Supervised Learning CW2

Shiqi Su

January 01, 2020

Part 1: Introduction to Multi-classification with Kernel Perceptron

1 Classic Perceptron binary classificaiton

This paper address the question of multi-classification by using kernelised Perceptron. In classic binary classification given training set $\{(\mathbf{x_1}, y_1), ..., \mathbf{x_m}, y_m)\} \in (\mathcal{R}^n, \{-1, +1\})^m$, where $\mathbf{x_i}$ is the tth instance and $y = \pm 1$ is the label. What's more the mapping between input and label can be expressed as weighted linear combination,

$$f(x) = sign(\mathbf{w} \cdot \mathbf{x_t})$$

where $\mathbf{w} = (w_1, ..., w_n)$ is weighted coefficient. This prime form is solvable when dimension smaller than number of training instances, i.e $n << m$. But when training data maps to higher dimension,i.e replace $\mathbf{x}$ with $\phi(\mathbf{x})$and involves complicated calculation we usually use dual form with Kernel trick which can reduce dimension and becomes low-cost computation. Thus binary classification in dual form can be presented as

$$f(x) = sign(\mathbf{w} \cdot \phi(\mathbf{x})) = sign(\sum_{i=1}^{m} \alpha_i \mathbf{K}(\mathbf{x_i}, \mathbf{x}))$$

the prediction of single label becomes

$$\hat{y}_t = sign(\sum_{i=1}^{m} \alpha_i \mathbf{K}(\mathbf{x_i}, \mathbf{x_t})) \tag{1}$$

In this paper, I choose 2 types kernel : Polynomial kernel and Gaussian Kernel. They are presented as following:

- Polynomial Kernel: $K_d(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_i})^d$
- Gaussian Kernel: $K(\mathbf{x_i}, \mathbf{x_j}) = e^{-c\|\mathbf{x_i} - \mathbf{x_j}\|^2}$

Besides discussing kernel trick we also need information about coefficient and according to Kernel Perceptron which requires updating coefficient $\alpha = (\alpha_1, ..., \alpha_m)^T$.

- **If** $\hat{y}_t = y_t$ then $\alpha_t = 0$
- **else** update $\alpha_t = y_t$

2 Extension: Multi-classification

By their nature perceptrons are essentially binary classifiers, however, they can be adopted to handle the multiple classification tasks common in remote sensing studies. The basic idea requires a multiclass analysis be broken down into a series of binary classifications, following either the one-against-one or one-against-all strategies.

2.1 One-Vs-All Algorithm

2.1.1 Training Dataset

In theoretical, OVA algorithm is described as :

- Learning: given a dataset $\{(\mathbf{x_1}, y_1), ..., (\mathbf{x_m}, y_m)\}, \mathbf{x_i} \in \Re^n, y_i \in \{1, 2, ..., k\}$
- Decompose to K binary classification tasks and learn K models: $f_1, ...f_k$
- For class k,construct a binary classification as:
  - $y_t^{(k)} = 1$: instance $\mathbf{x_t}$ in class k
  - $y_t^{(k)} = -1$: instance $\mathbf{x_t}$ not in class k
  - This strategy requires the base classifiers to produce a probability of class membership (confidence score) for its decision, rather than just a class label.

$$Confidence : f_k = \sum_{i=1}^{k-1} \alpha_i K(\mathbf{x_i}, \mathbf{x_t})$$

- Making decisions means applying all classifiers to an unseen sample x and predicting the label k for which the corresponding classifier reports the highest confidence score:

$$prediction : \hat{y} = \underset{k \in \{1,...,K\}}{argmax} \ f_k$$

  - If the prediction fails to correctly predict $\mathbf{x_t}$ in class k, update coefficient $\alpha_{t,k} = \alpha_{t,k} - 1$ where confidence score is positive meanwhile update $\alpha_{t,k} = \alpha_{t,k} + 1$ where where confidence score is non-positive.
  - else no updates needed

Algorithm 1 One-Vs-All Algorithm

```
1: Input:$(\mathbf{x_1}, y-1), ..., (\mathbf{x_m}, y_m) \in \Re^n \times \{-1, 1\}$
     1. Initialise $\alpha = \overrightarrow{0}, \mathbf{M} = 0$
     2. Repeat:
        (a) E=0
        (b) Receive pattern : $\mathbf{x_t} \in \Re^n$
        (c) For t=1 to m do $\mathbf{f_t} = \sum_{i=1}^{t-1} \alpha_i K(\mathbf{x_i}, \mathbf{x_t})$, where each elements denote as $f_t^{(k)}$
        (d) For k=1 to 10 predict $\hat{y}_t = \underset{k \in \{1,...,K\}}{argmax} \ \mathbf{f_t}$
        (e) Receive lable: $y_t$
        (f) If mistake $(\hat{y}_t * y_t \leq 0)$
            Then Update $\alpha_{t,k} = \alpha_{t,k} - 1, E = E + 1$ where $f_t^{(k)} > 0$ and $\alpha_{t,k} = \alpha_{t,k} + 1, E = E + 1$ where $f_t^{(k)} <= 0$.
            Else $\alpha_{t,k}, E$ unchanged
        (g) $M_i$ = E
     3. Until $|M_i - M_{i-1}| < 0$
   Output: Updated $\alpha$
```

2.1.2 Testing Trained Perceptron

Testing aims at measuring the performance of trained model, typically via errors. I can now use updated coefficient martix α to predict an unlabelled new instance $\mathbf{x_t}$ and recording testing error.

- Calculate kernel matrix $K(X_{train}, X_{test})$
- For $\mathbf{x_t}$ in testing set:
  - *confidence* $\mathbf{f_t} = \alpha^T \cdot K$
  - For each row $f_t$in confidence matrix, if $\underset{k\in\{1,\ldots,K\}}{argmax}\ f_t^{(k)} \neq \hat{y}_t$ then store testing error and prediction labels.

Through splitting data as 80% training and 20% testing, I perform 20 runs for dimension 1 to 7 and constructed a table which presents training errors and test errors with their mean and standard deviation.

