Homework 1

Reet Barik, WSU ID: 11630142
CptS 570 Machine Learning

October 2, 2018

Exercise 1. Answer the following questions with a yes or no along with proper justification.
a. Is the decision boundary of voted perceptron linear?
b. Is the decision boundary of averaged perceptron linear?

Answer:

A) No, the decision boundary of voted perceptron is not linear. This is so because, in case of Voted Perceptron the output is of the form (W1,C1), (W2,C2), (W3,C3),... where Wn are the classifiers that are stored along with the counts of how long each classifier survived which are denoted by Cn. Each of the Wn represents a classifier and consequently a linear decision boundary which can successfully classify only a subset of the training examples. Hence there is no global decision boundary for all the training examples. As a result, the individual decision boundaries of the classifiers might be linear but the global one isnt.

B) Yes, the decision boundary of averaged perceptron is linear. This is so because, in case of Averaged Perceptron, the average of all classifiers that is learned is taken as the output. Hence at the end of the run time of the algorithm, only 1 classifier is considered which has a linear decision boundary. Therefore, it can be said that the decision boundary of the Averaged Perceptron is linear.

Exercise 2. In the class, we saw the Passive-Aggressive (PA) update that tries to achieve a margin equal to one after each update. Derive the PA weight update for achieving margin M.

Answer:

In the aggressive update step, $w_{t+1} = min_w(1/2||w - w_t||^2)$ s.t. $y_t(w.x_t) >= 1$
But here, the margin is M and not 1. therefore, the constraint becomes $y_t(w.x_t) >= M$.
The Lagrangian is as follows: $L(w, T) = 1/2||w - w_t||^2 + T(M - y_t(w.x_t))$
Solve for the dual: $max_{T>=0} min_w L(w, T)$
Optimize for w: $dL/dw = w - w_t - Ty_t x_t$
Set the derivative to zero: $w = w_t + Ty_t x_t$
Substituting back into the Lagrangian: $L(T) = -1/2||x_t||^2 T^2 + T(M - y_t(w_t.x_t))$
Dual optimization problem: $max_{T>=0} - 1/2||x_t||^2 T^2 + T(M - y_t(w_t.x_t))$

On solving it, we get: $T = max(0, (M - y_t(w_t.x_t))/(||x_t||^2))$
This value of 'T' is substituted into the equation $w_{t+1} = w_t + Ty_tx_t$ to obtain the the PA weight update for achieving margin M.

Exercise 3. Consider the following setting. You are provided with n training examples: (x1, y1, h1),(x2, y2, h2),... ,(xn, yn, hn), where xi is the input example, yi is the class label (+1 or -1), and hi > 0 is the importance weight of the example. The teacher gave you some additional information by specifying the importance of each training example.
a. How will you modify the perceptron algorithm to be able to leverage this extra information? Please justify your answer.
b. How can you solve this learning problem using the standard perceptron algorithm? Please justify your answer. I'm looking for a reduction based solution.

Answer:

A) To incorporate this extra information in the form of h_i which is the importance of the i-th example, we can modify the standard perceptron algorithm as follows:
During the weight update in case of a mistake, the standard algorithm dictates that it occur as $w_{t+1} = w_t + y_tx_t$. But we are also supplied with the importance of each example in the form of h_i which can be used to modify the update in the following manner:
$w_{t+1} = w_t + h_t(y_tx_t)$.
This way, the magnitude of the update can be made dependant on the importance of that particular example.

B) To solve this learning problem using the standard perceptron algorithm, the former needs to be mapped to the latter. This can be done as follows:
Each h_i is incorporated into the i-th feature vector. This is so because h_i signifies the importance of the i-th example and hence can be thought of as a feature in itself. Once this has been decided, the learning problem reduces to one which can be solved by the standard perceptron algorithm. Therefore, for example, if $x_i = [1, 2, 3, 4, 5, 6]$ and $h_i = 9$, then in the standard perceptron algorithm, the feature vector will become $x_i = [1, 2, 3, 4, 5, 6, 9]$. The rest of the algorithm remains the same as the standard perceptron for binary classification.

Exercise 4. Consider the following setting. You are provided with n training examples: (x1, y1),(x2, y2),...,(xn, yn), where xi is the input example, and yi is the class label (+1 or -1). However, the training data is highly imbalanced (say 90% of the examples are negative and 10% of the examples are positive) and we care more about the accuracy of positive examples.
a. How will you modify the perceptron algorithm to solve this learning problem? Please justify your answer.
b. How can you solve this learning problem using the standard perceptron algorithm? Please justify your answer. I'm looking for a reduction based solution.

Answer:

A) To tackle highly imbalanced training data, the perceptron algorithm can be modified as follows:
We replace the update condition $y_n(w^T.x_n + b) < 0$ with $y_n(w^T.x_n + b) < \gamma$, where γ is the hyperparameter signifying minimum margin. To deal with the highly imbalanced classes, we take into consideration 2 different margins γ_+ and γ_- instead of a single margin for both the positive and negative classes. Since the algorithm needs to be more careful about the minority class (here, its the positive class constituting only 10% of the training examples), its corresponding margin needs to be the larger one. The relative size of the margins is computed by the factor by which the number of training examples in the negative class dominates the positive class. Here, the factor is 9. Therefore, here, $\gamma_+ = 9.\gamma_-$. The rest of the algorithm remains the same as the standard perceptron for binary classification.

B) This learning problem can be tackled by using the Standard Perceptron Algorithm if we can synthetically make the number of examples belonging to the minority class the same as that of the majority class. This can be done as follows:
For each iteration, we take all the examples of the minority class and a random subset of the examples of the majority class, such that the number of training examples from both classes are similar. For the subsequent iterations we choose a different subset of the majority class keeping the minority class examples constant. The rest of the algorithm remains the same as the standard perceptron for binary classification.
For example, if we have 100 training examples with 10 positive class labels and 90 negative class labels, for each iteration, we take the 10 positive examples (minority class) and 10 random examples of the negative class (majority class) and train a Standard Perceptron Algorithm, we can combat the unbalanced nature of the training data.

Exercise 5. Suppose we have n+ positive training examples and n negative training examples. Let C+ be the center of the positive examples and C be the center of the negative examples, i.e., $C+ = 1/n_+ \sum_{i:y_i=+1} x_i$ and $C- = -1/n_- \sum_{i:y_i=-1} x_i$. Consider a simple classifier called CLOSE that classifies a test example x by assigning it to the class whose center is closest.
a. Show that the decision boundary of the CLOSE classifier is a linear hyperplane of the form sign(w . x + b). Compute the values of w and b in terms of C+ and C.
b. Recall that the weight vector can be written as a linear combination of all the training examples: $w = \sum_{i=1}^{n_+ + n_-} \alpha i_i y_i x_i$. Compute the dual weights ($\alpha' s$). How many of the training examples are support vectors?

Answer:

A) The hyperplane for the CLOSE classifier consists of points which are equidistant from the 2 Centroids denoted by C+ and C-. Since this is a straight line or a linear hyperplane like in the standard perceptron algorithm, it is of the form sign(w . x + b).
We know that w is the distance between the 2 Centroids. Therefore it is the distance between C+ and C-. Hence, $w = |C_+ - C_-|$.
Now, we draw a line from C+ to C-. Let the point of intersection between that line and the hyperplane be P. Then, $OP = \frac{1}{2}|C_+ + C_-|$ [From Parallelogram law of vector addition]. The

point P satisfies the hyperplane. Hence, substituting, we get
$|C_+ - C_-|.1/2.|C_+ - C_-|) + b = 0$.
Therefore, $b = \frac{1}{2}|C_+^2 - C_-^2|$.

