CS534 – Machine Learning

Homework Assignment 2*

Fall 2018

Ali Raza 933620601 razaa@oregonstate.edu
Sheekha Jariwala 933620505 jariwalas@oregonstat.edu
Vivswan Shitole 933591615 shitolev@oregonstate.edu

Oregon State University
October 27, 2018

* Due 11:59PM Oct 27th, 2018

We worked as a team of three students and contribution of each member is 1/3. We coded our implementation in Python 3.6. The effects of different learning rates, regularization, and normalization are discussed in the following parts.

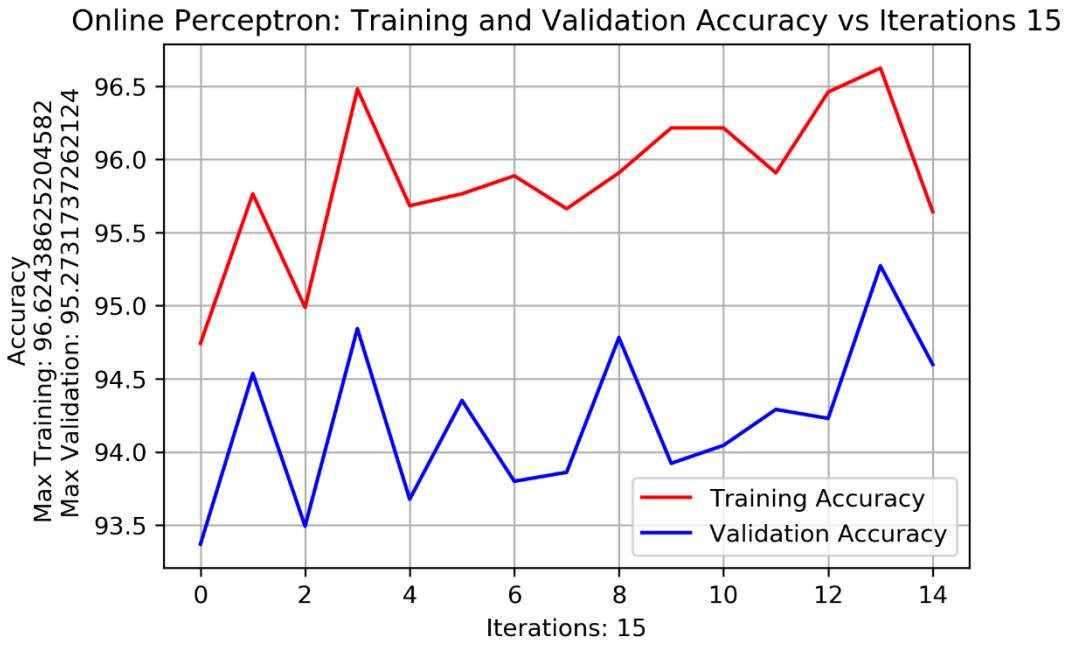
Part 1

a)

We implemented the Online Perceptron in Python. The source file is included with the assignment.

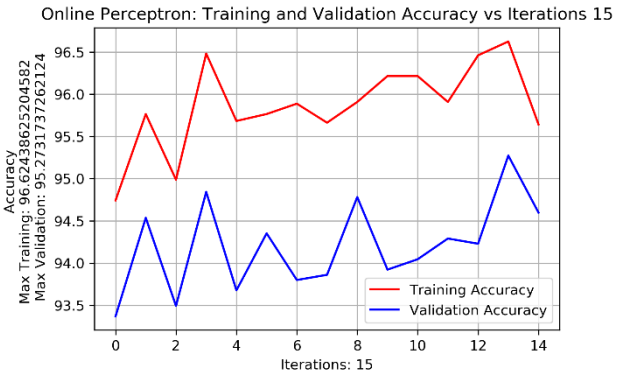
```
def onlinePerceptron(XTraining, YTraining, XValidation, YValidation, xTest, iterations):
    trainAccuracy = []
    validationAccuracy = []

    actualTrainingPrediction = []
    actualValidationPrediction = []

    YTestPrediction = []

    w = np.ones(XTraining.shape[1]) * 0

    for i in range(0,iterations):
        for t in range(0,XTraining.shape[0]):
            ut = np.sign(np.dot(w,XTraining[t, :]))

            if YTraining[t]*ut <= 0:
                w = w + YTraining[t]*XTraining[t, :]

        YTrainPrediction = np.sign(np.matmul(XTraining, w))
        actualTrainingPrediction = (sum(np.abs(YTrainPrediction+YTraining))/2)

        YValidationPrediction = np.sign(np.matmul(XValidation, w))
        actualValidationPrediction = (sum(np.abs(YValidationPrediction+YValidation))/2)

        if i == 13:
```

