Statistical Methods in AI

Distance Based and Linear Classifiers

**- Shrenik Lad,
200901097**

INTRODUCTION :

The aim of the project was to understand different types of classification algorithms by implementing some classifies, **distance based** and **linear classifiers,** and testing them on various datasets.

The datasets differ a lot in their nature. Some are **properly aligned** and the classes are separable whereas some datasets are **highly skewed**. It is very important to select appropriate classification algorithms for such datasets. Some classifiers work very well with aligned datasets but may fail with skewed ones and vice versa.

DATASETS USED

The following datasets were used for testing the above classifiers :-

A) ***Iris Dataset***

The dataset contains 3 classes of 50 instances each, where each class refers to a type of iris plant. 1 class is linearly separable from the other two while the latter are not linearly separable from each other.

Attributes :-

- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm

Classes :-

- Iris setosa
- Iris Versicolor
- Iris Virginica

B) ***Wine Dataset***

The dataset is about 3 types of wines grown in Italy. It determines the quantities of 13 constituents found in each of the three types of wines.

Number of Attributes :- 13
Number of Classes :- 3
Number of Instances :-

- class 1 - 59
- class 2 - 71
- class 3 – 48

C) ***Yeast Dataset***

The description of Yeast dataset is as follows :

Number of Instances :- 1484
Number of Attributes :- 8
Number of Classes :- 10

The dataset has very highly skewed distribution of samples among the classes.

D) ***P-R Dataset***

The dataset has samples for letter recognition. We are interested only in the samples having letters 'P' and 'R'. The classifier should distinguish between 'P' and 'R' samples. The images are black and white rectangular displays with letters written in 20 different fonts.

Number of instances :- 803 samples of 'P' and 758 sanples of 'R'.
Number of attributes :- 16

Attribute Information:

1.	lettr	capital letter (26 values from A to Z)	
2.	x-box	horizontal position of box	(integer)
3.	y-box	vertical position of box	(integer)
4.	width	width of box	(integer)
5.	high	height of box	(integer)
6.	onpix	total # on pixels	(integer)
7.	x-bar	mean x of on pixels in box	(integer)
8.	y-bar	mean y of on pixels in box	(integer)
9.	x2bar	mean x variance	(integer)
10.	y2bar	mean y variance	(integer)
11.	xybar	mean x y correlation	(integer)
12.	x2ybr	mean of x * x * y	(integer)
13.	xy2br	mean of x * y * y	(integer)

14.	x-ege	mean edge count left to right	(integer)
15.	xegvy	correlation of x-ege with y	(integer)
16.	y-ege	mean edge count bottom to top	(integer)
17.	yegvx	correlation of y-ege with x	(integer)

CLASSIFIERS

The following classifiers were implemented and tested on several datasets.

A) ***Distance Based Classifiers:***

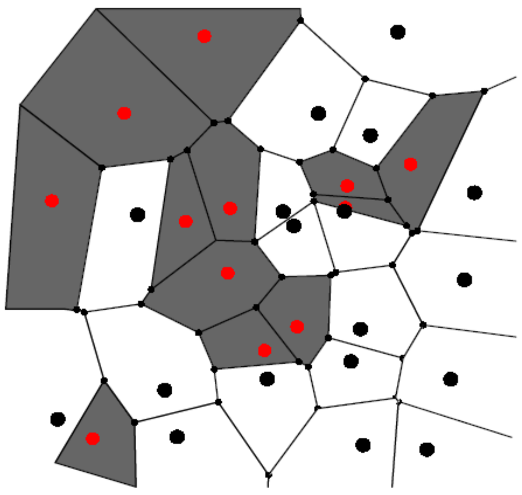
In such classification Algortithms, the distance of the test sample with the known (**training) samples** is considered to determine the class it belongs. Different distance metrics are available such as **Euclidean, Manhattan, and Minkowski** distances, and they may give different outputs.

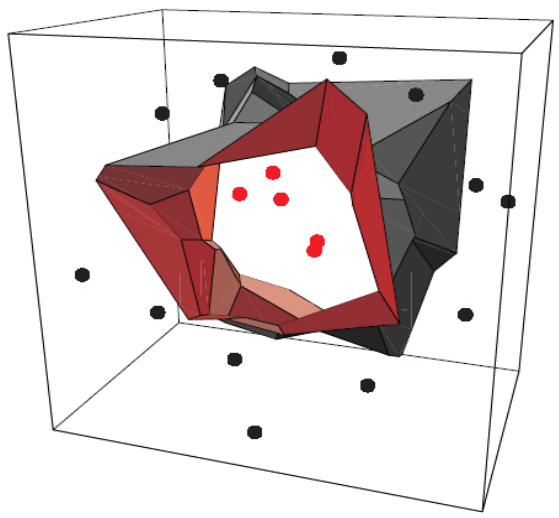
The 4 distance based classifiers are as follows :

1. Single Nearest Neighbour

In this classifier, the test sample is assigned the class of its **nearest neighbour**.

Let {x1 ,x2, x3 . . . xn} denote training samples in 'd' dimensional space, whose labels are known to us. For any test sample x, we find the distance of x with each of the training samples. If xi is nearest to x, x is assigned the label associated with xi. The feature space is partitioned into cells consisting of all points closer to a given training point xi than to any other training points. This is called **Voronoi tesselation** of the space.





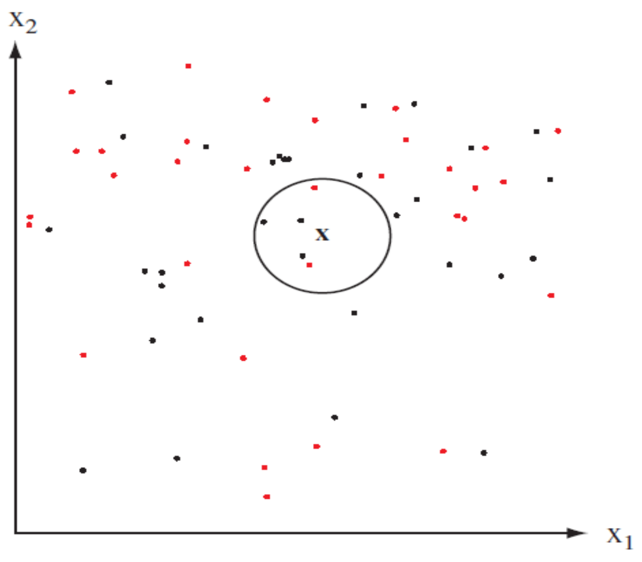
The nearest-neighbor rule is a sub-optimal procedure. It is **conceptually and computationally simple**.

2. K-Nearest Neighbour

In this classfiier, the K nearest neighbours of test sample x are searched. X is given the label which is most frequently represented among the K nearest samples.

Let {x1 ,x2, x3 . . . xn} denote training samples in 'd' dimensional space, whose labels are known to us. For any test sample x, we find the distance of x with each of the training samples. We find the K nearest training samples of x among them. If these samples are y1, y2, ... , yk, we find the class label which is most dominant among these K neighbours and assign the same label to test sample x.

Note that when **K=1**, this classifier becomes the single nearest neighbour classifier.



K=5 example

3. Weighted K-Nearest Neighbour

This is similar to K-nearest neighbour, except that weights are assigned to each class among the K nearest ones. Generally, weight is taken as inverse of **distance, i.e 1/d**. (This weight was equal to one in case of simple K-NN).

There are other variations of weighted K-NN as well.

4. Nearest Mean

In nearest mean classifier, distance of test sample from the mean of each class is found. The sample is assigned the class of its nearest mean.

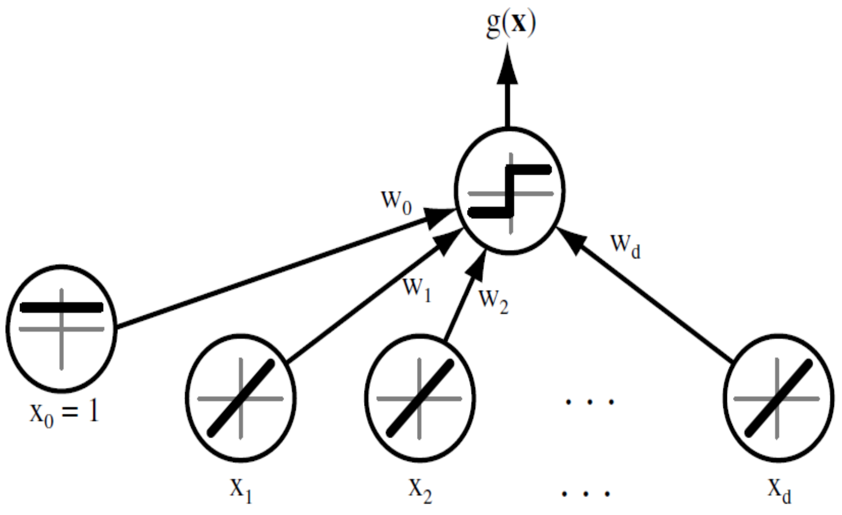
Let $\mu_1, \mu_2, \ldots, \mu_k$ denote the means of K classes respectively. We find the distance of x with each of these means. X is assigned the label of the mean which is nearest to it.

$\|\mathbf{x} - \boldsymbol{\mu}_i\|$ is the **Euclidean distance** between x and mean of class i.

B) ***Linear Classifiers***

Linear Classifiers classifies an object based on the value of a **linear combination** of the characteristics. For eg, in a 2 class, 2 dimensional problem, the linear classifier is a line separating the two classes. In higher dimensions, the linear classifier is a **hyperplane**.

$$g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0,$$



Linear classifiers can learn using the following methods :

1. <u>Perceptron Learning</u>

Perceptron learning uses basic **Gradient Descent approach** to find the solution vector. A criterion function J(a) is defined that is minimized if **a** is a solution vector. We start with some arbitarily chosen vector **a(1)** and compute the gradient vector at **J(a(1)**. The next value is obtained by moving some distance from a(1) along the negative of the gradient. In general, **a(k+1)** is obtained from **a(k)** by the equation

$$\mathbf{a}(k+1) = \mathbf{a}(k) - \eta(k)\nabla J(\mathbf{a}(k)),$$

where η is the learning rate that sets the step size. We hope that such a sequnce of weight vectors will converge to a solution minimizing J(a).

There are various variations of Perceptron Learning.

