Advanced Machine Learning
Online Learning

Robin Khatri Mohammed Poul Doust

In this report, we present results of Online Passive-Aggressive Algorithm, and its active learning and kernelized versions.

1 Passive-Agressive Online Learning Algorithm

We have implemented the algorithm described in [1].

1.1 Usage:

```
from pa import PA
clf = PA(C = 0.5, update_='fr') # update_: 'classic','fr','sr'
clf.fit(X_train, Y_train) # Y_train: labels should be in +-1.
y_pred = clf.predict_all(X_test)
```

1.2 Testing of algorithm with synthetic datasets

First, we tested our algorithm on synthetic datasets to see performance. Thereafter, we tested on real datasets and the results are presented later in this section.

Synthetic dataset 1: Generated from `make_moons` method of `sklearn.datasets`.

We generated 500 samples with noise level 0.5. The decision boundaries of our three strategies and of SVM are presented in figure 1.

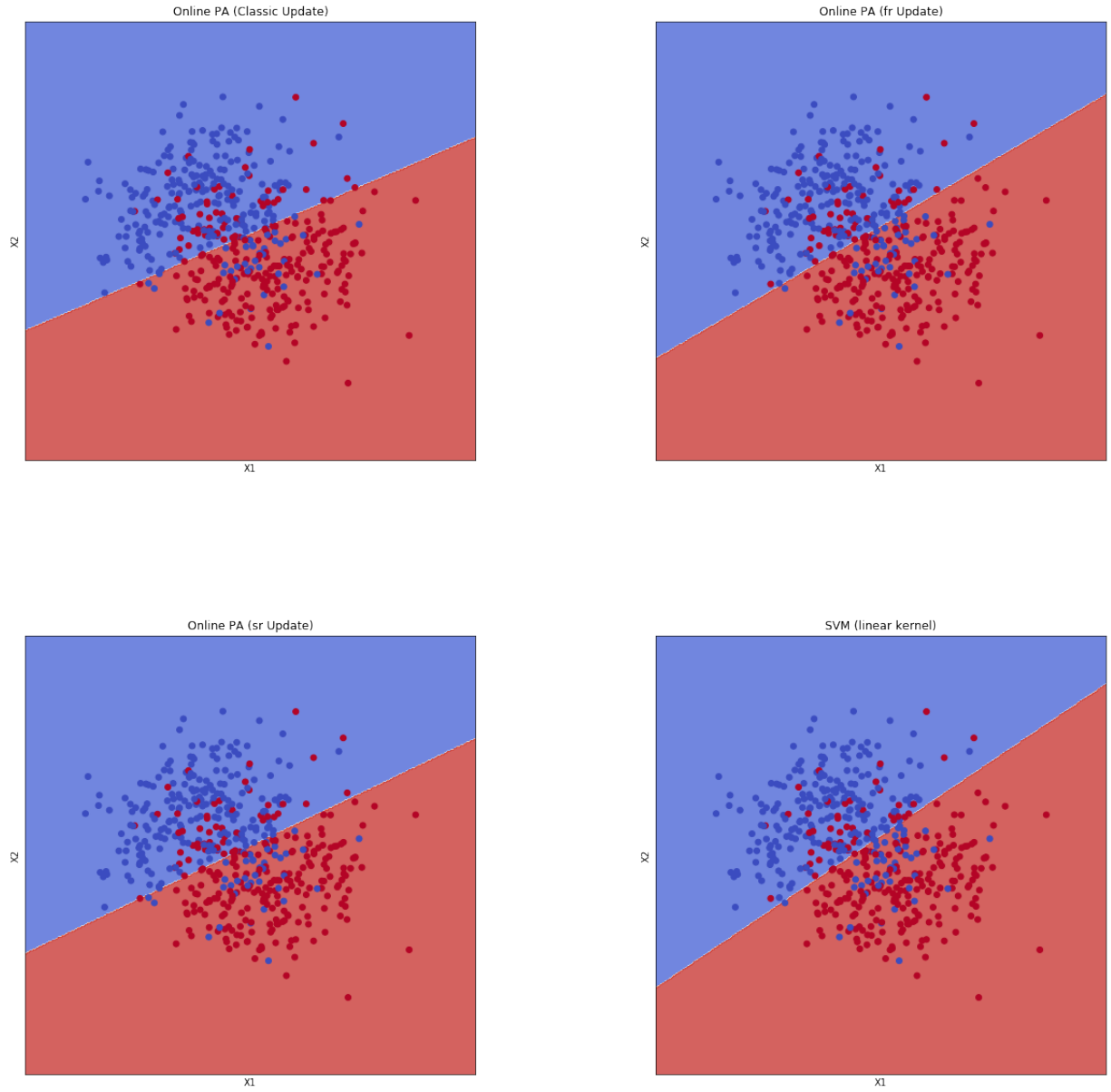

Figure 1: Decision boundaries of online passive aggressive algorithms and SVM on synthetic moons dataset.

Synthetic dataset 2: Generated from `make_blobs` method of `sklearn.datasets`.

We generated 500 samples with two centers. The decision boundaries of our three strategies and of SVM are presented in figure 2.

