Lab Course Machine Learning
Exercise 9

Exercise 1: A spam filter using SVM

Part A:

Function name : svm_format
Parameter : data to be converted to svm format

This function is used to convert the data into the format accessible by libsvm.

```
def svm_format(data):
    data = np.array(data)
    file = open("d3.File","w")
    for i in data:
        if i[-1] == 1:
            file.write("+1 ")
        else:
            file.write("-1 ")
        for j in range(len(i)-2):
            if i[j] != 0:
                file.write(str(j+1)+":"+str(i[j])+" ")

        file.write("\n")
    file.close()
```

Once the file is created, it's checked using "checkdataset.py" for its correctness using the following prompt command.

python checkdataset. py d3.File

For splitting the data into train,validation, the following command was used in the python console to use the file "subset.py"

python subset. py d3.File 900 *Test. File Train. File*

python subset. py Train. File 700 *Val. File Train. File*

Build a spam filter using libsvm:

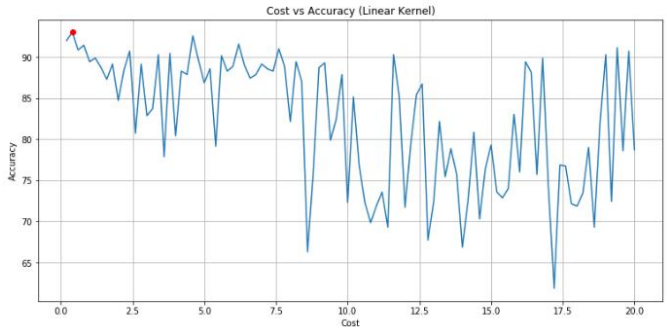
To build the spam filter using libsvm, the files Train.file, Val.file and Test.file were used for training validation and testing respectively. In this exercise, I've have created a model with Linear kernel and radial kernel for creating a better one. The range of hyper parameter 'C' that I have used is from 0.2 to 20 with an interval of 0.2.

```
train_y,train_x = svm_read_problem("Train.File")
val_y,val_x = svm_read_problem("Val.File")
test_y,test_x = svm_read_problem("Test.File")
crange = np.linspace(0.2,20,100)
f1score = []
accuracy_pred = []
mse_pred = []
bestmodel = None
MSE = 0
for i in crange:
    print("C: ",round(i,2))
    m = svm_train(train_y,train_x,'-c '+str(i))
    pred_label, pred_acc, pred_val = svm_predict(val_y,val_x,m )
    f1score.append(f1_score(val_y, pred_label))
    accuracy_pred.append(pred_acc[0])
    mse_pred.append(pred_acc[1])

    if bestmodel == None or MSE > pred_acc[1]:
        bestmodel = m
        MSE = pred_acc[1]
```