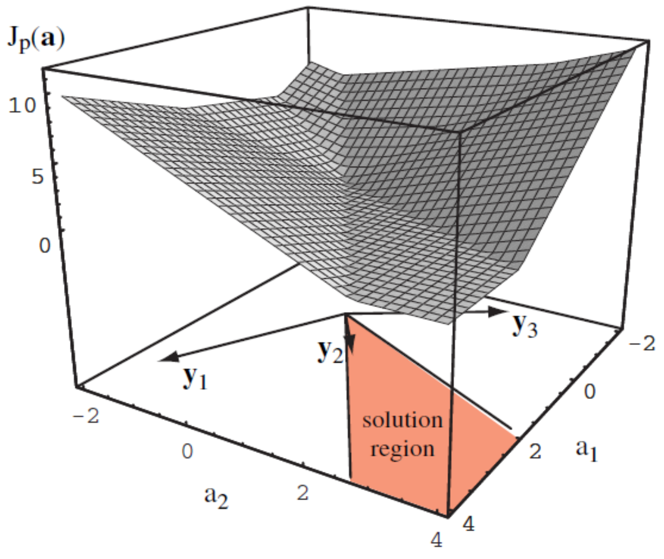
Batch Learning with fixed increment

Here the Perceptron Criterion function is

$$J_p(\mathbf{a}) = \sum_{\mathbf{y}\in\mathcal{Y}} (-\mathbf{a}^t\mathbf{y}),$$

where Y(a) is the set of samples misclassified by a. J(a) is never negative, it is zero when no samples are misclassified. The Gradient of J(a) is given by,

$$\nabla J_p = \sum_{\mathbf{y}\in\mathcal{Y}} (-\mathbf{y}),$$



J(a) when plotted for two dimensional solution vector. J(a) is **peicewise linear**.

So the update rule for batch perceptron becomes,

$$\mathbf{a(k+1) = a(k) + \eta\ (k)*Sum(y_i)}$$

where each y_i is misclassified by a(k).

The Algorithm **converges** when none of the samples are misclassified.

➢ Single sample Fixed increment

It is similar to **Batch Perceptron**, except that in the update rule, instead of taking sum of misclassified smaples, we take one of the misclassified samples randomly.

So, the update rule becomes,

$$\mathbf{a(k+1) = a(k) +} \; \eta \; \mathbf{(k) * y_k}$$

where y_k is one of the misclassified samples, chosen **randomly**.

The terminating condition is the same, when none of the samples are misclassified.

➢ Single sample Variable increment with margin

This is a variation of fixed increment rule algortithm. We introduce a **variable** $\eta(k)$ **increment** and a **margin b**, and it corrects the solution vector whenever $a^t(k).y^k$ fails to exceed the margin *b*.

The update rule in this case is given by,

$$\mathbf{a(k+1) = a(k) +}\eta(k)\mathbf{* y^k}.$$

where $a^t(k)*y(k) <= b$ for all k.

If the samples are linearly separable and :

$$\eta(k) \geq 0,$$

$$\lim_{m \to \infty} \sum_{k=1}^{m} \eta(k) = \infty \quad \text{and}$$

$$\lim_{m \to \infty} \frac{\sum\limits_{k=1}^{m} \eta^2(k)}{\left(\sum\limits_{k=1}^{m} \eta(k)\right)^2} = 0,$$

then a(k) converges to a solution vector satisfying $a^t.y_i > b$ for all i.

The common choices for $\eta(k)$ are 1/k and k, where k is the iteration number.

2. Minimum Squared Error classifier using Pseudoinverse method

Let Y be the nxd matrix, whose ith row is the vector y_i^t.

Let **b** be the margin vector b = (b1 , b2 , ... , $b_n)^t$.

We need to find a vector **a** satisfying

$$\begin{pmatrix} Y_{10} & Y_{11} & \cdots & Y_{1d} \\ Y_{20} & Y_{21} & \cdots & Y_{2d} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ Y_{n0} & Y_{n1} & \cdots & Y_{nd} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_d \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ \vdots \\ \vdots \\ b_n \end{pmatrix}$$

Y a = b

Since Y is generally rectangular, no exact solution exists to the above equation.

We define **error vector** by

e = Ya – b

We try to minimize the squared length of the error vector.

So, $$J_s(\mathbf{a}) = \|\mathbf{Ya} - \mathbf{b}\|^2 = \sum_{i=1}^{n} (\mathbf{a}^t\mathbf{y}_i - b_i)^2.$$

and, $$\nabla J_s = \sum_{i=1}^{n} 2(\mathbf{a}^t\mathbf{y}_i - b_i)\mathbf{y}_i = 2\mathbf{Y}^t(\mathbf{Ya} - \mathbf{b})$$

solving this, we get

$$\begin{aligned} \mathbf{a} &= (\mathbf{Y}^t\mathbf{Y})^{-1}\mathbf{Y}^t\mathbf{b} \\ &= \mathbf{Y}^\dagger\mathbf{b}, \end{aligned}$$

where $$\mathbf{Y}^\dagger \equiv (\mathbf{Y}^t\mathbf{Y})^{-1}\mathbf{Y}^t$$

is called the ***pseudoinverse*** of **Y**

It can be shown that a MSE solution always exists for Ya = b. The choice of margin vector should be carefully done.

3. LMS procedure for minimum squared error learning

We use the gradient descent technique to minimise the function

$$J(a) = || Ya - b ||^2$$

Gradient of above function becomes $\nabla J_s = 2\mathbf{Y}^t(\mathbf{Ya} - \mathbf{b})$

So, the update rule becomes,

$\mathbf{a(k+1) = a(k)} + \eta(k) \quad \mathbf{(b_k - a(k)^t . y^k)}$**,**

where y^k is any one of the **training samples.**

This method always yields a solution whether or not Y^tY is singular or not.

But the solution may not be the separating hyperplane for the classes.

Distance Based Classifiers :

1) Single Nearest Neighbour

Implementation:

- The dataset is randomly shuffled each time and depending on the ratio of training to testing size, we create training and testing samples.
- For a test sample x, we find distance of x with each of the training samples. The distance can be euclidean, manhattan or minkowski.
- Out of all the samples, we find the label of the one which has the minimum distance from x.
- x is assigned the label of that sample.
- We compare the output with the actual label of x to find the accuracy.

Results : (The 3 rows correspond to Euclidean , Manhattan and Minkowski distance resp)

IRIS

min accuracy	max accuracy	avg accuracy	Std Deviation
90.6667	98.6667	95.6	1.8319
89.3333	98.6667	94.8	2.0735
90.6667	98.6667	95.733	1.827

WINE

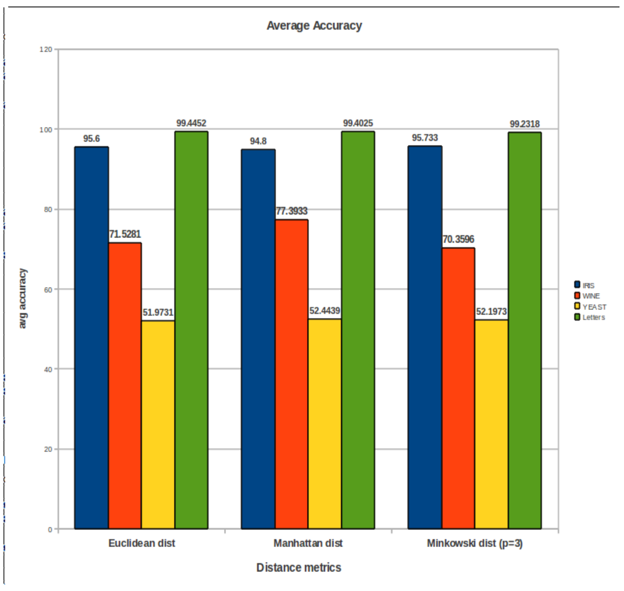
min accuracy	max accuracy	avg accuracy	Std Deviation
60.6742	77.5281	71.5281	3.9878
66.2921	83.1461	77.3933	3.6858
59.5506	78.6517	70.3596	4.0473

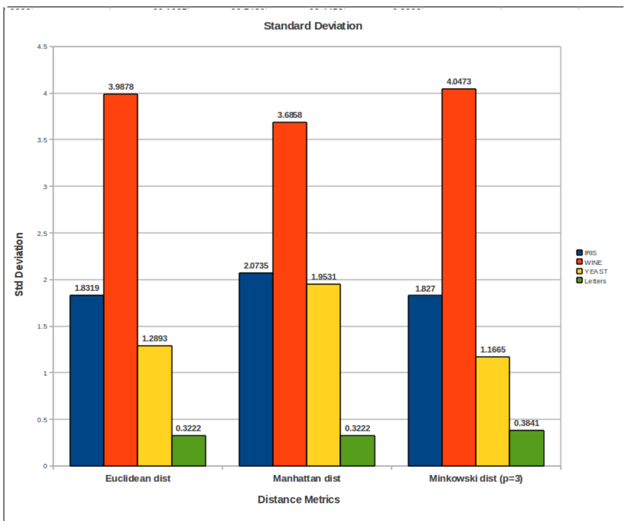
YEAST

min accuracy	max accuracy	avg accuracy	Std Deviation
49.3274	54.2601	51.9731	1.2893
49.1031	55.3812	52.4439	1.9531
51.1211	54.7085	52.1973	1.1665

Letters

min accuracy	max accuracy	avg accuracy	Std Deviation
99.1037	99.7439	99.4452	0.3222
99.1037	99.7439	99.4025	0.3222
98.8476	99.6159	99.2318	0.3841





K Nearest Neighbour

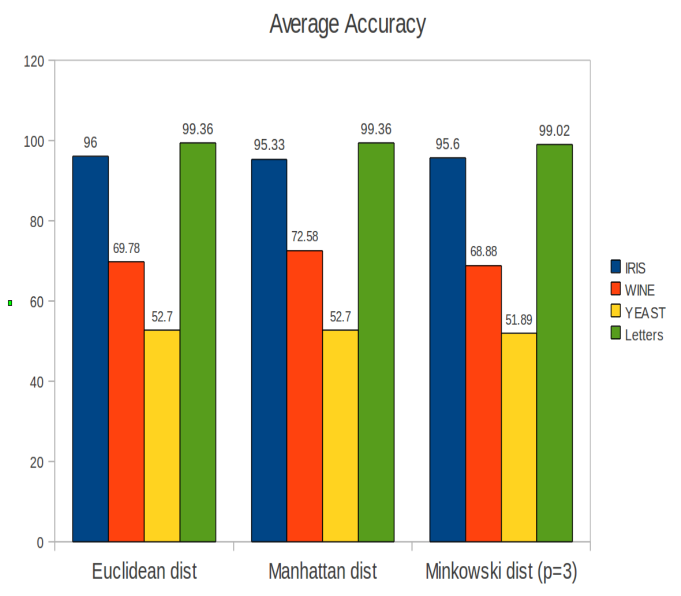
Implementation:

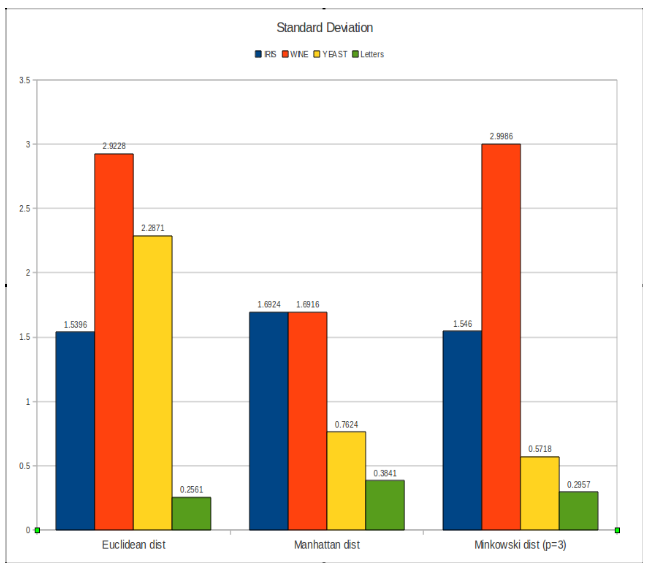
- The implementation is very similar to the single nearest-neighbor, but instead of finding the training sample which has minimum distance from x, we find K- samples which have minimum distance from x.
- We find the class label which has highest count among these K samples.
- X is assigned the label which has maximum count.
- In this method two or more classes can have the same number of count, out of which we choose any one randomly.