B) We know that the weight vector can be written as a linear combination of all the training examples: $w = \sum_{i=1}^{n_+ + n_-} \alpha i_i y_i x_i$
Also, $C+ = 1/n_+ \sum_{i:y_i=+1} x_i$ and $C- = -1/n_- \sum_{i:y_i=-1} x_i$.
Now, from part A, we know:

$$\begin{aligned} w &= |C_+ - C_-| \\ &= 1/n_+ \sum_{i:y_i=+1} x_i - 1/n_- \sum_{i:y_i=-1} x_i \\ &= 1/n_+ \sum_i^{n_+} y_i x_i + 1/n_- \sum_i^{n_-} y_i x_i \end{aligned}$$

Now, we know $w = \sum_i^{n_+} \alpha_i y_i x_i + \sum_i^{n_-} \alpha_i y_i x_i$ $\quad [Since, w = \sum_{i=1}^{n_+ + n_-} \alpha i_i y_i x_i]$
Hence, $\forall y = +1, \alpha_i = \frac{1}{n_+}$ and $\forall y = -1, \alpha_i = \frac{1}{n_-}$

Exercise 6. Suppose we use the following radial basis function (RBF) kernel: K(xi, xj) = $exp(-1/2||x_i - x_j||^2)$ which has some implicit unknown mapping $\phi(x)$.
a. Prove that the mapping $\phi(x)$ corresponding to RBF kernel has infinite dimensions.
b. Prove that for any two input examples xi and xj , the squared Euclidean distance of their corresponding points in the higher-dimensional space defined by f is less than 2, i.e., $||\phi(x_i) - \phi(x_j)||^2 <= 2$.

Answer:

A) The RBF Kernel is as follows:

$$\begin{aligned} K(x_i, x_j) &= exp(-1/2||x_i - x_j||^2) \\ &= exp(-1/2(||x_i||^2 + ||x_j||^2 - 2 < x_i.x_j >)) \\ &= exp(-1/2||x_i||^2 - 1/2||x_j||^2) exp(-1/2 - 2 < x_i.x_j >) \\ &= Ce^{<x_i.x_j>} \qquad [Since, C := exp(-1/2||x_i||^2 - 1/2||x_j||^2) \quad is \quad a \quad constant] \\ &= \sum_{n=0}^{\infty} \frac{< x_i.x_j >^n}{n!} \qquad [Taylor \quad Expansion \quad of \quad e^x] \\ &= \sum_{n=0}^{\infty} \frac{(||x_i||||x_j||cos\theta)^n}{n!} \\ &= \sum_{n=0}^{\infty} \phi(x_i)\phi(x_j) \end{aligned}$$

This means that the vectors x_i and x_j will have infinite dimensions and hence the RBF Kernel maps into infinite dimensional space.

B) The required proof is as follows:

$$\begin{aligned}||\phi(x_i)-\phi(x_j)||^2 &= <\phi(x_i),\phi(x_i)> + <\phi(x_j),\phi(x_j)> -2. <\phi(x_i),\phi(x_j)> \\ &= K(x_i,x_i)+K(x_j,x_j)-2.K(x_i,x_j) \\ &= 1+1-2exp(-1/2||x_i-x_j||^2) \\ &\leq 2\end{aligned}$$

Exercise 7. The decision boundary of a SVM with a kernel function (via implicit feature mapping $\phi(.)$) is defined as follows:
$w.\phi(x)+b=\sum_{i\in SV} y_i\alpha_i K(x_i,x)+b=f(x;\alpha,b)$, where w and b are parameters of the decision boundary in the feature space phi defined by the kernel function K, SV is the set of support vectors, and α_i is the dual weight of the i-th support vector. Let us assume that we use the radial basis function (RBF) kernel $K(x_i,x_j)=exp(1/2||x_i-x_j||^2$; also assume that the training examples are linearly separable in the feature space ϕ and SVM finds a decision boundary that perfectly separates the training examples.
If we choose a testing example x_{far} that is far away from any training instance x_i (distance here is measured in the original feature space R^d). Prove that $f(x_{far};\alpha,b)\approx b$.

Answer:

Since x_{far} is far away from any training example x_i, we can say that,

$$\begin{aligned}&||x_{far}-x_i|| \gg 0, \forall i\in SV \\ \Rightarrow\ & K(x_{far},x_i)\approx 0, \forall i\in SV \\ \Rightarrow\ & \sum_{i\in SV} y_i.\alpha_i.K(x_{far},x_i)\approx 0 \\ \Rightarrow\ & f(x_{far};\alpha,b)\approx b.[Proved]\end{aligned}$$

Exercise 8. The function $K(x_i,x_j)=-<x_i,x_j>$ is a valid kernel. Prove or Disprove it.

Answer:

From Mercer's Theorem, we know that a function K is a Kernel function if and only if its corresponding Kernel matrix is symmetric and positive semi-definite.
We know that K $=<x_i,x_j>$ is a valid Kernel. Let K' $=-<x_i,x_j>$.
We observe that K' = - K. We know that any symmetric matrix multiplied by a scalar results in another symmetric matrix. We know that the matrix for K is symmetric. Hence, the Kernel matrix for -K will be symmetric as well.
We know that the Kernel matrix for K is positive semi-definite (Since K is a Kernel). Which implies, $x^TKx\geq 0$. But we know that K' = - K. Which means, $x^TK'x \ngeq 0$.
Hence, the Kernel matrix for K' is not positive semi-definite.
That is why, $K(x_i,x_j)=-<x_i,x_j>$ is NOT a valid Kernel.

Exercise 9. You are provided with n training examples: (x1, y1),(x2, y2), ...,(xn, yn,), where xi is the input example, yi is the class label (+1 or -1). The teacher gave you some additional information by specifying the costs for different mistakes C+ and C, where C+ and C- stand for the cost of miss-classifying a positive and negative example respectively.
a. How will you modify the Soft-margin SVM formulation to be able to leverage this extra information? Please justify your answer.

Answer:

We know that the expression for the Soft Margin SVM is as follows:
$min_{w,b} 1/2||w||^2 + C\sum_{i=1}^{N}\xi_i$, subject to: $y^i(w.x^i + b) \geq 1 - \xi_i; i = 1, ..., N; \xi_i \geq 0;$
Now, we have been provided additional information in the form of Cost for different mistakes C+ and C-, where C+ and C- stand for the cost of miss-classifying a positive and negative example respectively. We can modify the Soft-margin SVM formulation in the following way: For a positive class, we take the C parameter value as C_+ and for the negative class, C_-. So, the formulation of the Soft Margin SVM becomes the following:

For all the 'p' positive examples,
$min_{w,b} 1/2||w||^2 + C_+\sum_{i=1}^{p}\xi_i$, subject to: $y^i(w.x^i + b) \geq 1 - \xi_i; i = 1, ..., p; \xi_i \geq 0;$

For all the 'q' negative examples,
$min_{w,b} 1/2||w||^2 + C_-\sum_{i=1}^{q}\xi_i$, subject to: $y^i(w.x^i + b) \geq 1 - \xi_i; i = 1, ..., q; \xi_i \geq 0;$

C is the parameter which trades off between the slackness of margin and the accuracy. Hence, it makes sense to change C to C_+ and C_- for positive and negative classes respectively in order to make it dependant on the different costs.