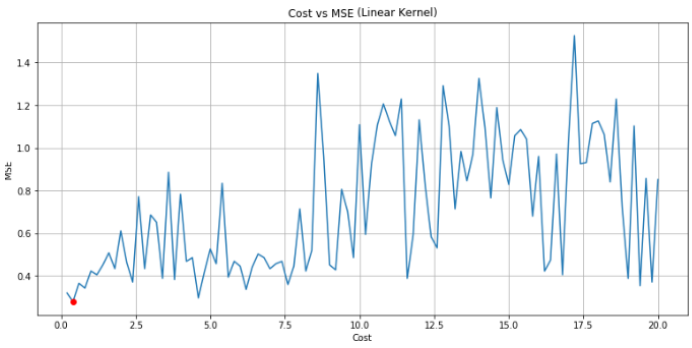
Linear Kernel

Best Cost (C) : 0.4
Kernel: Linear
Accuracy: 92.11
MSE: 0.316

Cost vs Accuracy



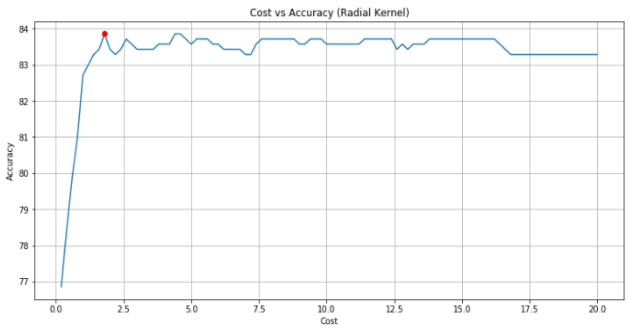
Cost vs MSE



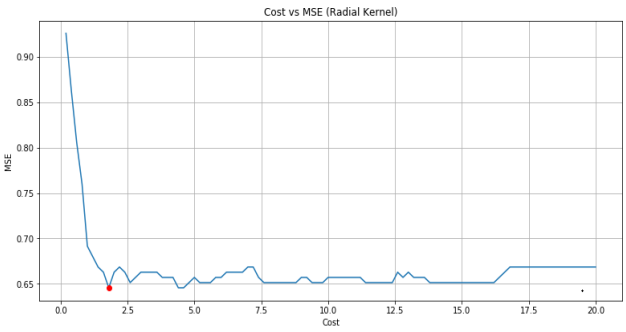
Radial Kernel

Best Cost (C): 1.8
Kernel: Radial
Accuracy: 82.11
MSE: 0.716

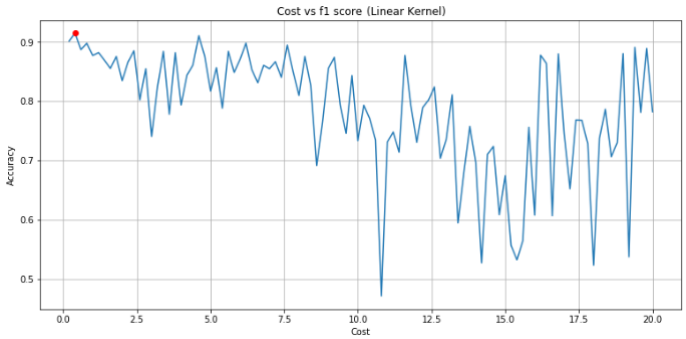
Cost vs Accuracy



Cost vs MSE



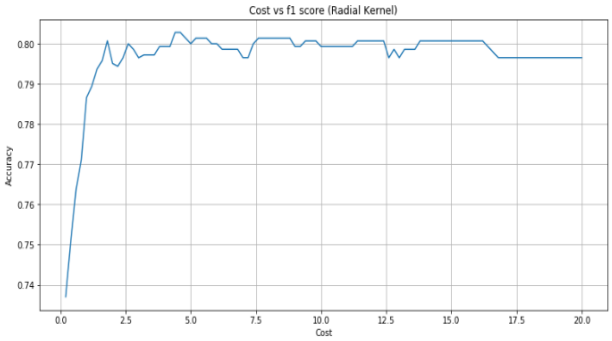
Cost vs f1 score



Prediction on test set with optimum hyperparameters

	Ground Truth	Prediction
35	1.0	1.0
598	-1.0	-1.0
248	1.0	1.0
264	1.0	1.0
574	-1.0	-1.0
589	-1.0	-1.0
78	1.0	1.0
739	-1.0	-1.0
548	-1.0	-1.0
666	-1.0	-1.0
70	1.0	1.0
462	-1.0	-1.0
325	1.0	1.0
44	1.0	1.0
421	-1.0	-1.0
629	-1.0	-1.0
466	-1.0	-1.0
818	-1.0	-1.0
768	-1.0	-1.0
164	1.0	1.0

Cost vs f1 score



Prediction on test set with optimum hyperparameters

	Ground Truth	Prediction
21	1.0	1.0
272	1.0	1.0
142	1.0	1.0
0	1.0	1.0
755	-1.0	-1.0
274	1.0	1.0
169	1.0	1.0
872	-1.0	1.0
663	-1.0	-1.0
885	-1.0	-1.0
446	-1.0	-1.0
721	-1.0	-1.0
581	-1.0	-1.0
345	1.0	1.0
52	1.0	1.0
648	-1.0	-1.0
135	1.0	1.0
575	-1.0	1.0
394	-1.0	-1.0
384	-1.0	-1.0