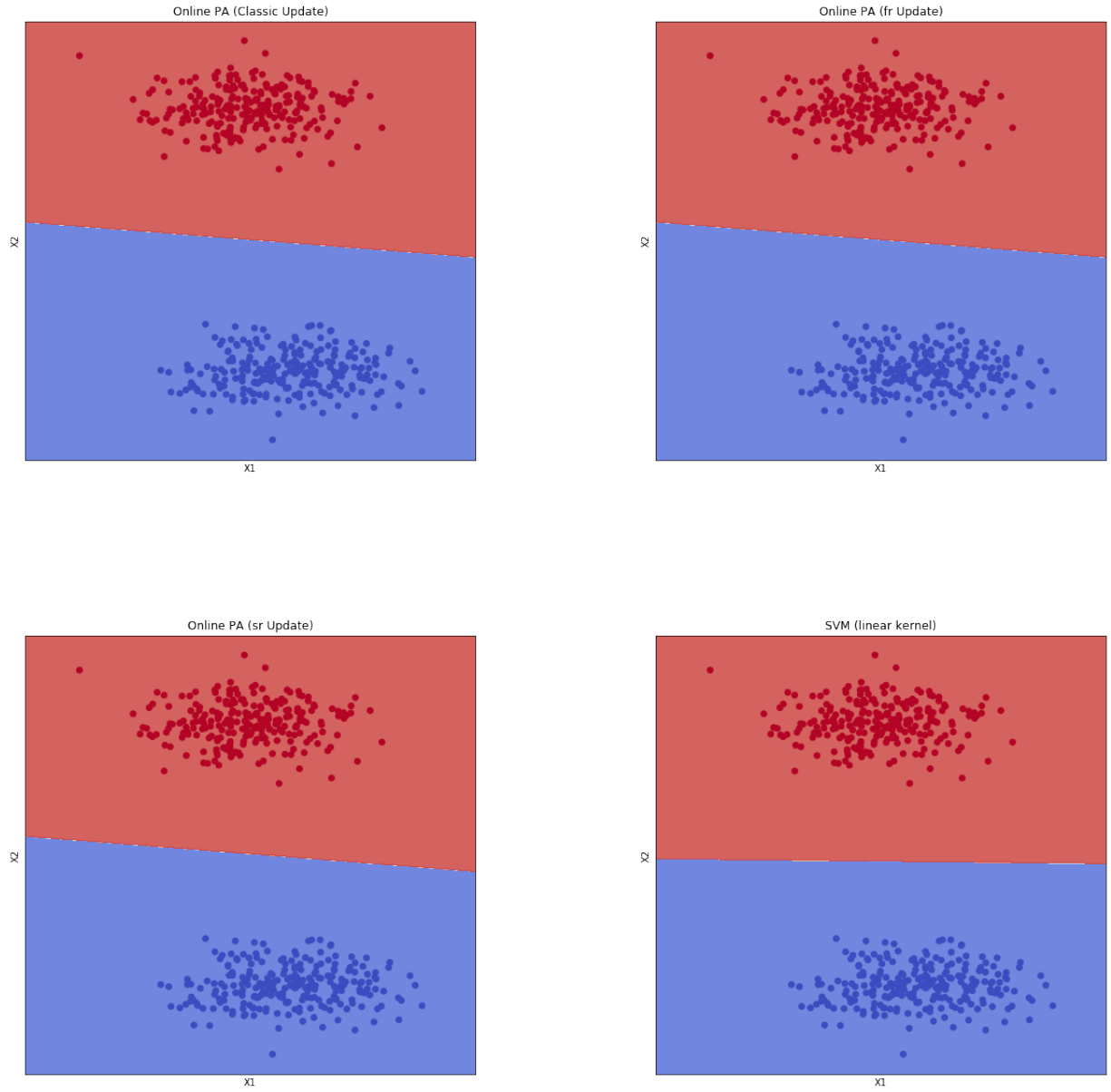

Figure 2: Decision boundaries of online passive aggressive algorithms and SVM on synthetic blobs dataset.

Figure 3: Time-Complexity on a real dataset (Banknote authentication dataset).

2 Real datasets: from LIBSVM page

We tested our algorithm on 3 binary datasets - `mushrooms, ionosphere, sonar` mentioned in the LIBSVM webpage. We also tested on noisy datasets by randomly flipping labels. We defined an experimental setup in a function `test_on_data` included in `utils.py` which takes in X and labels Y along with a dictionary consisting of classifier name and class. It divides X and Y into train and test sets (70-30) and prints the accuracy and total (train+test) time for different classifiers.

Table 2 presents the accuracies of different classifiers while table presents accuracy for all three update strategies and SVM (Linear).

Algorithm	**Mushrooms (8124 × 22)**	**Ionosphere (351 × 34)**	**Sonar (208 × 60)**
SVM (Linear)	79.37 %	88.68 %	79.37 %
Classic, $C = 0.1$	69.84%	84.91%	69.84 %
First relaxation, $C = 0.1$	68.25%	83.96%	69.84 %
Second relaxation, $C = 0.1$	73.02%	84.91%	69.84 %

Table 2 presents the run-time (train+test) for different classifiers while table presents accuracy for all three update strategies and SVM (Linear).

Algorithm	**Mushrooms (8124 × 22)**	**Ionosphere (351 × 34)**	**Sonar (208 × 60)**
SVM (Linear)	0.0041 sec.	0.0075 sec.	0.006 sec.
Classic	0.0089 sec.	0.017 sec.	0.015 sec.
First relaxation	0.0081 sec.	0.016 sec.	0.014 sec.
Second relaxation	0.0079 sec.	0.012 sec.	0.008 sec.

Evidently, the time taken for training and testing is considerably lesser with first and second relaxation when compared with the classic version. The accuracy remains mostly same. SVM from `sklearn`, however, outperforms the online passive aggressive algorithm, at least in linear case.