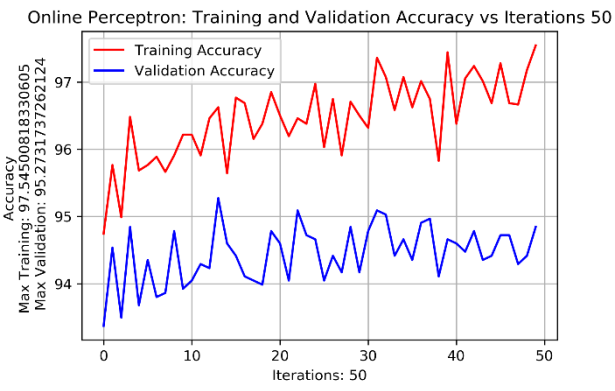
```
        YTestPrediction = np.sign(np.matmul(xTest, w))
        np.savetxt('oplabel.csv', YTestPrediction)
    trainAccuracy.append((100*actualTrainingPrediction)/YTraining.shape[0])
    validationAccuracy.append((100*actualValidationPrediction)/YValidation.shape[0])
```



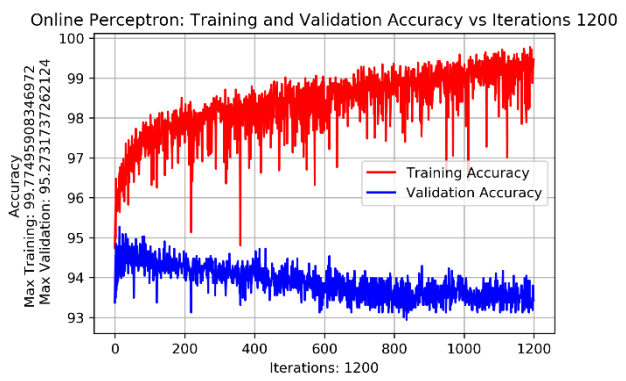
b)

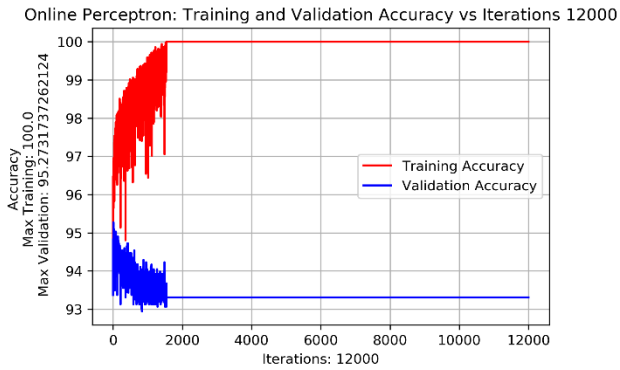
We ran online perceptron for various iterations as shown below. It can bee seen that train accuracy reaches 100% after about 1700 iterations. It means that training data is linearly separable. Although by looking at the validation accuracy, we can say that overfitting is happening.





Online perceptron for Iterations = 15 and 50





Online perceptron for Iterations = 1200 and 12000

c)

As seen from the graphs above, overfitting is happening after iteration 14. We used the weights generated after 14 iterations to make predictions on the samples in test data. The prediction file oplabel.csv is submitted with the assignment.

