CS 312: Artificial Intelligence Laboratory

Task 6: Support Vector Machine Classifier

Group 17

190030026 [Mayank Mittal]

190030007 [Ayush Gupta]

1.Library used :-

- **Pandas**:- pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. To know more visit here.

- **Sklearn**:- Scikit-learn (Sklearn) is **the most useful and robust library for machine learning in Python**. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistent interface in Python. To know more visit here.

- **Tqdm**:- Instantly make your loops show a smart progress meter - just wrap any iterable with tqdm(iterable). To know more visit here.

- **SVC**:- The objective of a Linear SVC (**Support Vector Classifier**) is to fit the data you provide, returning a "best fit" hyperplane that divides, or categorizes, your data. From there, after getting the hyperplane, you can then feed some features to your classifier to see what the "predicted" class is. To know more visit here.

- **Matplotlib**:- Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. To know more visit here.

2.Detail of SVM package:-

- **StandardScaler**:-
  - Imported from sklearn.preprocessing
  - For normalization of data and scaling

- **Train_test_split:-**
  - Imported from sklearn.model_selection
  - Splits the data in train and test sets in given proportion

- **Warnings**:-
  - Imported warnings
  - To ignore warnings for clean board

- **SVC :-**
  - Import from sklearn.svm
  - Support vector model from sklearn

- **fit_transform() method:-**
  - Fits the model on input data (x_train and y_train)

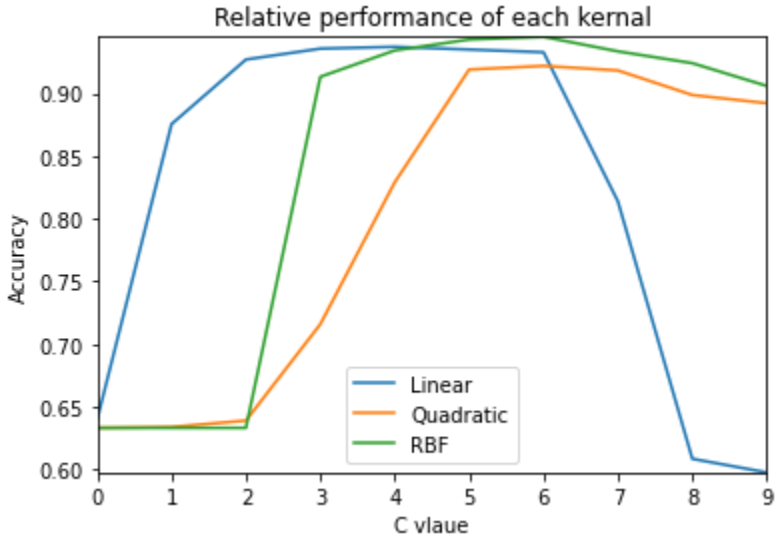
- **predict() method :-**
  - Predict the output for input data (x_test and y_test)

3.Experimental Results & Observation:-

	C	Linear	Quadratic	RBF
0	0.00005	(0.6046583850931677, 0.6398550724637682)	(0.5944099378881987, 0.6333333333333333)	(0.5944099378881987, 0.6333333333333333)
1	0.00050	(0.8574534161490683, 0.8753623188405797)	(0.59472049689441, 0.6340579710144928)	(0.5944099378881987, 0.6333333333333333)
2	0.00500	(0.9114906832298136, 0.9268115942028986)	(0.6046583850931677, 0.6391304347826087)	(0.5944099378881987, 0.6333333333333333)
3	0.05000	(0.9307453416149069, 0.9355072463768116)	(0.6900621118012422, 0.7159420289855073)	(0.8950310559006212, 0.9130434782608695)
4	0.50000	(0.9322981366459627, 0.9369565217391305)	(0.8220496894409938, 0.8289855072463768)	(0.937888198757764, 0.9340579710144927)
5	5.00000	(0.9332298136645962, 0.9347826086956522)	(0.9322981366459627, 0.9188405797101449)	(0.9580745341614907, 0.9427536231884058)
6	50.00000	(0.9304347826086956, 0.9326086956521739)	(0.9658385093167702, 0.9217391304347826)	(0.9816770186335404, 0.9449275362318841)
7	500.00000	(0.8204968944099379, 0.813768115942029)	(0.9819875776397515, 0.9181159420289855)	(0.9922360248447205, 0.9333333333333333)
8	5000.00000	(0.6267080745341614, 0.6086956521739131)	(0.991304347826087, 0.8985507246376812)	(0.9940993788819876, 0.9239130434782609)
9	50000.00000	(0.6080745341614907, 0.5978260869565217)	(0.993167701863354, 0.8920289855072464)	(0.9962732919254659, 0.9057971014492754)

We can see that as the C increases the accuracy will also increase as we know the C presents the SVM optimization of how much you want to avoid misclassifying each training example. So in the linear kernel case the overfitting problem will occur when c >= 50.