Table 1: Basic results for Perceptron(Polynomial Kernel)

d	Training set error rate(%)	Test set error rate(%)
1	7.5867 ± 1.5759	9.5403 ± 1.5621
2	0.7065 ± 0.3968	3.6909 ± 0.5596
3	0.1828 ± 0.0894	3.2070 ± 0.3963
4	0.1102 ± 0.0892	2.9543 ± 0.2832
5	0.1418 ± 0.3603	2.9758 ± 0.5591
6	0.0437 ± 0.0272	2.8548 ± 0.3562
7	0.0383 ± 0.0177	2.9624 ± 0.3309

2.1.3 Add Cross Validation

K-fold cross-validation (CV) is widely adopted as a model selection criterion.[4] The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. Hence, in our situation 4 folds are used to train perceptron and the rest is for testing so that choosing best parameter d^* in range (1,7). Detailed implementation as follows:

Algorithm 2 k-fold Cross-Validation

```
1: Define set of hyperparameter combinations, D for current Kernel Perceptron
    1. Shuffle data into 80% training data and 20% testing data
    2. outer loop: For parameter combination $d_j$ in D:
        (a) Divide training data into K folds with approximately equal distribution:
        (b) inner loop: For fold $k_i$ in K folds
            i. Set fold $k_i$ as the validation set
            ii. Train model on remaining K-1 folds
            iii. Evaluate model performance on $k_i$ using validated error rate $v_i$
        (c) Calculate average performance (validated error rate $v_i$) over K folds, i.e $V_j = \frac{1}{K}\sum_{i=1}^{K} v_i$
    3. select $d^*$ such that $d^* = \underset{d\in D=\{1,\ldots,7\}}{argmin}\ \mathbf{V_j}$
```

Output: Selected d^*

For run in 20, repeat previous described procedure 20 times. Hence, I got results from 5-Fold Cross-Validation:

- Time taken: 0:34:02.583907
- Mean d^*: 5.4 with std: 0.86023
- Mean test error: 2.98925% with std: 0.41901%

2.1.4 Confusion Matrix

After adding 5-Fold Cross-Validation to choose d^*, it's easier to apply it on 20% test data so that can calculate predictions and comparing to true label in order to construct confusion matrix.

- For a single run do:
- Initialise confusion matrix $C = \overrightarrow{\mathbf{0}}_{(10,10)}$
- Testing using best parameter d^* chosen through 5-Fold Cross-Validation. Output prediction row.
- For tth row in prediction matrix, find largest element $f_t^{(k)}$ and its corresponding index k. If $f_t^k \neq \hat{y}$, update $C_{t,k} += 1$.

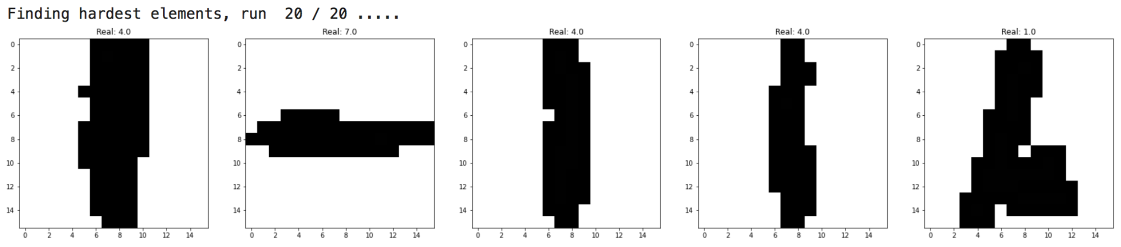
Figure 1: Confusion Matrix for Perceptron(%)(Polynomial Kernel)

	0	1	2	3	4	5	6	7	8	9
0	0.00 +- 0.00	30.00 +- 45.83	60.00 +- 91.65	50.00 +- 92.20	45.00 +- 58.95	40.00 +- 58.31	75.00 +- 94.21	15.00 +- 35.71	30.00 +- 45.83	40.00 +- 58.31
1	0.00 +- 0.00	0.00 +- 0.00	20.00 +- 40.00	5.00 +- 21.79	70.00 +- 78.10	0.00 +- 0.00	45.00 +- 66.90	10.00 +- 30.00	25.00 +- 43.30	10.00 +- 43.59
2	50.00 +- 80.62	30.00 +- 45.83	0.00 +- 0.00	80.00 +- 67.82	135.00 +- 101.37	20.00 +- 50.99	30.00 +- 55.68	70.00 +- 64.03	105.00 +- 92.06	15.00 +- 35.71
3	30.00 +- 55.68	30.00 +- 55.68	120.00 +- 107.70	0.00 +- 0.00	35.00 +- 57.23	335.00 +- 224.22	0.00 +- 0.00	85.00 +- 101.37	180.00 +- 132.66	30.00 +- 45.83
4	15.00 +- 35.71	110.00 +- 83.07	100.00 +- 77.46	15.00 +- 35.71	0.00 +- 0.00	50.00 +- 86.60	65.00 +- 85.29	45.00 +- 58.95	10.00 +- 30.00	180.00 +- 124.90
5	100.00 +- 94.87	15.00 +- 35.71	55.00 +- 58.95	135.00 +- 106.18	85.00 +- 96.31	0.00 +- 0.00	135.00 +- 135.19	10.00 +- 30.00	80.00 +- 97.98	50.00 +- 67.08
6	125.00 +- 94.21	50.00 +- 92.20	45.00 +- 66.90	5.00 +- 21.79	60.00 +- 86.02	55.00 +- 58.95	0.00 +- 0.00	0.00 +- 0.00	25.00 +- 53.62	5.00 +- 21.79
7	0.00 +- 0.00	45.00 +- 92.06	30.00 +- 45.83	5.00 +- 21.79	150.00 +- 156.52	15.00 +- 35.71	0.00 +- 0.00	0.00 +- 0.00	65.00 +- 72.63	140.00 +- 115.76
8	105.00 +- 107.12	80.00 +- 74.83	115.00 +- 90.97	195.00 +- 146.54	95.00 +- 80.47	170.00 +- 130.77	50.00 +- 92.20	45.00 +- 58.95	0.00 +- 0.00	25.00 +- 43.30
9	30.00 +- 45.83	10.00 +- 30.00	35.00 +- 65.38	50.00 +- 74.16	175.00 +- 189.41	30.00 +- 55.68	5.00 +- 21.79	145.00 +- 128.35	10.00 +- 30.00	0.00 +- 0.00

2.1.5 Hardest to Predict

In order to find 5 hardest $\mathbf{x_t}$ using least confidence score as a benchmark. The same as before, we need d^* to do the testing and return confidence matrix. For every $\mathbf{x_t}$ in test set, increase the confidence score by adding extra confidence of the predicted label,i.e $confidence_{(t,y_t)}/5$. And store updated score in a least confidence array with the same as test size. Finally,Sort the array of least confidence array in ascending order and use indices o first five elements.

