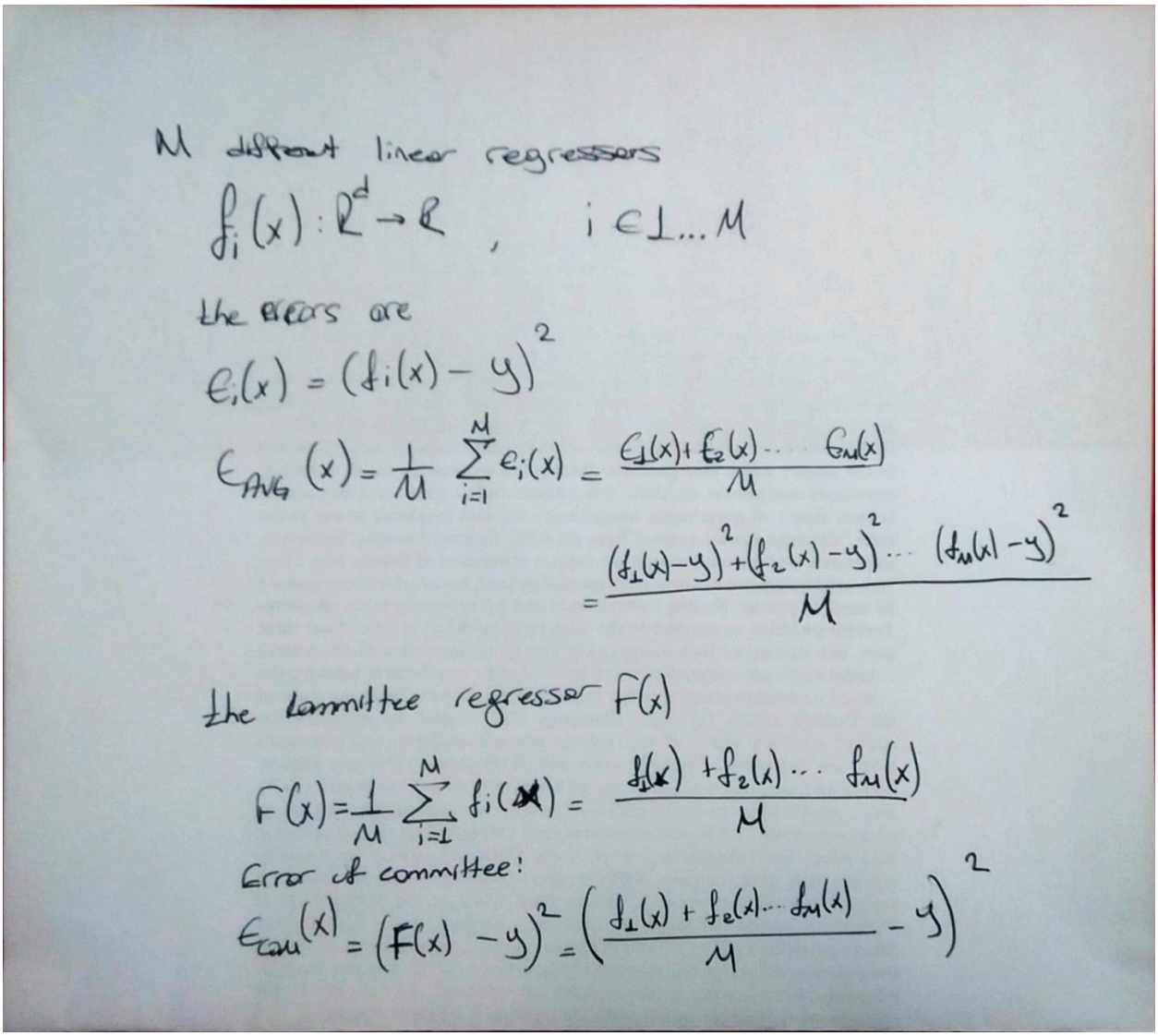
M. Yusa Erguven - 19260

Machine Learning HW2

Question 1 - Ensembles

I made my calculations on paper, so that I am putting the scanned versions.



