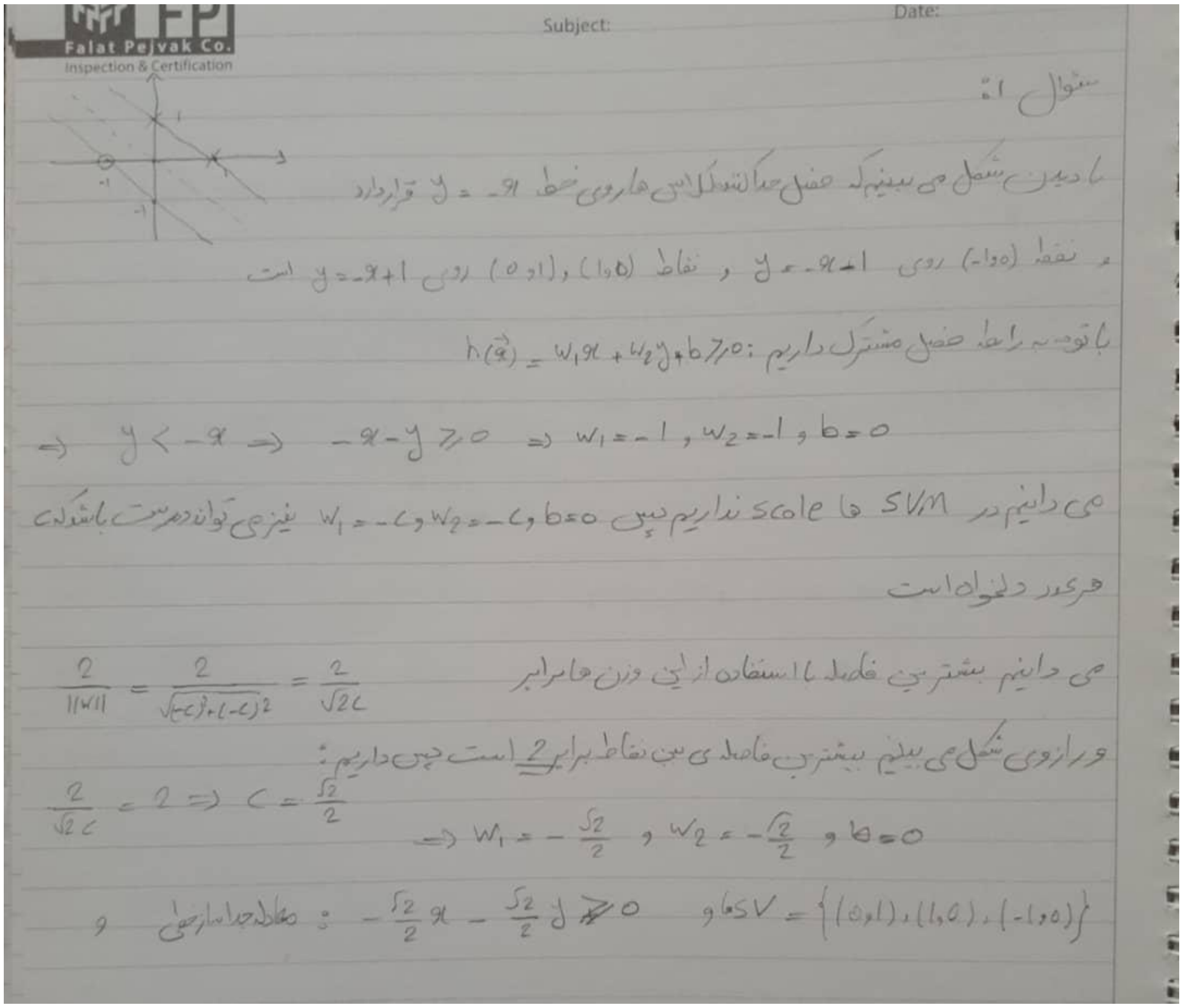
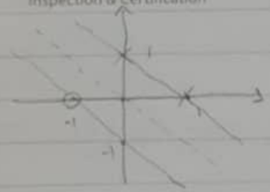
MACHINE LEARNING

Soheil Shirvani 810195416

Question 1:



Subject: Date:



سوال ۱:

با دیدن شکل می بینیم که فصل جداکننده کلاس ها روی خط $y=-x$ قرار دارد

و نقطه $(-1,0)$ روی $y=-x-1$ و نقاط $(1,0),(0,1)$ روی $y=-x+1$ است

با توجه به رابطه فصل مشترک داریم: $h(\vec{x}) = w_1x+w_2y+b \geq 0$

$\Rightarrow y<-x \Rightarrow -x-y\geq 0 \Rightarrow w_1=-1, w_2=-1, b=0$

می دانیم در SVM ها scale نداریم پس $w_1=-c, w_2=-c, b=0$ نیز می تواند درست باشد که c

هر عدد دلخواه است

می دانیم بیشترین فاصله با استفاده از این وزن ها برابر $\frac{2}{\|w\|} = \frac{2}{\sqrt{(-c)^2+(-c)^2}} = \frac{2}{\sqrt{2}c}$

و از روی شکل می بینیم بیشترین فاصله ی بین نقاط برابر $\underline{2}$ است پس داریم:

$\frac{2}{\sqrt{2}c}=2 \Rightarrow c=\frac{\sqrt{2}}{2}$

$\Rightarrow w_1=-\frac{\sqrt{2}}{2}, w_2=-\frac{\sqrt{2}}{2}, b=0$

و SV ها $=\{(0,1),(1,0),(-1,0)\}$ و معادله جداساز خطی: $-\frac{\sqrt{2}}{2}x-\frac{\sqrt{2}}{2}y \geq 0$ و

Question 2:

1)

Solving the primal problem, we obtain the optimal w, but **know nothing about the** α_i. In order to classify a query point x we need to explicitly compute the scalar product $w^T x$, which may be **expensive** if d is large.

Solving the dual problem, we obtain the α_i (where $\alpha_i=0$ for all but a few points - the support vectors). In order to classify a query point x, we calculate

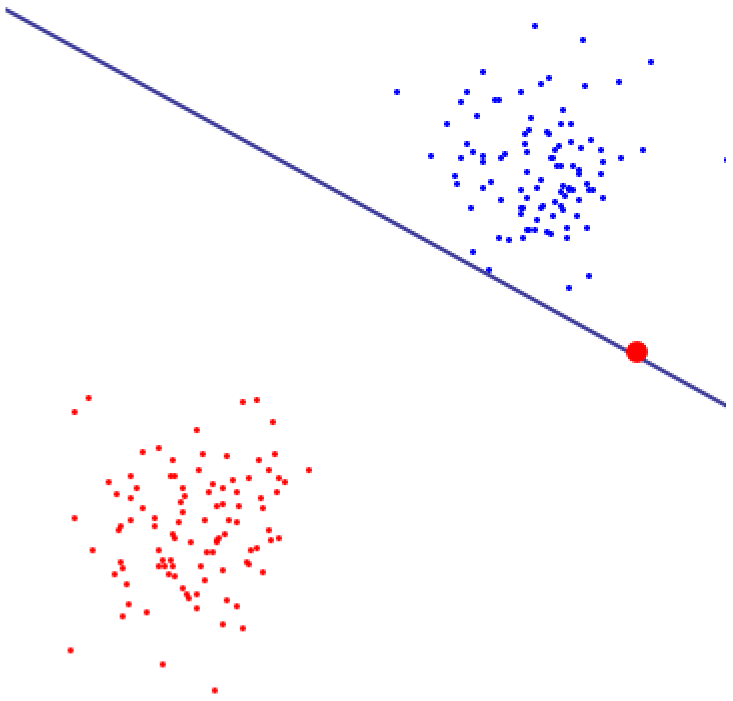
$$w^T x + w_0 = \left(\sum_{i=1}^{n} \alpha_i y_i x_i\right)^T x + w_0 = \sum_{i=1}^{n} \alpha_i y_i \langle x_i, x \rangle + w_0$$

This term is very **efficiently calculated** if there are only few support vectors. Further, since we now have a scalar product only involving *data* vectors, we may **apply the kernel trick**.

2)

I would expect soft-margin SVM to be better even when training dataset is linearly separable. The reason is that in a hard-margin SVM, a single outlier can determine the boundary, which makes the classifier overly sensitive to noise in the data.

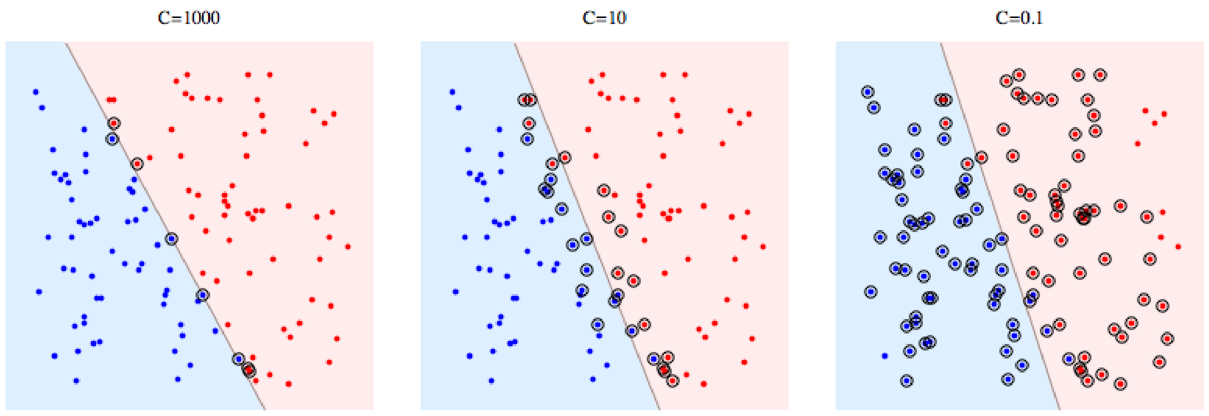
In the diagram below, a single red outlier essentially determines the boundary, which is the hallmark of overfitting



To get a sense of what soft-margin SVM is doing, it's better to look at it in the dual formulation, where you can see that it has the same margin-maximizing objective (margin could be negative) as the hard-margin SVM, but with an additional constraint that each lagrange multiplier associated with support vector is bounded by C. Essentially this bounds the influence of any single point on the decision boundary, for derivation, see Proposition 6.12 in Cristianini/Shaw-Taylor's "An Introduction to Support Vector Machines and Other Kernel-based Learning Methods".

The result is that soft-margin SVM could choose decision boundary that has non-zero training error even if dataset is linearly separable, and is less likely to overfit.

Here's an example using libSVM on a synthetic problem. Circled points show support vectors. You can see that decreasing C causes classifier to sacrifice linear separability in order to gain stability, in a sense that influence of any single datapoint is now bounded by C.