Figure 2: 5 hardest elements to predict



```
Finding hardest elements, run  20 / 20 .....
```

2.1.6 Gaussian Kernel

In this section, Gaussian Kernel is applying instead of Polynomial Kernel.

Table 2: Basic results for Perceptron(Gaussian Kernel)

d	Training set error rate(%)	Test set error rate(%)
1	0.0013 ± 0.0040	6.7554 ± 0.5632
2	0.0000 ± 0.0000	6.9677 ± 0.4629
3	0.0013 ± 0.0040	7.0027 ± 0.5443
4	0.0034 ± 0.0058	6.8333 ± 0.6265
5	0.0034 ± 0.0072	7.1720 ± 0.6043
6	0.0013 ± 0.0040	8.4032 ± 0.4818
7	0.0027 ± 0.0069	10.3629 ± 0.6160

Using the previous defined process, I got following result when using 5-Fold Cross-Validation:

- Time taken: 0:37:39.421615
- Mean d*: 2.45 with std: 1.32193
- Mean test error(%): 6.94624 with std(%): 0.48015

2.1.7 Discussion: Polynomial Kernel and Gaussian Kernel

In machine learning, kernel functions are various based on specific algorithm. Such as polynomial kernel is a kernel function commonly used with support vector machines (SVMs),that represents the similarity of vectors (training samples) in a feature space over polynomials of the original variables, allowing learning of non-linear models.

In basic results,we observe that the Gaussian Kernel seems to be overfitting on the training data, which leads as a result to a higher test error rate, than the Polynomial Kernel.

After adding cross validation,the Gaussian Kernel seems to be underperforming in comparison with the Polynomial Kernel, in which we observed a mean test error rate of 2.98925% with std 0.41901%. While Gaussian Kernel we observed a mean test error rate of 6.94624% with std 0.48015%. The above results may demonstrate the fact that Gaussian Kernel selects solution more smooth so overfitting is possible.

Polynomial Kernel Perceptron, whose time complexity is quadratic $O(n^d)$ while Gaussian Kernel Perceptron's time complexity is exponential. Hence, Polynomial Kernel Perceptron becomes faster than Gaussian Kernel Perceptron when input size is large. This can be seen from empirical experiment, Gaussian Kernel costs almost 40 minutes while Polynomial costs 35 minutes.

2.2 One-Vs-One Algorithm

2.2.1 Training Dataset

An alternative is to introduce $k(k-1)/2$ binary discriminant functions, one for every possible pair of classes.[2] This is known as a one-versus-one classifier. Each point is then classified according to a majority vote amongst the discriminant functions.

- Given a dataset $\{(\mathbf{x_1}, y_1), ..., (\mathbf{x_m}, y_m)\}, \mathbf{x_i} \in \Re^n, y_i \in \{1, 2, ..., k\}$
- For every $1 \leq i < j \leq k$ we construct a binary training srquence $S_{i,j}$, containing all examples form S whose label is either i or j.
- Set binary label in $S_{i,j}$ to be +1 if the multiclass label in S is i and -1 if the multiclass label in S is j.

- Train binary perceptron based on every $S_{i,j}$ to get voting for class i and j.
- Construct a multiclass classifier by predicting the class that had the highest votes.
- If predict wrong then updating α is needed. $S_{i,j}$ as the pth multiclassifer, if i is the true label then the pth non-positive elements in confidence vector predicts wrong, update $\alpha_{i,y} = \alpha_{i,y} + 1$; if j is the true label then the pth positive elements in confidence vector predicts wrong, update $\alpha_{i,\hat{y}} = \alpha_{i,\hat{y}} - 1$

A pseudocode of the OVO approach is given in the following.

Algorithm 3 One-vs-One Algorithm

```
input:
training set $S = \{(\mathbf{x_1}, y_1), ..., (\mathbf{x_m}, y_m)\}$,$\alpha$ be empty matrix, $tol \in R$, $\mathbf{V} = \overrightarrow{\mathbf{0}}_{(10,10)}$,E=0
foreach $i, j \in y$ s.t. $i < j$ let $S_{i,j}$ to be the $p$th classifier in total number $(10 * 9)/2 = 45$
while True do
  for t=1,...,m do
    1. If $y_t = i$ add 1 to $V_{i,j}$
    2. If $y_t = j$ add -1 to $V_{i,j}$
    3. $V_i = \sum_{i}^{10} V_{i,j}$ s.t $\hat{y} = \underset{i \in \{1,...,K\}}{argmax} V_i$, starts voting procedure
    4. If $\hat{y_t} * y_t < 0$:
       (a) $\alpha_{i,y_t} = \alpha_{i,y_t} + 1$, column belonging to correct label class +1;$E_t = E_t + 1$
       (b) $\alpha_{i,\hat{y}} = \alpha_{i,\hat{y}} - 1$ column belonging to false predicted class -1;$E_t = E_t + 1$
  end for
  if $|E_i - E_{i-1}| < tol$
end while
```

2.2.2 Testing Trained Perceptron

I can now use updated coefficient martix α to predict an unlabelled new instance $\mathbf{x_t}$ and recording testing error.

Table 3: Basic results for One-vs-One Algorithm in Perceptron

d	Training set error rate(%)	Test set error rate(%)
1	14.3211 ± 0.5788	13.8172 ± 1.0472
2	6.9797 ± 0.4245	8.0484 ± 0.7157
3	4.4165 ± 0.5087	6.7473 ± 0.6515
4	3.0869 ± 0.2195	5.5887 ± 0.5264
5	2.3407 ± 0.3937	5.6210 ± 0.6192
6	2.0933 ± 0.3671	5.1022 ± 0.5467
7	1.8412 ± 0.2261	5.1989 ± 0.5956

2.2.3 Adding Cross Validation

The results of O.v.O Algorithm after adding 5-fold cross validation.

- Time taken: 5:04:58.301049 d= 7 ....

- Mean d*: 6.6 with std(%): 66.33249580710799
- Mean test error(%): 5.123655913978494 with std(%): 0.5344259100202382

2.3 Discussion: O.V.A and O.V.O algorithm

The difference of OVA and OVO algorithm is that OVA focus on comparing one class and the rest classes while OVO classifies pairwise labels. Hence, this will causes difference in time complexity and accuracy.

In disussion of time complexity, it appears that OVA method performing faster than OVO algorithm based on my experiment.And in accuracy part which is usually measured by test errors, we observe that both the training and test set error rates seem to be higher in OVO method comparing the results with the one-vs-all polynomial kernel perceptron, even for higher values of d.However, accuracy in OvO is usually outperform other methods theoretically since it requires more time to compute and satisfies trade off between time complexity and accuracy. Hence, my experiment exists drawbacks in OVO method.