Trying to prove $\Rightarrow E_{con}(x) \leq E_{AVG}(x)$

$$E_{con}(x) \Rightarrow (F(x) - y)^2$$
$$= \left(\frac{1}{M}\sum_{i=1}^{M} f_i(x) - y\right)^2$$

$$E_{AVG}(x) \Rightarrow \frac{1}{M}\sum_{i=1}^{M}\left(f_i(x) - y\right)^2$$

According to Jensen's inequality:

$$g\left(\frac{1}{n}\sum_{i=1}^{n} z_i\right) \leq \frac{1}{n}\sum_{i=1}^{n} g(z_i)$$

Consider z_i and $g(z)$ as follows:

$z_i = f_i(x)$ and also $n = M$

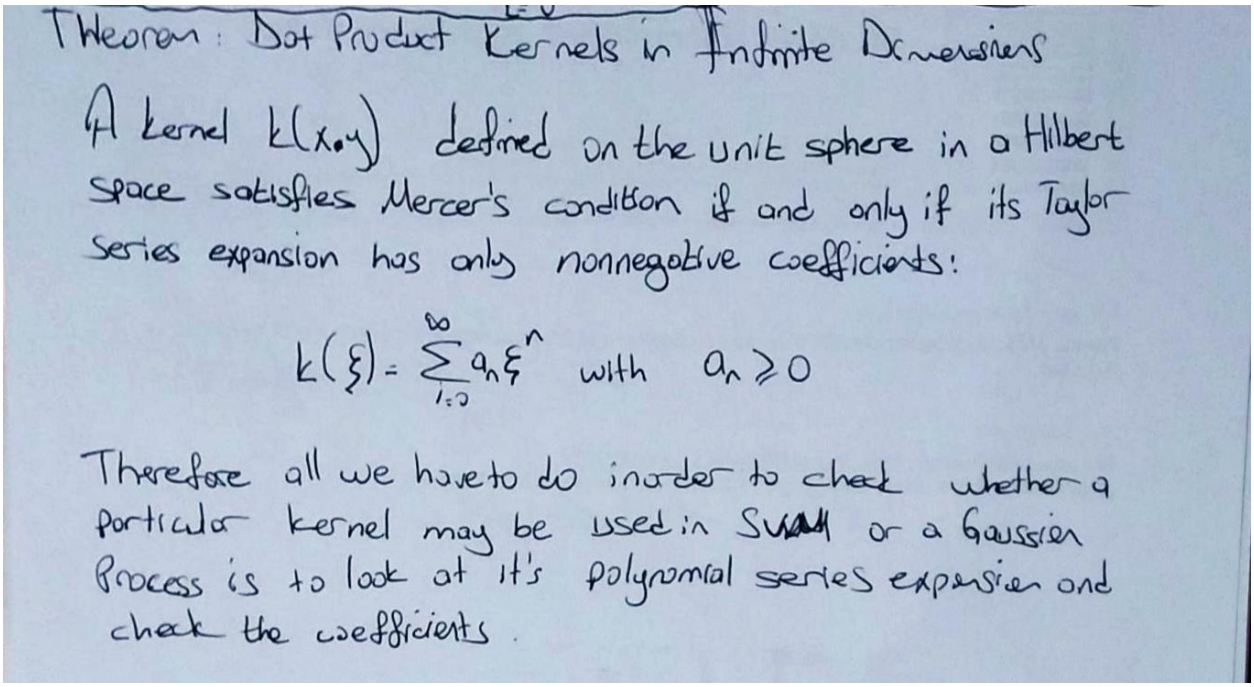
$g(z) = (z-y)^2$

then replace them in our inequality $\Rightarrow E_{con}(x) \leq E_{AVG}(x)$

$$\underbrace{\left(\left(\frac{1}{M}\sum_{i=1}^{M} f_i(x)\right) - y\right)^2}_{g\left(\frac{1}{M}\sum_{i=1}^{M} z_i\right)} \leq \underbrace{\frac{1}{M}\sum_{i=1}^{M}\left(f_i(x) - y\right)^2}_{\frac{1}{M}\sum_{i=1}^{M} g(z_i)}$$