Meaning of support vectors:

For hard margin SVM, support vectors are the points which are "on the margin". In the picture above, C=1000 is pretty close to hard-margin SVM, and you can see the circled points are the ones that will touch the margin (margin is almost 0 in that picture, so it's essentially the same as the separating hyperplane)

For soft-margin SVM, it's easier to explain them in terms of dual variables. Your support vector predictor in terms of dual variables is the following function.

$$\begin{aligned} f(\mathbf{x}, \boldsymbol{\alpha}^*, b^*) &= \sum_{i=1}^{\ell} y_i \alpha_i^* \langle \mathbf{x}_i \cdot \mathbf{x} \rangle + b^* \\ &= \sum_{i \in \mathrm{sv}} y_i \alpha_i^* \langle \mathbf{x}_i \cdot \mathbf{x} \rangle + b^*. \end{aligned}$$

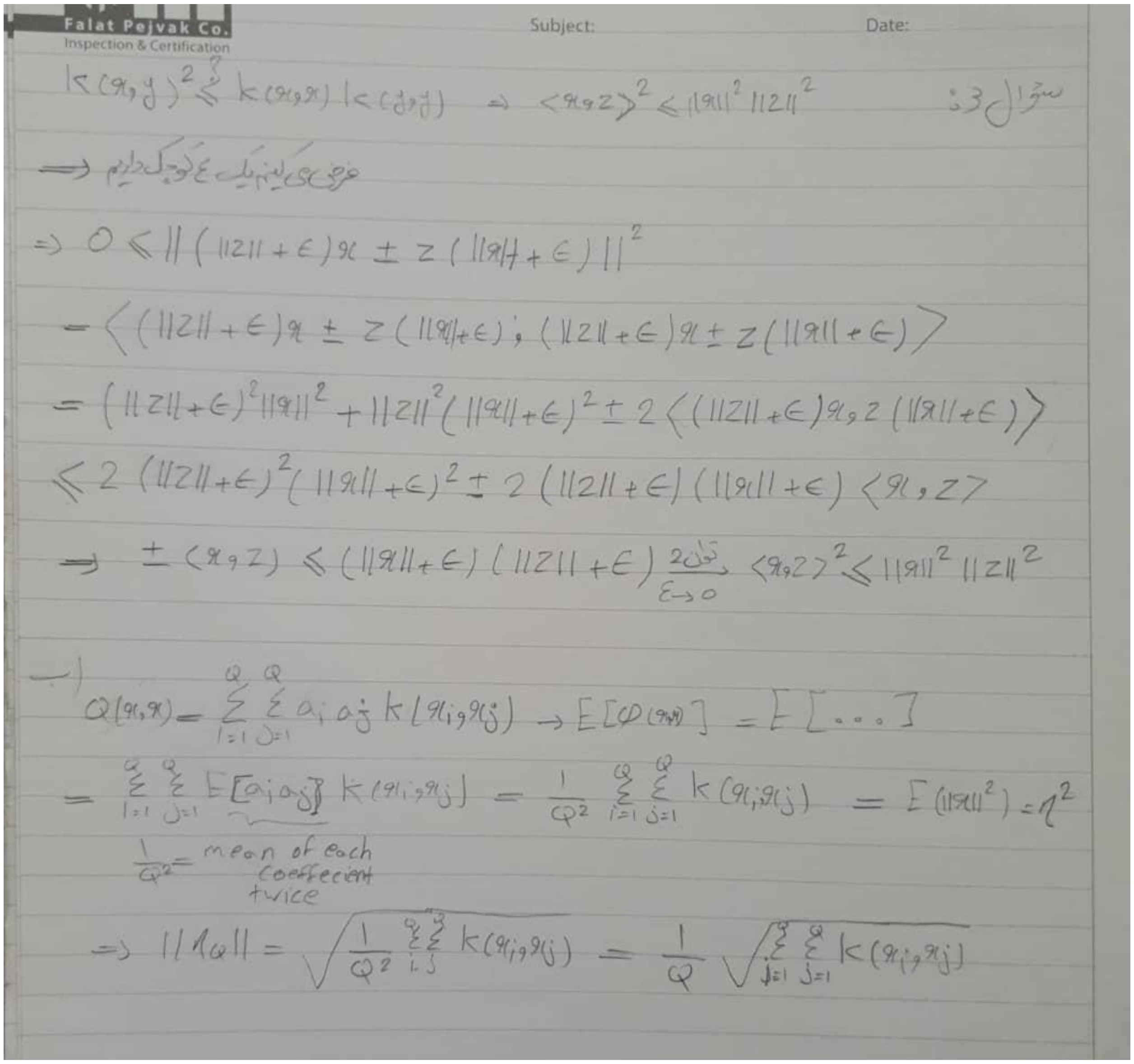
Here, alphas and b are parameters that are found during training procedure, xi's, yi's are your training set and x is the new datapoint. Support vectors are datapoints from training set which are are included in the predictor, ie, the ones with non-zero alpha parameter.

3)

The C parameter tells the SVM optimization how much you want to avoid misclassifying each training example. 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. For very tiny values of C, you should get misclassified examples, often even if your training data is linearly separable.

Question 3:

Kernel Function is a method used to take data as input and transform into the required form of processing data. “Kernel” is used due to set of mathematical functions used in Support Vector Machine provides the window to manipulate the data. So, Kernel Function generally transforms the training set of data so that a non-linear decision surface is able to transformed to a linear equation in a higher number of dimension spaces. Basically, it returns the inner product between two points in a standard feature dimension.



Falat Pejvak Co.
Inspection & Certification

Subject:

Date:

سؤال 3:

$$k(x,y)^2 \overset{?}{\leq} k(x,x)\,k(y,y) \Rightarrow \langle x,z\rangle^2 \leq \|x\|^2\,\|z\|^2$$

$\Longrightarrow$ فرض می‌کنیم یک ϵ کوچک داریم

$$\Rightarrow 0 \leq \|(\|z\|+\epsilon)x \pm z(\|x\|+\epsilon)\|^2$$

$$= \langle(\|z\|+\epsilon)x \pm z(\|x\|+\epsilon), (\|z\|+\epsilon)x \pm z(\|x\|+\epsilon)\rangle$$

$$= (\|z\|+\epsilon)^2\|x\|^2 + \|z\|^2(\|x\|+\epsilon)^2 \pm 2\langle(\|z\|+\epsilon)x, z(\|x\|+\epsilon)\rangle$$

$$\leq 2(\|z\|+\epsilon)^2(\|x\|+\epsilon)^2 \pm 2(\|z\|+\epsilon)(\|x\|+\epsilon)\langle x,z\rangle$$

$$\Longrightarrow \pm\langle x,z\rangle \leq (\|x\|+\epsilon)(\|z\|+\epsilon) \xrightarrow[\epsilon\to 0]{\text{2توان}} \langle x,z\rangle^2 \leq \|x\|^2\,\|z\|^2$$

ب)

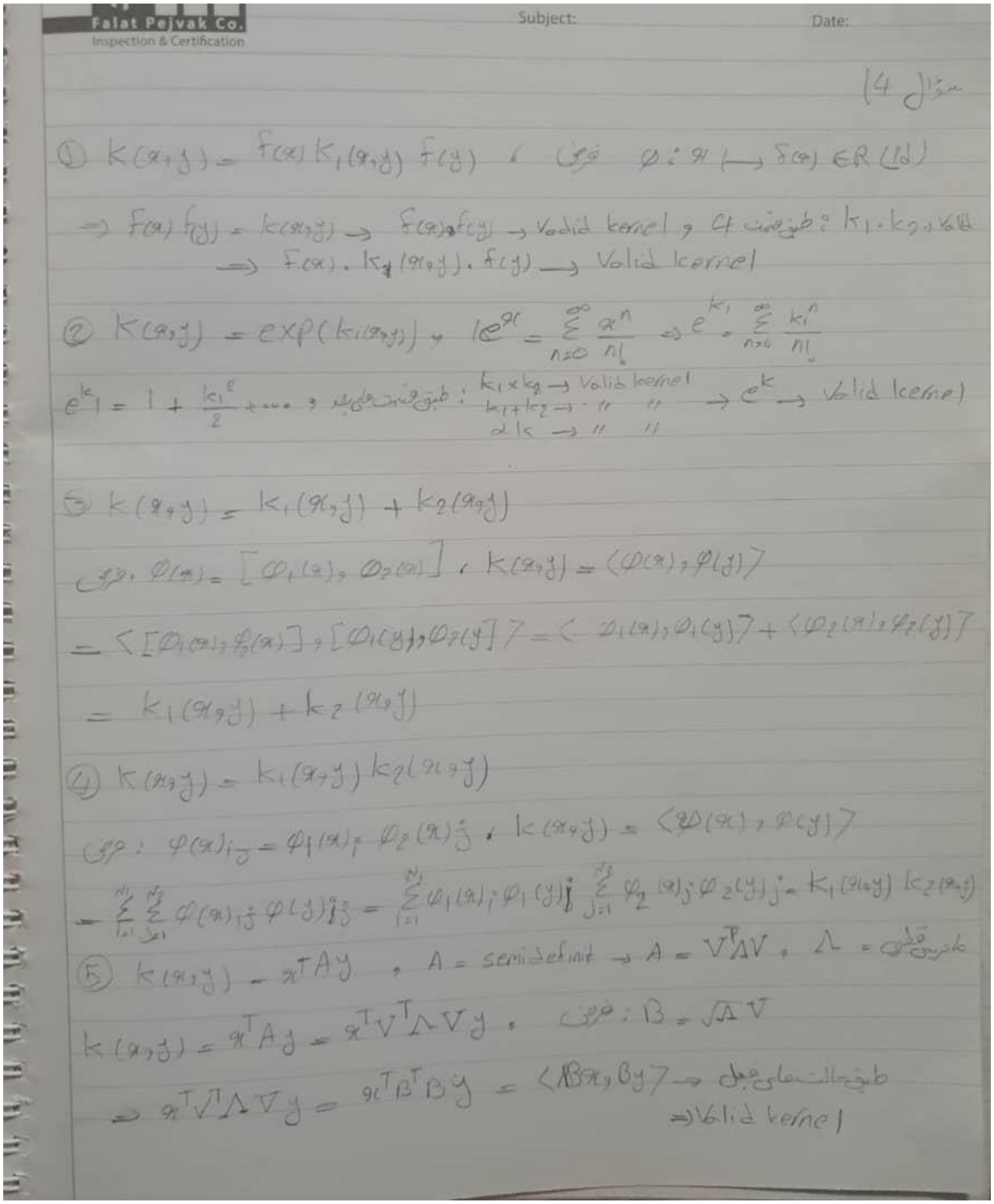
$$Q(x,x) = \sum_{i=1}^{Q}\sum_{j=1}^{Q} a_i a_j\, k(x_i,x_j) \rightarrow E[\varphi(x,x)] = E[\ldots]$$

$$= \sum_{i=1}^{Q}\sum_{j=1}^{Q} \underbrace{E[a_i a_j]}\, k(x_i,x_j) = \frac{1}{Q^2}\sum_{i=1}^{Q}\sum_{j=1}^{Q} k(x_i,x_j) = E(\|x\|^2) = \eta^2$$

$\frac{1}{Q^2}$ = mean of each coeffecient twice

$$\Rightarrow \|\eta_\omega\| = \sqrt{\frac{1}{Q^2}\sum_{i}^{Q}\sum_{j}^{Q} k(x_i,x_j)} = \frac{1}{Q}\sqrt{\sum_{j=1}^{Q}\sum_{j=1}^{Q} k(x_i,x_j)}$$

Question 4:


Falat Pejvak Co.
Inspection & Certification
Subject:
Date:

سوال 4)

① $k(x,y) = f(x)\, k_1(x,y)\, f(y)$ ، فرض $\phi : x \longmapsto f(x) \in R\ (1d)$

$\Rightarrow f(x)\, f(y) = k(x,y) \rightarrow f(x) f(y) \rightarrow$ Valid kernel , طبق خاصیت C4 : $k_1 . k_2$, Valid