Although One-vs-All is popular, it is a heuristic that suffers from several problems. Firstly, the scale of the confidence values may differ between the binary classifiers. Second, even if the class distribution is balanced in the training set, the binary classification learners see unbalanced distributions because typically the set of negatives they see is much larger than the set of positives[2].

3 Another Two Algorithms

There exists various binary classification such like Support Vector Machine and Trees. In this section will use this two Python built-in library and adjust their parameters to see whether fits data well.

3.1 Support Vector Machine

A support-vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks like outliers detection.[1] Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin, the lower the generalization error of the classifier. In this section, I will use SVM(svc) built in *sklearn* library based on selected parameter: C, gamma, kernel choice. In the purpose of discussion, assign various value to one parameter while others to be fixed.

3.1.1 Kernel Choice

This is the choice of kernel for the SVM classifier. I will treat this as a hyperparameter as well and evaluate two possible values. Polynomial Kernel for 'poly' and Gaussian Kernel for 'rbf'.

- 'poly': $K_d(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_i})^d$
- 'rbf': $K(\mathbf{x_i}, \mathbf{x_j}) = e^{-\frac{1}{2\sigma^2}\|\mathbf{x_i}-\mathbf{x_j}\|^2}$

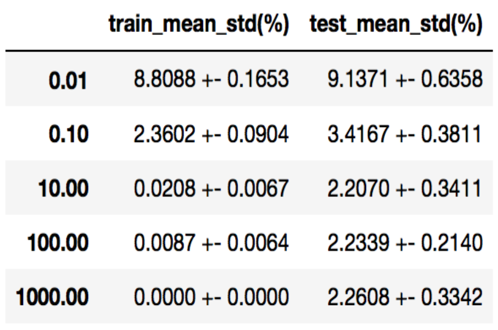
3.1.2 Parameter C

It controls the trade-off between $\|\mathbf{W}\|^2$ and training error $\sum_i^m V_{i,j}\xi_i$For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points.

- **Basic Results for Parameter C**

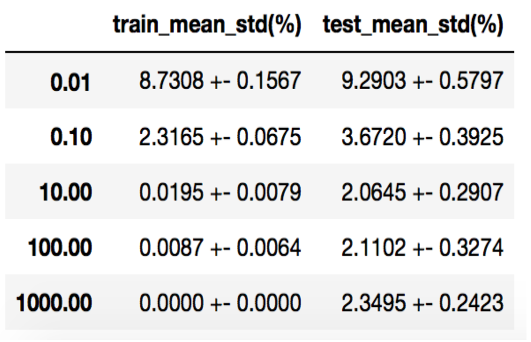
I choose c=[0.01, 0.1, 10, 100, 1000] and see the effects if only split data as 80% training and 20% testing. Besides, I observe that even for higher values of C,there is no significant increase in test error. That being said, for C= 10, I observe the lowest combination of test error rate and variance (i.e. for higher values of C, variance increases). The figure demonstrates Gaussian Kernel has less training and test error rate regardless the choice of C, hence, Gaussian Kernel fits dataset better than Polynomial Kernel but there is no much significance difference. And considering computational cost, Gaussian Kernel needs more time to compute.

Figure 3: basic results for SVM various kernel choice(%)



	train_mean_std(%)	test_mean_std(%)
0.01	8.8088 +- 0.1653	9.1371 +- 0.6358
0.10	2.3602 +- 0.0904	3.4167 +- 0.3811
10.00	0.0208 +- 0.0067	2.2070 +- 0.3411
100.00	0.0087 +- 0.0064	2.2339 +- 0.2140
1000.00	0.0000 +- 0.0000	2.2608 +- 0.3342

(a) basic results for SVM (Polynomial Kernel)



	train_mean_std(%)	test_mean_std(%)
0.01	8.7308 +- 0.1567	9.2903 +- 0.5797
0.10	2.3165 +- 0.0675	3.6720 +- 0.3925
10.00	0.0195 +- 0.0079	2.0645 +- 0.2907
100.00	0.0087 +- 0.0064	2.1102 +- 0.3274
1000.00	0.0000 +- 0.0000	2.3495 +- 0.2423

(b) basic results for SVM (Gaussian Kernel)

- **Adding Cross Validation for Parameter C**

In aspect of computational time, Gaussian needs more time which has been discussed previous in section 2.1.7. Results for C with Polynomial Kernel:

1. Time taken: 0:52:12.543600
2. Mean c*: 253.0 with std: 375.5276
3. Mean test error(%): 2.18817 with std(%): 0.278378

Results for C with Gaussian Kernel:

1. Time taken: 1:10:14.348494
2. Mean c*: 329.5 with std: 440.42565
3. Mean test error(%): 2.20161 with std(%): 0.28876

3.1.3 Parameter Gamma

Gamma is the kernel coefficient for 'rbf', the Gaussian Kernel. It's a hyperparmeter which we have to set before training model.The behavior of the model is very sensitive to the gamma parameter. If gamma is too large, the radius of the area of influence of the support vectors only includes the support vector itself and no amount of regularization with C will be able to prevent overfitting.

When gamma is very small, the model is too constrained and cannot capture the complexity or "shape" of the data. The region of influence of any selected support vector would include the whole training set. The resulting model will behave similarly to a linear model with a set of hyperplanes that separate the centers of high density of any pair of two classes.

Table 4: Basic results for Perceptron(Polynomial Kernel)

c	Training set error rate(%)	Test set error rate(%)
0.01	21.5858 ± 0.3486	22.0887 ± 0.8792
0.10	4.5335 ± 0.1084	5.094 ± 0.4055
10	0.0309 ± 0.0121	2.2258 ± 0.2437
100	0.0128 ± 0.0029	2.3522 ± 0.2767
1000	0.0000 ± 0.0000	2.2742 ± 0.2845

3.2 Random Forest

Random forest is an ensemble tool which takes a subset of observations and a subset of variables to build a decision trees. It builds multiple such decision tree and uses averaging to improve the predictive accuracy and control over-fitting.

3.2.1 Number of Trees: n_estimators

This is the number of trees you want to build before taking the maximum voting or averages of predictions. Higher number of trees give you better performance but computational time consuming. You should choose as high value as your processor can handle because this makes your predictions stronger and more stable. In Python library, its default value is 100.