Question 2 - Kernels

Even though I didn't get exactly what we are supposed to do in this question, I read about it and according to my findings, Schoenberg proved his theorem on Dot Product Kernels in Infinite Dimensions which is as follows and relates to our topic:



Explicitly not computing the dot product of vectors in this case is actually related to kernel trick, which helps us to avoid it.

If the dot product is defined in an infinite dimensional space Taylor series expansion of kernel function k as a function of <x1, x2> should have no negative coefficients.

Question 3 - Letter recognition using SVM

a) Linear Kernel

First I split my data to to %20 test, %80 train datasets. Then again split the train set to %25 validation, %75 train set. After that, for the hyperparameter optimization part, with the C values = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100], I trained my SVM classifier and found the optimum C value. The code below shows this cross validation process.

Linear Kernel

```
# C is the SVM regularization parameter
C_values = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100]

#Let's create a validation set
X_train_val, X_test_val, y_train_val, y_test_val = train_test_split(X_train, y_train, test_size = 0.25)



accuracies = []
for C in C_values:
    svc = SVC(kernel ='linear', C = C)
    svc.fit(X_train_val, y_train_val)
    y_pred_val = svc.predict(X_test_val)
    print(confusion_matrix(y_test_val,y_pred_val))
    print(classification_report(y_test_val,y_pred_val))
    accuracies.append(accuracy_score(y_test_val,y_pred_val))
```

The accuracy results are as follows:

`0.9554140127388535,`

`0.9681528662420382,`

`0.9745222929936306,`

`0.9840764331210191,`

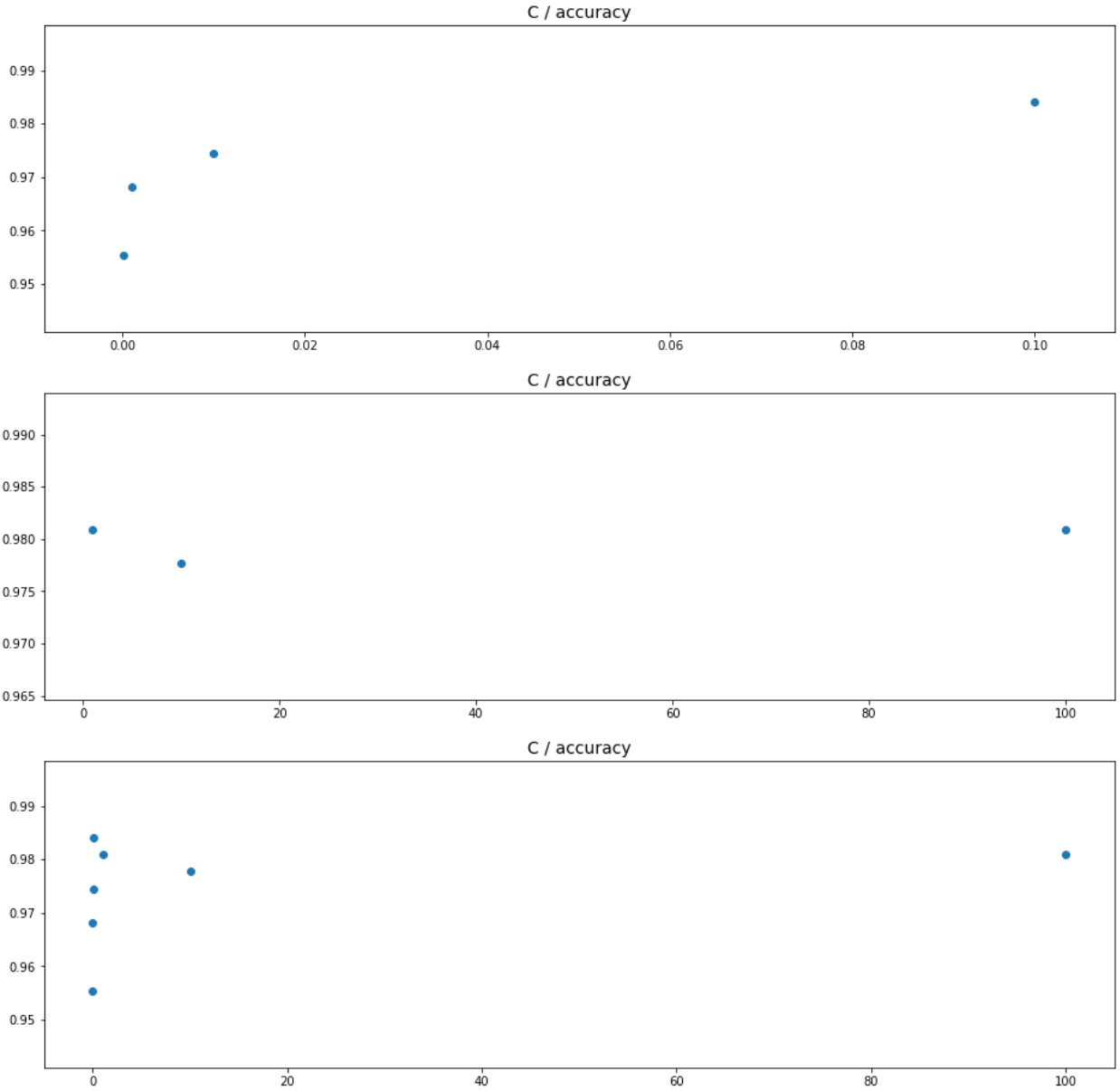
`0.9808917197452229,`

`0.9777070063694268,`

`0.9808917197452229`

I plotted the accuracies with respect to C values. For the sake of simplicity I have 3 graphs, since first 4 values are too close to each other, it is hard to interpret the overall graph.

The first graph shows the C values: 0.0001, 0.001, 0.01, 0.1
The second graph shows the C values: 1, 10, 100
The last one shows all of them: 0.0001, 0.001, 0.01, 0.1, 1, 10, 100



I re-trained a model using the best C value which is 0.01 with accuracy 0.9840764331210191 and then run the model on test set with the code below. Results are as follows:

```
svclassifier = SVC(kernel='linear', C = 0.1)
svclassifier.fit(X_train, y_train)
y_pred = svc.predict(X_test)
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

```
[[152   4]
 [  6 152]]
              precision    recall  f1-score   support

           0       0.96      0.97      0.97       156
           1       0.97      0.96      0.97       158

    accuracy                           0.97       314
   macro avg       0.97      0.97      0.97       314
weighted avg       0.97      0.97      0.97       314
```