$\Rightarrow f(x) . k_1(x,y) . f(y) \longrightarrow$ Valid kernel

② $k(x,y) = \exp(k_1(x,y))$, $e^{x} = \sum_{n=0}^{\infty} \frac{x^n}{n!} \Rightarrow e^{k_1} = \sum_{n=0}^{\infty} \frac{k_1^n}{n!}$

$e^{k_1} = 1 + \frac{k_1^2}{2} + \dots$, طبق خواص کرنل ها : $k_1 \times k_2 \rightarrow$ Valid kernel ; $k_1 + k_2 \rightarrow$ " " ; $\alpha k \rightarrow$ " " $\rightarrow e^{k} \rightarrow$ Valid kernel

③ $k(x,y) = k_1(x,y) + k_2(x,y)$

فرض . $\phi(x) = [\phi_1(x), \phi_2(x)]$, $k(x,y) = \langle \phi(x), \phi(y) \rangle$

$= \langle [\phi_1(x), \phi_2(x)], [\phi_1(y), \phi_2(y)] \rangle = \langle \phi_1(x), \phi_1(y) \rangle + \langle \phi_2(x), \phi_2(y) \rangle$

$= k_1(x,y) + k_2(x,y)$

④ $k(x,y) = k_1(x,y)\, k_2(x,y)$

فرض : $\phi(x)_{ij} = \phi_1(x)_i\, \phi_2(x)_j$, $k(x,y) = \langle \phi(x), \phi(y) \rangle$

$= \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} \phi(x)_{ij}\, \phi(y)_{ij} = \sum_{i=1}^{N_1} \phi_1(x)_i\, \phi_1(y)_i \sum_{j=1}^{N_2} \phi_2(x)_j\, \phi_2(y)_j = k_1(x,y)\, k_2(x,y)$

⑤ $k(x,y) = x^T A y$, $A =$ semidefinit $\rightarrow A = V^T \Lambda V$, $\Lambda =$ ماتریس قطری

$k(x,y) = x^T A y = x^T V^T \Lambda V y$, فرض : $B = \sqrt{\Lambda}\, V$

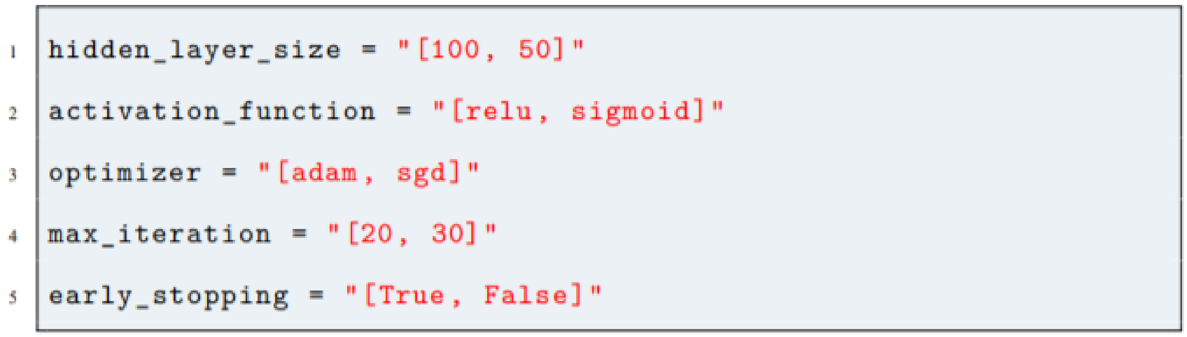
$\Rightarrow x^T V^T \Lambda V y = x^T B^T B y = \langle Bx, By \rangle \rightarrow$ طبق استدلال های قبل

$\Rightarrow$ Valid kernel

Question 7: Coding:

In this part we are going to change hyper parameters of a 2-layer perceptron and compare the parameters. It has a input layer, hidden layer, and an output layer

Out parameters are:



```
hidden_layer_size = "[100, 50]"
activation_function = "[relu, sigmoid]"
optimizer = "[adam, sgd]"
max_iteration = "[20, 30]"
early_stopping = "[True, False]"
```

First we are going to discuss about each individual parameter here:

1) Hidden Layer Size:
   This parameter defines number of neurons in the hidden layer. We can increase or decrease the size of the hidden layer using number of neurons in our code.

2) Activation Function:
   - Relu: The rectified linear activation function or Relu for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. ... The rectified linear activation function overcomes the vanishing gradient problem, allowing models to learn faster and perform better. $F(x) = x \, ; x \geq 0$
   - Sigmoid: In sigmoid activation, the input to the function is transformed into a value between 0.0 and 1.0. Inputs that are much larger than 1.0 are transformed to the value 1.0, similarly, values much smaller than 0.0 are snapped to 0.0. Shape of the function for all possible inputs is an S-shape from 0 up through 0.5 to 1.0. $F(x) = \frac{1}{1+ e^{-x}}$

3) Optimizer:
   - Adam: Adam is a replacement optimization algorithm for stochastic gradient descent for training deep learning models. Adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems. Adam (short for Adaptive Moment Estimation) is an update to the RMSProp optimizer. In this optimization algorithm, running averages of both the gradients and the second moments of the gradients are used.
   - SGD: Stochastic gradient descent (often abbreviated SGD) is an iterative method for optimizing an objective function with suitable smoothness properties. classical stochastic gradient descent is generally sensitive to the step size η. Fast convergence requires large step sizes but this may induce numerical instability. The problem can be largely solved[13] by considering implicit updates whereby the stochastic gradient is evaluated at the next iterate rather than the current one.
4) Max Iteration:
   Max Iteration is the number of epochs which our model will run before ending. More epochs mean more learning and the overfitting will be increased.
5) Early Stopping:

   stopping the training of neural network early before it has overfit the training dataset can reduce overfitting and improve the generalization of deep neural networks. The challenge of training a neural network long enough to learn the mapping, but not so long that it overfits the training data. Model performance on a holdout validation dataset can be monitored during training and training stopped when generalization error starts to increase. The use of early stopping requires the selection of a performance measure to monitor, a trigger for stopping training, and a selection of the model weights to use.

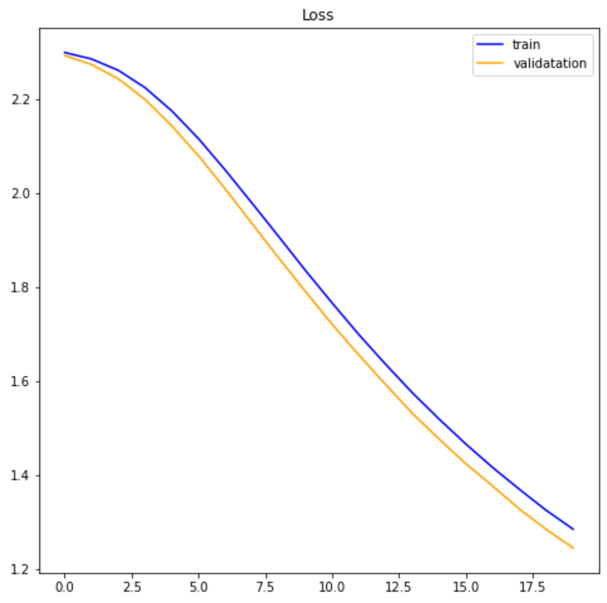
Our Default model is Default Network: Optimizer=Adam, NumberOfNeuron=100, Activation=Relu, EarlyStoping=False, Epoch=20, And for comparing we just change that specific parameter and save all the rest. For the next part, we then write back our default network and change the parameters with respect to the question again.

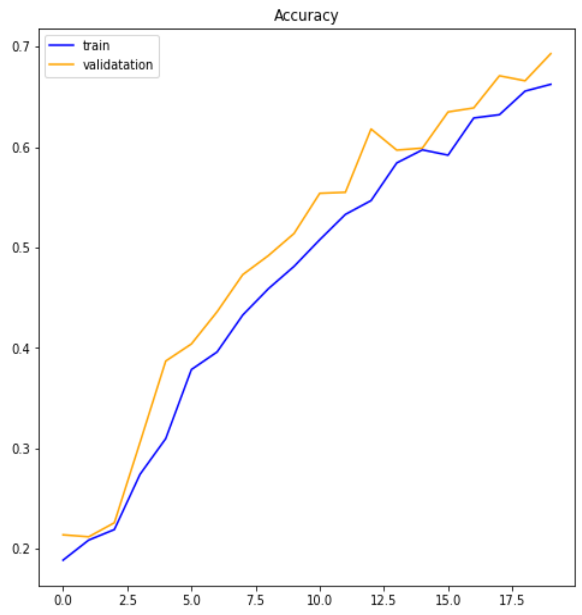
Now We can see The Result of each change here:

1. Changing number of neurons in hidden layer:

1) 50:

```
Epoch 19/20
125/125 [==============================] - 0s 2ms/step - loss: 1.3242 - accuracy: 0.6557 - val_loss: 1.2837 - val_accuracy: 0.6
660
Epoch 20/20
125/125 [==============================] - 0s 2ms/step - loss: 1.2843 - accuracy: 0.6625 - val_loss: 1.2444 - val_accuracy: 0.6
930
79/79 [==============================] - 0s 1ms/step - loss: 1.3651 - accuracy: 0.6180
```

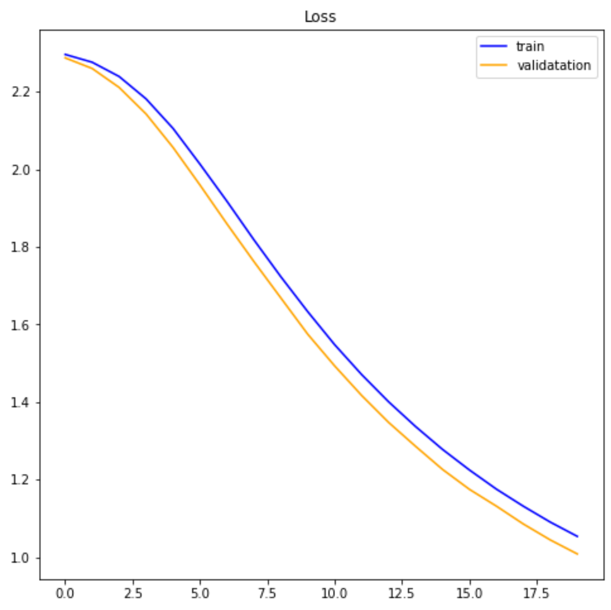


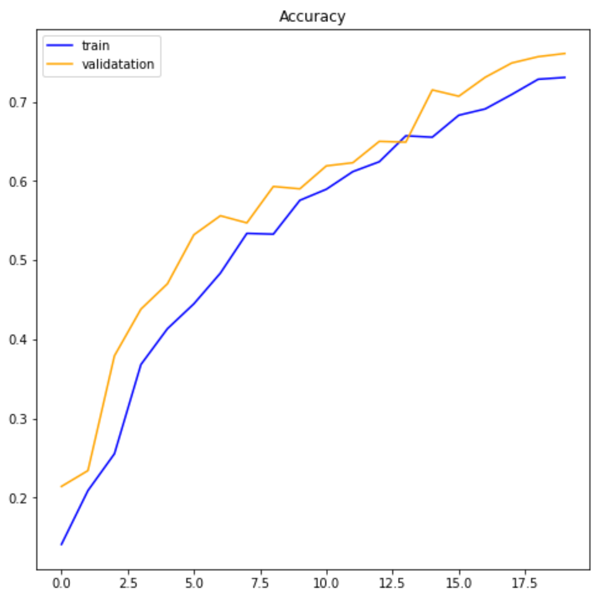


```
Test Accuracy is :  61.800
Train Accuracy is :  66.250
Train Confusion Matrix
[[474   0   2  22   2   1  19   0   2   0]
 [  0 552   3   7   0   0   0   2   0   0]
 [ 11  43 374  31   5   0  32   3   3   1]
 [ 14  41  14 433   1   0  10  18   4   4]
 [  7  22   2   0 250   0  13  13   2 136]
 [ 32  48   4 195  15  31  19  33  46   6]
 [  9  23  25   4   3   0 448   0   1   0]
 [  4  51   0   2   7   0   1 437   0  18]
 [ 27  92  27 153   5   0  17   4 136  21]
 [ 13  22   1   5  56   0   2 152   1 231]]
Test Confusion Matrix
[[199   0   0   5   0   1  13   0   1   0]
 [  0 281   1   3   0   0   2   0   0   0]
 [  9  39 167  30   2   0  22   6   0   1]
 [  1  20  10 201   1   0   2  14   1   4]
 [  6  12   3   0 121   0  12  16   0 105]
 [ 16   5   6 111   4  10  14  31  15   9]
 [  7  13  12   5   4   0 183   0   0   1]
 [  1  38   2   1   4   0   0 199   3   9]
 [ 11  33  12  91   6   0   5   9  56  19]
 [  3  16   1   4  31   0   1  58   2 128]]
```