We also tested our implementation on noisy dataset. For this, we used `Mushrooms` dataset and flipped labels depending upon a probability. We generated a random integer between 1 and 10 and if it belonged to say (1,2,3) tuple, we flipped the labels. Therefore, in that case, the probability of flipping labels for a row was 3/10. Similarly, we defined noise levels between 0.1 and 0.5. After 0.5, it doesn't make sense to increase noise further, as the flipped labels in that case will be considered true labels and the accuracy for 0.1 noise level would be approximately same as accuracy for 0.9 noise level. We have defined 'noise-level' as presented in the following function.

```
def flip_labels(row):
    if random.randint(1, 10) in (1, 2, 3): # Noise level = 0.3
        row[0] = -1 * row[0]
    else:
        row[0] = row[0]
    return row
```

The table 2 presents the accuracy while the table 2 presents the time taken for different noise levels.

Algorithm	**Noise = 0.1**	**Noise = 0.2**	**Noise = 0.3**	**Noise =0.4**	**Noise=0.5**
SVM (Linear)	86.09 %	75.58%	68.65 %	59.62 %	48.83 %
Classic	79.15 %	73.2 %	63.6 %	50.96%	48.42 %
First relaxation	79.15 %	73.2 %	63.6 %	50.96%	48.42 %
Second relaxation	79.28 %	73.12 %	63.64 %	50.96%	48.42 %

Algorithm	**Noise = 0.1**	**Noise = 0.2**	**Noise = 0.3**	**Noise =0.4**	**Noise=0.5**
SVM (Linear)	17.93 sec.	16.85 sec.	15.21 sec.	11.17 sec.	3.68 sec.
Classic	0.29 sec.	0.31 sec.	0.35 sec.	0.32 sec.	0.309 sec.
First relaxation	0.32 sec.	0.32 sec.	0.35 sec.	0.34 sec.	0.39 sec.
Second relaxation	0.42 sec.	0.35 sec.	0.38 sec.	0.37 sec.	0.49 sec.