Results: (K=3)

IRIS				WINE			
min accuracy	max accuracy	avg accuracy	Std Deviation	min accuracy	max accuracy	avg accuracy	Std Deviation
93.3333	98.6667	96	1.5396	62.9213	73.0337	69.7753	2.9228
93.3333	98.6667	95.3333	1.6924	69.6629	75.2809	72.5843	1.6916
93.3333	97.3333	95.6	1.546	61.7978	71.9101	68.8764	2.9986

YEAST				Letters			
min accuracy	max accuracy	avg accuracy	Std Deviation	min accuracy	max accuracy	avg accuracy	Std Deviation
51.0782	54.3127	52.6954	2.2871	99.1037	99.6159	99.3598	0.2561
52.1563	53.2345	52.6954	0.7624	98.9757	99.7439	99.3598	0.3841
51.4825	52.2911	51.8868	0.5718	98.8476	99.3598	99.0184	0.2957

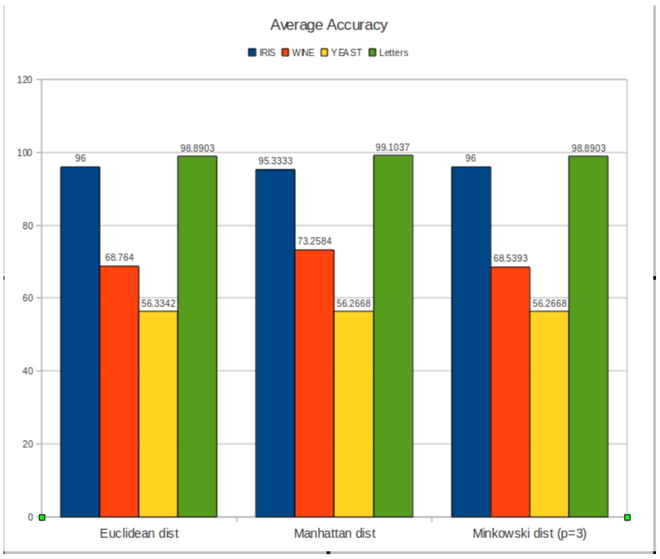


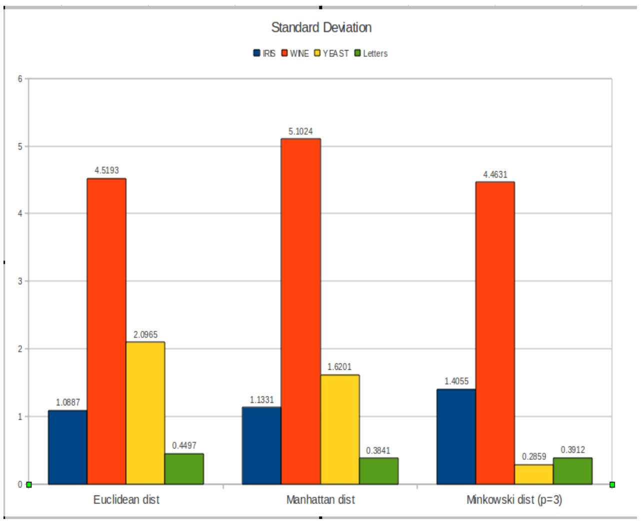


Results (K=7):

IRIS				WINE			
min accuracy	max accuracy	avg accuracy	Std Deviation	min accuracy	max accuracy	avg accuracy	Std Deviation
94.6667	97.3333	96	1.0887	60.6742	76.4045	68.764	4.5193
93.3333	97.3333	95.3333	1.1331	62.9213	78.6517	73.2584	5.1024
94.6667	98.6667	96	1.4055	60.6742	75.2809	68.5393	4.4631

WINE				YEAST			
min accuracy	max accuracy	avg accuracy	Std Deviation	min accuracy	max accuracy	avg accuracy	Std Deviation
60.6742	76.4045	68.764	4.5193	54.8518	57.8167	56.3342	2.0965
62.9213	78.6517	73.2584	5.1024	55.1213	57.4124	56.2668	1.6201
60.6742	75.2809	68.5393	4.4631	56.0647	56.469	56.2668	0.2859





Nearest Mean Classifer

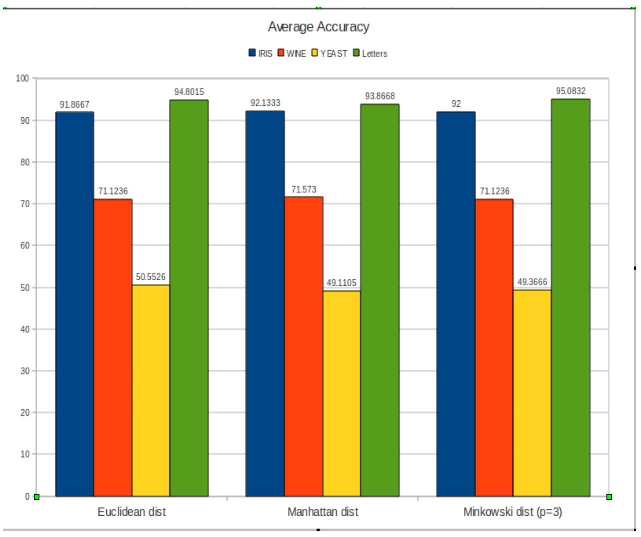
Implementation :

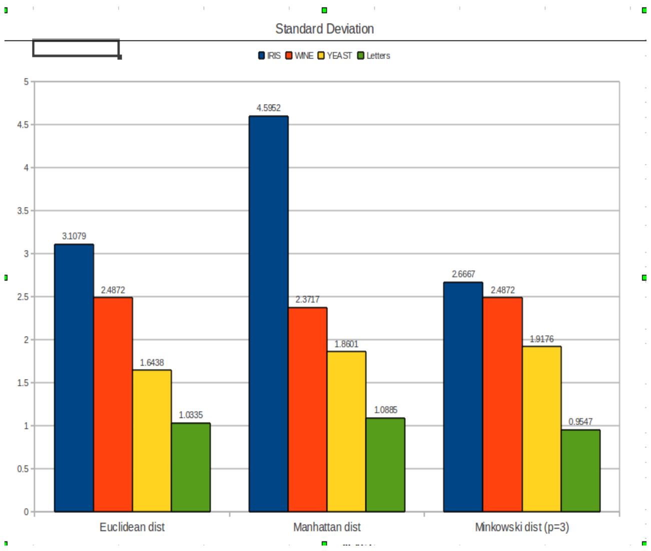
- Once the training and testing data is ready, we find the number of classes
- If there are **C** classes, we find the means of all the C classes from the training set.
- For a test sample x, we find the distance of x from each of these means.
- X is assigned the class of the closest mean.
- We compare the output with actual class for finding accuracy.

Results:

IRIS				WINE			
min accuracy	max accuracy	avg accuracy	Std Deviation	min accuracy	max accuracy	avg accuracy	Std Deviation
86.6667	96	91.8667	3.1079	66.2921	75.2809	71.1236	2.4872
82.6667	97.3333	92.1333	4.5952	66.2921	75.2809	71.573	2.3717
88	96	92	2.6667	66.2921	75.2809	71.1236	2.4872

YEAST				Letters			
min accuracy	max accuracy	avg accuracy	Std Deviation	min accuracy	max accuracy	avg accuracy	Std Deviation
48.6523	53.0997	50.5526	1.6438	93.0858	96.1588	94.8015	1.0335
46.2264	52.0216	49.1105	1.8601	91.9334	95.2625	93.8668	1.0885
46.2264	51.8868	49.3666	1.9176	93.598	96.5429	95.0832	0.9547





Weighted K – NN (K=3)

Implementation:

- ➢ This is similar to K- nearest-neighbor. First the K training samples are found which have minimum distance from test sample x.
- ➢ For each of the samples among the K nearest, we see the label of that class and increase that class's value by a weight equal to 1/d, where d is the distance between that sample and x.
- ➢ At the end, we see which class has the maximum value
- ➢ x is assigned this class which has max value.

Results (K=3):

IRIS

min accuracy	max accuracy	avg accuracy	Std Deviation
92	100	96.4	2.356
92	98.6667	95.733	1.9676
93.333	100	96.6667	2.0123

WINE

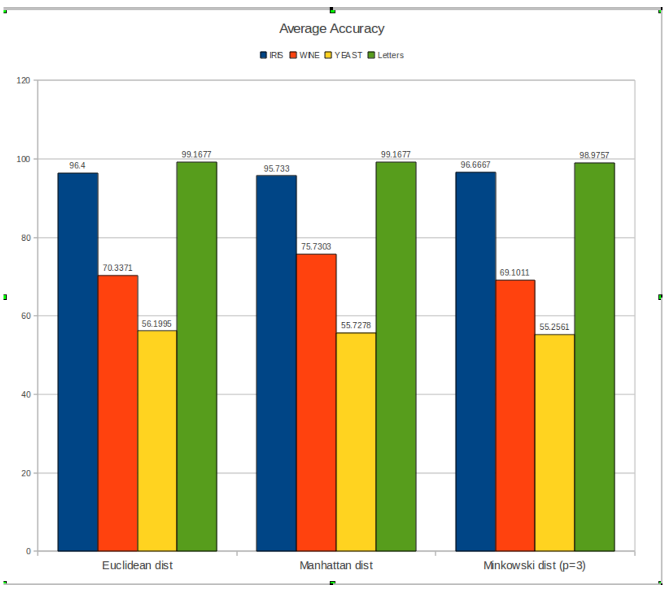
min accuracy	max accuracy	avg accuracy	Std Deviation
60.6742	75.2809	70.3371	4.0059
64.0449	79.7753	75.7303	5.0859
59.5506	73.0337	69.1011	3.9012