2) 100 Neurons:

```
Epoch 19/20
125/125 [==============================] - 0s 1ms/step - loss: 1.0902 - accuracy: 0.7285 - val_loss: 1.0442 - val_accuracy: 0.7
570
Epoch 20/20
125/125 [==============================] - 0s 2ms/step - loss: 1.0535 - accuracy: 0.7308 - val_loss: 1.0082 - val_accuracy: 0.7
610
79/79 [==============================] - 0s 1ms/step - loss: 1.1392 - accuracy: 0.7092
```





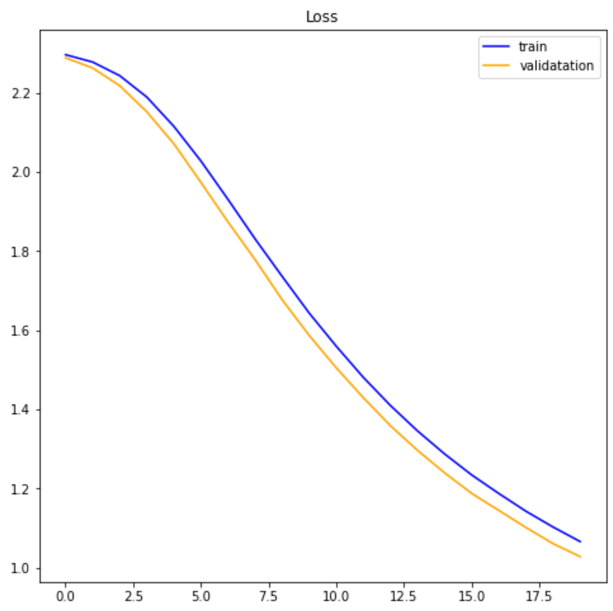
```
Test Accuracy is :  70.920
Train Accuracy is :  73.075
Train Confusion Matrix
[[479   0   7   7   2   8  11   0   8   0]
 [  0 548   5   2   0   0   0   2   7   0]
 [  8  35 413  12   7   0  19   3   6   0]
 [  6  23  29 434   2   5   3  17  17   3]
 [  1  11   1   0 359   0  15  10   0  48]
 [ 28  46  17 116  24 105  12   8  55  18]
 [  7  21  14   0   3   2 464   0   2   0]
 [  3  33   2   0   8   0   0 452   1  21]
 [ 21  63  41  68  11   1  14   5 240  18]
 [ 12   9   2   5  86   0   1 151   2 215]]
Test Confusion Matrix
[[201   0   1   1   0   2  10   0   4   0]
 [  0 279   1   1   0   0   2   0   4   0]
 [  9  23 207   9   2   0  16   7   3   0]
 [  0   7  15 209   0   1   1  14   5   2]
 [  1   8   1   0 194   0  15   8   0  48]
 [ 15   4  11  81  11  51   7  15  14  12]
 [  6  11   9   1   4   2 192   0   0   0]
 [  1  26   3   0   3   0   0 214   3   7]
 [ 10  19  28  39  10   0   3  11 109  13]
 [  3   7   1   3  56   1   1  51   4 117]]
```

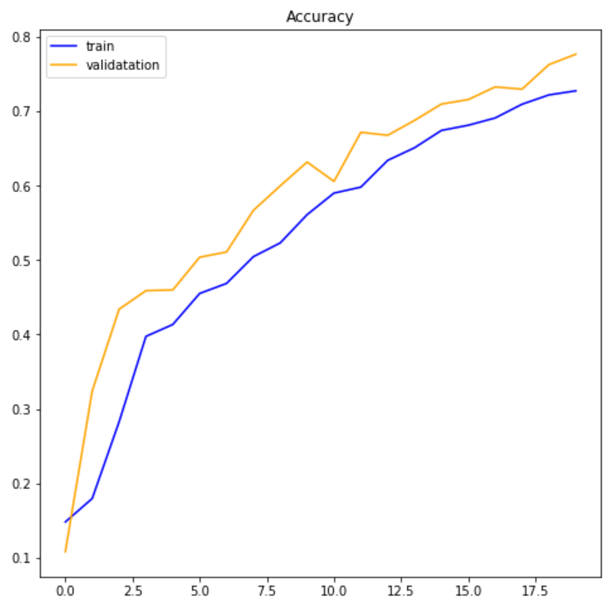
Here we can see by increasing the number of neurons to 100, a better accuracy was achieved.
There is no particular way to determine what number of neurons is better for the model, we should find it by experiments. In this question since our input shape had 196 feature it was kind of guessing that 50 is small. It was proved by experiment as 100 neurons achieved a better accuracy.

2. Activation Functions:

1) Relu:

```
Epoch 19/20
125/125 [==============================] - 0s 2ms/step - loss: 1.1031 - accuracy: 0.7222 - val_loss: 1.0611 - val_accuracy: 0.7
630
Epoch 20/20
125/125 [==============================] - 0s 2ms/step - loss: 1.0664 - accuracy: 0.7278 - val_loss: 1.0282 - val_accuracy: 0.7
770
79/79 [==============================] - 0s 1ms/step - loss: 1.1662 - accuracy: 0.7060
```

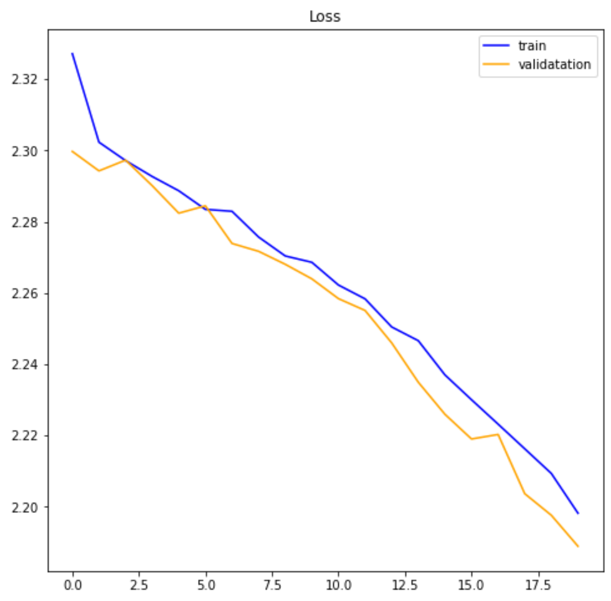


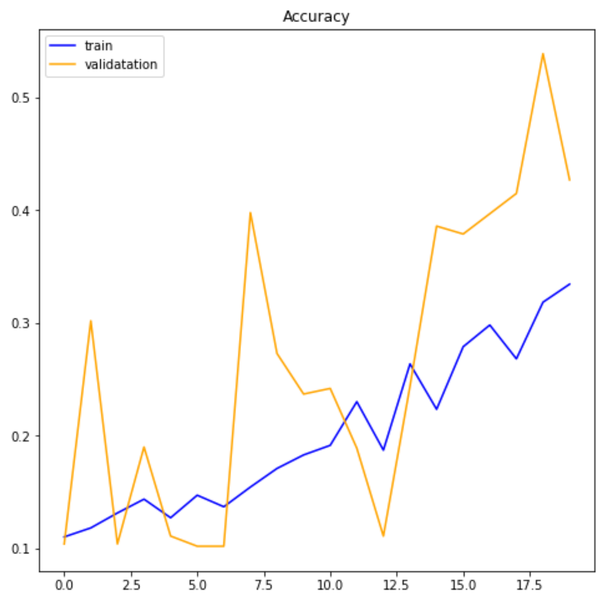


```
Test Accuracy is :  70.600
Train Accuracy is :  72.775
Train Confusion Matrix
[[485   0   2   5   4   8  10   0   8   0]
 [  0 548   4   4   0   4   0   1   2   1]
 [ 12  31 388  14   9   0  37   3   9   0]
 [ 11  24  22 427   5   8   4  14  15   9]
 [  2   6   0   0 339   0  15   2   0  81]
 [ 22  33   3 129  24 136  15  12  29  26]
 [  5  19   6   0   3   2 475   0   3   0]
 [  4  24   1   0   8   0   1 436   5  41]
 [ 27  56  20  82   6   0  19   5 241  26]
 [ 12   9   0   5  62   0   3  56   1 335]]
Test Confusion Matrix
[[202   0   0   0   2   4   9   0   2   0]
 [  0 279   0   3   0   0   3   0   2   0]
 [ 10  19 192  10   2   1  27   7   7   1]
 [  0   8  10 205   0   4   3  12   6   6]
 [  1   6   0   0 189   0  14   1   0  64]
 [ 12   4   6  89   8  45  10  21   8  18]
 [  7   9   2   1   6   1 199   0   0   0]
 [  1  19   5   0   4   0   0 188   4  36]
 [  9  20  14  51   9   0   4   8 103  24]
 [  3   7   1   4  47   0   1  15   3 163]]
```

2) Sigmoid:

```
Epoch 19/20
125/125 [==============================] - 0s 2ms/step - loss: 2.2094 - accuracy: 0.3185 - val_loss: 2.1976 - val_accuracy: 0.5
390
Epoch 20/20
125/125 [==============================] - 0s 1ms/step - loss: 2.1982 - accuracy: 0.3345 - val_loss: 2.1889 - val_accuracy: 0.4
270
79/79 [==============================] - 0s 1ms/step - loss: 2.2143 - accuracy: 0.3808: 0s - loss: 2.2136 - accuracy: 0.38
```





```
Test Accuracy is :  38.080
Train Accuracy is :  33.450
Train Confusion Matrix
[[473   2   0  46   0   0   1   0   0   0]
 [  0 554   0  10   0   0   0   0   0   0]
 [ 83 116  44 219   0   0  41   0   0   0]
 [ 10  36   0 489   0   0   3   0   0   1]
 [ 68 135   0 104   0   0  39   0   0  99]
 [ 34 133   0 258   0   0   2   0   0   2]
 [ 38  70   0  22   0   0 383   0   0   0]
 [ 45 269   0  89   0   0   1   0   0 116]
 [ 43 143   0 287   0   0   6   0   0   3]
 [ 51 158   0 101   0   0   0   0   0 173]]
Test Confusion Matrix
[[202   0   0  16   0   0   1   0   0   0]
 [  0 279   0   7   0   0   1   0   0   0]
 [ 20  82  13 144   0   0  16   0   0   1]
 [  1  15   0 237   0   0   0   0   0   1]
 [ 45  78   0  87   0   0  16   0   0  49]
 [ 16  32   0 171   0   0   1   0   0   1]
 [ 28  37   0  16   0   0 144   0   0   0]
 [ 11 163   0  59   0   0   0   0   0  24]
 [ 13  47   0 176   0   0   1   0   0   5]
 [ 12  69   0  85   0   0   1   0   0  77]]
```