- **Basic Results for n_estimators**

I assign this hyperparameter as $N = n\_estimators \in \{10, 100, 500, 1000\}$ and perform 20 runs. If only 10 trees are in the experiment, both training error and test error are larger than the other values. It can be seen that even for large number of trees training error rate tends to zero while there is no significant increase in test error rate. Besides, variance is also controlled in a small range which supports this ensemble method does prevent overfitting and improve accuracy.

Table 5: Basic results for N

N	Training set error rate(%)	Test set error rate(%)
10	0.1116 ± 0.0438	6.4167 ± 0.4464
100	0.0000 ± 0.0000	3.6989 ± 0.3512
500	0.0000 ± 0.0000	3.3118 ± 0.3576
1000	0.0000 ± 0.0000	3.5806 ± 0.4617

- **Adding Cross Validation for n_estimators**

During practical operation, I found that the more number of trees are applied the more time is need to computing. The mean value of best parameter N is 382 for this classification problem. It does make sense because when $N > 500$, the test error rate raises slightly maybe a little overfitting. Overall, the tree performance is well with 3% test error rate.

1. Time taken: 1:26:53.404987
2. Mean N*: 382.0 with std: 363.45013
3. Mean test error(%): 3.25806 with std(%): 0.28571

4 Comparing three Algorithms

This section will discuss above used three Algorithms: Kernel Perceptron, Support Vector Machine, Random Forest.

4.1 Time complexity

Table 6 summaries time complexity for 3 algorithms using Big O notation. And the parameter m is the number of feature vectors, n stands for the number of dimension, n_{SV} the number of support vectors, M the number trees in Random Forest and T to be the number of examples in the test set.

- **Perceptron** For the O.v.A. perceptron, the main cost during the trainingphase occurs when calculating the kernel matrix. Regardless of kernel choice, it requires multiplication between matrices which is the most expensive part in computation.
  - Take m to be the size of the parameter vector α and assume time taken to compute the inner product between examples is O(mn),thus in training section. Time complexity could be $O(m^2n)$
  - Assume T is the size of the training set and running time of the dual-form perceptron is O(T mn)
- **SVM** Bottou and Lin[5] show that the training examples in the SVM problem can be split into the below categories: examples which are not support vectors, examples which are support vectors(i.e.they appear in the margin as $y_i(\mathbf{w}^T\mathbf{x_i} + b) = \pm 1$). To analyse the complexity of SVMs, let's consider the following observations:
  - In practice, state-of-the-art SVM implementations typically have a training time complexity that scales between $O(m)$and $O(m^3)$.This can be further driven down to the computational cost of SVM is either $O(n^3)$ for large numbers of C or $O(n^2)$ for smaller values.
  - Prediction complexity of kernel SVM depends on the choice of kernel and is typically proportional to the number of support vectors. For most kernels, including polynomial and RBF, this is $O(n_{SV}n)$,where n_{SV}he number of support vectors.
- **Random Forest** It's an ensemble method, so time complexity going to be close to the sum of the complexities of building the individual decision trees in the model.If you have n instances and m attributes, the computational cost of building a tree is $O(mnlog(n))$.If you grow M trees, then the complexity is $O(Mmnlog(n))$. This is not an exact complexity, because the trees in the model are grown using a subset of the features, and additional time may be added in to handle the randomization processes.

Table 6: Comparison of different classification algorithms' time complexity

Algorithm	Training complexity(%)	Testing complexity(%)
Perceptron O.v.A. with polynomial kernel	$O(m^2n)$	$O(mTn)$
Perceptron O.v.A. with gaussian kernel	$O(m^2n)$	$O(mTn)$
SVM	$O(m^2) or O(m^3)$	$O(n_{SV}n)$
RandomForest	$O(Mmnlog(n))$	$O(n * M)$

Table 7: Empirical Computational Time

Algorithm	Time observed(h)
Perceptron O.v.A. with polynomial kernel	0:14:07.533529
Perceptron O.v.A. with Gaussian kernel	0:16:06.326975
Perceptron O.v.O. with polynomial kernel	1:11:39.246889
SVM with polynomial kernel	0:25:49.845641
SVM with Gaussian kernel	0:26:11.290748
RandomForest	0:18:13.635669

Table 8: Empirical Computational Time Adding CV

Algorithm	Time observed(h)
Perceptron O.v.A. with polynomial kernel	0:34:02.583907
Perceptron O.v.A. with Gaussian kernel	0:37:39.421615
Perceptron O.v.A. with polynomial kernel	5:04:58.301049
SVM with polynomial kernel	0:52:12.543600
SVM with Gaussian kernel	1:10:14.348494
RandomForest	1:26:53.404987

4.2 Accuracy

Table 9 summarises the mean test error rates observed in the different classification algorithms examined in this paper. These are the error rates as calculated,over a series of 20 runs, in the randomly allocated 20% test set, after using the cross-validation process defined in Question 2.
We can observe that SVM outperforms other methods with smallest test error rate and standard deviation. The reason why SVM did a good job is that in this case lies within its objectives.

- $\rho(S) := \frac{1}{\underset{w,b}{min}\{\|\mathbf{w}\| : y_i(\mathbf{w}^T\mathbf{x_i} + b) \geq 1\}}$

SVM has a quadratic objective due to the L2 Regulariser.This technique works very well to avoid over-fitting issue.And also sparsity of support vectors implies good generlisation performance. By applying a quadratic optimiser on SVM, the margin of the separating canonical hyperplane $\rho_s(\mathbf{w}, b) = \frac{1}{\|\mathbf{w}\|}$ is also maximized, ensuring that the sum of square distances between each pair of points from different sides of the hyperplane is maximum.

Table 9: Comparison of different classification algorithms' test error rates

Algorithm	Test set error rate(%)
Perceptron O.v.A. with polynomial kernel	2.98924 ± 0.41901
Perceptron O.v.A. with Gaussian kernel	6.94623 ± 0.48015
Perceptron O.v.O. with polynomial kernel	5.12365 ± 0.53442
SVM with polynomial kernel	2.18817 ± 0.27837
SVM with Gaussian kernel	2.201613 ± 0.28876
RandomForest	3.25806 ± 0.28571

Part 2:
Estimating Sample Complexity for Four Algorithms

5 Background

5.1 Understanding Generalization Error

In supervised learning applications in machine learning and statistical learning theory, generalization error[1] (also known as the out-of-sample error[3] or the risk) is a measure of how accurately an algorithm is able to predict outcome values for previously unseen data.
Let $h_s = ERM_{\mathcal{H}}(S), h_s$ minimize the empirical risk. We can decompose the risk of h_s as:

$$L_{\mathcal{D}}(h_s) = \epsilon_{app} + \epsilon_{est}$$

Follow the described question, it's obvious that the algorithms always predict with the first feature of $\mathbf{x}$, therefore ϵ_{app} will be zero. Hence we can narrow done the error to estimation error, where it can be presented as:

$$\epsilon_{est} = L_{\mathcal{D}}(h_s) - \min_{h \in \mathcal{H}} L_{\mathcal{D}}(H) = L_{\mathcal{D}}(h_s)$$

It is obvious that ϵ_{est} is only an estimate of $L_{\mathcal{D}}$. Besides, it decreases with the size of S and increases with the complexity of $\mathcal{H}$. Above expression also means that $L_{\mathcal{D}}(h_s)$ for this specific question only depends on the constraint of S_m and n.