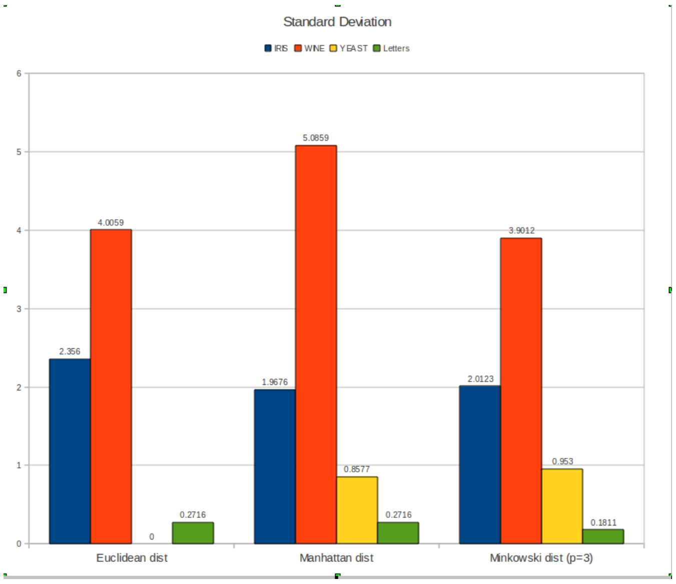
YEAST

min accuracy	max accuracy	avg accuracy	Std Deviation
56.1995	56.1995	56.1995	0
55.1213	56.3342	55.7278	0.8577
54.5822	55.9299	55.2561	0.953

Letters

min accuracy	max accuracy	avg accuracy	Std Deviation
98.9757	99.3598	99.1677	0.2716
98.9757	99.3598	99.1677	0.2716
98.8476	99.1037	98.9757	0.1811





Results (K=7) :

IRIS

min accuracy	max accuracy	avg accuracy	Std Deviation
92	100	96.6667	2.5337
92	98.6667	96	2.5142
93.3333	100	96.6667	2.2879

WINE

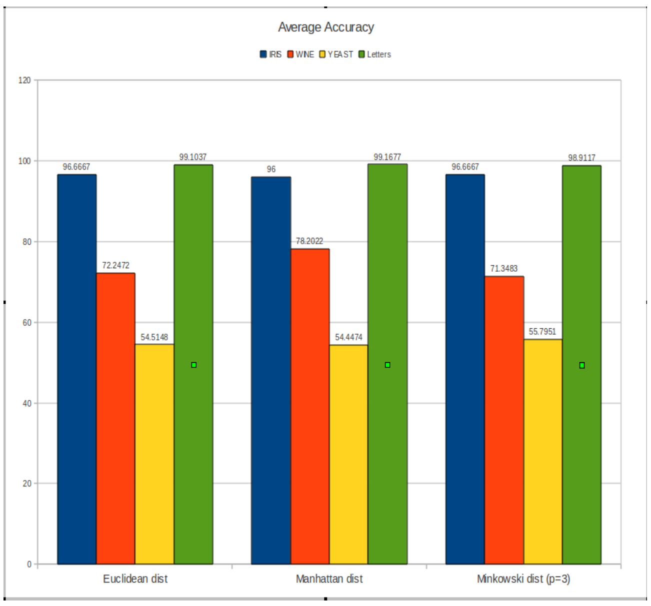
min accuracy	max accuracy	avg accuracy	Std Deviation
64.0449	77.5281	72.2472	4.1385
73.0337	83.1461	78.2022	3.4408
62.9213	76.4045	71.3483	4.0077

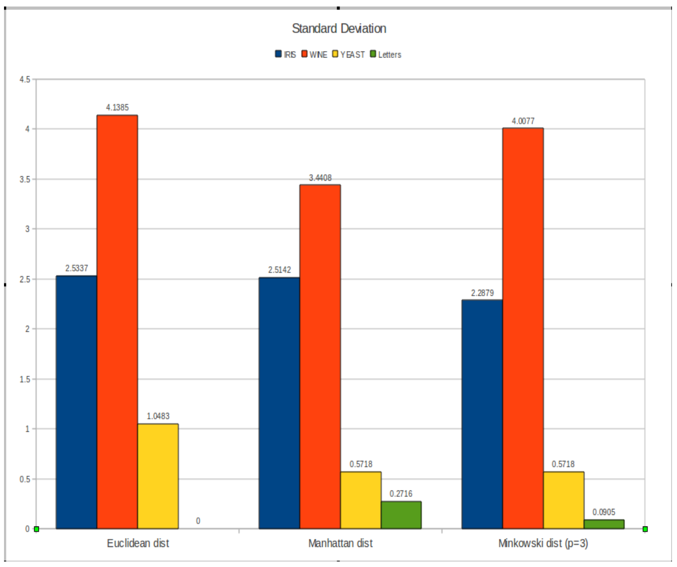
YEAST

min accuracy	max accuracy	avg accuracy	Std Deviation
53.7736	55.2561	54.5148	1.0483
54.0431	54.8518	54.4474	0.5718
55.3908	56.1995	55.7951	0.5718

Letters

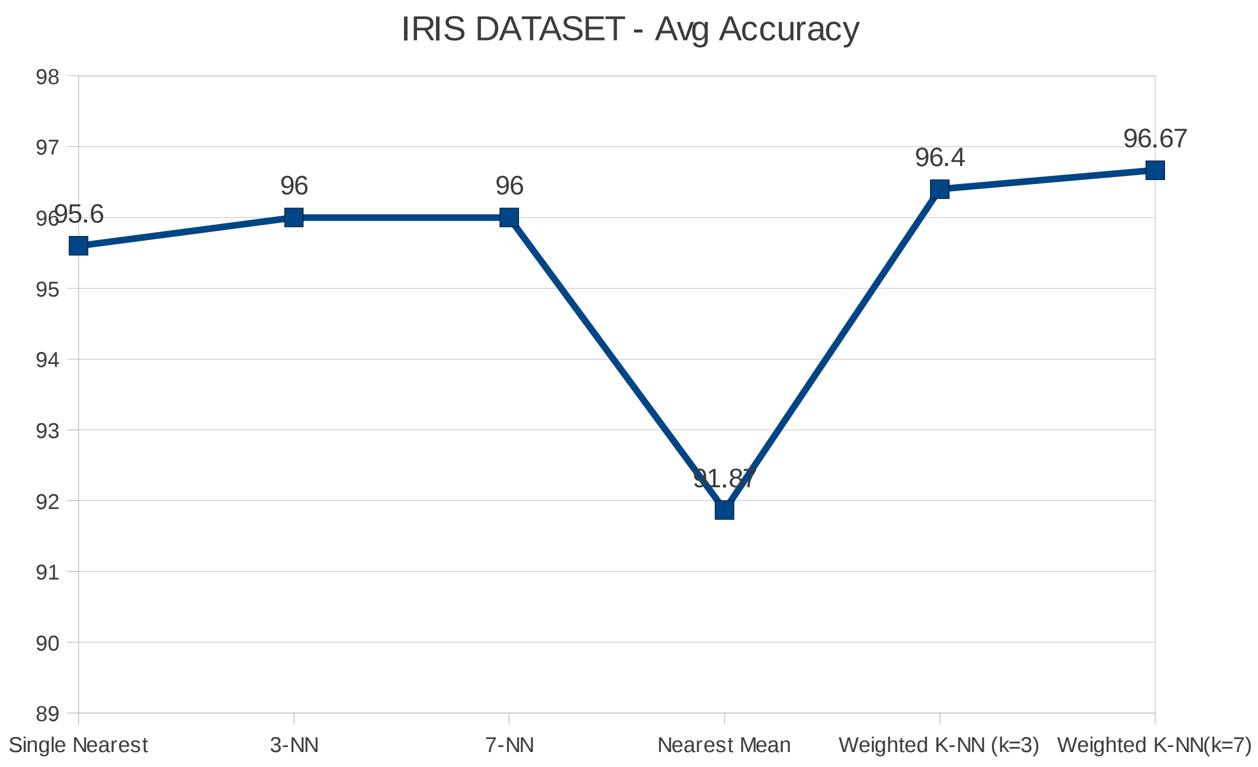
min accuracy	max accuracy	avg accuracy	Std Deviation
99.1037	99.1037	99.1037	0
98.9757	99.3598	99.1677	0.2716
98.8476	98.9757	98.9117	0.0905



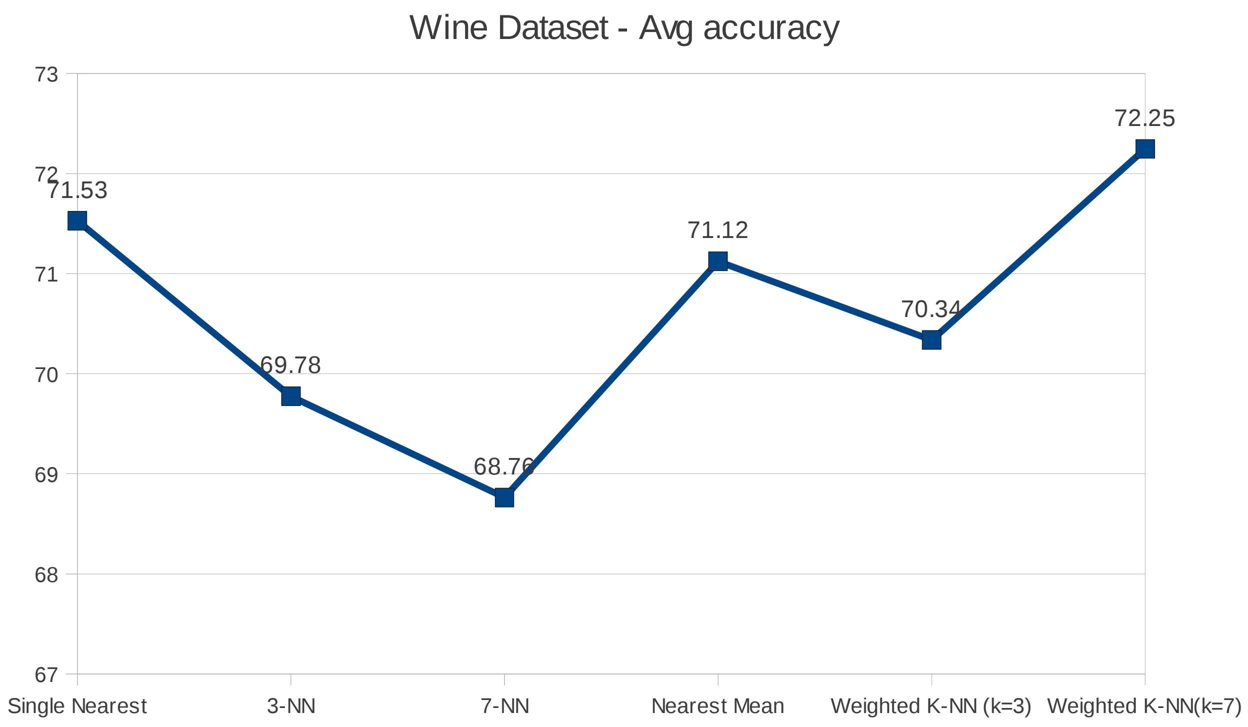


Analysis :