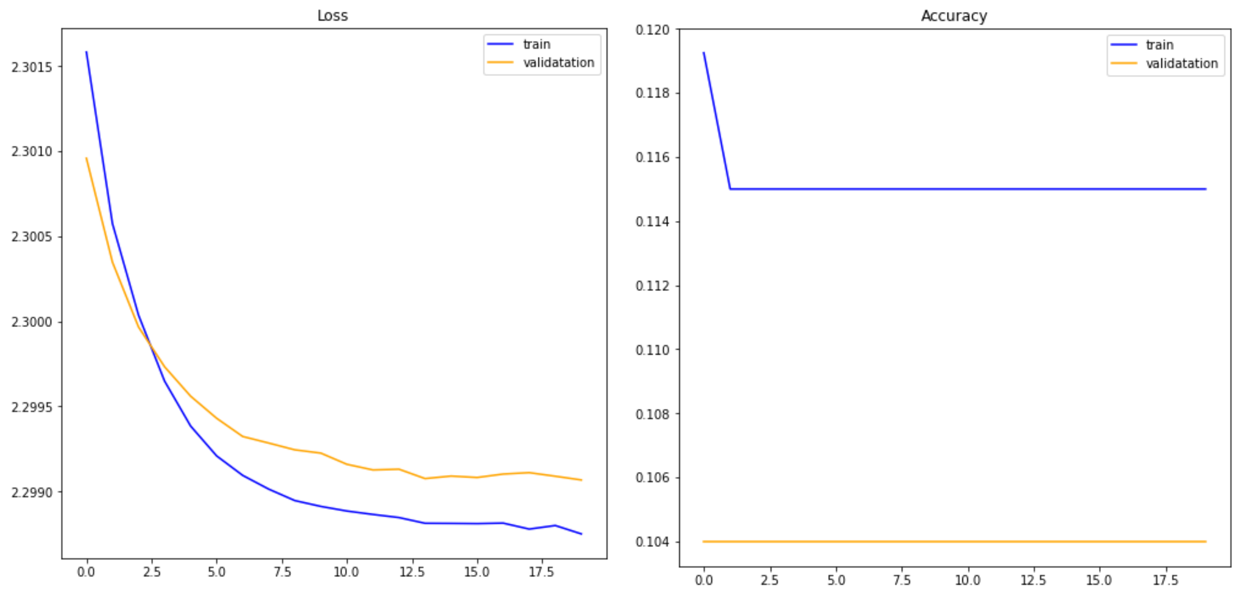
Here we test 2 different activation functions Relu and Sigmoid.
As we can see, model using sigmoid could not fit on the dataset and its loss is noisy. Its accuracy is low too. Sigmoid have a lot of problem including climbing, gradient vanishing and etc. It is not a good activation function; it was used in the passed before the appearance of new activations like Relu.

Relu achieved a great accuracy and it is good activation for general purposes and will often result in good accuracy.

3. Optimizer:

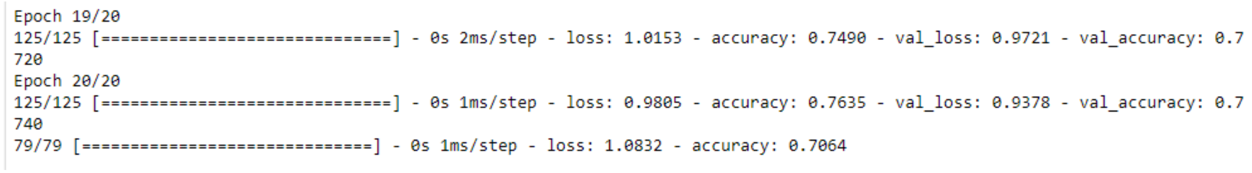
1) SGD

```
Epoch 19/20
125/125 [==============================] - 0s 2ms/step - loss: 2.2988 - accuracy: 0.1150 - val_loss: 2.2991 - val_accuracy: 0.1
040
Epoch 20/20
125/125 [==============================] - 0s 2ms/step - loss: 2.2988 - accuracy: 0.1150 - val_loss: 2.2991 - val_accuracy: 0.1
040
79/79 [==============================] - 0s 1ms/step - loss: 2.3020 - accuracy: 0.1148
```

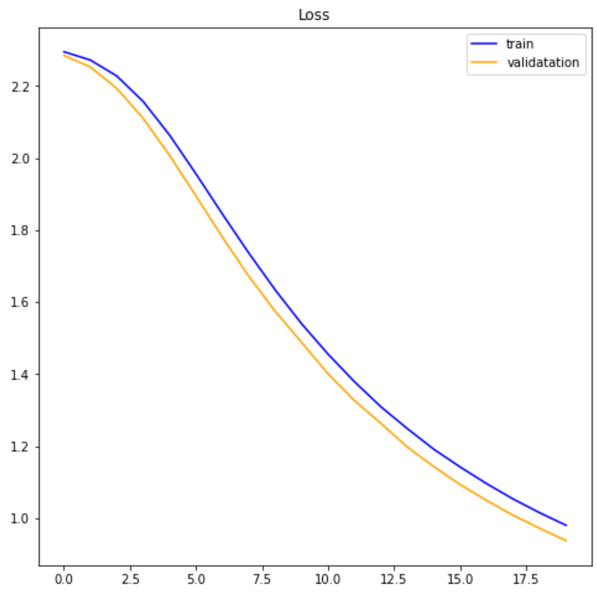


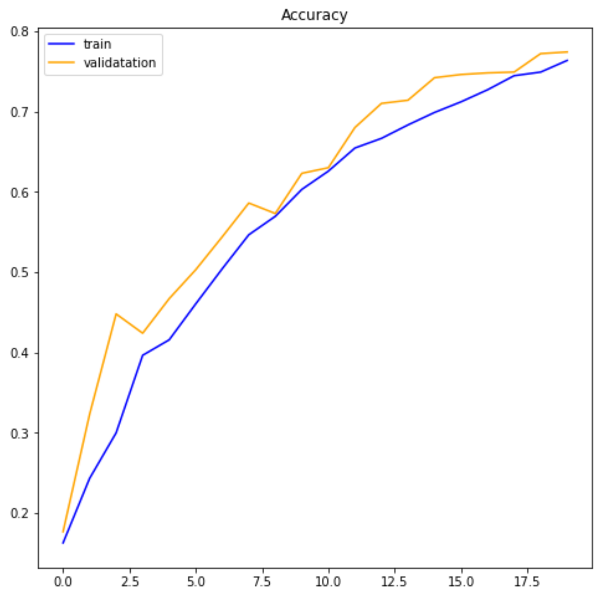
```
Test Accuracy is :  11.480
Train Accuracy is :  11.500
Train Confusion Matrix
[[  0 522   0   0   0   0   0   0   0   0]
 [  0 564   0   0   0   0   0   0   0   0]
 [  0 503   0   0   0   0   0   0   0   0]
 [  0 539   0   0   0   0   0   0   0   0]
 [  0 445   0   0   0   0   0   0   0   0]
 [  0 429   0   0   0   0   0   0   0   0]
 [  0 513   0   0   0   0   0   0   0   0]
 [  0 520   0   0   0   0   0   0   0   0]
 [  0 482   0   0   0   0   0   0   0   0]
 [  0 483   0   0   0   0   0   0   0   0]]
Test Confusion Matrix
[[  0 219   0   0   0   0   0   0   0   0]
 [  0 287   0   0   0   0   0   0   0   0]
 [  0 276   0   0   0   0   0   0   0   0]
 [  0 254   0   0   0   0   0   0   0   0]
 [  0 275   0   0   0   0   0   0   0   0]
 [  0 221   0   0   0   0   0   0   0   0]
 [  0 225   0   0   0   0   0   0   0   0]
 [  0 257   0   0   0   0   0   0   0   0]
 [  0 242   0   0   0   0   0   0   0   0]
 [  0 244   0   0   0   0   0   0   0   0]]
```

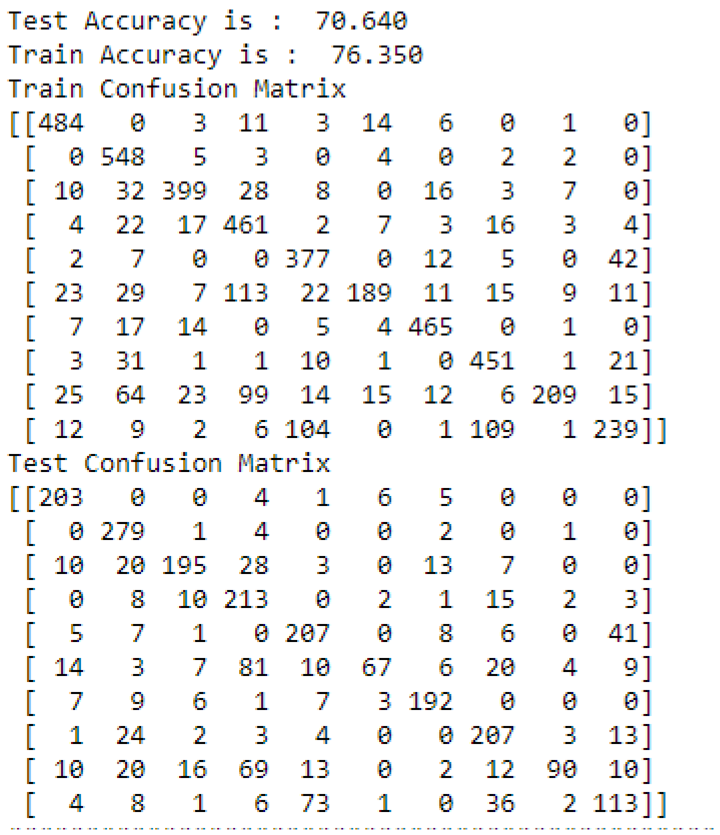
2) Adam:



```
Epoch 19/20
125/125 [==============================] - 0s 2ms/step - loss: 1.0153 - accuracy: 0.7490 - val_loss: 0.9721 - val_accuracy: 0.7
720
Epoch 20/20
125/125 [==============================] - 0s 1ms/step - loss: 0.9805 - accuracy: 0.7635 - val_loss: 0.9378 - val_accuracy: 0.7
740
79/79 [==============================] - 0s 1ms/step - loss: 1.0832 - accuracy: 0.7064
```







```
Test Accuracy is :  70.640
Train Accuracy is :  76.350
Train Confusion Matrix
[[484   0   3  11   3  14   6   0   1   0]
 [  0 548   5   3   0   4   0   2   2   0]
 [ 10  32 399  28   8   0  16   3   7   0]
 [  4  22  17 461   2   7   3  16   3   4]
 [  2   7   0   0 377   0  12   5   0  42]
 [ 23  29   7 113  22 189  11  15   9  11]
 [  7  17  14   0   5   4 465   0   1   0]
 [  3  31   1   1  10   1   0 451   1  21]
 [ 25  64  23  99  14  15  12   6 209  15]
 [ 12   9   2   6 104   0   1 109   1 239]]
Test Confusion Matrix
[[203   0   0   4   1   6   5   0   0   0]
 [  0 279   1   4   0   0   2   0   1   0]
 [ 10  20 195  28   3   0  13   7   0   0]
 [  0   8  10 213   0   2   1  15   2   3]
 [  5   7   1   0 207   0   8   6   0  41]
 [ 14   3   7  81  10  67   6  20   4   9]
 [  7   9   6   1   7   3 192   0   0   0]
 [  1  24   2   3   4   0   0 207   3  13]
 [ 10  20  16  69  13   0   2  12  90  10]
 [  4   8   1   6  73   1   0  36   2 113]]
```