```
print(accuracy_score(y_test,y_pred))
```

```
0.9681528662420382
```

The accuracy is: 0.9681528662420382

The decision values of test set is at the file named: **Decisions_Linear_Kernel.txt**.

a) Radial Basis Function Kernel

I again split my data to to %20 test, %80 train datasets. Then again split the train set to %25 validation, %75 train set. After that, for the hyperparameter optimization part, with the C values = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100] and gamma values = [0.0625, 0.125, 0.25, 0.5, 1], I applied cross validation on my SVM classifier and found the optimum C value. The code below shows this cross validation process.

Radial Basis Function Kernel

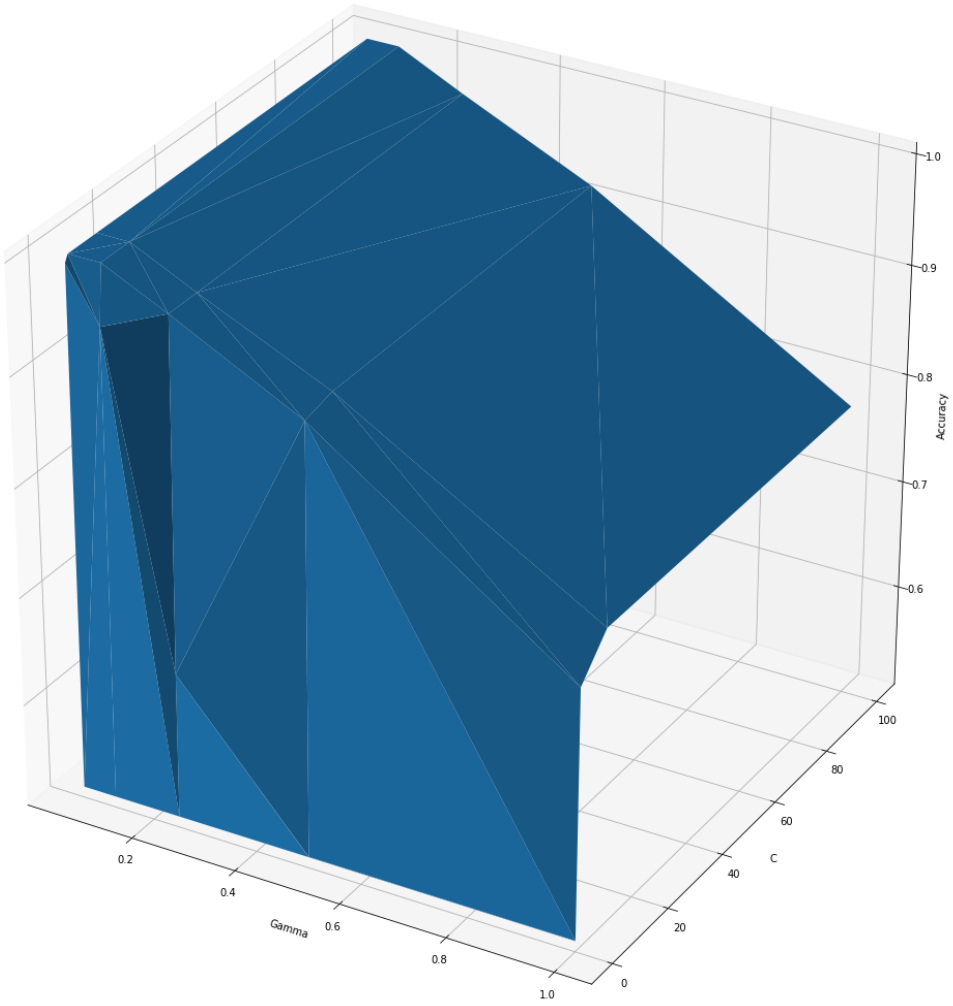
```
#Let's create a validation set
X_train_val, X_test_val, y_train_val, y_test_val = train_test_split(X_train, y_train, test_size = 0.25)
```

```
C_values = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100]
gamma_values = [0.0625, 0.125, 0.25, 0.5, 1]