Part 2

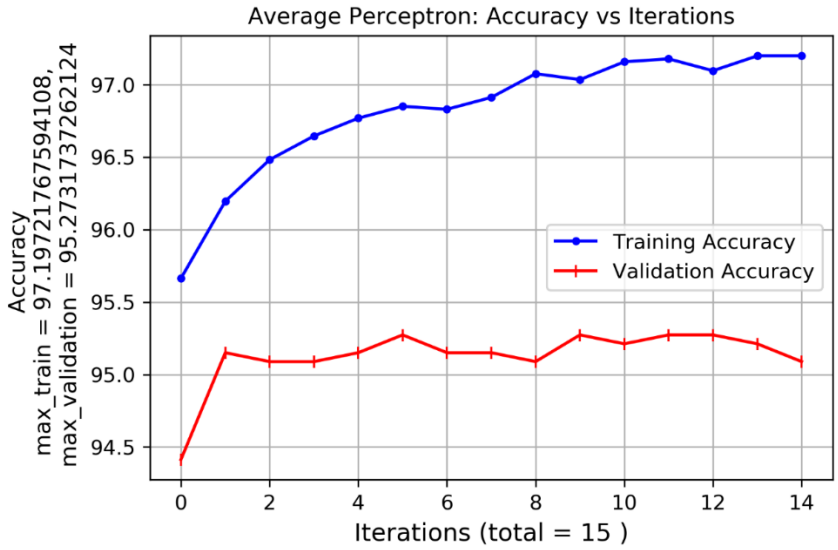
a)

We implemented the average perceptron. Source file is included with the assignment.

```
def Part2(X, Y, X_validation, Y_validation,X_testing, MAX_ITERATIONS):
    accuracy_training = []
    accuracy_validation = []
    w = np.ones(X.shape[1]) * 0
    w_average = np.ones(X.shape[1]) * 0
    c = 0
    s = 0
    for current_iteration in range(1,MAX_ITERATIONS+1):
        for index_sample in range(0,X.shape[0] ):
            ut = np.sign(np.dot(w,X[index_sample,:]) )
            if(Y[index_sample]*ut <= 0):
                if (s+c > 0):
                    w_average = (s*w_average + c*w)/(s+c)
                s = s+c
                w = w + Y[index_sample]*X[index_sample,:]
            else:
                c = c+1
        Y_train_predicted = np.sign( np.matmul(X, w_average)  )
        correct_predictions_training = sum(np.abs(Y_train_predicted + Y))/2
        accuracy_training.append(100*(correct_predictions_training)/Y.shape[0])

        Y_validation_predicted = np.sign( np.matmul(X_validation, w_average)  )
        correct_predictions_validation = sum(np.abs(Y_validation_predicted + Y_validation))/2
        accuracy_validation.append(100*(correct_predictions_validation)/Y_validation.shape[0])
    if (c>0):
        w_average = (s*w_average + c*w)/(s+c)

    plot_Part2(accuracy_training, accuracy_validation, MAX_ITERATIONS)
```

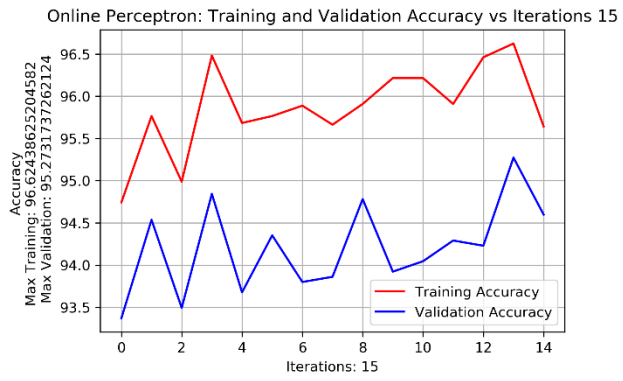
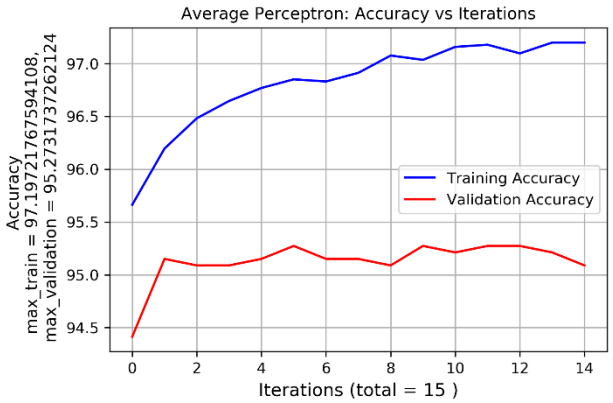
b)