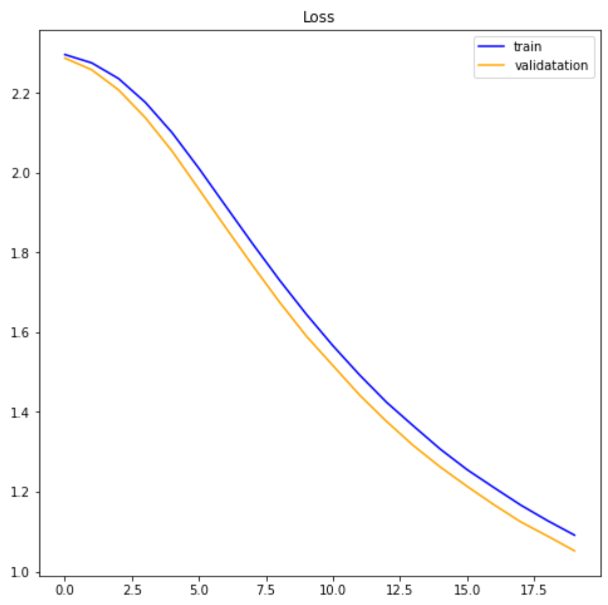
Here we tried different Optimization algorithms including SGD and ADAM. SGD could not fit on the dataset at all and as its confusion matrix shows all the output was predicted to be the class 2. This is because it could not be learned at all and the class 2 had the most elements between all.

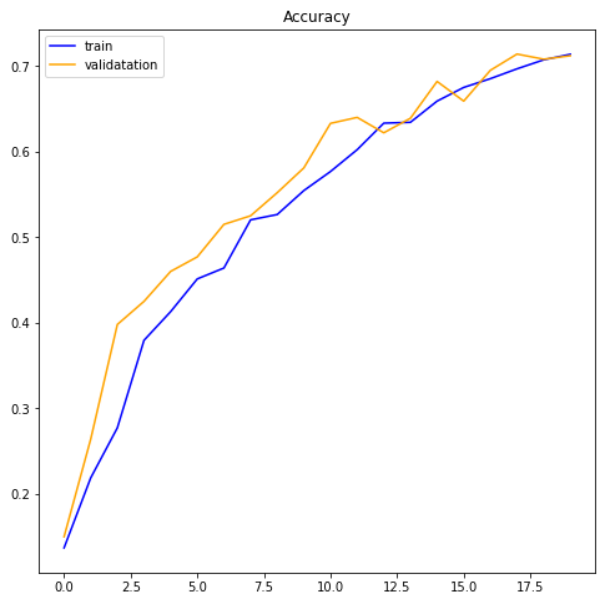
Adam is a newer and better optimization algorithm which nowadays, developers use a lot. It is based on RMSProp activation and uses all the benefits of the optimizers. Adam is a common used optimizer and generally will result in good accuracy as it can be seen here.

4. Early Stopping:

1) True:

```
Epoch 19/20
125/125 [==============================] - 0s 2ms/step - loss: 1.1275 - accuracy: 0.7072 - val_loss: 1.0888 - val_accuracy: 0.7
080
Epoch 20/20
125/125 [==============================] - 0s 1ms/step - loss: 1.0913 - accuracy: 0.7138 - val_loss: 1.0520 - val_accuracy: 0.7
120
79/79 [==============================] - 0s 1ms/step - loss: 1.1888 - accuracy: 0.6632
```

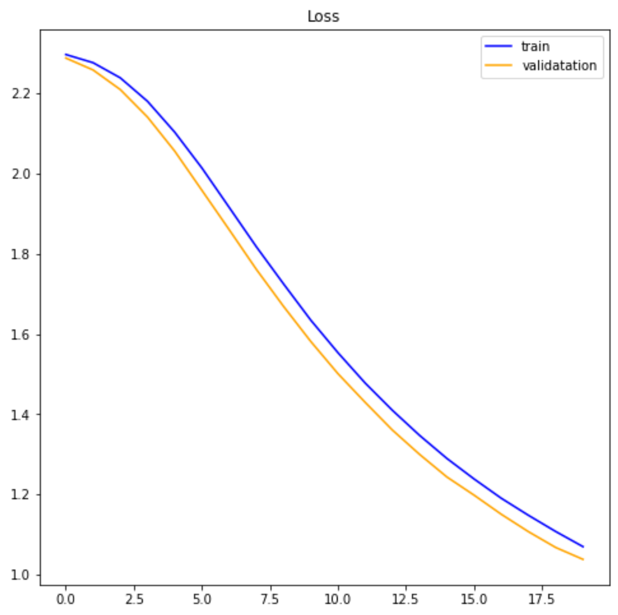


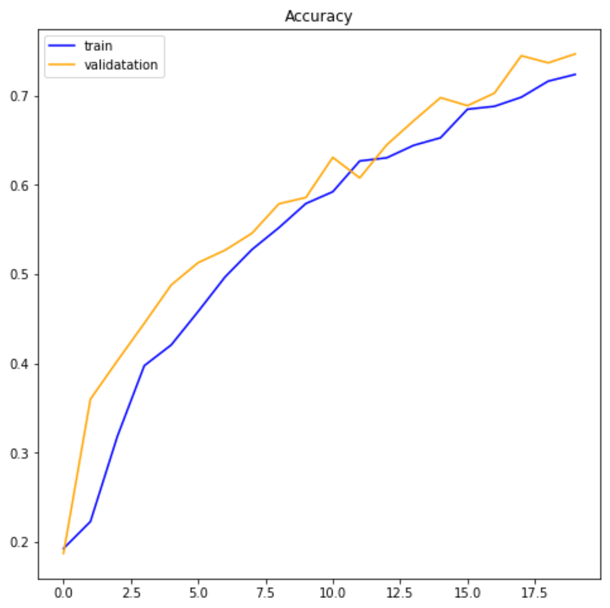


```
Test Accuracy is :  66.320
Train Accuracy is :  71.375
Train Confusion Matrix
[[470   0   5  16   4   8  11   0   8   0]
 [  0 551   2   5   0   0   0   2   4   0]
 [  9  47 368  30  11   0  29   3   6   0]
 [  4  31  15 454   1   1   3  18   8   4]
 [  0  13   1   0 301   0  12  10   0 108]
 [ 18  43   5 138  21  46   7  20 117  14]
 [  6  25  19   0   3   1 454   0   5   0]
 [  4  42   0   0   4   0   1 458   1  10]
 [ 22  88  15  94  10   1  11  14 214  13]
 [ 12  13   1   5  62   0   2 162   1 225]]
Test Confusion Matrix
[[193   0   0   8   2   3  10   0   3   0]
 [  0 282   2   2   0   0   1   0   0   0]
 [  8  39 173  28   2   0  18   7   1   0]
 [  0  10  10 213   0   0   1  16   3   1]
 [  2   9   1   0 161   0  13  12   0  77]
 [  8   6   6  98   5  21   6  23  38  10]
 [  6  14   7   1   5   2 189   0   1   0]
 [  1  31   2   1   3   0   0 209   2   8]
 [  8  33  10  62   9   0   2  17  90  11]
 [  2  12   1   4  42   0   2  52   2 127]]
#######################################################
```

2) False:

```
Epoch 19/20
125/125 [==============================] - 0s 2ms/step - loss: 1.1076 - accuracy: 0.7165 - val_loss: 1.0678 - val_accuracy: 0.7
370
Epoch 20/20
125/125 [==============================] - 0s 1ms/step - loss: 1.0700 - accuracy: 0.7240 - val_loss: 1.0383 - val_accuracy: 0.7
470
79/79 [==============================] - 0s 1ms/step - loss: 1.1730 - accuracy: 0.6792
```





```
Test Accuracy is :  67.920
Train Accuracy is :  72.400
Train Confusion Matrix
[[471   0   3   4   4  11  11   1  17   0]
 [  0 550   3   1   0   0   0   1   8   1]
 [ 11  45 342  31  19   0  39   3  13   0]
 [  6  29  22 404   4   3   2  17  45   7]
 [  1   9   0   0 354   0  11   3   0  67]
 [ 21  37  12  72  31  55  14  12 154  21]
 [  7  23   3   0   6   1 469   0   4   0]
 [  2  34   0   0  11   0   0 430   2  41]
 [ 19  68  14  36  16   1  13   6 290  19]
 [ 12  11   1   1  81   0   1  67   6 303]]
Test Confusion Matrix
[[192   0   0   3   2   5  11   0   6   0]
 [  0 281   1   1   0   0   2   0   2   0]
 [  9  33 164  21   4   0  29   7   8   1]
 [  0   8  18 187   0   0   1  17  18   5]
 [  0   8   1   0 197   0  10   3   0  56]
 [ 13   3  11  54  13  14   7  20  76  10]
 [  6  12   2   1  10   1 191   0   2   0]
 [  1  26   2   0   4   0   0 196   3  25]
 [ 10  23  13  23  11   0   3  11 129  19]
 [  4   8   1   2  58   0   0  20   4 147]]
##########################################################################
```

Here we tried Early Stopping.

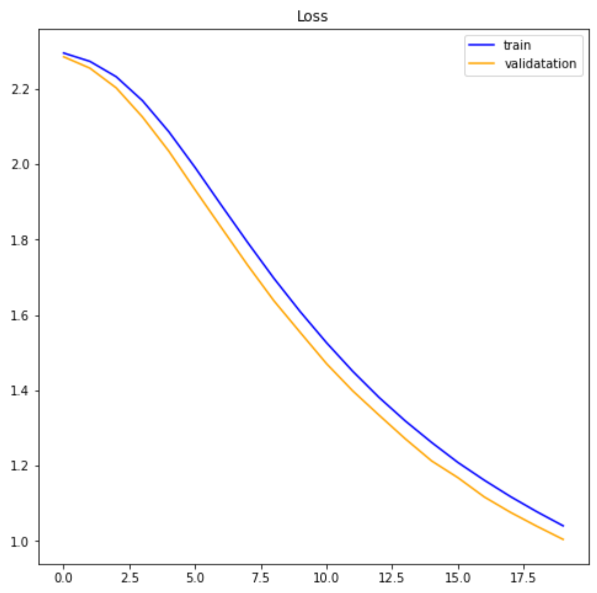
In many cases our model will stop training after it reach a converge or a local or global minimum. After that it should be stopped since learning on the same data with little update makes the model overfitted to that dataset. So, after a couple of epochs when there are small changes in loss our model should be stopped to prevent overfitting.