For accuracy for different values of C on each of these three LIBSVM datasets, please follow the notebook. Generally, when C increases, the accuracy for first and second relaxations becomes constant and closer to classic update strategy.

The time taken by SVM on noisy `mushrooms` dataset is very high. Online passive aggressive algorithms are considerably faster while the difference in accuracy is not much on most of these settings.

3 Active Online Learning

Module `paActive.py` contains AL class for active learning. Its usage is presented below:

```
from paActive import AL
clf = AL(C = 0.5, update_ = 'fr', delta = 1) # update_: 'classic', 'fr', 'sr'
clf.fit(X_train,Y_train) # Y_train should be in +1 and -1 scale
y_pred = clf.predict_all(X_test)
```

For detailed experiments and code, see the script (`paActive.py`) and notebook - `OnlineLearning_Parts_1_and_2.ipynb`.

Table 4 presents accuracy and training time of linear SVM, Online passive aggressive algorithm and active learning algorithms on `ionosphere` dataset. For online algorithms (Passive aggressive and active, first relaxation is presented). We tried different values for the δ parameter.

Algorithm	**Accuracy (in %)**	**Training time (in seconds)**
SVM (Linear)	87.74	0.003
Online PA, First relaxation $C = 0.5$	84.91	0.0089
Active Learning, First relaxation, $C = 0.5, \delta = 3$	78.30	0.0058
Active Learning, First relaxation, $C = 0.5, \delta = 7$	79.25	0.009
Active Learning, First relaxation, $C = 0.5, \delta = 10$	83.96	0.009

Clearly, as the value of δ increases, the run-time increases. The similar trend was obtained in general with other update strategies. The lower time taken is for lower values of δ. The test-accuracy increases as the value of δ increases which is expected as more data labels are used for prediction.
The effect of δ can be better seen by seeing the impact of its value on the number of labels used for training. The parameter δ is smoothing parameter and controls the amount of labels used for training. On `ionosphere` dataset, as we increase δ, the fraction of labels used for training increases as given in figure 4.

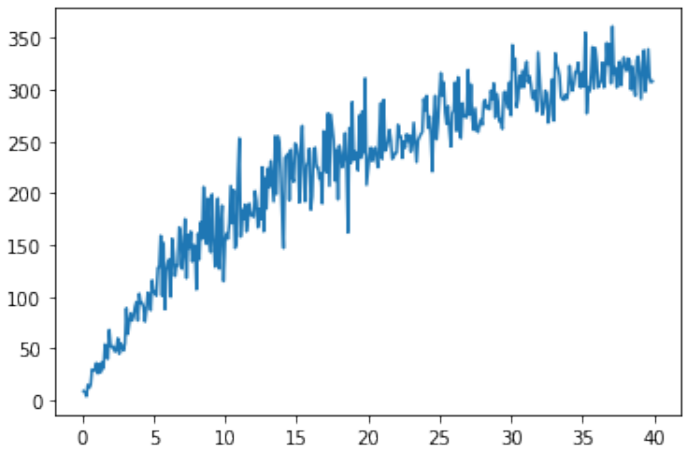

Figure 4: δ (x-axis) vs. Used labels (y-axis) - `Ionosphere` dataset.

To further see this, we make use of a synthetic dataset shown in the figure 5.

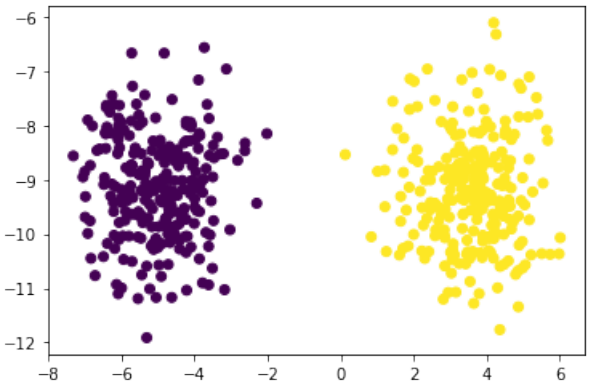

Figure 5: Synthetic binary dataset

With $\delta = 3$, the active online learning algorithm uses 67 out of 500 labels (*i.e.* 86.6% labels are not used.)