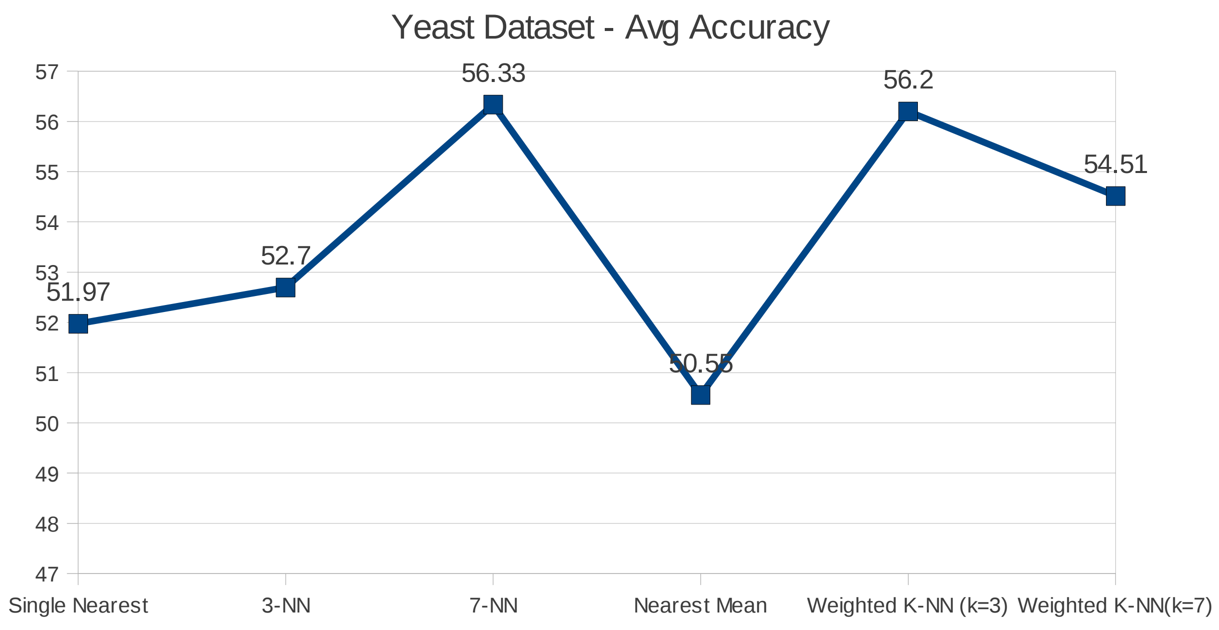
Iris Performance



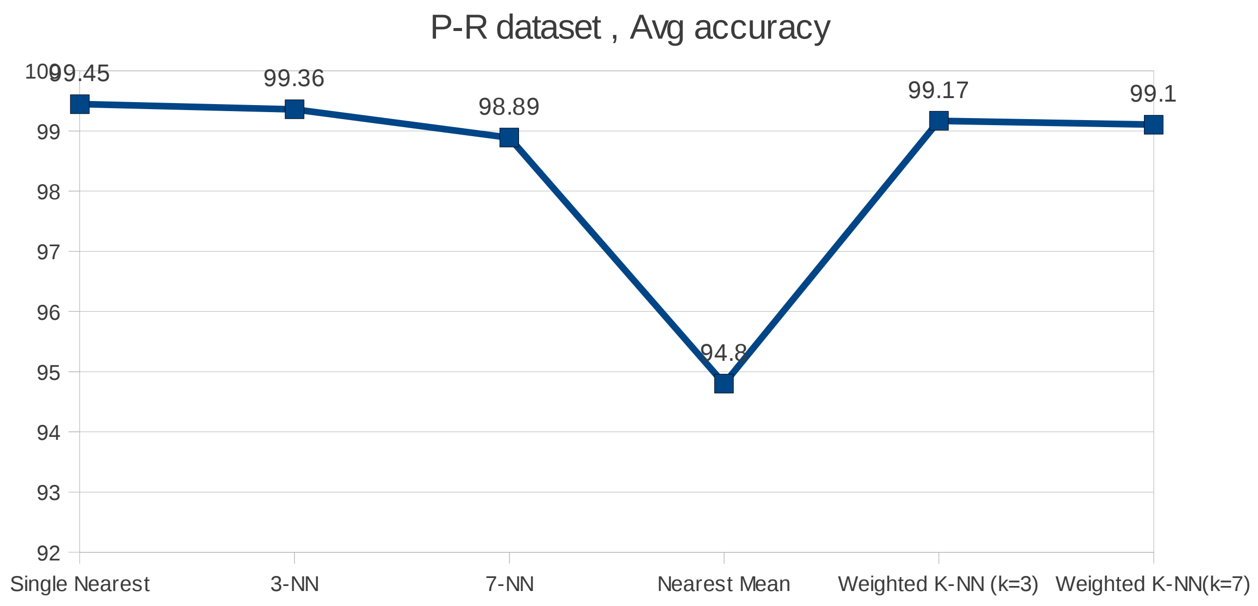
Wine performance



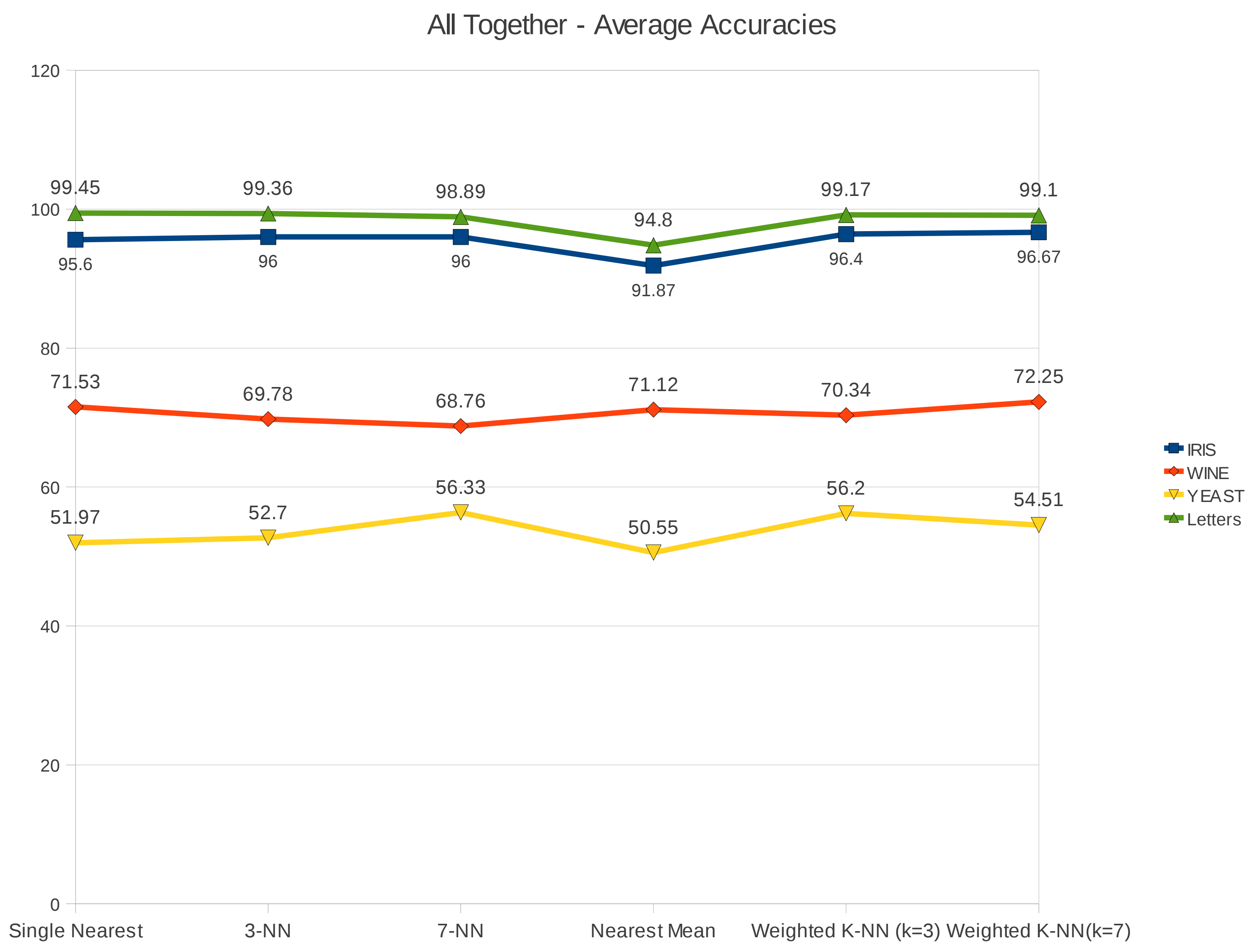
Yeast performance



P-R performance



All together



Linear Classifiers :

1) Batch Perceptron

Implementation

- The given dataset is randomly shuffled. Depending on the ratio of training size to testing size, samples are selected for the training set.
- Since the dataset have more than 1 classes, we have to use either the **one vs rest** approach or the **one vs one** approach for the multicategory case.
- In one vs rest approach, if there are **c** classes, we reduce the problem to **c** two-class problems, where the **i**th problem is to find the hyperplane which separates **i**th class from all other classes. But this approach leads to ambiguous regions where no class can be assigned to the sample.
- Let's say for the **i**th problem, we take all the samples in class i as one class, and all others as the other class. We take the samples to **augmented feature set** where we add one more dimension to each sample, which is always one. The problem reduces to finding a Hyperplane **a** such that $\mathbf{a^t y_i > 0}$ for all i.
- We start with a randomly chosen vector a(k) and k=0. We then find the set of samples which is misclassified by current solution a(k).
- We then apply the update rule $\mathbf{a(k+1)=a(k) + N(k).SUM(y_i)}$, for all i, where each y_i belongs to the set of misclassified samples. N(k) is a constant in this case.
- The update rule is applied iteratively until $\mathbf{|N(k).Sum(y_i)|}$ is less than **theta** which is fixed.
- If the samples are linearly separable, then the procedure converges to a solution vector. But if the samples are not linearly separable, we limit the number of iterations to some fixed value. When update rule is applied, that many times, we take the vector a(k) which misclassified least number of samples, as our final solution vector.

  In this way we get **C** weigth vectors, each separating a particular class from the others.