5.2 Sample Complexity

The sample complexity of a machine learning algorithm represents the number of training-samples that it needs in order to successfully learn a target function. More precisely, the sample complexity is the number of training-samples that we need to supply to the algorithm, so that the function returned by the algorithm is within an arbitrarily small error, i.e generalization error smaller than 0.1, of the best possible function, with probability arbitrarily close to 1.
In our problem, $\mathcal{S}_m$ denote a set of m instances drawn uniformly at random from $\{-1, 1\}^n$ with first feature as their labels.Then let $\mathcal{A}_s(\mathbf{x})$ denote the prediction of an algorithm $\mathcal{A}$ trained from data sequence $\mathcal{S}$ on instance $\mathbf{x}$. As discussed above, we conclude that $L_{\mathcal{D}}(h_s)$ for this particular problem only depends on the constraint of S_m and n. Thus the generlisation error is then

$$\varepsilon(\mathcal{A}_s) := 2^{-n} \sum_{\mathbf{x} \in \{-1,1\}^n} \mathcal{I}[\mathcal{A}_s(\mathbf{x}) \neq x_1]$$

and thus the sample complexity on average at 10% generlisation error is

$$\mathcal{C}(\mathcal{A}) := \min\{m \in \{1, 2, ...\} : E[\varepsilon(\mathcal{A}_s)] \leq 0.1\}$$

6 Binary Search for Estimating Sample Complexity

Given a range of n we estimate sample complexity through binary search algorithm. The following code leads to how to estimate generalisation error which will used in sample complexity estimation.

Algorithm 4 Estimated Generalisation Error

```
  Input: Algorithm $\mathcal{A}$, $\mathcal{S}_m$
  Define T runs for current $\mathcal{S}_m$
  **Outer Loop**: For t in T runs
      1. Train Algorithm $\mathcal{A}$
      2. Test Algorithm $\mathcal{A}$ and store test errors $\mathcal{I}[\mathcal{A}_s(\mathbf{x}) \neq x_1]$
      3. Calculate test error rate $\epsilon_{test} = 2^{-n} * \mathcal{I}[\mathcal{A}_s(\mathbf{x}) \neq x_1]$
  **END**
  Output: taking average $\varepsilon_{est}(\mathcal{A}_s) := 2^{-n} \sum_{\mathbf{x} \in \{-1,1\}^n} \mathcal{I}[\mathcal{A}_s(\mathbf{x}) \neq x_1]$
```

The following pseudo-code describe process in detail for a single n.

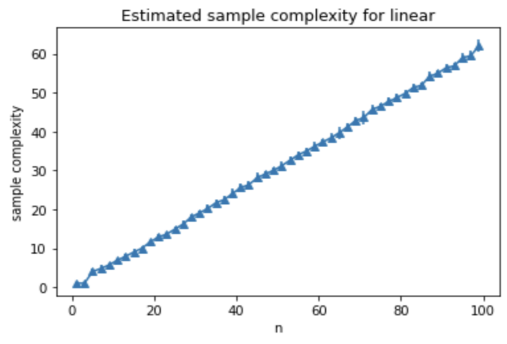
Algorithm 5 Binary Search Algorithm

```
Input: Algorithm $\mathcal{A}, m \in [m_l, m_r]$
Define 10 runs for current n in binary search
**Outer Loop**: For i in r runs
    1. **While True**:
        (a) $m = \frac{m_l+m_r}{2}$
        (b) Receive estimated generalization error $\epsilon_{est}(\mathcal{A}_S m)$ for n and m based on algorithm $\mathcal{A}$
        (c) **IF** $\varepsilon_{est}(\mathcal{A}_{Sm}) < \varepsilon(\mathcal{A}_{Sm}) = 0.1$:
            i. **IF** $m_r - m_l < 2$, break and return m
            ii. **ELSE** decrease the upper search bound, let $m_r = m$ and **continue**
        (d) **ELSE** $\varepsilon_{test}(\mathcal{A}_{Sm}) > \varepsilon(\mathcal{A}_{Sm}) = 0.1$:
            i. **IF** $m_r - m_l < 2$,break and return m
            ii. **ELSE** increase the lower search bound let $m_l = m$ and **continue**
    2. store 10 times estimated sample complexity $\mathcal{C}_i(\mathcal{A})$
**END LOOP**
Output: taking average $\mathcal{C}(\hat{\mathcal{A}}) = \frac{1}{r}\sum_{i}^{r} \mathcal{C}_i(\mathcal{A})$
```

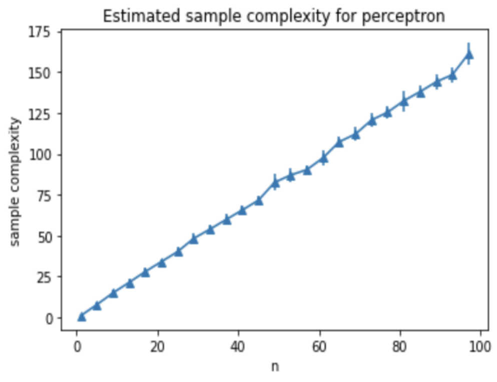
6.1 Results for 4 Algorithm

Repeat above procedure n_{max} times in range $m \in [m_l, m_r]$ based on algorithm $\mathcal{A}$.Finally can get each n with sample complexity $\mathcal{C}(\hat{\mathcal{A}})$. And from computational with code,I got results for 4 algorithms.