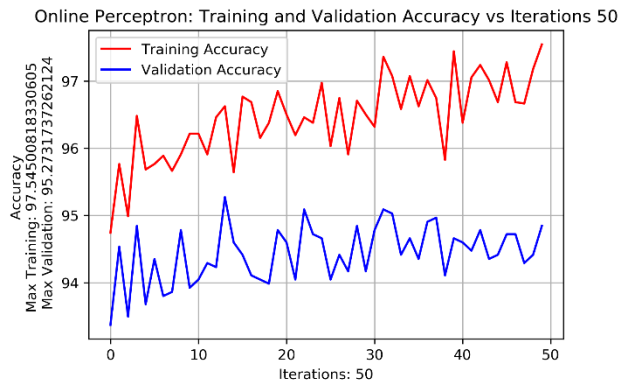
Train and validation accuracies versus the iterations number are shown below:

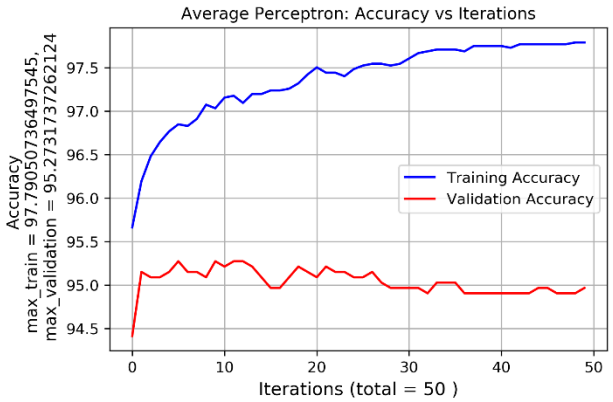


c)

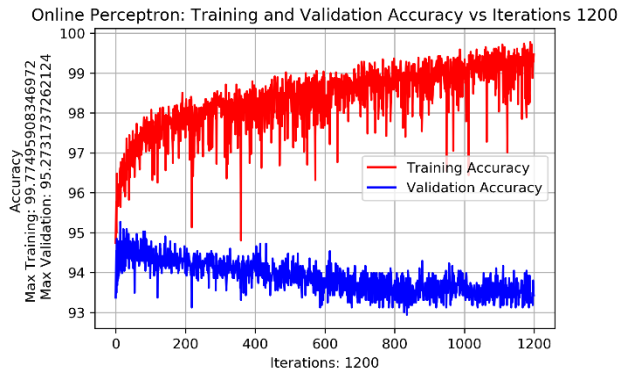
Validation accuracy is smooth when using the average model. It has less oscillations as compared to that of vanilla perceptron. Validation accuracy has reached its maximum accuracy earlier (iterations=5) in average perceptron as compared 14 iterations of vanilla perceptron. To gain more insight about the behavior of average model, we plotted train and validation accuracies for various number of iterations

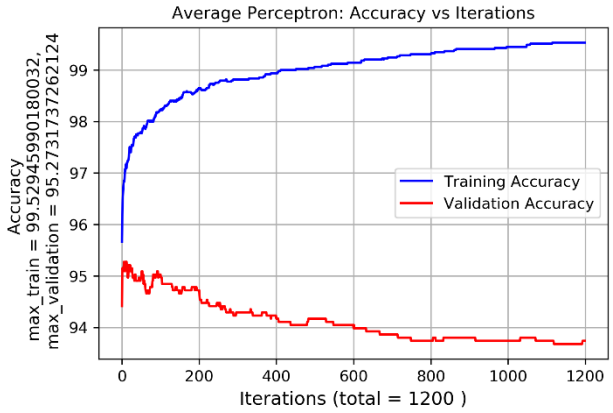
Online Vanilla Perceptron	**Average Perceptron**
	