Exercise 10. Consider the following setting. You are provided with n training examples: (x1, y1, h1),(x2, y2, h2), ... ,(xn, yn, hn), where xi is the input example, yi is the class label (+1 or -1), and hi > 0 is the importance weight of the example. The teacher gave you some additional information by specifying the importance of each training example.
a. How will you modify the Soft-margin SVM formulation to be able to leverage this extra information? Please justify your answer.
b. How can you solve this learning problem using the standard SVM training algorithm? Please justify your answer.

Answer:

A) We know that the expression for the Soft Margin SVM is as follows:
$min_{w,b} 1/2||w||^2 + C\sum_{i=1}^{N}\xi_i$, subject to: $y^i(w.x^i + b) \geq 1 - \xi_i; i = 1, ..., N; \xi_i \geq 0;$. Here, C is the cost of error and ξ is the slack variable.
Now, we are given additional information in the form of importance of each training example, h_i. We have to minimize the cost of error for proportional to the importance of each example. For example, if an example with high importance h is miss-classified, then we need to correct

it with minimum cost. To reduce the cost, we multiply $1/h_i$ to $C\sum_{i=1}^{N}\xi_i$. As a result, the modified Soft Margin SVM formulation is as follows:
$min_{w,b}1/2||w||^2 + C\sum_{i=1}^{N}\frac{1}{h_i}\xi_i$, subject to: $y^i(w.x^i + b) \geq 1 - \xi_i; i = 1, ..., N; \xi_i \geq 0;$

B) Here, the standard SVM training algorithm can be used if we can manipulate the data such that there it reflects the importance of each example. This can be achieved by over-sampling of the important data points. Over-sampling will increase the number of more important training examples available. Though there is a chance of over-fitting, it will over-fit more on the more important examples which is a trade-off we are willing to make.

Exercise 11. You are provided with a set of n training examples: (x1, y1),(x2, y2), ,(xn, yn,), where xi is the input example, yi is the class label (+1 or -1). Suppose n is very large (say in the order of millions). In this case, standard SVM training algorithms will not scale due to large training set.
Tom wants to devise a solution based on Coarse-to-Fine framework of problem solving. The basic idea is to cluster the training data; train a SVM classifier based on the clusters (coarse problem); refine the clusters as needed (fine problem); perform training on the finer problem; and repeat until convergence. Suppose we start with k+ positive clusters and k- negative clusters to begin with (a cluster is defined as a set of examples). Please specify the mathematical formulation (define all the variables used in your formulation) and concrete algorithm for each of the following steps to instantiate this idea:
a) How to define the SVM training formulation for a given level of coarseness: a set of k+ positive clusters and a set of k negative clusters?
b) How to refine the clusters based on the resulting SVM classifier?
c) What is the stopping criteria?

Answer:

A) We are given a set of k_+ positive clusters and k_- negative clusters. We find out the Centroid of each such cluster using the information provided in question 5 of the Homework Assignment. Once we have identified the Centroids of the clusters, we train a linear SVM classifier on those centroids. This will give us a soft margin SVM formulation with $k_+ + k_-$ constraints. Therefore, the SVM formulation becomes:
$min_{w,b}1/2||w||^2 + C\sum_{i=1}^{k_++k_-}\xi_i$, subject to: $y^i(w.x^i + b) \geq 1 - \xi_i; i = 1, ..., (k_+ + k_-); \xi_i \geq 0;$

B) The 'fine' part of the Course to fine framework can be implemented by further slitting the clusters. We split those clusters that are closer to the hyperplane and thereby increase the accuracy. Instead of splitting in a naive way, we can split them into an increasing number of clusters so that there is a significant increase in accuracy at each step. The number of splits at each step can be determined sequentially (linear) or by a steeper increasing function (exponential) to speed it up.

C) If the accuracy after a splitting step is not significantly better than the previous (determined by a pre-defined threshold), then we choose the number of splits from the previous level. This is our stopping criteria.

Exercise 12. You are provided with a set of n training examples: (x1, y1),(x2, y2), ... ,(xn,

yn,), where xi is the input example, yi is the class label (+1 or -1). Suppose n is very large (say in the order of millions). In this case, online kernelized Perceptron algorithms will not scale if the number of allowed support vectors are unbounded.
a) Suppose you have trained using kernelized Perceptron algorithm (without any bounds on support vectors) and got a set of support vectors SV . Tom wants to use this classifier for real-time prediction and cannot afford more than B kernel evaluations for each classification decision. Please give an algorithm to select B support vectors from SV . You need to motivate your design choices in order to convince Tom to use your solution.
b) Tom wants to train using kernelized Perceptron algorithm, but wants to use at most B support vectors during the training process. Please modify the standard kernelized Perceptron training algorithm (from class slides) for this new setting. You need to motivate your design choices in order to convince Tom to use your solution.

Answer:

A) The setting of the problem is as follows: While training, we do not have a bound on the number of Support Vectors and as a result, we get 'm' number of Support Vectors at the end of training. But during real-time prediction, there is a bound on the number of kernel evaluations: We can have at most 'B' kernel evaluations which means, we can use at most 'B' number of Support Vectors for prediction purposes. Hence, we need to come up with an algorithm that helps us choose 'B' Support Vectors from the 'm' learned SVs.
To choose one SV over another, we need to determine whether the former has more influence on the decision boundary than the latter which is possible when it is closer to the decision boundary. Therefore we need to find the expression of the distance of each Support Vector from the decision boundary and choose the best 'B' among those, or in other words, the 'B' closest Support Vectors to the decision boundary.
We can find the distance 'r' of a Support Vector 'x' from the decision boundary as follows: We know that the shortest distance of a point from a hyperplane is perpendicular to it and hence, parallel to $\vec{w}$. The unit vector in this direction is given by $\vec{w}/|\vec{w}|$. If the projection of the point x on the hyperplane is taken as x', then
$\vec{x'} = \vec{x} - yr\frac{\vec{w}}{|\vec{w}|}$. Here, multiplying by 'y' changes the sign for the 2 cases of $\vec{x}$ being on either side of the decision surface.
Moreover, since x' lies on the decision boundary, it satisfies $\vec{w}^T\vec{x} + b = 0$. Hence,
$\vec{w}^T(\vec{x} - yr\frac{\vec{w}}{|\vec{w}|}) + b = 0$. Solving for 'r', we get
$r = y\frac{\vec{w}^T\vec{x}+b}{|\vec{w}|}$.
This gives us the expression to compute the distance of a Support Vector from the decision boundary. We use this to sort the 'm' Support Vectors w.r.t 'r' and choose the 'B' best ones with the least 'r' value. Since we are using 'r' as the criteria to choose the best 'B' SVs, we ensure that the chosen 'B' SVs are the B most influential among all the Support Vectors.
[Note: The distance of an SV from the decision boundary was computed following the steps at https://nlp.stanford.edu/IR-book/html/htmledition/support-vector-machines-the-linearly-separable-case-1.html]