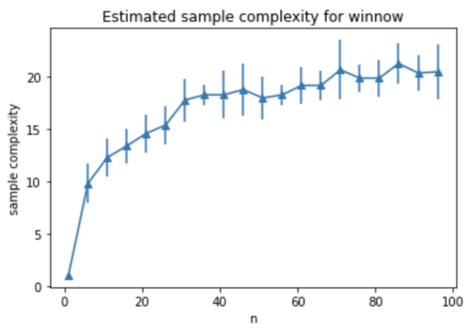
Figure 4: Estimated Sample Complexity for 4 algorithms



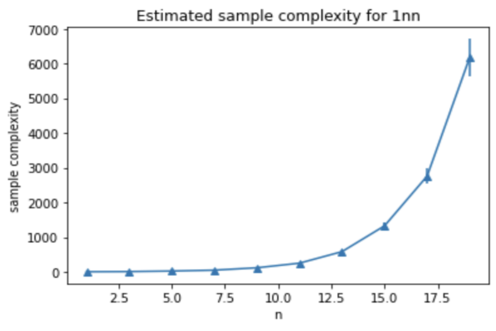
(a) Estimate Sample Complexity for least square



(b) Estimate Sample Complexity for perceptron



(a) Estimate Sample Complexity for winnow



(b) Estimate Sample Complexity for 1 NN

6.2 Trade off and Bias

6.2.1 Trade off

I refer this trade off is between computational time and accuracy. Total computational time cost estimating sample complexity up to n_{max}, with r runs for each n can be decomposed to: binary search complexity $O(log(M))$, training algorithm complexity $f_{train}(m,n)$, testing algorithm $f_{test}(m,n)$.
To start with total computational cost is:

$$C(n_{max}) = \sum_{n}^{n_{max}} B(n) \times r \tag{2}$$

where B(n) is the computational cost of running binary search with averaged test errors across T random sampling trials. Let the computational time of the chosen training algorithm to be $f_{train}(m,n)$,test algorithm to be $f_{test}(m,n)$ For a fix number of m, we train and test the algorithm for T times to find the average test error.So the cost to estimate the generalization error for fixed m with T trials

$$(f_{test}(m,n) + f_{train}(m,n)) * T \tag{3}$$

Let range of m considered in Binary search to be $R = m_r - m_l$ which depend on n and classifier, let R=O(M(n)) where M(n) is function between m and n, can choose linear, exponential etc..In each binary search it contain the own time complexity O(log(M)).Combining all the information above, I got:

$$\begin{align} B(n) &\sim O(log(R) * (f_{train}(R,n) + f_{test}(n)) \tag{4}\\ &= O(log(M(n)) * (f_{train}(M(n),n) + f_{test}(n)) \tag{5}\end{align}$$

Overall,the total computational time of estimating sample complexity for $n \in [1, n_{max}]$:

$$\begin{align} C(n_{max}) &= \sum_{n}^{n_{max}} B(n) \times r \tag{6}\\ &= O(n_{max} B(n_{max})) \tag{7}\end{align}$$

Least square time complexity
Note T stands for test size.

$$\begin{aligned} M(n) &\sim O(n)\\ f_{train}(M(n),n) &\sim O(n^3 + n^2 m) \sim O(n^3)\\ f_{test}(n) &\sim O(nT)\\ B(n) &\sim O(log(n) \times (O(nT) + O(n^3)))\\ C(n_{max}) &\sim O(n_{max} B(n_{max})) \sim O((n_{max}^4 + n_{max}^2 T) log(n_{max})) \end{aligned}$$

Perceptron time complexity

$$\begin{aligned}
M(n) &\sim O(n) \\
f_{train}(M(n), n) &\sim O(mn) \sim O(n^2) \\
f_{test}(n) &\sim O(nT) \\
B(n) &\sim O(log(n) \times (O(n^2) + O(nT))) \\
C(n_{max}) &\sim O(n_{max} B(n_{max})) \sim O((n_{max}^3 + n_{max}^2 T) log(n_{max}))
\end{aligned}$$

Winnow time complexity
Similar to Perceptron.

$$\begin{aligned}
M(n) &\sim O(log(n)) \\
f_{train}(M(n), n) &\sim O(mn) \sim O(log(n)n) \\
f_{test}(n) &\sim O(nT) \\
B(n) &\sim O(log(n) \times (O(log(n)n) + O(nT))) \\
C(n_{max}) &\sim O(n_{max} B(n_{max})) \sim O((n_{max}^2 log(n_{max}) + n_{max}^2 T) log(log(n_{max})))
\end{aligned}$$

In practical computing, I found Winnow runs faster than perceptron may be the reason that function M(n) is an logarithm which faste than linear case.
1NN time complexity

$$\begin{aligned}
M(n) &\sim O(n^3) \\
f(m, n) &\sim O(mn \times T) \sim O(n^4 T) \\
B(n) &\sim O(log(n^3) \times (O(n^2) + O(nT))) \\
C(n_{max}) &\sim O(n_{max} B(n_{max})) \sim O((n_{max}^5 + T log(n_{max}^3))
\end{aligned}$$

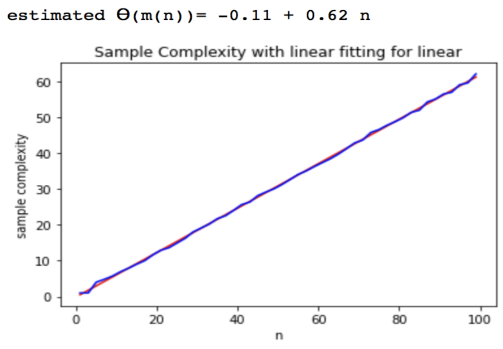
6.2.2 Bias

The bias term may come from we treat $\epsilon_{app} = 0$. And this has been discussed in background section.

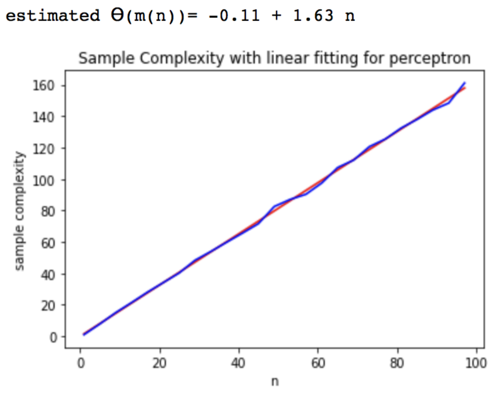
7 Estimate m(n) Function

This section estimates how m grows as a function of n as $n \to \infty$.