Iterations = 15	



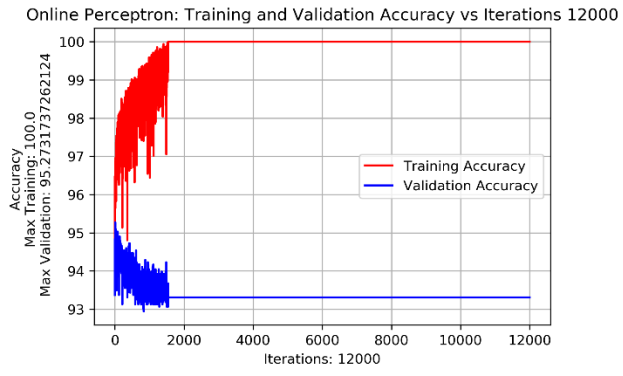


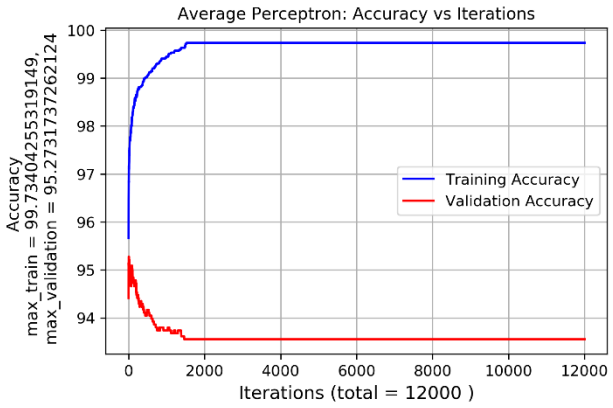
Iterations = 50





Iterations = 1200





Iterations = 12000

These graphs verify that vanilla perceptron counts later data samples more than it counts earlier samples, and has more oscillations.

Part 3

a) Implemented polynomial kernel function:

```
def kp(x,y,p):
    value = math.pow( (1 + np.dot(x,y)) , p)
    return value
```

b) Implemented Gram Matrix K:

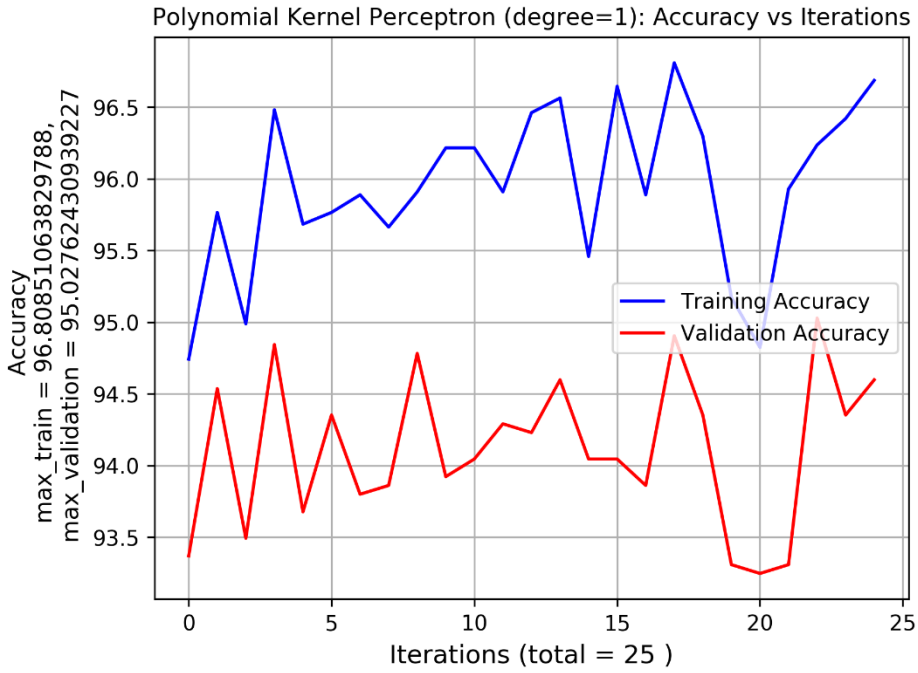
```
K = np.zeros((N,N)).astype(float)
    for i in range(0,N):
        for j in range(0,N):
            K[i][j] = kp(X[i],X[j],p)
```

c) Best Accuracies for each p:

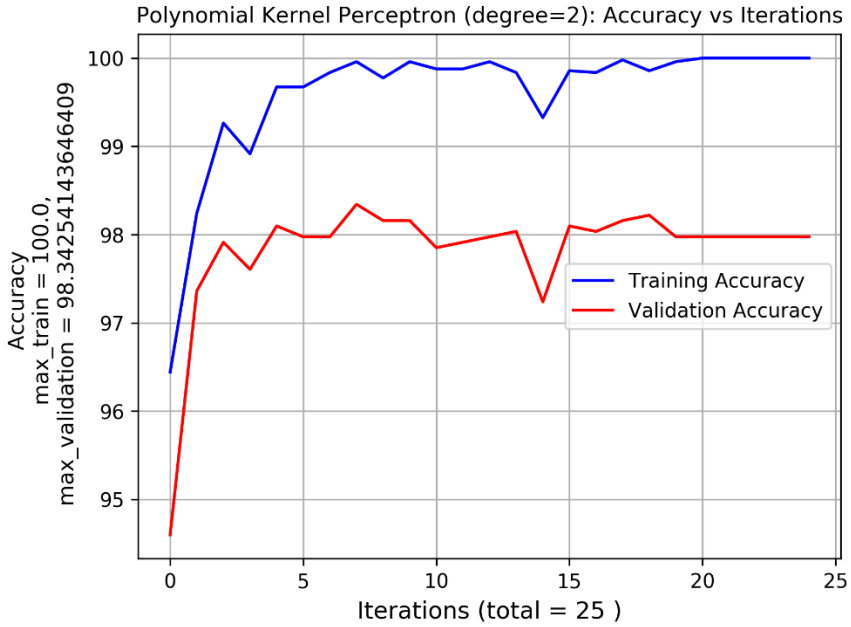
P	Training Accuracies	Validation Accuracy
1	96.8	95.02
2	100	98.32
3	100	98.46
7	100	97.73
15	99.9	96.56

Plots:

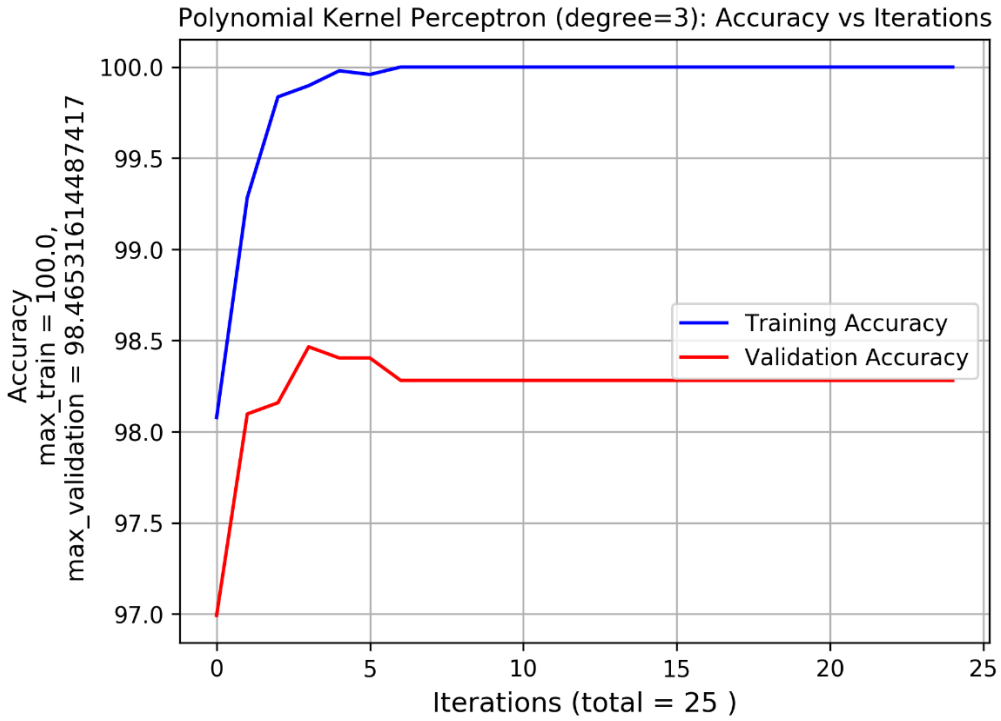
1. Train and Validation Accuracies for $P = 1$