B) The difference from the previous problem is that here, we don't have the luxury to learn all the 'm' Support Vectors in an unbounded setting. We have a bound 'B' on the

number of Support Vectors we can use during training. For that, we need to keep a track of the number of Support Vectors we learn during training. and after the count reaches 'B', for every SV it learns, we check whether it is further away from the decision boundary than any of the already learned 'B' SVs. If it is, we discard the current SV, else, we keep the current SV and pop that SV from the set which is the least important thereby keeping the number of SVs a constant 'B'. The importance of each SV is computed by taking into account the number of times it is updated. The more number of times an SV is updated, it is further away from the decision boundary and consequently its importance decreases. The standard kernelized Perceptron training algorithm can modified as follows to incorporate this idea:
Initialize α to an all-zero vector of length 'm' such that the count of times each SV is updated is stored as well.

```
for itr in (iterations):
    for each training example xj,yj:
        $y\_hat = sign(\sum_{i=1}^{B} \alpha_i y_i K(x_i, x_j))$ [Use only the best B SVs i.e. with the least counts]
        if $y\_hat \neq y_j$:
            $\alpha_j = \alpha_j + 1$
            count($\alpha_j$) += 1.
```

Exercise 13. Implement a binary classifier with both perceptron and passive-aggressive (PA) weight update.
• Convert all features into binary format (0 or 1). Describe your conversion. How many features do you have (i.e., the dimensionality)?
• Implement Perceptron and Passive-Aggressive based classifier from these binary features.
• Compute the online learning curve for both Perceptron and PA algorithm by plotting the number of training iterations (1 to 5) on the x-axis and the number of mistakes on the y-axis. Compare the two curves and list your observations.
• Compute the accuracy of both Perceptron and PA algorithm on the training data, development data, and testing data for each training iteration (1 to 5). So you will have three accuracy curves for Perceptron and another three accuracy curves for PA algorithm. Compare the six curves and list your observations.
• Implement the averaged perceptron algorithm in both the naive way (one I wrote on the white board) and the smart way (Algorithm 7 from Hals book chapter). Measure the training time to perform 5 iterations for both implementations. Compute the accuracies on training data, development data, and testing data with averaged classifier and compare with those obtained using standard perceptron. List your observations.
• Compute the general learning curve (vary the number of training examples starting from 5000 in the increments of 5000) for 5 training iterations. Plot the number of training examples on x-axis and the accuracy (development and testing) on the y-axis. List your observations from these curves.

Answer:

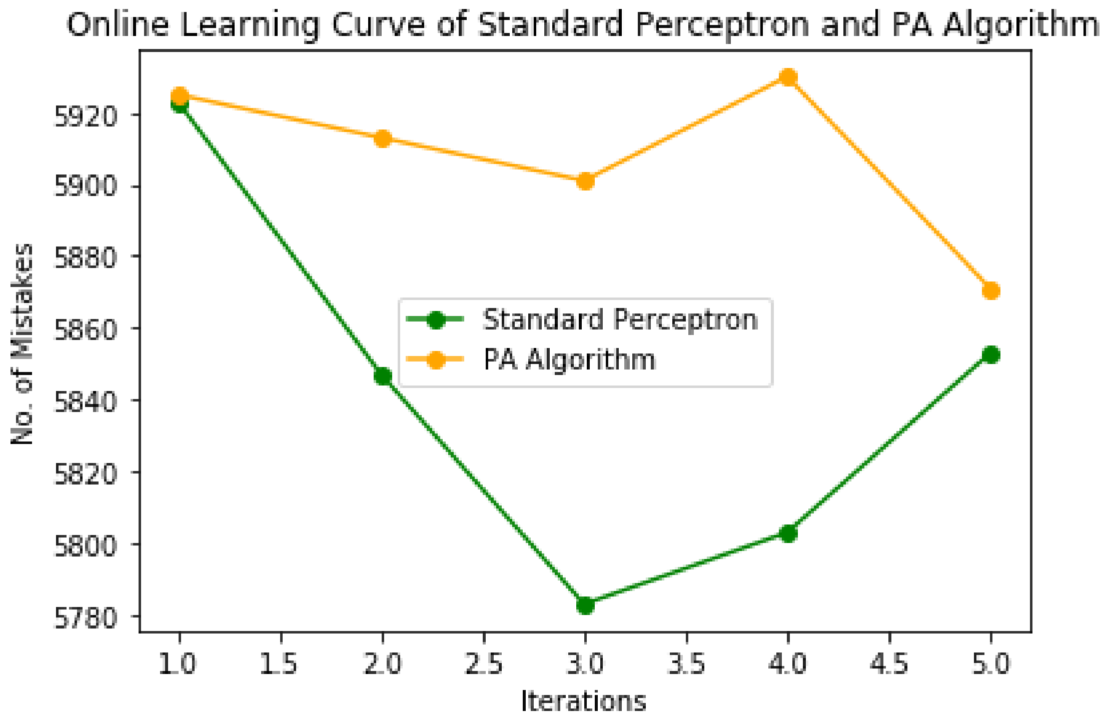
• The features have been converted to binary as follows:
1. Clean data: Drop rows which has missing values (Optional here since there are no rows

with missing values).
2. The numerical data was standardized using StandardScaler. (i.e. Mean of the data is 0 and Variance 1).
3. 'Age' and 'Workinghours' was then converted into bins. i.e. workinghours between 20 to 40 was labelled 'bin1', etc.
4. One hot encoding of the training, testing, and dev data to convert the qualitiative data to binary.
5. Reduce the number of columns (Done only in case of SVM training). Here, I have considered those columns which contain less than 2% of the total training example number of 1's and dropped them from the data-set since they don't contribute much to the weights. This significantly reduces the number of computations in case of SVM classifiers.

Here, the dimensonality of the features used for Q13 is 87 and that in Q14 (after reducing the number of columns) is 35.

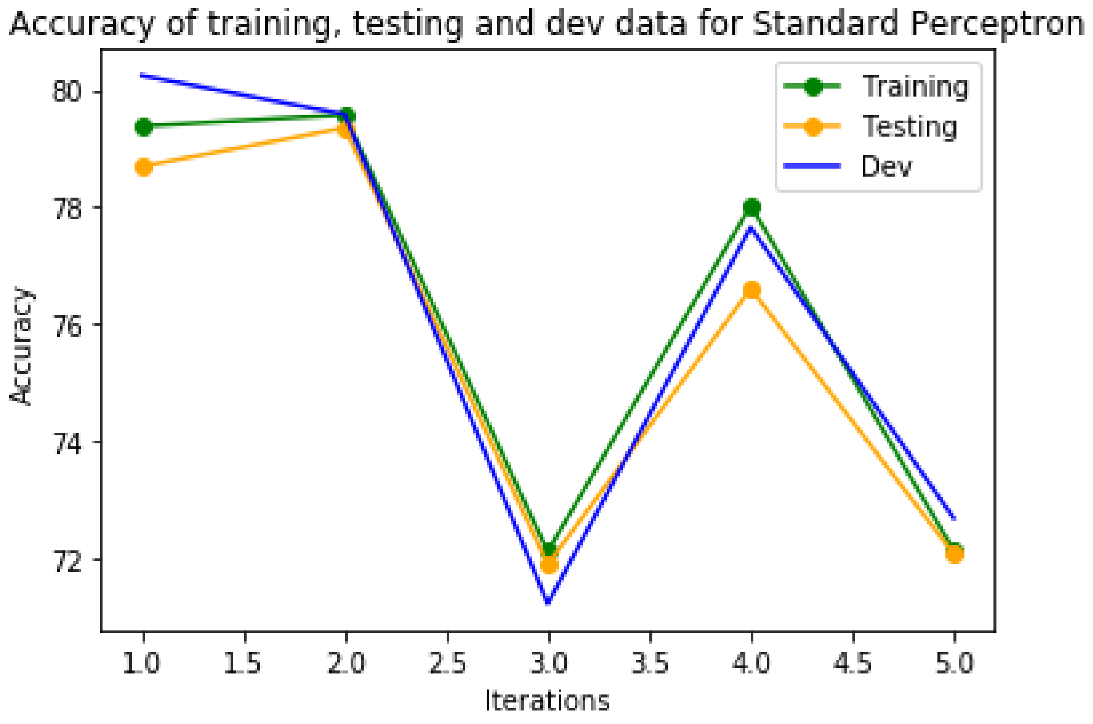
- Code for standard perceptron and passive aggressive classifier has been submitted.