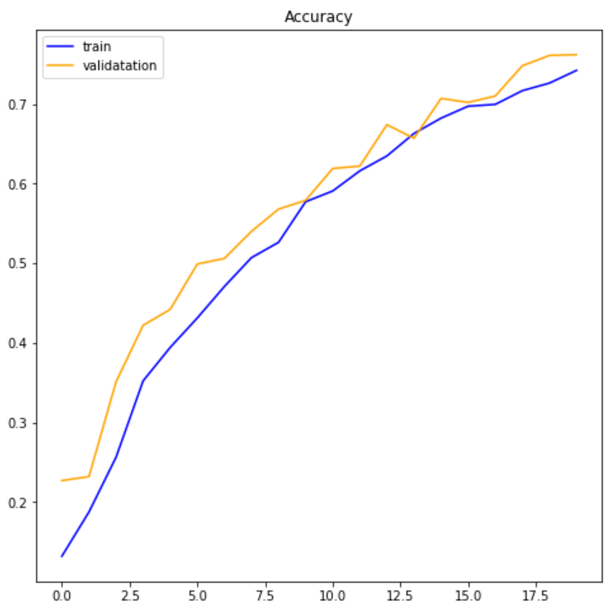
In this question, unfortunately, we could not see the early stopping result since it could not reach a convergence after 20 epoch and loss value still had big changes. To see the results, we should have increased the epochs, but we know the result, after few epochs in which loss had small changes, our model will stop the training to prevent overfitting.

5. Epochs:

1) 20 Epoch:

```
Epoch 19/20
125/125 [==============================] - 0s 2ms/step - loss: 1.0779 - accuracy: 0.7262 - val_loss: 1.0393 - val_accuracy: 0.7
610
Epoch 20/20
125/125 [==============================] - 0s 2ms/step - loss: 1.0405 - accuracy: 0.7423 - val_loss: 1.0044 - val_accuracy: 0.7
620
79/79 [==============================] - 0s 1ms/step - loss: 1.1435 - accuracy: 0.6952
```

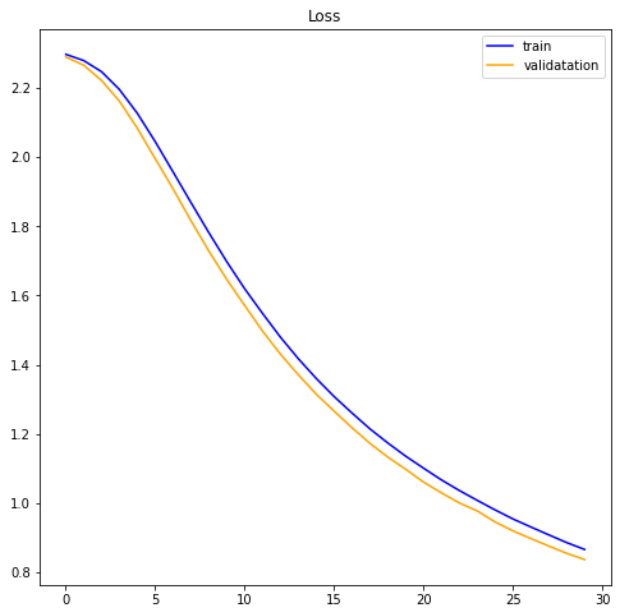


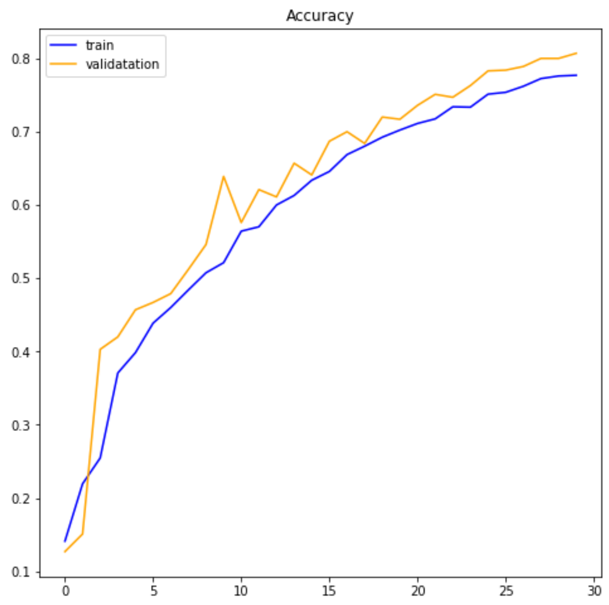


```
Test Accuracy is :  69.520
Train Accuracy is :  74.225
Train Confusion Matrix
[[489   0   1   3   2  14  12   0   1   0]
 [  0 549   4   4   0   4   0   1   1   1]
 [ 13  36 371  26   8   1  36   3   8   1]
 [  6  25  15 448   2   9   5  19   4   6]
 [  2   7   0   0 311   0  15   7   0 103]
 [ 26  24   4 106  23 192  20  21   0  13]
 [  7  17   2   0   3   5 479   0   0   0]
 [  4  27   0   0   5   0   0 447   3  34]
 [ 24  65  25  96   8  16  20  11 198  19]
 [ 12   8   1   4  50   0   2 121   1 284]]
Test Confusion Matrix
[[203   0   0   0   0   6  10   0   0   0]
 [  0 280   0   4   0   0   3   0   0   0]
 [ 11  25 169  21   2   1  30   9   7   1]
 [  1   8   6 205   0   7   3  18   2   4]
 [  6   8   0   0 167   0  10   8   0  76]
 [ 15   3   6  73   7  70  11  26   1   9]
 [  7   7   0   1   6   2 202   0   0   0]
 [  1  21   1   1   3   0   0 203   4  23]
 [ 10  22  15  58   9   9   3  16  83  17]
 [  4   6   1   3  37   2   0  33   2 156]]
#####################################################
```

2) 30 Epoch:

```
Epoch 29/30
125/125 [==============================] - 0s 2ms/step - loss: 0.8859 - accuracy: 0.7760 - val_loss: 0.8549 - val_accuracy: 0.8
000
Epoch 30/30
125/125 [==============================] - 0s 2ms/step - loss: 0.8662 - accuracy: 0.7770 - val_loss: 0.8370 - val_accuracy: 0.8
070
79/79 [==============================] - 0s 1ms/step - loss: 0.9802 - accuracy: 0.7364
```





```
Test Accuracy is :  73.640
Train Accuracy is :  77.700
Train Confusion Matrix
[[488   0   2   3   2  16   8   0   3   0]
 [  0 544   7   3   0   2   0   2   6   0]
 [ 10  19 406  24  10   0  23   3   8   0]
 [  4  14  21 446   1  13   3  16  14   7]
 [  2   4   1   0 357   0  11   4   1  65]
 [ 24  14   9  89  21 217  12  14  19  10]
 [  7  10   6   0   4   3 481   0   2   0]
 [  4  19   2   1   8   1   0 442   5  38]
 [ 20  34  27  54  16  10  11   3 288  19]
 [ 12   7   2   5  84   0   1  66   3 303]]
Test Confusion Matrix
[[208   0   0   0   1   6   4   0   0   0]
 [  0 279   1   1   0   0   2   0   4   0]
 [  8  11 202  17   2   1  23   7   4   1]
 [  1   4  13 203   0   7   1  12   8   5]
 [  1   5   1   0 198   0   9   2   0  59]
 [ 15   0   8  65  10  79   7  22   7   8]
 [  7   3   4   0   5   3 203   0   0   0]
 [  2  15   7   1   4   0   0 197   3  28]
 [ 10  12  18  40   9   2   2   9 125  15]
 [  4   5   1   3  53   2   0  24   5 147]]
##########################################################################
```

Here we are comparing the result of different Epochs.

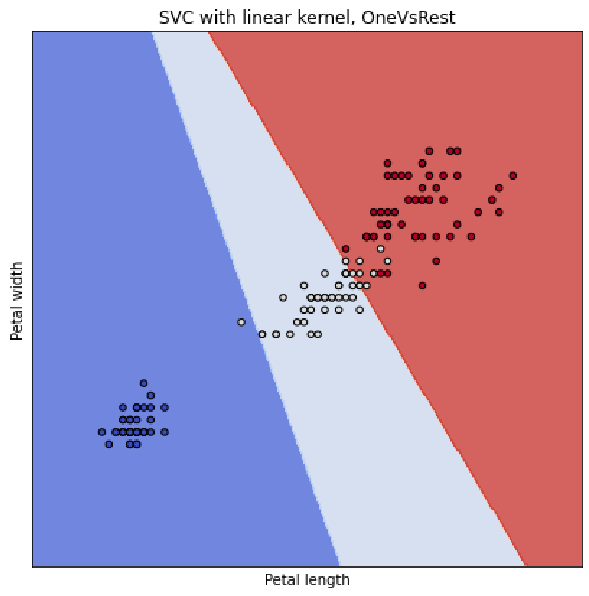
As it can be seen, for 30 epochs we had a better accuracy. This is because after 20 epochs our model did not reach a converge and it was still learning, so more epochs were needed to learn and reach a minimum.

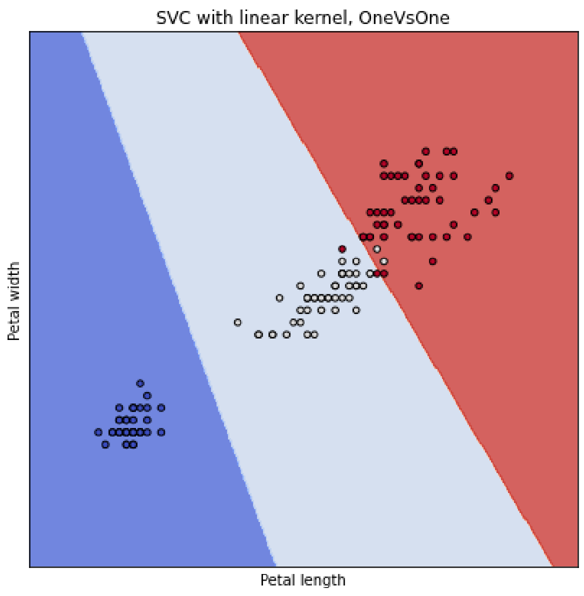
After 30 epochs, we can see our model still did not reached a minimum but it is going closer to it. We still need more epochs for the model to be trained perfectly but as we can see 30 epochs reached a better accuracy than 20 because of the fact mentioned above.