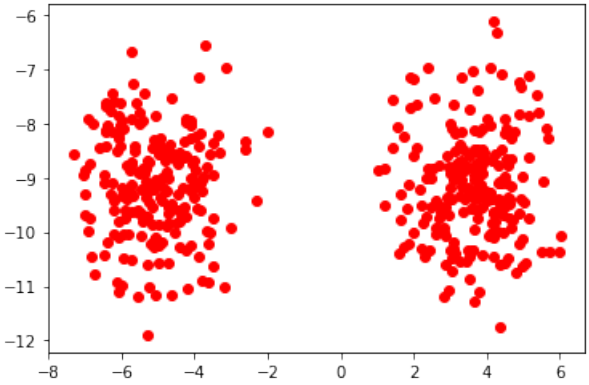

Figure 6: Unused labels (86.6 %)

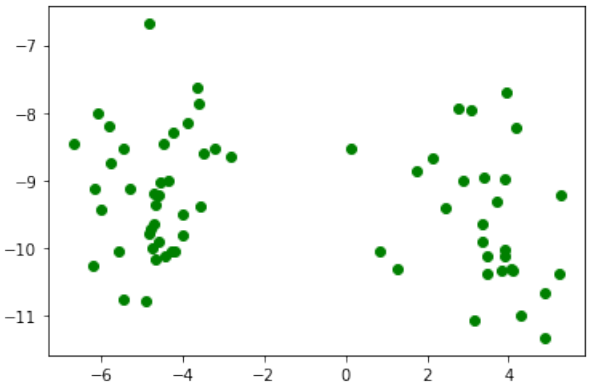

Figure 7: Used labels (13.4 %)

4 Kernelized online PA algorithm

Kernelized version of online passive aggressive are given in module `paKernel.py`. Its usage is presented below:

```
from paKernel import KPA
clf = KPA(C = 0.1, update_ = 'fr', kernel = 'rbf', gamma = 0.5) # kernel: 'rbf', 'poly'
clf.fit(X_train, Y_train) # Y_train: labels should be in +-1.
y_pred = clf.predict_all(X_test)
```

For detailed experiments and source code, see the script (`paKernel.py`) and notebook - `OnlineLearning_Parts_3_Kernels.ipynb`.

On a synthetic dataset (1000 samples of moons dataset with noise = 0.5), the accuracies of Kernelized online passive-algorithms with RBF kernel $\gamma = 0.5$ is presented below:

Algorithm	**Accuracy (in %)**	**Train-time**
SVM (RBF, $\gamma = 0.5$)	84.5	0.01
KPA - Classic, (RBF, $\gamma = 0.5$)	80	0.24
KPA - First relaxation, (RBF, $C = 0.1, \gamma = 0.5$)	85	0.22
KPA - Second relaxation, (RBF, $C = 0.1, \gamma = 0.5$)	85	0.22

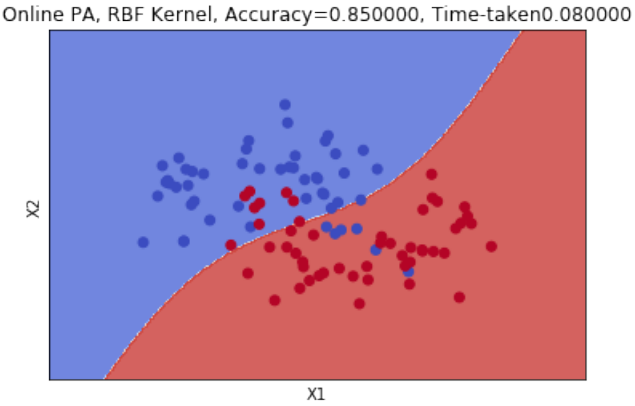

Figure 8: Kernelized passive aggressive algorithm (RBF kernel), $\gamma = 0.5$. Sample size = 100, Accuracy = 0.85.