From the above comparisions, it can be seen that the SVM with Linear Kernel performs better compared to the model with Radial kernel

PART B – Using SVC (sklearn):

The dataset is imported using the following command and after preprocessing, it is split into train and test split.

```
Data = pd.read_csv("SMSSpamD2/SMSSpamCollection", sep="\t", header=None)
Data.columns = ['Type', 'Message']
Data.head()
```

	Type	Message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

To encode the type of message, I've used pd.get_dummies() and the following is the command for it.

```
Data['Type'] = pd.get_dummies(Data.Type)
```

Function name : traintestsplit
Parameter : data to split and size of train data

```
def traintestsplit(data, splitize):

        train = data.sample(frac = splitize)
        test = data.drop(train.index)
        train.reset_index(inplace = True)
        test.reset_index(inplace = True)
        return train, test
```

Once the dataset is split to train and test set, a set of stop words is made for pre-processing the messages.

```
stopwordlist = stopwords.words("english")
stopwords = set(stopwordlist)
```

To streamline the flow of the process, a pipeline is created with Vectorizer and classifier. The vectorizer includes TFIDF for converting the text in terms of frequency for feature estimation and the SVC classifier is used for classifying the message into ham or spam

```
Pipe = Pipeline([
    ("vectorizer", TfidfVectorizer(stop_words = stopwords, lowercase = True)),
    ("classifier", SVC())
])
```

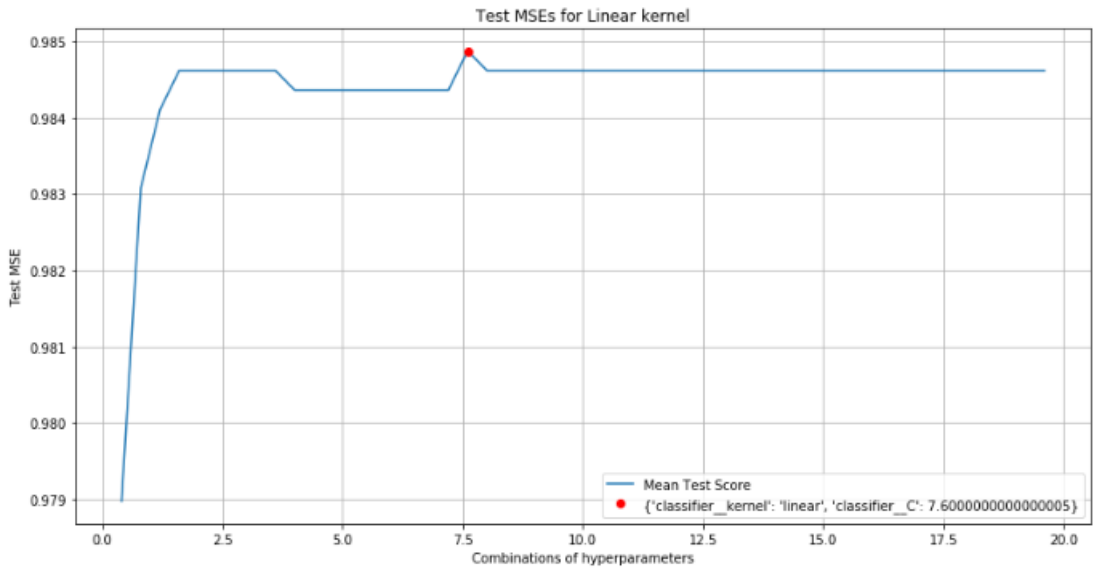
Once the pipeline is set, a dictionary of parameters is initialized for grid search.

```
param_grid = {
    "classifier__C":np.arange(0.4, 20, 0.4),
    "classifier__kernel":["rbf", "linear"],
}
```