accuracies = {}
df

for C in C_values:
    for gamma in gamma_values:
        # Create a SVC classifier using an RBF kernel
        svc = SVC(kernel='rbf', random_state=0, gamma=gamma, C=C)
        svc.fit(X_train_val, y_train_val)
        y_pred_val = svc.predict(X_test_val)
        accuracy = accuracy_score(y_test_val,y_pred_val)
        accuracies[f"{C}_{gamma}"] = accuracy
        print(f"C: {C}\t Gamma: {gamma}\t Accuracy: {accuracy}")
```

Accuracy results are as follows:

	C	gamma	accuracy
0	0.0001	0.0625	0.515924
1	0.0001	0.125	0.515924
2	0.0001	0.25	0.515924
3	0.0001	0.5	0.515924
4	0.0001	1	0.515924
5	0.001	0.0625	0.515924
6	0.001	0.125	0.515924
7	0.001	0.25	0.515924
8	0.001	0.5	0.515924
9	0.001	1	0.515924
10	0.01	0.0625	0.515924
11	0.01	0.125	0.515924
12	0.01	0.25	0.515924
13	0.01	0.5	0.515924
14	0.01	1	0.515924
15	0.1	0.0625	0.993631
16	0.1	0.125	0.945860
17	0.1	0.25	0.649682
18	0.1	0.5	0.515924
19	0.1	1	0.515924
20	1	0.0625	1.000000
21	1	0.125	1.000000
22	1	0.25	0.971338
23	1	0.5	0.910828
24	1	1	0.745223
25	10	0.0625	1.000000
26	10	0.125	1.000000
27	10	0.25	0.971338
28	10	0.5	0.917197
29	10	1	0.777070
30	100	0.0625	1.000000
31	100	0.125	1.000000
32	100	0.25	0.971338
33	100	0.5	0.917197
34	100	1	0.777070

I plotted the accuracies with respect to C values and gamma values.



Then found the best accuracy values and corresponding C and gamma values.

In [136]:
```
def getBestAcc(df_acc):
    bests = []
    max = 0
    for row in df_acc.itertuples(index=True, name='Pandas'):
        if getattr(row, "accuracy") > max:
            bests = []
            toApp = [getattr(row, "C"), getattr(row, "gamma"), getattr(row, "accuracy")]
            bests.append(toApp)
            max = getattr(row, "accuracy")
        elif getattr(row, "accuracy") == max:
            toApp = [getattr(row, "C"), getattr(row, "gamma"), getattr(row, "accuracy")]
            bests.append(toApp)
    return bests
```

In [137]:
```
getBestAcc(df)
```

Out[137]:
```
[['1', '0.0625', 1.0],
 ['1', '0.125', 1.0],
 ['10', '0.0625', 1.0],
 ['10', '0.125', 1.0],
 ['100', '0.0625', 1.0],
 ['100', '0.125', 1.0]]
```

So, After tuning, we have several optimum C and gamma values

C: 1, Gamma: 0.0625, Accuracy: 1.0
C: 1, Gamma: 0.125, Accuracy: 1.0
C: 10, Gamma: 0.0625, Accuracy: 1.0
C: 10, Gamma: 0.125, Accuracy: 1.0
C: 100, Gamma: 0.0625, Accuracy: 1.0
C: 100, Gamma: 0.125, Accuracy: 1.0

So Let's pick one of them and test our data
Picked:
C: 10, Gamma: 0.0625

In [139]:
```
gamma = 0.0625
C = 10
# Create a SVC classifier using an RBF kernel
svc = SVC(kernel='rbf', random_state=0, gamma=gamma, C=C)
svc.fit(X_train, y_train)
y_pred = svc.predict(X_test)
accuracy = accuracy_score(y_test,y_pred)
accuracies[f"{C}_{gamma}"] = accuracy
print(f"C: {C}\t Gamma: {gamma}\t Accuracy: {accuracy}")
```

```
C: 10    Gamma: 0.0625    Accuracy: 1.0
```

Resulting Accuracy: 1.0

The decision values of test set is at the file named: **Decisions_RBF_Kernel.txt**.

The accuracy for Linear Kernel was: 0.9681528662420382 and the accuracy for Radial Basis Function Kernel is: 1.0 which clearly states the SVM with RBF kernel is better.

This means our prediction gets better when we use non-linear model which is SVM with RBF kernel. Since SVM with Linear kernel is linear and tries to separate the labels with linear line and RBF on the contrary uses curves to separate it, it is likely to have a better performance with RBF. The kernel defines the function class that we work with. The squared exponential kernel such as RBF defines a function space that is a lot larger than that of the linear kernel. So that it is more flexible and with the risk of overfitting, it fits the data better than linear kernel. That’s why prediction of RBF is better than Linear kernel.