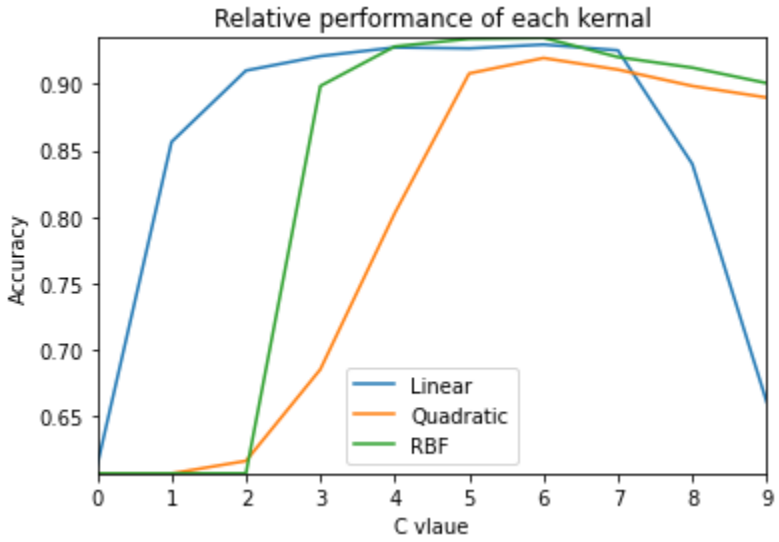
We can also see that the more C the more good accuracy we are getting and RBF is one who is performing the best.



After a certain point in each kernel as we increase the C the accuracy decreases. So the best accuracy is at **C = 50 for RBF.** This data and result is for max iteration in SVC equal to $1e^6$.

We tried for max iteration $1e^7$ but the total time taken was more than 6 minutes so drop the idea to take in the final result.

	C	Linear	Quadratic	RBF
0	0.00005	(0.6111801242236025, 0.6130434782608696)	(0.6055900621118012, 0.6072463768115942)	(0.6055900621118012, 0.6072463768115942)
1	0.00050	(0.865527950310559, 0.8565217391304348)	(0.6059006211180125, 0.6072463768115942)	(0.6055900621118012, 0.6072463768115942)
2	0.00500	(0.918944099378882, 0.9101449275362319)	(0.6124223602484472, 0.6166666666666667)	(0.6055900621118012, 0.6072463768115942)
3	0.05000	(0.9313664596273292, 0.9210144927536232)	(0.6965838509316771, 0.6855072463768116)	(0.8993788819875776, 0.8985507246376812)
4	0.50000	(0.9357142857142857, 0.927536231884058)	(0.824223602484472, 0.8028985507246377)	(0.9397515527950311, 0.9282608695652174)
5	5.00000	(0.937888198757764, 0.9268115942028986)	(0.9394409937888198, 0.9079710144927536)	(0.9627329192546584, 0.9340579710144927)
6	50.00000	(0.9388198757763975, 0.9297101449275362)	(0.9683229813664597, 0.9195652173913044)	(0.9835403726708074, 0.9347826086956522)
7	500.00000	(0.9372670807453416, 0.9253623188405797)	(0.9854037267080745, 0.9108695652173913)	(0.9928571428571429, 0.9202898550724637)
8	5000.00000	(0.8419254658385094, 0.8398550724637681)	(0.9928571428571429, 0.8985507246376812)	(0.9953416149068323, 0.9123188405797101)
9	50000.00000	(0.6481366459627329, 0.6608695652173913)	(0.9947204968944099, 0.8898550724637682)	(0.9968944099378882, 0.9007246376811594)



We also tried for max iteration $1e^8$ but the total time taken was more than 40 minutes so drop the idea to take in the final result.

FinalTable

	C	Linear	Quadratic	RBF
0	0.00005	(0.6083850931677018, 0.6260869565217392)	(0.6, 0.6202898550724638)	(0.6, 0.6202898550724638)
1	0.00050	(0.8624223602484472, 0.8760869565217392)	(0.6003105590062112, 0.6202898550724638)	(0.6, 0.6202898550724638)
2	0.00500	(0.9161490683229814, 0.9101449275362319)	(0.6074534161490683, 0.6260869565217392)	(0.6, 0.6202898550724638)
3	0.05000	(0.9304347826086956, 0.9217391304347826)	(0.6854037267080745, 0.7028985507246377)	(0.8950310559006212, 0.8978260869565218)
4	0.50000	(0.9338509316770186, 0.9282608695652174)	(0.8139751552795031, 0.8333333333333334)	(0.9391304347826087, 0.9253623188405797)
5	5.00000	(0.9347826086956522, 0.9268115942028986)	(0.9350931677018633, 0.9014492753623189)	(0.9627329192546584, 0.9391304347826087)
6	50.00000	(0.9338509316770186, 0.9297101449275362)	(0.9670807453416149, 0.9181159420289855)	(0.9819875776397515, 0.9391304347826087)
7	500.00000	(0.9332298136645962, 0.9260869565217391)	(0.9822981366459628, 0.8985507246376812)	(0.9922360248447205, 0.9210144927536232)
8	5000.00000	(0.9301242236024845, 0.9246376811594202)	(0.9909937888198758, 0.8804347826086957)	(0.9950310559006211, 0.9152173913043479)
9	50000.00000	(0.8403726708074534, 0.8514492753623188)	(0.9944099378881988, 0.8818840579710145)	(0.9972049689440994, 0.908695652173913)

```
p = [*range(0,10,1)]
plt.plot(p, Linear_test, label = "Linear")
plt.plot(p, Quadratic_test, label = "Quadratic")
plt.plot(p, RBF_test, label = "RBF")

plt.xlabel("C vlaue")
plt.ylabel("Number of New States Visited")
plt.title("Diff vlaue")
# plt.ylim(0 , 1)
plt.margins(0)
plt.legend()
plt.show()
```