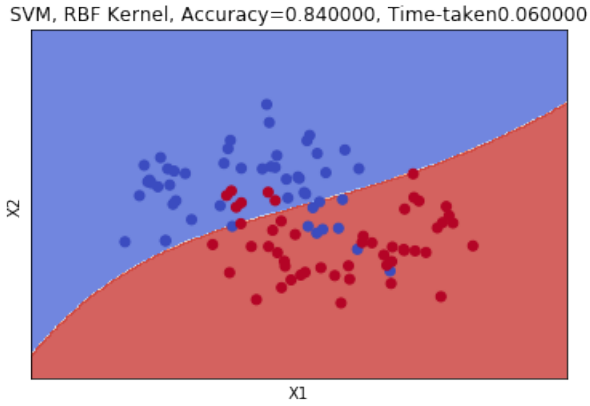

Figure 9: Kernelized passive aggressive algorithm (polynomial kernel), degree=3. Sample size = 100, Accuracy = 0.84.

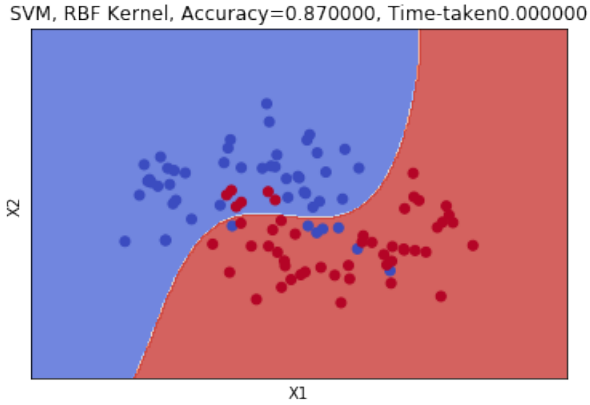

Figure 10: SVM with RBF kernel, $\gamma = 0.5$. Sample size = 100, Accuracy = 0.87.

Figure 11 gives the plot between C and accuracy of the kernelized online passive aggressive algorithm with first relaxation. As C increases, the accuracy seems to increase. However, the effect of C is not uniform.

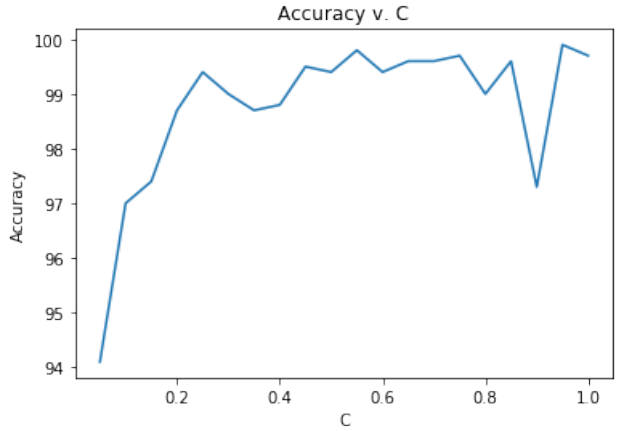

Figure 11: Effect of C on accuracy with RBF kernel (C=0.1). Sample size = 1000.

Ionosphere dataset

On `Ionosphere` dataset, the accuracy of the kernelized passive aggressive algorithm is given in table 4. For this, we used our experimental setup as defined in previous sections (Train-test ratio = 70-30).

Algorithm	**Accuracy (in %)**	**Time-taken**
SVM (RBF, $\gamma = 0.5$**)**	97.17	0.013
Linear PA - First relaxation, $(C = 0.5)$	84.91	0.02
Kernelized PA - First relaxation, (RBF, $C = 0.5, \gamma = 0.5$**)**	91.51	0.08
Kernelized PA - Second relaxation, (Polynomial, $C = 0.5, = 3$**)**	89.62	0.04

With kernels, accuracy is clearly better. On `Ionosphere` dataset, RBF kernel performs better than the polynomial one. The accuracy is much higher than the linear version, however, it takes more time due to kernel computation.

References

[1] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7(Mar):551–585, 2006.