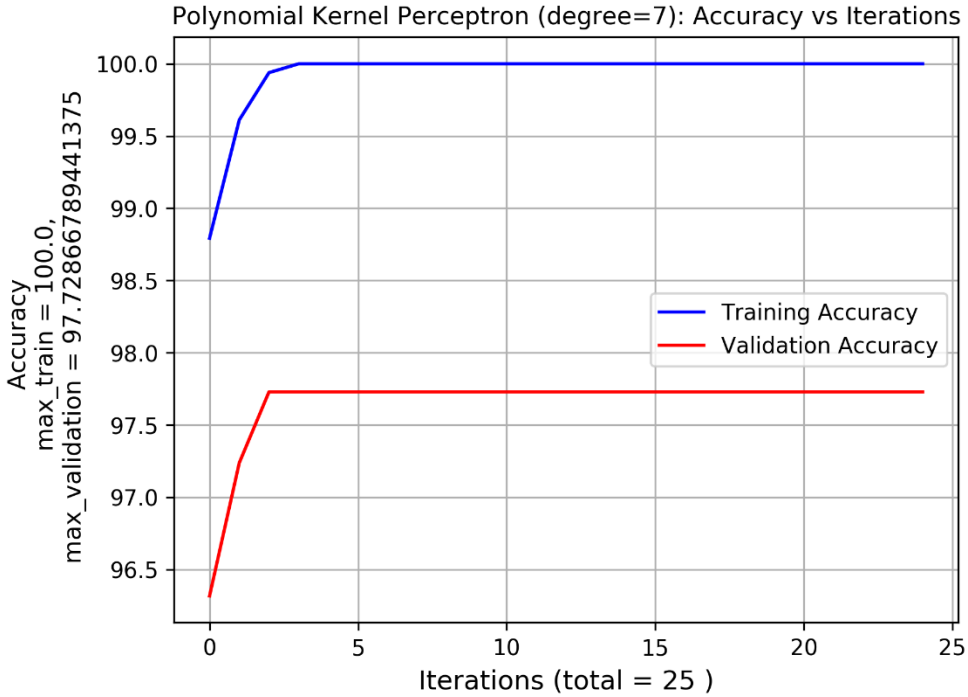
2. Train and Validation Accuracies for $P = 2$



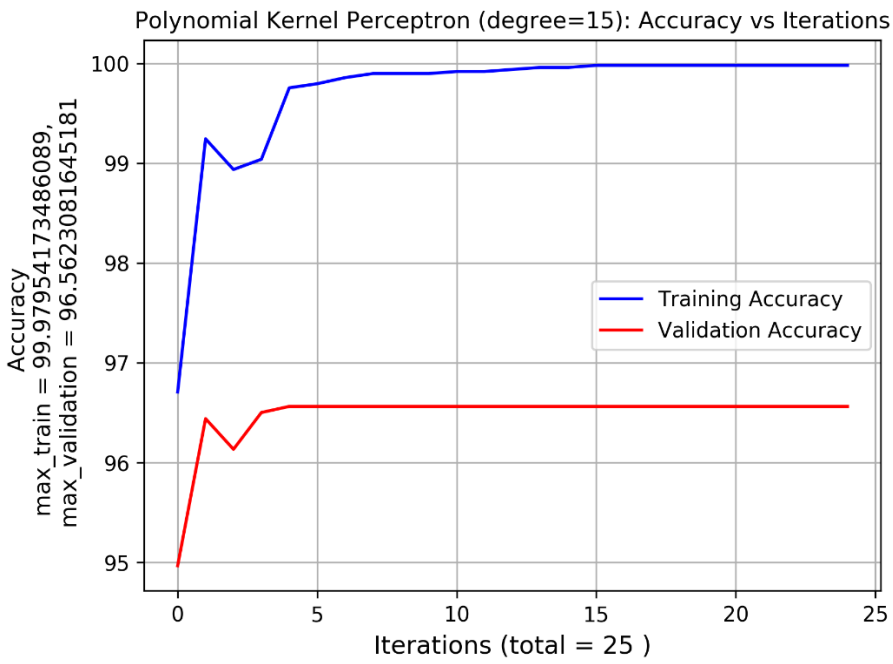
3. Train and Validation Accuracies for $P = 3$



