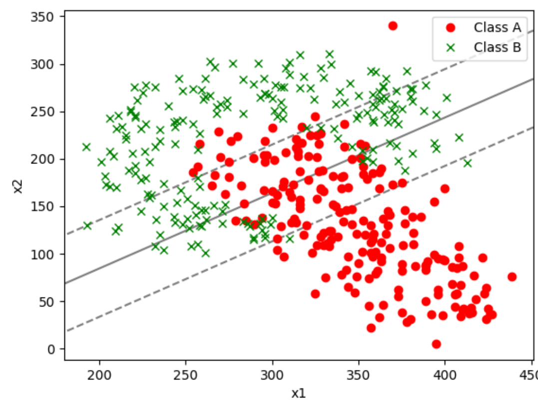
(a) c=0.1



(b) c=1



(c) c=10



(d) c=100

Fig. 3. Different decision boundaries of linear SVM classifiers with margins

```
        h.fit(training_X,training_Y) # train using training set
        Y_predicted = h.predict(X_data) # predict using all samples

        epsilon = h_error(Y_predicted,Y_data,D[t])

        # if error is more than 50%, start the loop again
        # This means we are discarding the classifier corresponding error
        # rate more than 50% and we are trying to resample another training
        # set --> Therefore, in the end we will have 50 trained classifiers
        if epsilon >= 0.5:
            # Not update t, so that we can iterate with the same t value
            continue

        beta = epsilon / (1-epsilon)
        D.append( update_weights(Y_predicted,Y_data,beta,D[t]) ) #--> D[t+1]
        h_list.append(h)
        beta_list.append(beta)
        t+=1 # While loop!

    return h_list, beta_list, D
```

D. Check Accuracy of Adaboost-M1 classifier via 10-times-10-fold Cross Validation

Obtained mean and variance of accuracy for 10-times-10-fold cross validation approach on adaboost-M1 classifier (with weak classifier linear SVM and c=1) can be seen below:

```
Mean of the accuracy:  0.8372256097560975 --> 83.72%
Var of the accuracy:  7.904603999270937e-07
Best accuracy:  0.915609756097561 --> 91.56%
```

E. Plot Decision Boundary of Adaboost M1 Classifier and Compare Accuracy of it with Linear SVM

Decision boundary of the ensemble model can be seen in Figure 4. If one compare this decision boundary with the decision boundary of weak classifier (linear SVM with c value 1) in Figure 2, s(he) can clearly see that thanks to boosting approach, the trained classifier were improved.

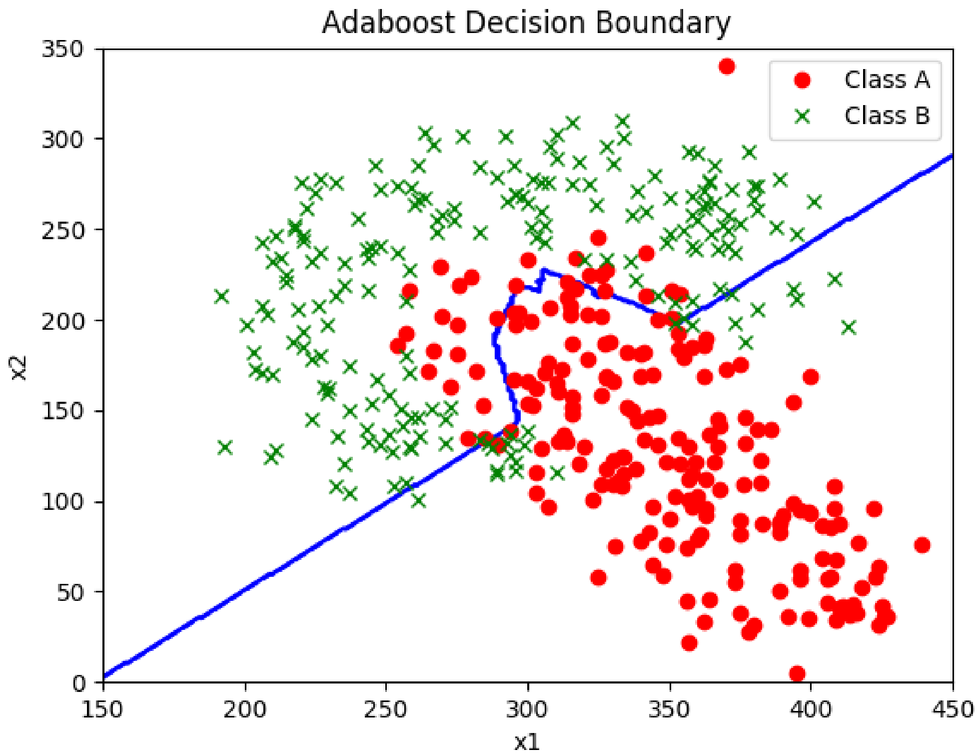


Fig. 4. Decision Boundary of the Ensemble Model