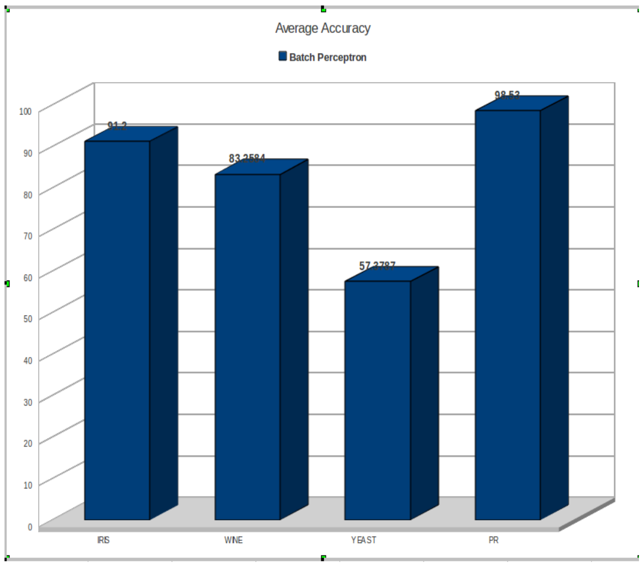
  Testing:

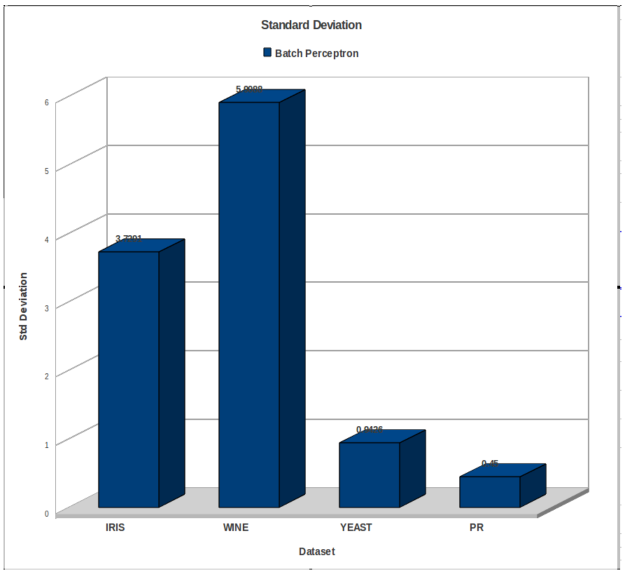
- For each test sample x, we first convert to the augmented feature set vector y, and we find the weight vectors **a** for which $a^t y>0$.
- If there are more than one such weight vectors, we find distance of x from each weight vector and the one which has maximum distance is the output.
- We compare the output with the actual class of x to find the accuracy.

Results:

	min accuracy	max accuracy	average accuracy	Standard Deviation
IRIS	84	96	91.2	3.7291
WINE	74.1573	93.2584	83.2584	5.9088
YEAST	56.469	58.4906	57.3787	0.9426
P-R	97.96	99.49	98.53	0.45

Average Accuracy on Datasets **Standard Deviation on Datasets**





2) Single Sample Perceptron

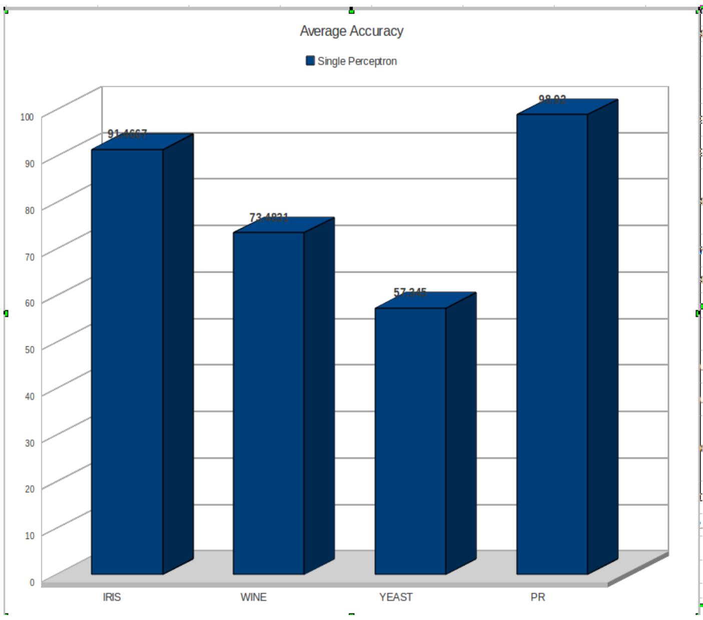
Implementation:

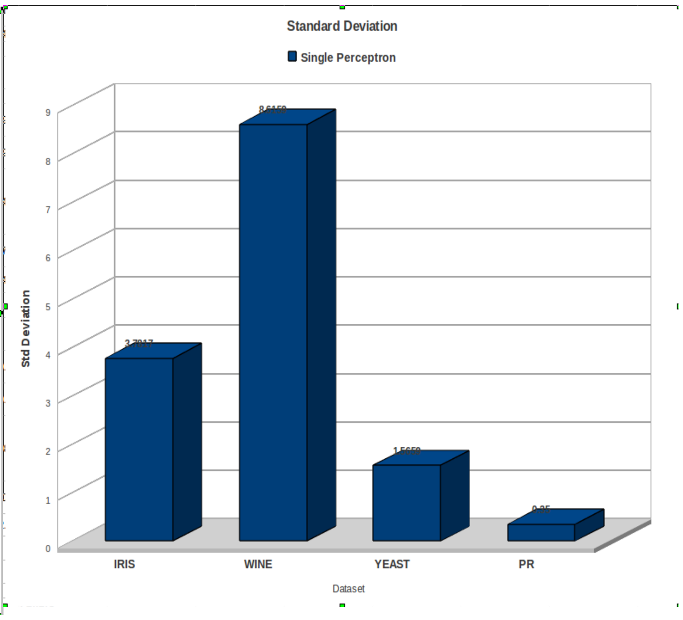
- The single sample perceptron is implemented in the same way as above except the update rule.
- Here, instead of taking the sum of misclassified samples, we take one of the misclassfied samples at random.
- So, the update rule becomes **a(k+1) = a(k) + N(k) . $\mathbf{y_k}$**, where $a(k)^t.y_k < 0$ and N(k) is fixed.
- a(k) should be updated until $|N(k).y_k|$ becomes less than **theta**. But because of performance issues, we compare $|N(k).Sum(y_i)|$ with theta like in Batch Perceptron.
- Ideally, the corrections should be done until all the samples are correctly classified. But for non separable cases, we limit the number of iterations to some fixed value.
- When **k > limit**, the a(k) which misclassified minimum number of samples is the output.
- Testing procedure is same as in Batch Perceptron.

Results:

	min accuracy	max accuracy	average accuracy	Standard Deviation
IRIS	86.6667	97.3333	91.4667	3.7817
WINE	59.5506	79.7753	73.4831	8.6159
YEAST	55.9299	58.8949	57.345	1.5659
P-R	98.4	99.36	98.92	0.35

Average Accuracy on Datasets **Standard Deviation on Datasets**





3) Perceptron with margin and Variable Increment

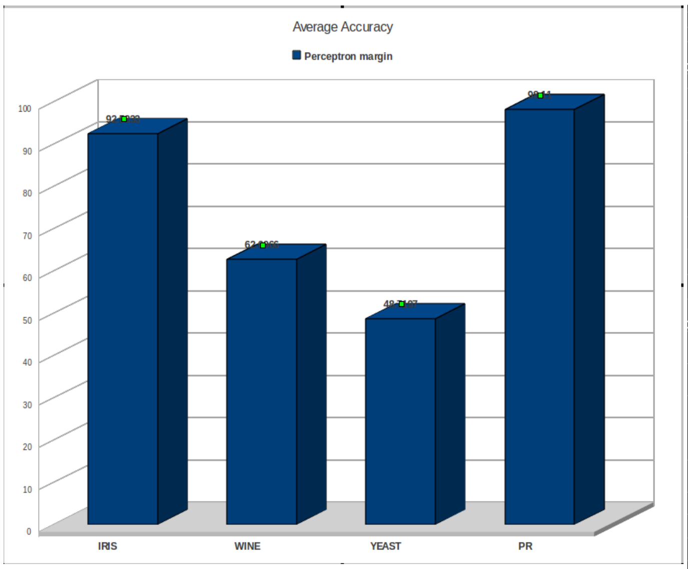
Implementation:

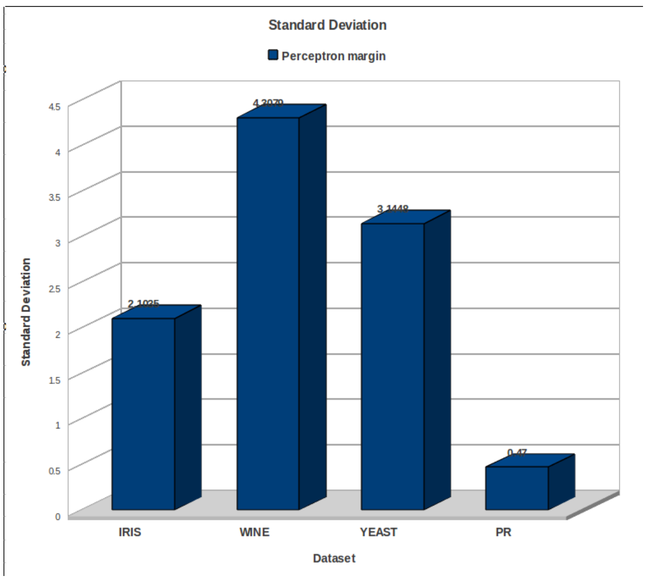
- Here also all the steps are same except the update rule.
- The N(k) in the update rule is variable, and also a margin **b** is introduced.
- A sample y_i is misclassified if $a(k)^t.y_i < b$.
- The update rule becomes $\mathbf{a(k+1) = a(k) + N(k).y_k}$ where $\mathbf{a(k)^t.y_k < b}$
- The value of N(k) depends on the nature of dataset. For separable data, N(k) can be proportional to k, whereas in other cases N(k) is some constant **C * 1/k**.
- Here also, there is limit on iterations and testing process is the same.

Results:

	min accuracy	max accuracy	average accuracy	Standard Deviation
IRIS	89.3333	96	92.5333	2.1035
WINE	57.3034	67.4157	62.6966	4.3079
YEAST	46.496	50.9434	48.7197	3.1448
P-R	97.28	98.9	98.11	0.47

Average Accuracy on Datasets **Standard Deviation on Datasets**





4) MSE (Pseudo – Inverse Method)

Implementation:

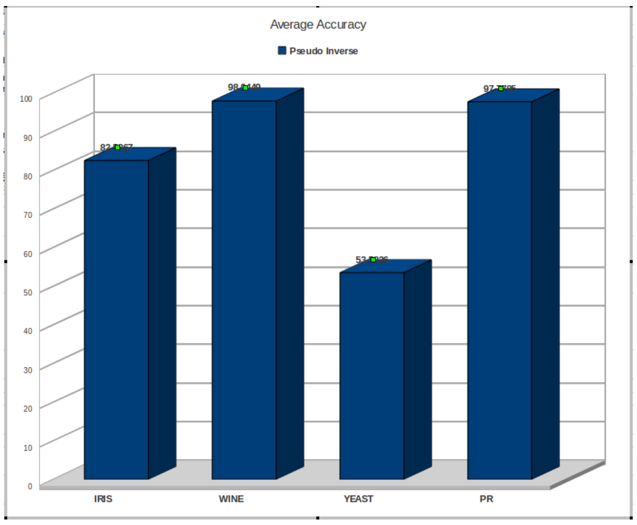
- As stated in the Theory section, the MSE solution for Y.a=b is given by

  a=Y$^+$b, where **Y$^+$ = (Y^{t}Y)$^{-1}$Y^t** is the **pseudoinverse of Y**
- We generate the Y matrix in which each row is a training sample.
- Margin vector **b** is taken as n x 1 matrix of all ones.
- The solution vector is found for each class by solving the above equation and then testing is done in the same way as above.
- This is not an iterative process. The solution vector is found only by doing the above matrix multiplications.