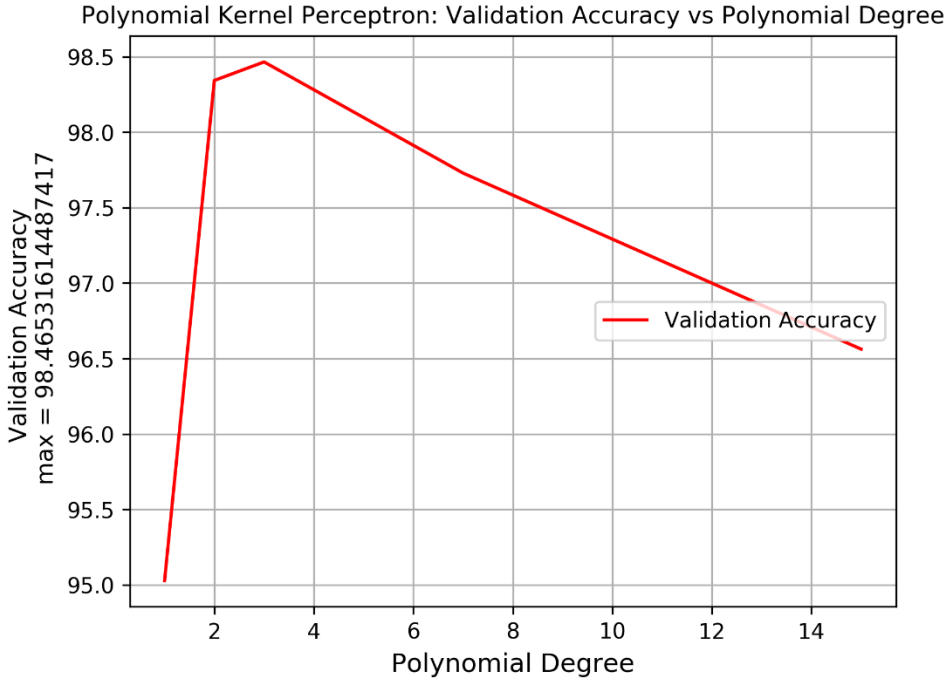
4. Train and Validation Accuracies for $P = 7$



5. Train and Validation Accuracies for $P = 15$



6. Validation Accuracies v/s P



d) Effect of P on Training and Validation Accuracies:

Increasing P for the Kernel Function in the dual space is equivalent to increasing the dimensionality of corresponding basis functions in the primal space. Hence increasing P increases the model capacity and makes it more prone to overfitting.

This is confirmed from the obtained plots for Training and Validation Accuracies versus P. We see that both the Training and Validation Accuracies increase for P = { 1, 2, 3 }. For higher values of P, the Training Accuracy increases to 100 percent while the Validation Accuracy starts dropping from 98.5 percent to 96 percent – hence signifying overfitting

Conclusion

For the given Training set we generated a perceptron model. We explored vanilla, average, and kernel perceptron. In addition, we discussed overfitting, and experimented with different polynomial kernels. Source file, report, oplabel, and kplabel are included with the assignment.