Figure 5: Estimated m(n) for 4 algorithms

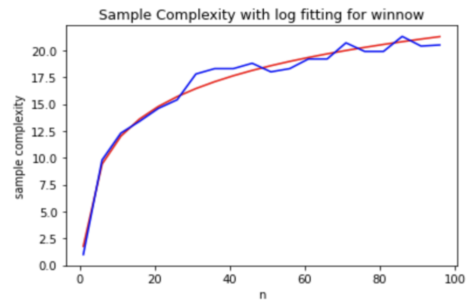


(a) least square:$\Theta(m(n)) = -0.11 + 0.62n$



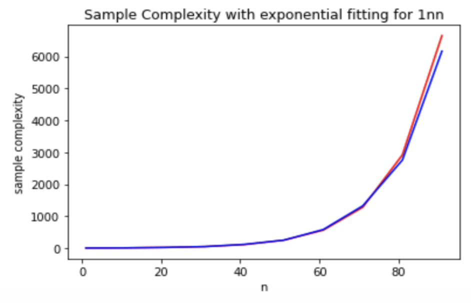
(b) perceptron:$\Theta(m(n)) = -0.11 + 1.63n$

estimated Θ(m(n))= 1.76 + 4.28 log(n)



(a) winnow: $\Theta(m(n)) = 1.76 + 4.28log(n)$

estimated Θ(m(n))= 3.69 exp(0.08 n)



(b) 1 NN:$\Theta(m(n)) = 3.69e^{0.08n}$

7.1 Discuss of Observations

Linear Regression vs Perceptron
Given training data S,the primal form of linear regression is derived by mini- mizing the empirical risk $\epsilon(\mathcal{S}, w)$ of the entire training set.As the first agent, x_0 is always consistent with the label, the primal form of linear regres- sion directly computes the optimal weight, where all the entires are close to zero except for the first one. The same weight will generalize well on the test data, because the test data is also sampled with its first agent consistent to label.

8 Upper Bound Probability for Perceptron

From mistake bound theory brought by Novikoff and Block(1962), suppose we have a binary classification dataset with n dimensional inputs.If the data is separable,then the Perceptron algorithm will find a separating hyperplane after making a finite number of mistakes. In more mathematical way, it can be presented as:

- Let $\{(\mathbf{x_1}, y_1), ..., \mathbf{x_m}, y_m)\} \in (\mathcal{R}^n, \{-1, +1\})^m$ be a sequence of training examples such that for all i,the feature vector $\mathbf{x_i} \in \Re^n, \|\mathbf{x_i}\| \leq R$ and the label $y \in \{-1, +1\}$. That is all training examples are contained in a ball of size R.
- The training data is separable by margin γ using a unit vector $\mathbf{u}$ (i.e. $\|\mathbf{u}\| = 1$),we have $\mathbf{y_i}(\mathbf{u}^T\mathbf{x_i}) \geq \gamma$
- Then, the perceptron algorithm will make at most $(\frac{R}{\gamma})^2$ mistakes on the training sequence.

To prove this upper bound $(\frac{R}{\gamma})^2$, we claim:

1. After t mistakes $\mathbf{u}^T\mathbf{w_t} \geq t\gamma$.

$$\mathbf{u}^T\mathbf{w_{t+1}} = \mathbf{u}^T\mathbf{w_t} + y_i(u)^T\mathbf{x_i} \geq \mathbf{u}^T\mathbf{w_t} + \gamma$$

because the data is separable by a margin γ.Besides,due to initialization $\mathbf{w_0} = 0$(i.e $\mathbf{u^T w_0} = 0$),straightforward induction gives us $\mathbf{u}^T\mathbf{w_t} \geq t\gamma$

2. After t mistakes, $\|\mathbf{w_t}\|^2 \leq tR^2$.

$$\begin{aligned}\|\mathbf{w_{t+1}}\|^2 &= \|\mathbf{w_t} + y_i\mathbf{x_i}\|^2 \\ &= \|\mathbf{w_t}\|^2 + 2y_i(\mathbf{w_t}^T\mathbf{x_i}) + \|\mathbf{w_i}\|^2 \\ &\text{updating } \mathbf{w} \text{ when mistake occurs,that's } {}_i\mathbf{w_t}^T\mathbf{x_i} < 0 \\ &\|\mathbf{w_i}\|^2 \leq R\text{,the defination of R} \\ &\leq \|\mathbf{w_t}\|^2 + R^2\end{aligned}$$

because the data is separable by a margin γ.Besides,due to initialization $\mathbf{w_0} = 0$(i.e $\mathbf{u^T w_0} = 0$),straightforward induction gives us $\|\mathbf{w_t}\|^2 \leq tR^2$

3. From claim (2), simple transformation made

$$R\sqrt{t} \geq \|\mathbf{w_t}\| \geq \mathbf{u}^T\mathbf{w_t} \geq t\gamma$$

   (a) the second inequality holds for $\mathbf{u}^T\mathbf{w_t} = \|\mathbf{w_t}\|\|\mathbf{u}\|cos <$ angles between them $>$. But $\|\mathbf{u}\| = 1$ and cosine is less than 1. So $\mathbf{u}^T\mathbf{w_t} \leq \|\mathbf{w_t}\|$(Cauchy-Schwarz inequality)

   (b) third inequality comes form claim (1)

4. over all gives us Number of mistakes $M \leq \frac{R^2}{\gamma^2} = B$

Let S_m be a set of m training examples i.i.d from D.Let t be drawn uniformly from {1,...,m} where $P(t = t') = \frac{1}{m}$. So the lower bound or supremum probability of hitting a mistake p_t at any th trail is claimed to be $\frac{B}{m}$. That is:

$$P(\mathcal{A}_S(X) \neq y) = P(\frac{B}{m}) \leq \frac{R^2}{m\gamma^2} = \frac{n}{m\gamma^2}.$$

In part 2 we know that the maximum norm of x from D is $\sqrt{n}$, giving $R = \max(\|x\|) = \sqrt{n}$

References

[1] *"1.4. Support Vector Machines — scikit-learn 0.20.2 documentation.* Nov. 2017. URL: `https://scikit-learn.org/stable/modules/svm.html`.

[2] Christopher M Bishop. *Pattern recognition and machine learning.* springer, 2006.

[3] Olivier Bousquet, Stéphane Boucheron, and Gábor Lugosi. “Introduction to statistical learning theory”. In: *Summer School on Machine Learning.* Springer. 2003, pp. 169–207.

[4] Yoonsuh Jung. “Multiple predicting K-fold cross-validation for model selection”. In: *Journal of Nonparametric Statistics* 30.1 (2018), pp. 197–215. URL: `https://www.tandfonline.com/doi/full/10.1080/10485252.2017.1404598`.

[5] Chi-Jen Lin. *Large-scale kernel machines.* MIT press, 2007.