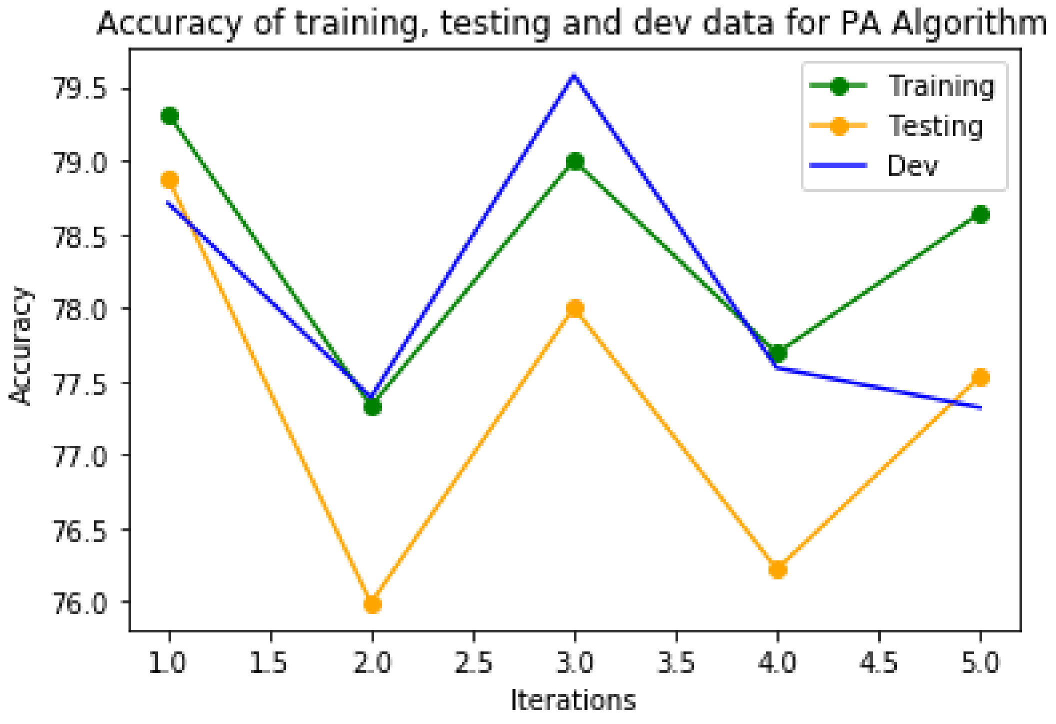
- Online learning curve for both Perceptron and PA algorithm is as follows:



Observations:
1. The number of mistakes is least at number of iterations equal to 3 and 5 for Standard Perceptron and PA algorithm respectively. Therefore, the hyperparamter, which is the number of iterations, could be set at 3 and 5 for Perceptron and PA algorithm respectively.
2. The standard perceptron performs better than the PA algorithm on the training set.

• Accuracy curves on training, testing and dev set for both standard Perceptron and PA algorithm:

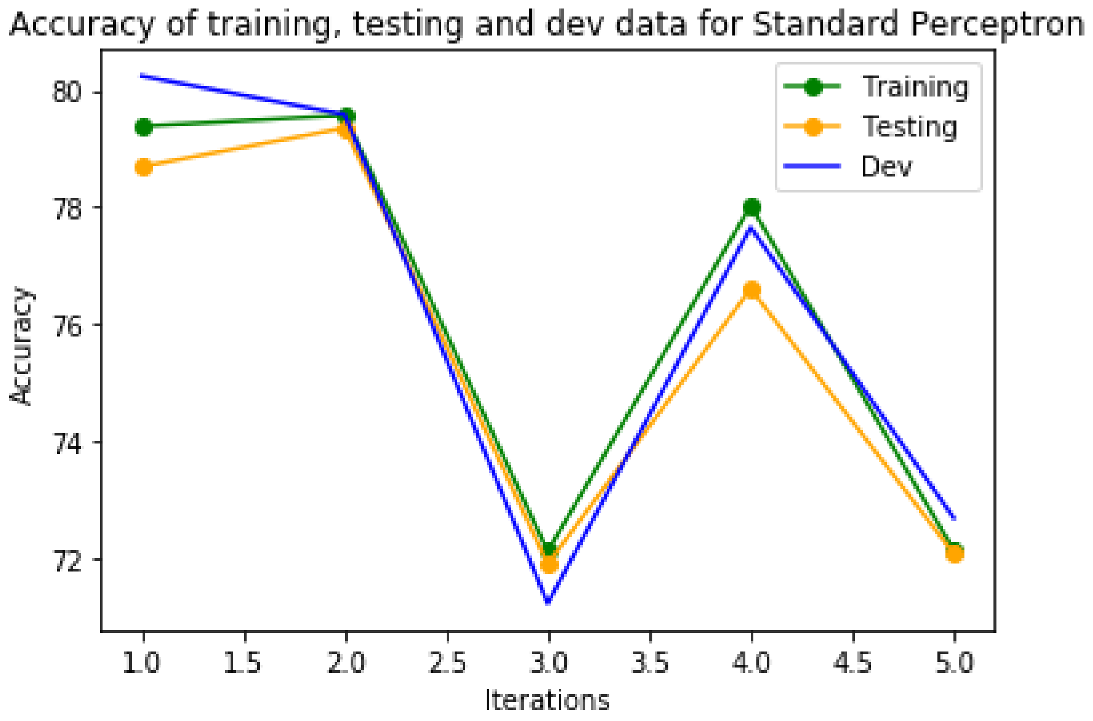


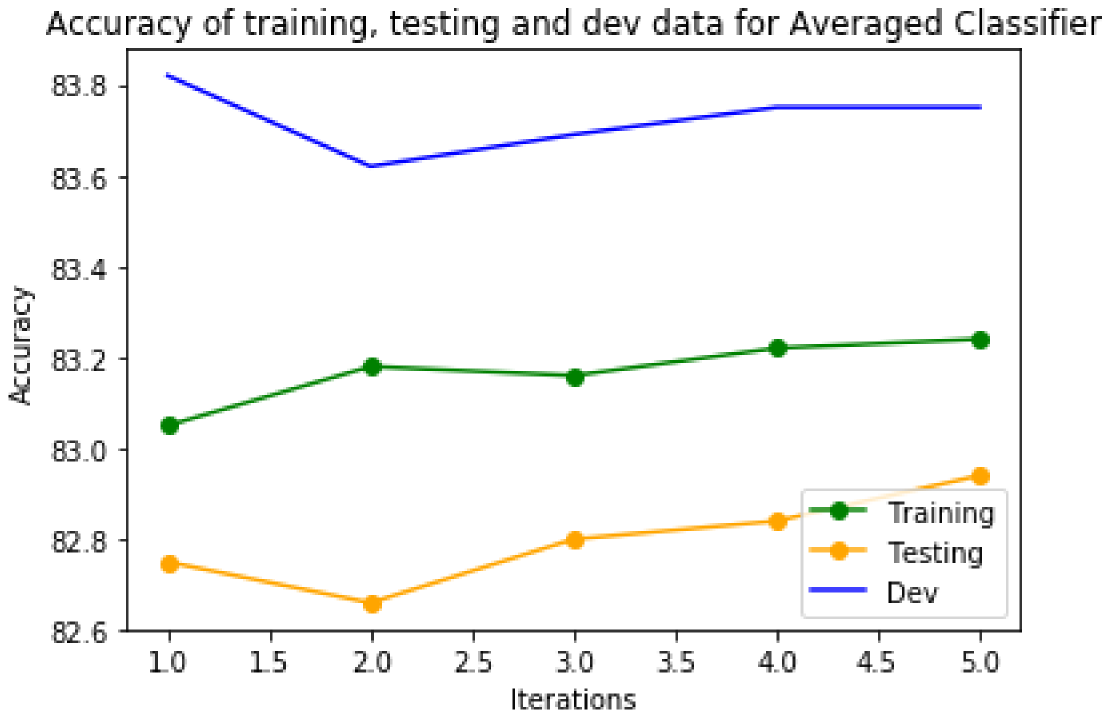


Observations:
1. The accuracy curve for training, testing and dev data follow the same nature for both Perceptron and PA algorithm.
2. Based on the performance on the dev data set, the hyperparamter iterations should be

set at 1 and 3 for Perceptron and PA algorithm respectively

• Code for Averaged Perceptron (naive and smart) has been submitted. Accuracy curve of training, testing and dev set for averaged perceptron and standard perceptron is as follows:

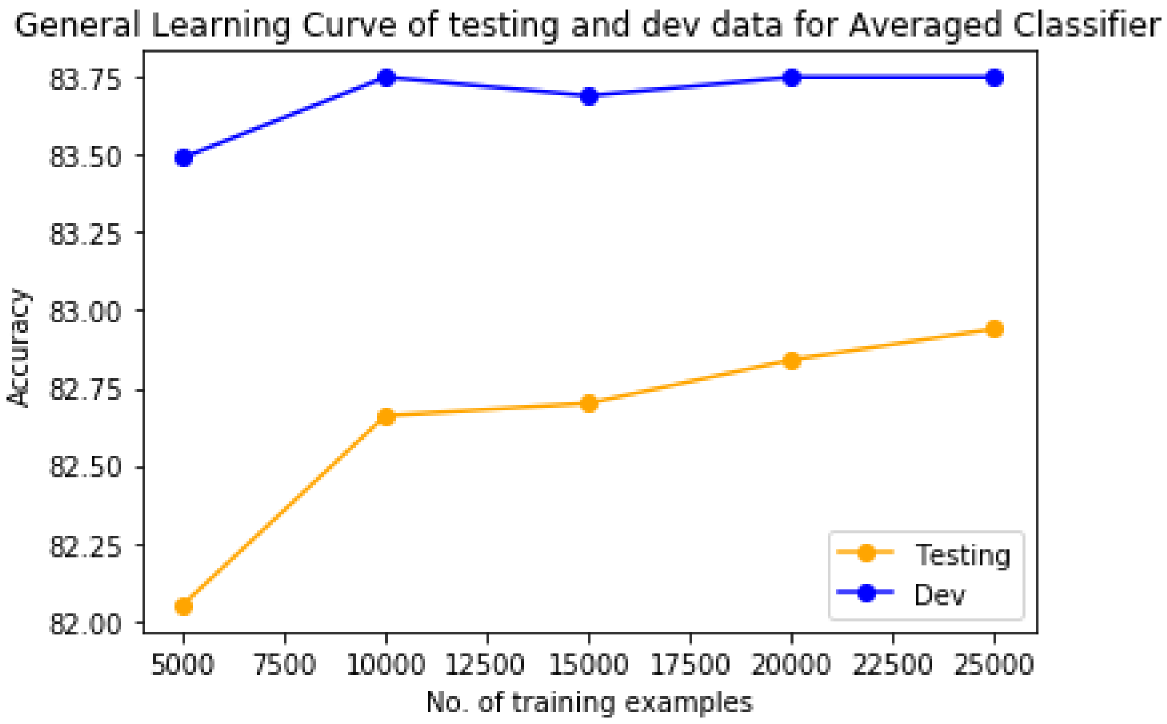




Training time need for naive averaged perceptron = 0.599637508392334 seconds.

Training time need for smart averaged perceptron = 0.48772096633911133 seconds.
Observations:
1. The accuracy curve is relatively more stable for Averaged Perceptron when compared to that of Standard Perceptron
2. The accuracy curve for all 3 sets have similar nature in both Standard Perceptron and Avergaed Perceptron.
3. Based on the performance on the dev data set, the hyperparamter iterations should be set at 1 for Averaged Perceptron.

- General Learning curve for Averaged Perceptron is as follows:



Observations:
1. The general learning curve for both the testing and dev set follow similar nature.
2. There is a general increase in accuracy with increase in the number of examples.