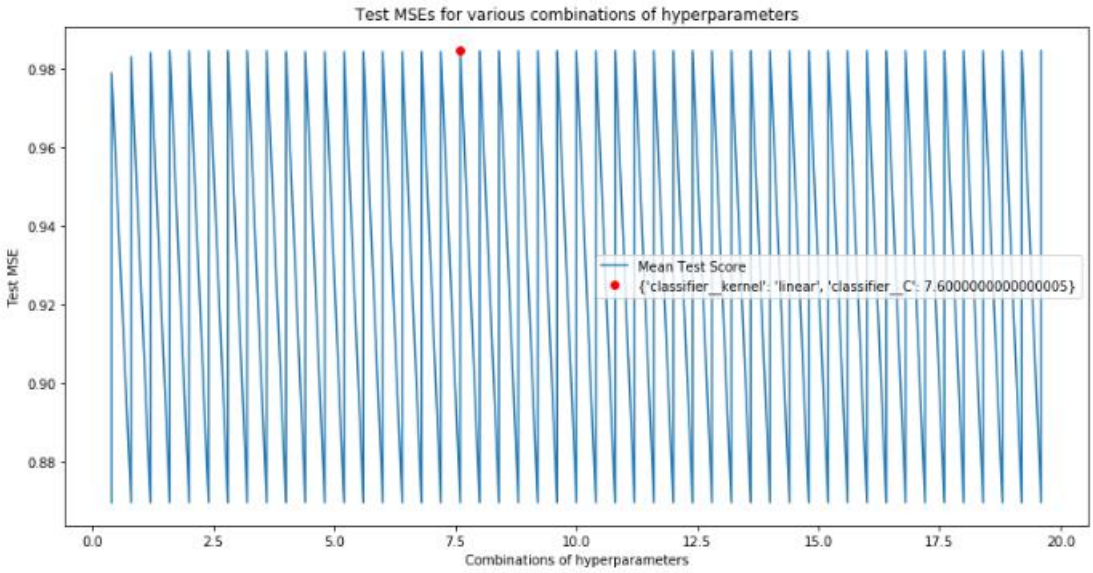
To perform grid search with the initialized hyper parameters and cross validation of 5 folds.

```
model = GridSearchCV(Pipe, cv = 5, n_jobs = 2, param_grid = param_grid)

model.fit(xTrain, yTrain)
```

	Mean Test Score	Mean Train Score	Cost	Kernel
1	0.978974	0.983077	0.4	linear
3	0.983077	0.993910	0.8	linear
5	0.984103	0.998333	1.2	linear
7	0.984615	0.998910	1.6	linear
9	0.984615	0.999487	2	linear
11	0.984615	0.999615	2.4	linear
13	0.984615	0.999808	2.8	linear
15	0.984615	1.000000	3.2	linear
17	0.984615	1.000000	3.6	linear
19	0.984359	1.000000	4	linear
21	0.984359	1.000000	4.4	linear
23	0.984359	1.000000	4.8	linear
25	0.984359	1.000000	5.2	linear
27	0.984359	1.000000	5.6	linear
29	0.984359	1.000000	6	linear
31	0.984359	1.000000	6.4	linear
33	0.984359	1.000000	6.8	linear
35	0.984359	1.000000	7.2	linear
37	0.984872	1.000000	7.6	linear
39	0.984615	1.000000	8	linear



```
Best Model:

Kernel:  linear
Cost C:  7.6000000000000005
MSE Test:  0.9849

[]
```



Prediction on test set:

```
ypred = Pipe.predict(xTest)
```

MSE: 0.019

	precision	recall	f1-score	support
ham	0.98	0.89	0.93	238
spam	0.98	1.00	0.99	1434
micro avg	0.98	0.98	0.98	1672
macro avg	0.98	0.94	0.96	1672
weighted avg	0.98	0.98	0.98	1672

Predictions

	Ground Truth	Prediction
0	spam	spam
1	ham	ham
2	spam	spam
3	spam	spam
4	ham	ham
5	ham	ham