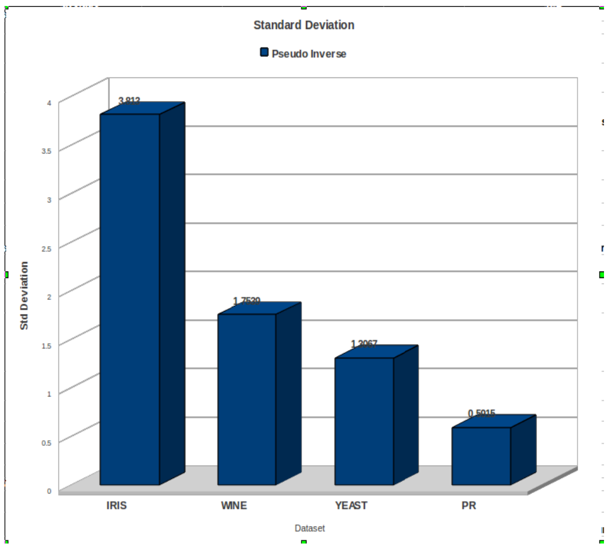
Results:

	min accuracy	max accuracy	average accuracy	Standard Deviation
IRIS	73.3333	92	82.5867	3.813
WINE	87.6404	100	98.0449	1.7539
YEAST	50.6739	57.2776	53.5836	1.3067
P-R	95.9027	99.1037	97.7785	0.5915

Average Accuracy on Datasets



Standard Deviation on Datasets



LMS (Widrow – Hoff rule)

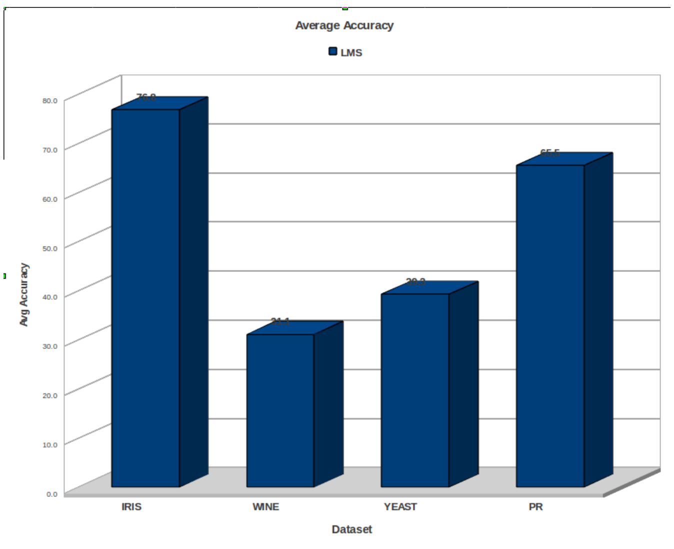
Implementation:

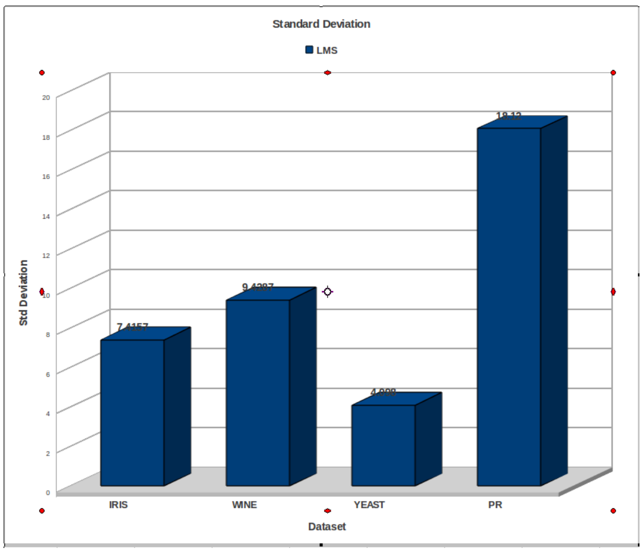
- The procedure is same as batch perceptron and single sample perceptron except the update rule.
- Here, the update rule is based on the MSE function **Mod(Y.a – b)^2**
- The update rule in this case is, **a(k+1) = a(k) + N(k).(b – a(k).y^k).y^k**, where y^k is any of the training samples.
- We apply the update rule until ǀN(k).(b – a(k).y^k).y^{k}ǀ is less than **theta.**
- We also keep a limit on the number of iterations.
- The value of eta is appropriately chosen depending on the nature of dataset.
- Also, the value of a(k) becomes very large after some iterations. Therefore, we **normalize** it at every step so that it does not reach infinity.

Results:

	min accuracy	max accuracy	average accuracy	Standard Deviation
IRIS	62.6667	84	76.8	7.4157
WINE	6.7416	58.427	31.1124	9.4287
YEAST	28.0323	46.496	39.2871	4.098
P-R	24.84	92.39	65.48	18.12

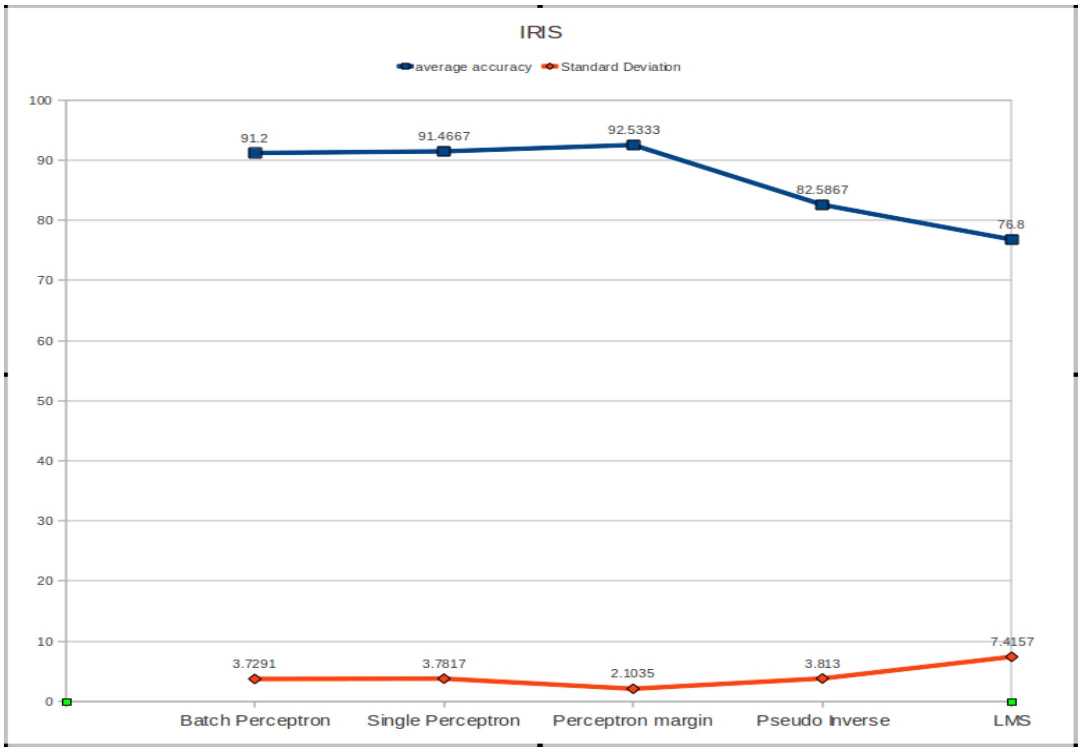
Average Accuracy on Datasets **Standard Deviation on Datasets**