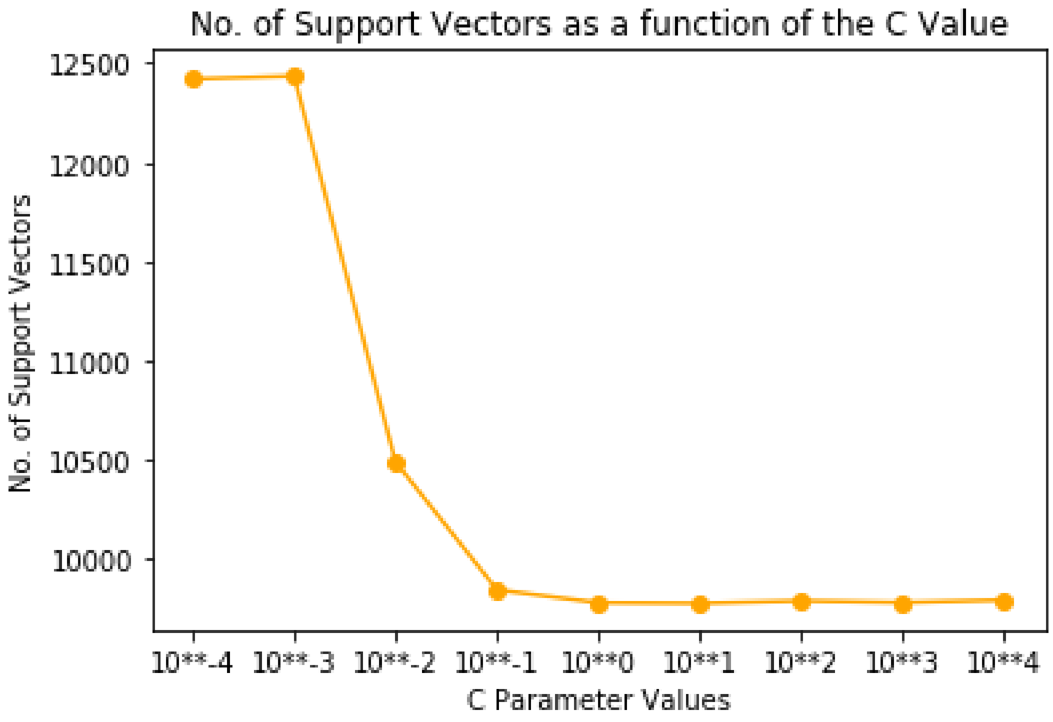
Exercise 14. Empirical analysis question.
•You can use a publicly available SVM classifier implementation (e.g., LibSVM or scikit-learn) for SVM related experiments. LibSVM (http://www.csie.ntu.edu.tw/ cjlin/libsvm/ and scikit-learn (http://scikit-learn.org/stable/modules/svm.html).
(a) Using a linear kernel, train the SVM on the training data for different values of C parameter: $10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{-3}, 10^{4}$. Compute the training accuracy, validation accuracy, and testing accuracy for the SVM obtained with different values of the C parameter. Plot the training accuracy, validation accuracy, and testing accuracy as a function of C (C value on x-axis and Accuracy on y-axis) one curve each for training, validation, and testing data. Also, plot the number of support vectors (see the training log at the end of SVM training for LibSVM or use the scikit-learn API if applicable) as a function of C. List

your observations.
(b) Select the best value of hyper-parameter C based on the accuracy on validation set and train a linear SVM on the combined set of training and validation examples. Compute the testing accuracy and the corresponding confusion matrix: a 2 2 matrix.
(c) Repeat the experiment (a) with the best C value from (a) with polynomial kernel of degree 2, 3, and 4. Compare the training, validation, testing accuracies, and the number of support vectors for diffent kernels (linear, polynomial kernel of degree 2, polynomial kernel of degree 3, and polynomial kernel of degree 4). List your observations.
•You will implement the standard kernelized Perceptron training algorithm (from class slides) for binary classification.
a. Train the binary classifier for 5 iterations with polynomial kernel (pick the best degree out of 2, 3, and 4 from the above experiment). Plot the number of mistakes as a function of training iterations. Compute the training, development, and testing accuracy at the end of 5 iterations.

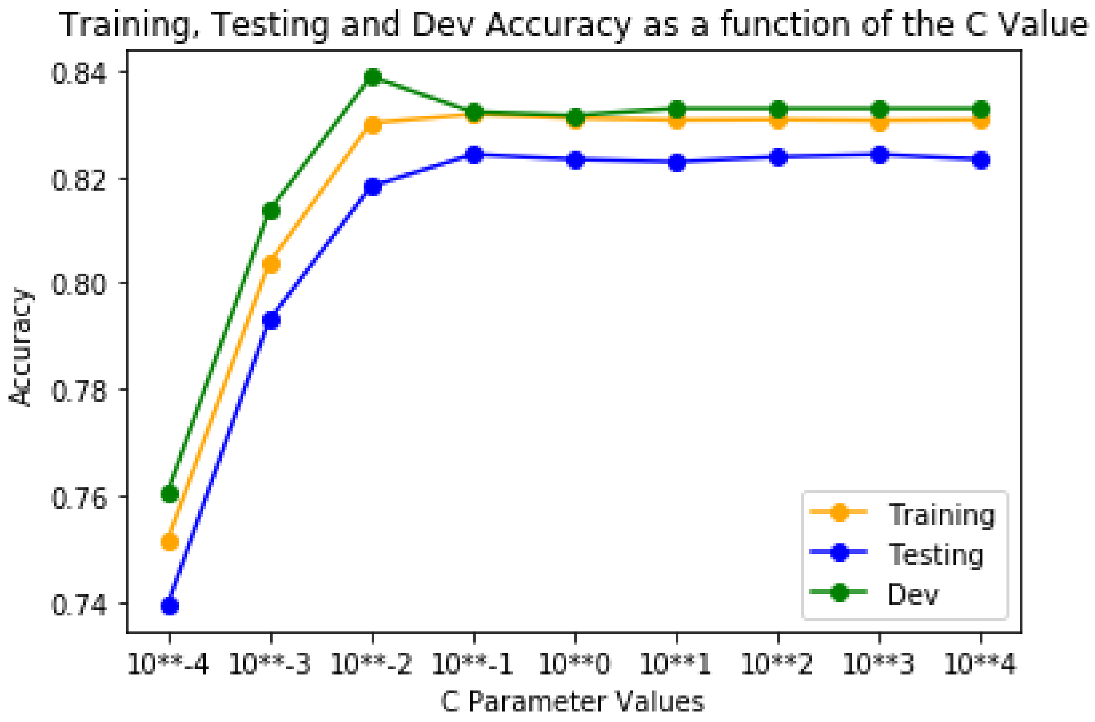
Answer:

- Empirical Analysis question:

A)
Plot of number of Support Vectors for different values of C:



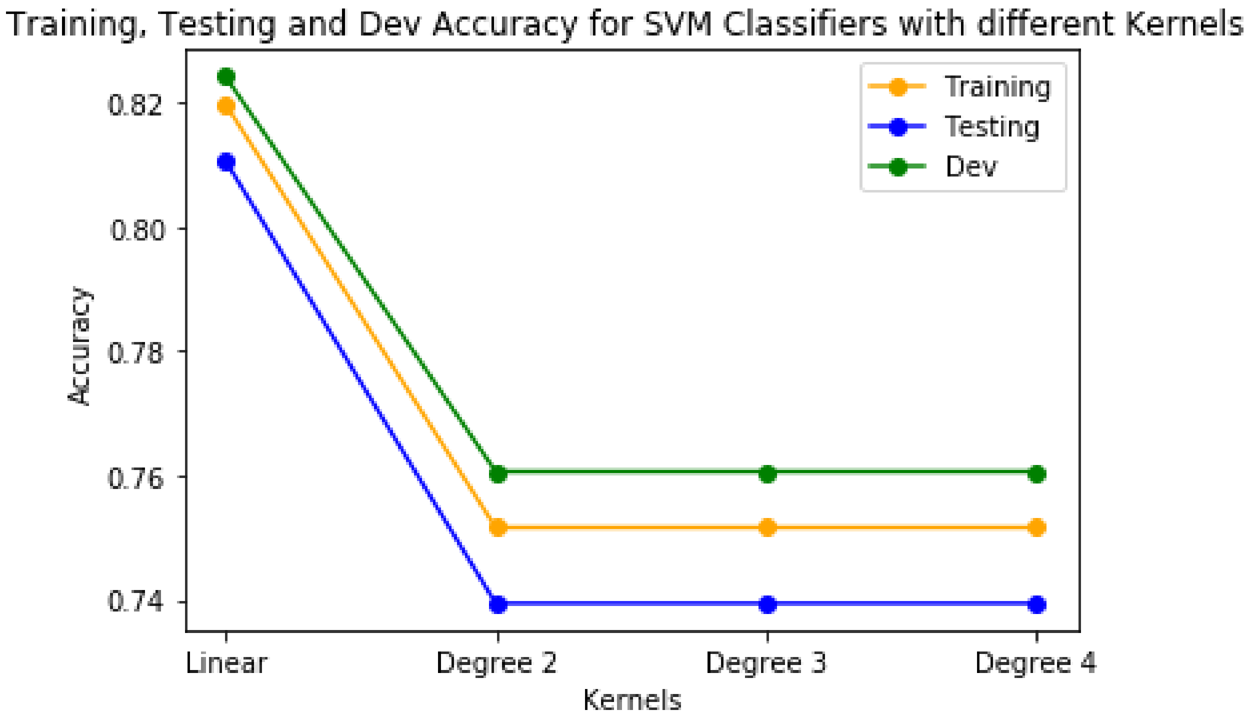
Plot of training, testing and dev accuracies for different values of C:



B) The testing accuracy for a linear SVM trained on the combined set of training and dev data is : 0.8107

The Confusion Matrix is : [[1425, 161], [245, 314]]

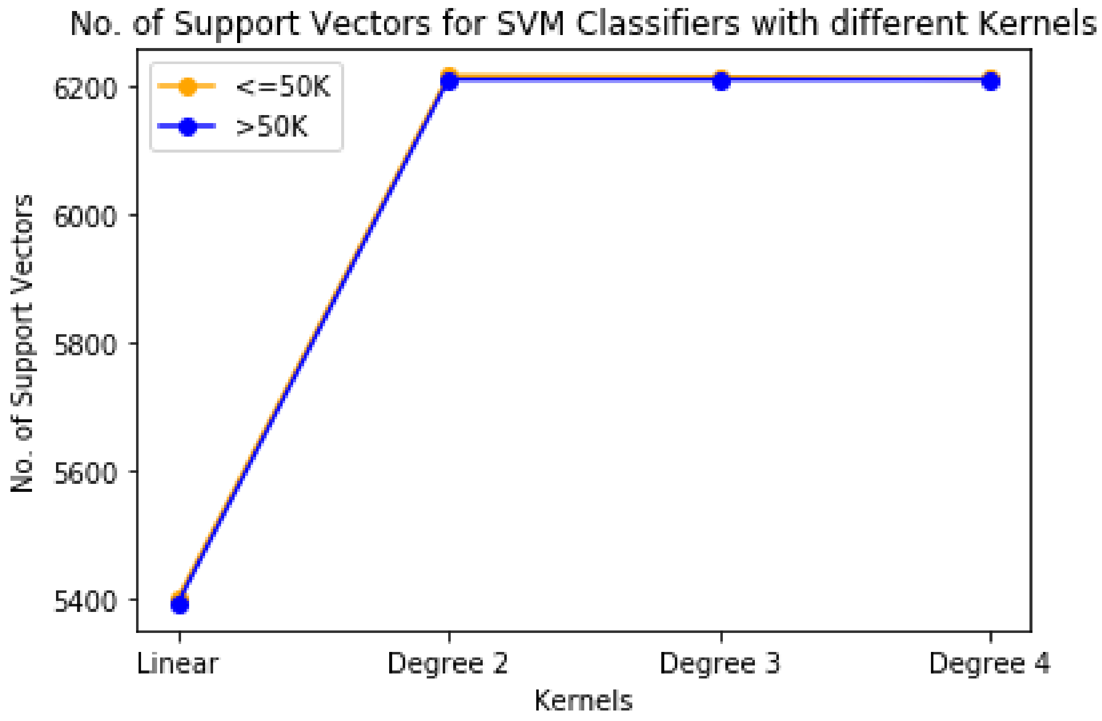
C) The training, validation, testing accuracies for different Kernels:



Observations:

1. The accuracy curve for training, testing and dev data follow the same nature for all the

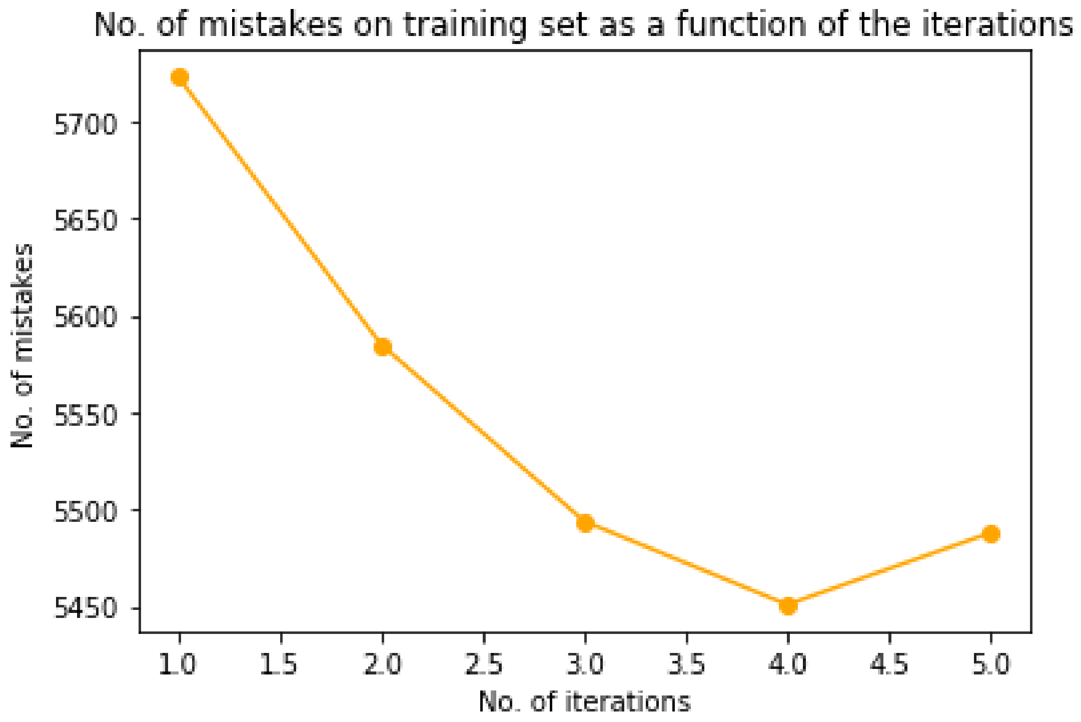
different kernels.
2. Linear Kernel gives the best accuracy when compared to the other polynomial Kernels
Number of Support Vectors for different Kernels:



Observations:
1. The curves for the number of both postive and negative support vectors are almost the same.
2. Linear Kernel has the least number of Support Vectors when compared to the other polynomial kernels.

- Kernelized Perceptron:

Plot of number of mistakes as a function of iterations:



Observations:
1. Number of mistakes is the least for the number of iterations = 4. Hence, the value of the hyperparameter could be taken as 4.
Training accuracy = 24.94
Testing accuracy = 76.06
Dev accuracy = 76.06