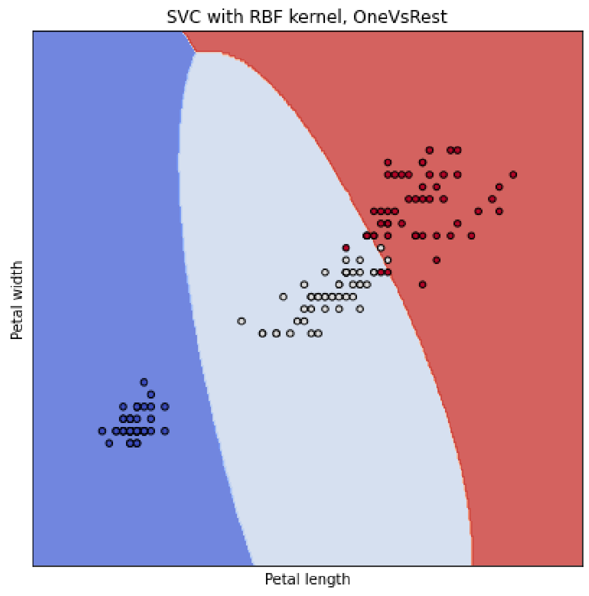
Question 9: Coding:

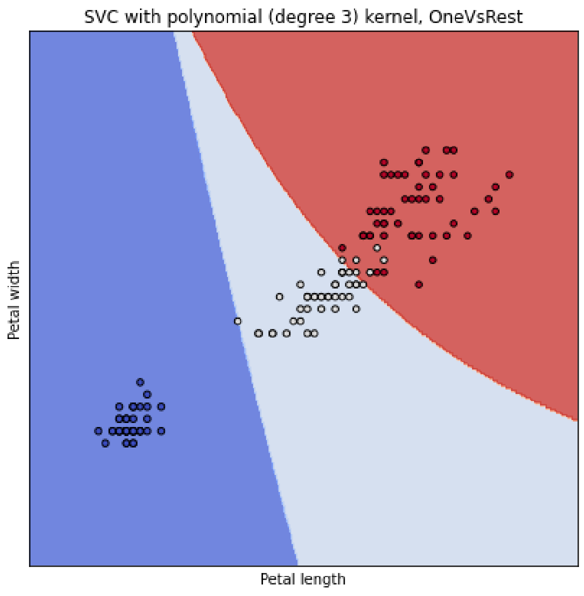
In this Question we are going to use Petal Length and Petal Width features of Iris dataset to Classify the dataset using different SVM kernels using different methods including (OneVsRest and OneVsOne). Then we are going to plot the areas devided using SVM classifier along with the accuracy and confusion matrix of each part:

To compare the kernels area we have the plot below:









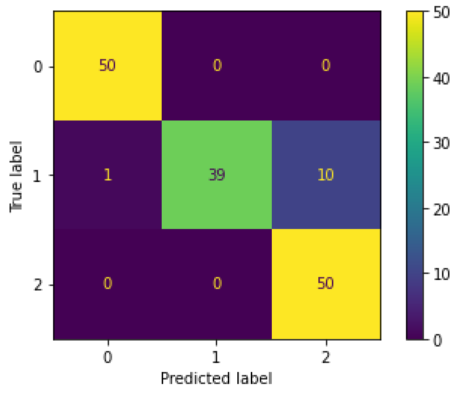
For Each one we have:

1)

```
SVC with linear kernel, OneVsRest
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        50
           1       1.00      0.78      0.88        50
           2       0.83      1.00      0.91        50

    accuracy                           0.93       150
   macro avg       0.94      0.93      0.93       150
weighted avg       0.94      0.93      0.93       150
```

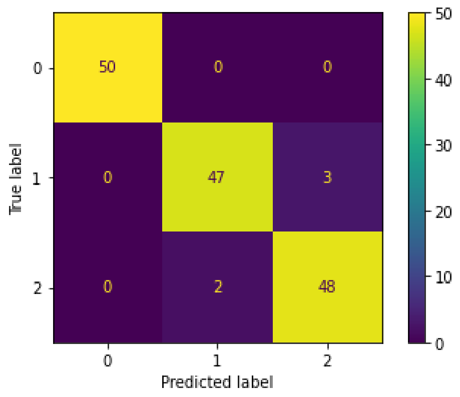


2)

```
SVC with linear kernel, OneVsOne
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        50
           1       0.96      0.94      0.95        50
           2       0.94      0.96      0.95        50

    accuracy                           0.97       150
   macro avg       0.97      0.97      0.97       150
weighted avg       0.97      0.97      0.97       150
```

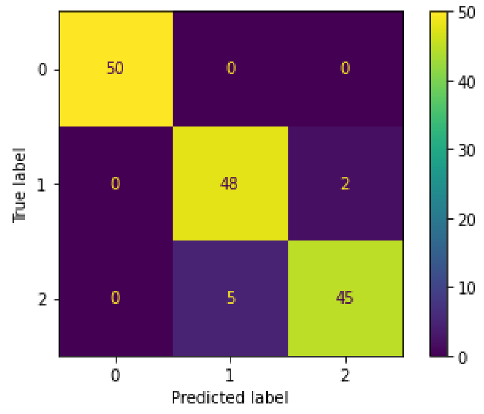


3)

```
SVC with RBF kernel, OneVsRest
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        50
           1       0.91      0.96      0.93        50
           2       0.96      0.90      0.93        50

    accuracy                           0.95       150
   macro avg       0.95      0.95      0.95       150
weighted avg       0.95      0.95      0.95       150
```

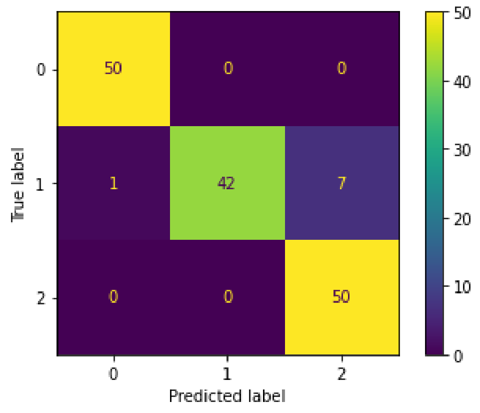


4)

```
SVC with polynomial (degree 3) kernel, OneVsRest
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        50
           1       1.00      0.84      0.91        50
           2       0.88      1.00      0.93        50

    accuracy                           0.95       150
   macro avg       0.95      0.95      0.95       150
weighted avg       0.95      0.95      0.95       150
```



```
##################################
```

We can see OneVsOne Linear kernel achieved best accuracy. Even in the fist plot it is obvious that Linear with One vs One had the lowest error in classification. All the confusion matrix plots and accuracies are shown. Linear Kernel Using one vs one method reached an accuracy of 97% which was the highest of all.

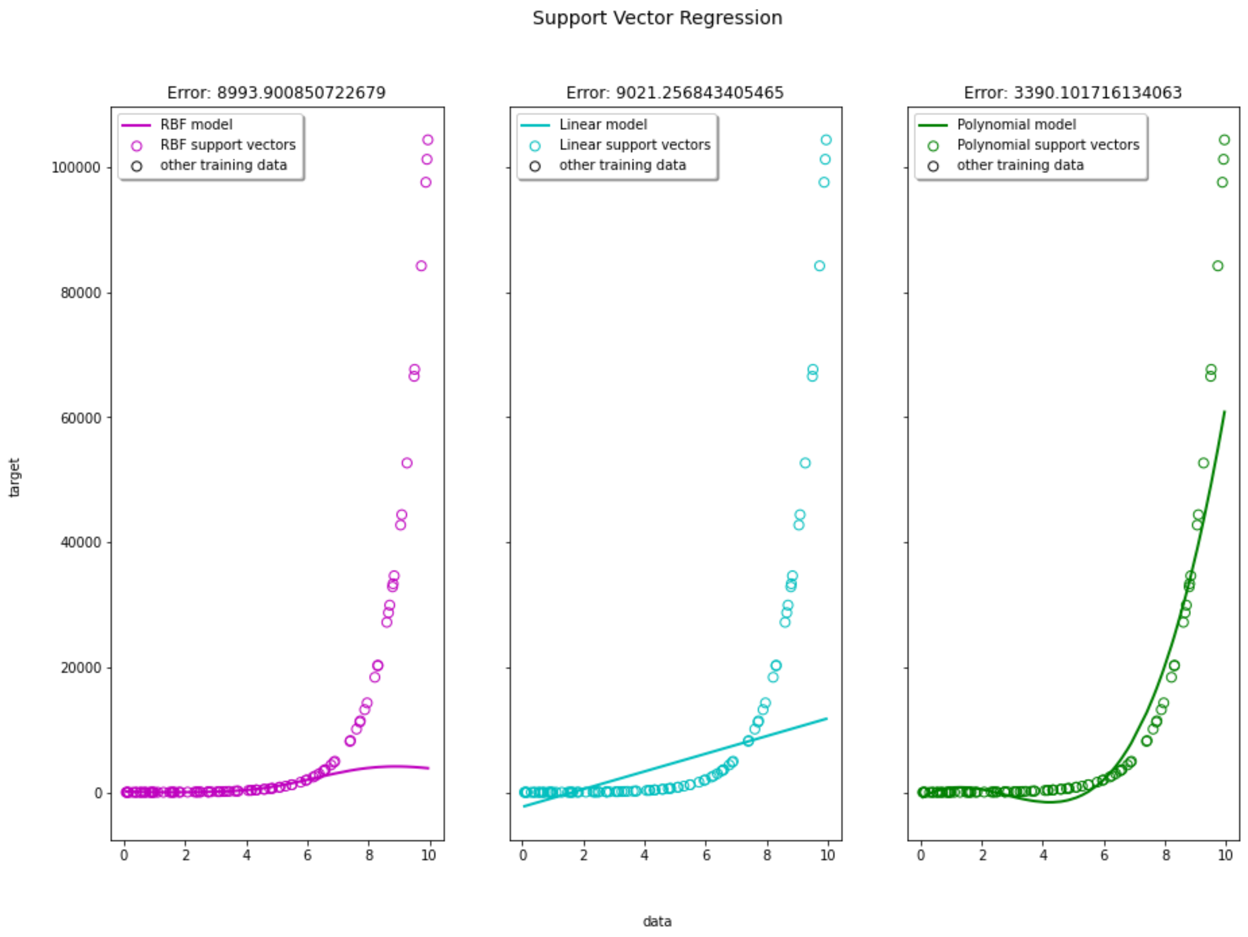
Question 10: Coding:

1) S

Here first we are going to produce 100 point between 0 and 10 with respect to:

$$y = 5e^{x} + 3; x \in [0, 10]$$

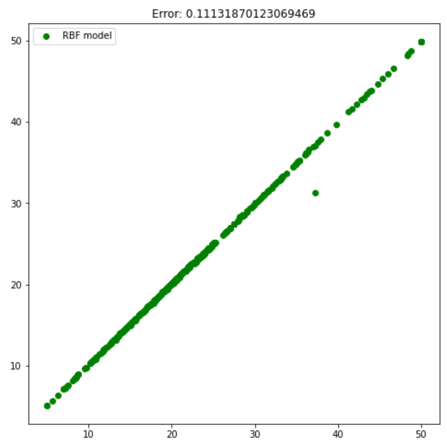
And then run 3 SVM regression with polynomial third degree, rbf, and exponential kernels. Our result is like below:



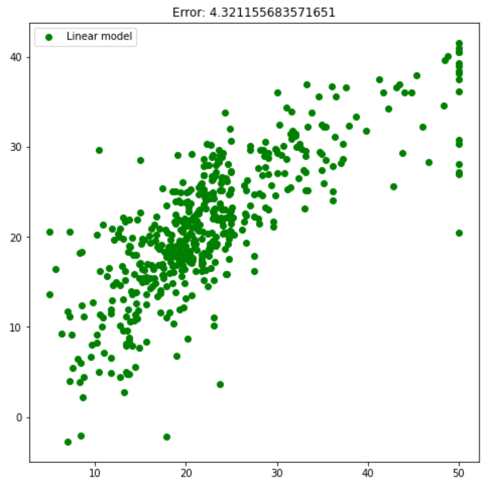
Errors are come as title for each plot. Polynomial third degree had the best regressor with lowest error among all.

2) Here we do the same thing as in part 1 but on the Boston housing dataset. We are going to predict the price of each house using a svm regression with rbf, polynomial third degree, and exponential kernels. Our results are:

For RBF:



For Linear:



For Polynomial with degree 3: It took a lot of time solving a polynomial problem. So, results for this kernel could not be shown.

SVM for a dataset consisting many samples is not a good idea. Since SVM should solve a quadratic optimization problem it took a lot of time for rbf or polynomial to be solved. So, with Boston housing dataset, it took a lot of time fitting a polynomial third degree on the dataset. But as it can be seen a RBF kernel SVR fitted well.

All Errors are Mean Absolute Error which are on the title for each plot.