Exercise 2 – Using Decision Tree classifier:

Initializing classifier, vectorizer and creating a pipeline for the same

```
treeclassifier = DecisionTreeClassifier()

Pipe = Pipeline([
    ("vectorizer", TfidfVectorizer(stop_words = stopwords, lowercase = True)),
    ("classifier", DecisionTreeClassifier())
])
```

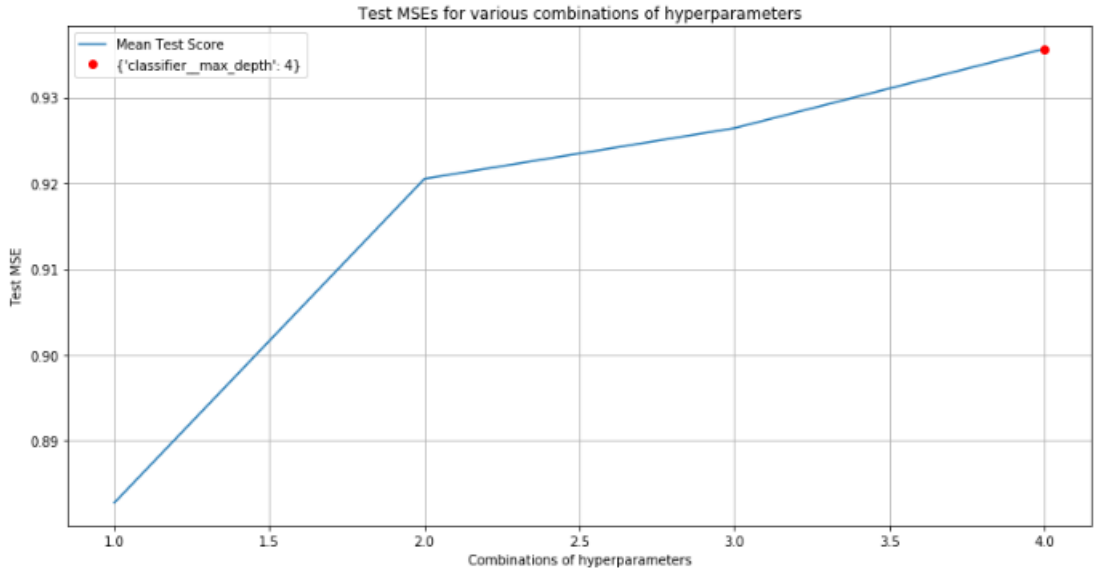
Creating a parameter grid with maximum depth of the tree with range between 1 and 5

```
param_grid = {
    "classifier__max_depth":np.arange(1, 5, 1)
}
```

```
treemodel = GridSearchCV(Pipe, cv = 5, n_jobs = 3,param_grid=param_grid)

treemodel.fit(xTrain, yTrain)
```

	Mean Test Score	Mean Train Score	Depth
0	0.882821	0.887500	1
1	0.920513	0.927115	2
2	0.926410	0.941859	3
3	0.935641	0.949231	4

```
Best Model:

Depth:  4
MSE Test:  0.9356

[]
```



Prediction on test set

```
ypred = Pipe.predict(xTest)
```

MSE: 0.074

	precision	recall	f1-score	support
ham	0.81	0.62	0.70	238
spam	0.94	0.98	0.96	1434
micro avg	0.93	0.93	0.93	1672
macro avg	0.87	0.80	0.83	1672
weighted avg	0.92	0.93	0.92	1672

Predictions

	Ground Truth	Prediction
0	spam	spam
1	ham	ham
2	spam	spam
3	spam	spam
4	ham	ham
5	ham	ham

From the above results and predictions by the models SVC and Decision Tree, it can be seen that the model generated with SVC classifier performs better than the Decision tree classifier. The precision, recall and f1-score of SVC model is better compared to the Decision tree model. Also, the MSE of SVC is 0.019 whereas, the MSE of decision tree classifier is 0.074. So, it can be concluded that SVC performs better than Decision tree classifier.