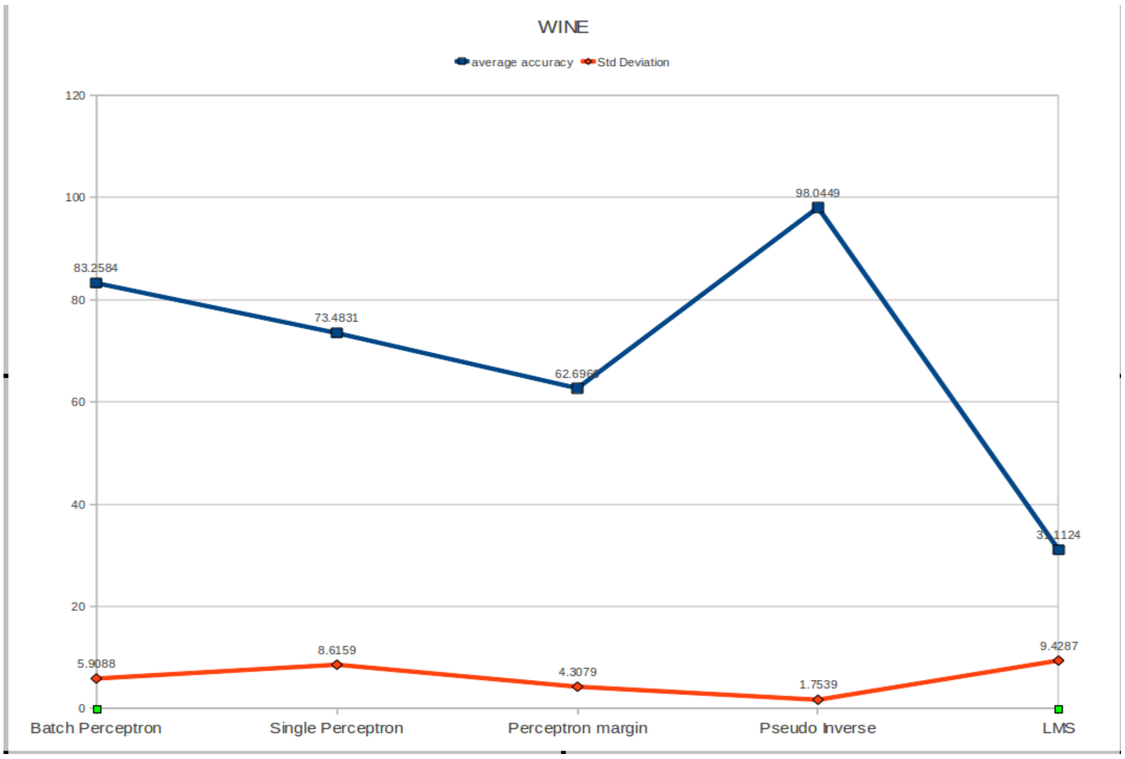


Analysis and Comparisions

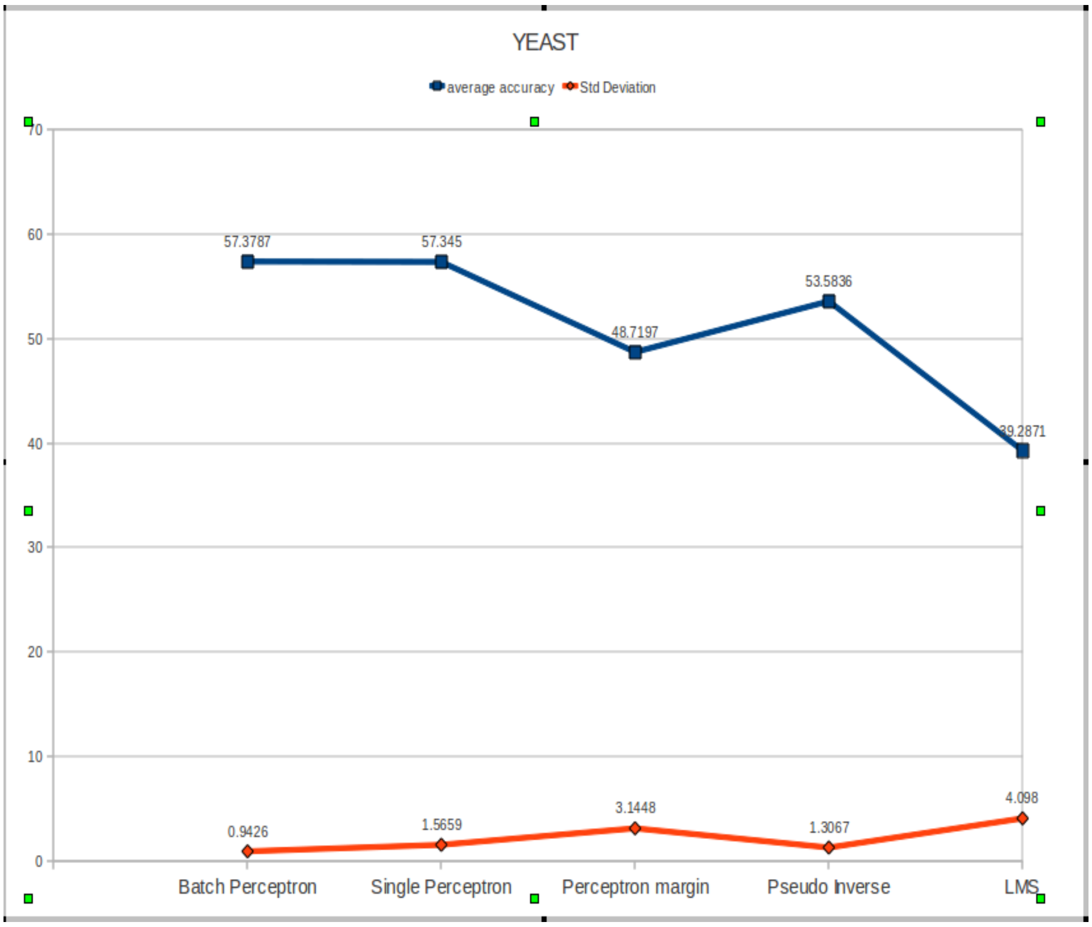
Iris Dataset



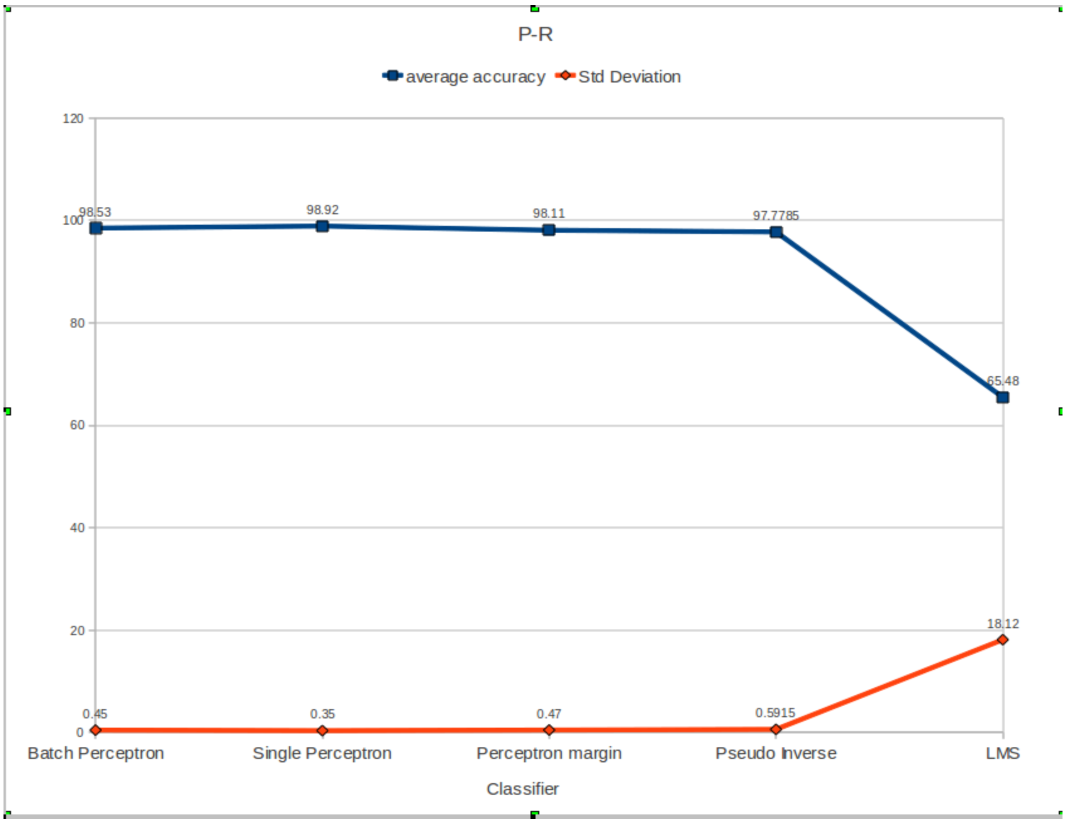
Wine Dataset



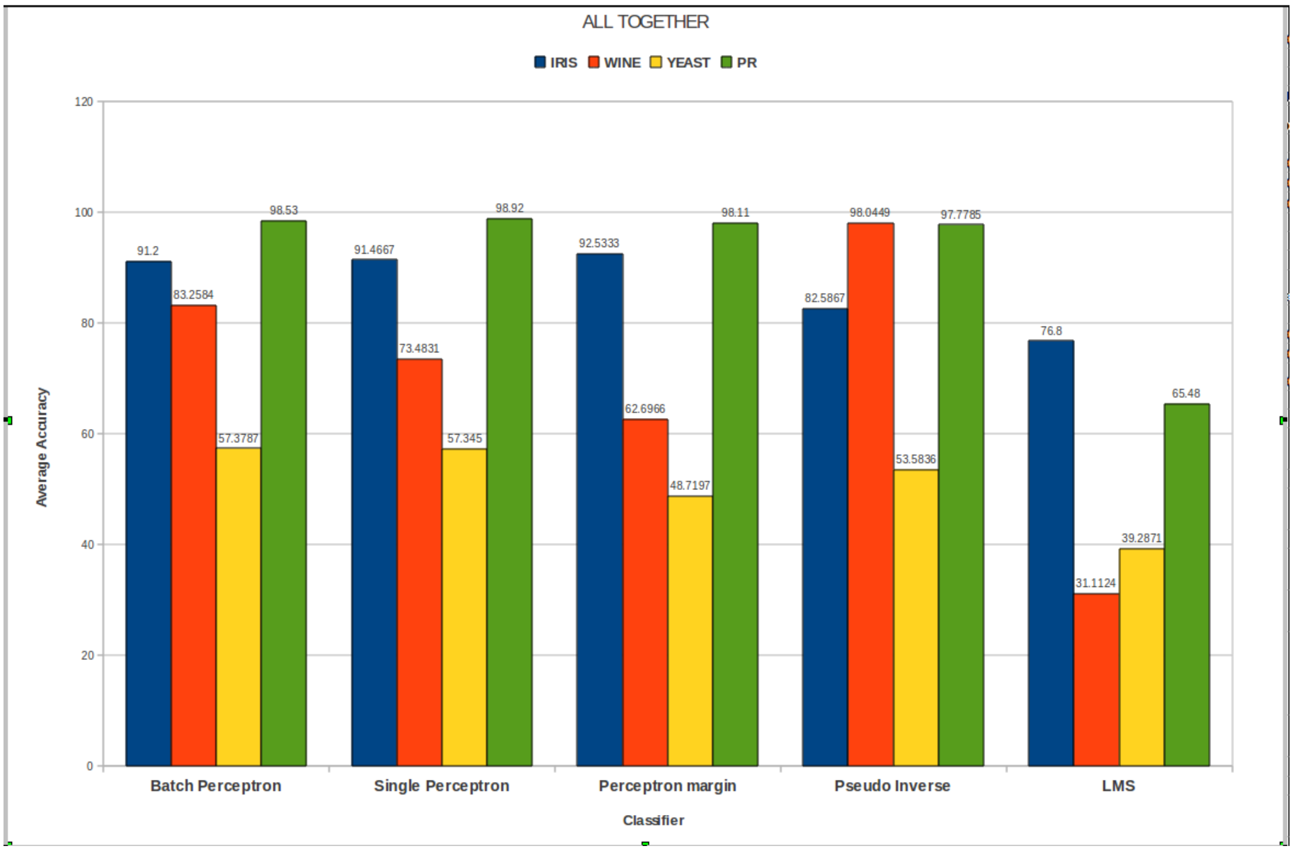
Yeast Dataset



P-R Dataset



All Together



Time and Space complexities

<u>Distance based Classifiers:</u>

- ### Single NN

  Time Complexity :- N^2
  Space Complexity :- (d+2) * N , where d is the dimension.

- ### K-NN and Weighted K-NN

  Time Complexity :- N^2
  Space Complexity :- (d+2) * N , where d is the dimension.

- ### Nearest mean classifier

  Time Complexity :- N * C , where C is the number of classes
  Space Complexity :- (d+2) * N , where d is the dimension.

<u>Linear Classifiers :</u>

- ### Perceptron Learning

  Time Complexity :- *limit* * N , where *limit* is the maximum number of iterations allowed
  Space Complexity :- (d+2) * N , where d is the dimension.

- ### MSE - Pseudoinverse

  Time Complexity :- $(d+1)^2$ * N , where *d* is the dimension
  Space Complexity :- (d+2) * N , where d is the dimension.

- ### LMS

  Time Complexity :- *limit* * N , where *limit* is the maximum number of iterations allowed
  Space Complexity :- (d+2) * N , where d is the dimension.

<u>Conclusion</u> :

- The performance of distance based classifiers and linear classifiers was studied in detail.
- Classification Algorithms give variable performance depending on the nature of datasets and other factors.
- Some Algorithms have parameters like N(k), **theta,** margin vector **b**, b etc. It is very important to select proper values for these parameters. The performance of the same Algorithm on the same dataset may vary largely based on what parameters are chosen. Parameters which give maximum performance must be chosen.
- In distance based classifiers, performance also depends on the distance metrics chosen. Euclidean, Manhattan and Minkowski distances were tested and analysed in the project.

<u>References</u> :

1) "Pattern Classification (second edition)" , Duda, Hart, Stock, Wiley Interscience, ISBN-0-476-05669-3