Graph Convolution Neural Network Test with Cora Dataset

20185141 용권순

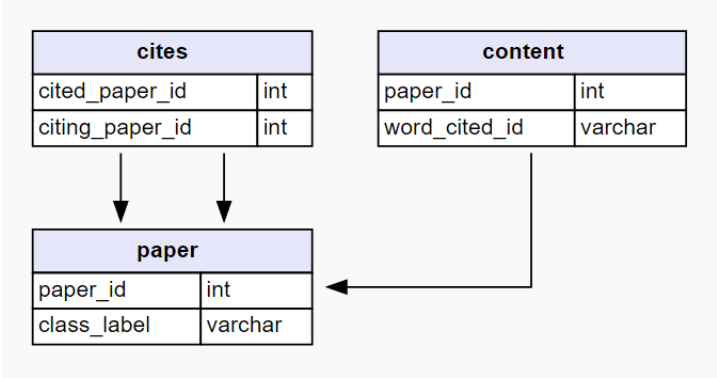
20185141

KwonSoon Yong

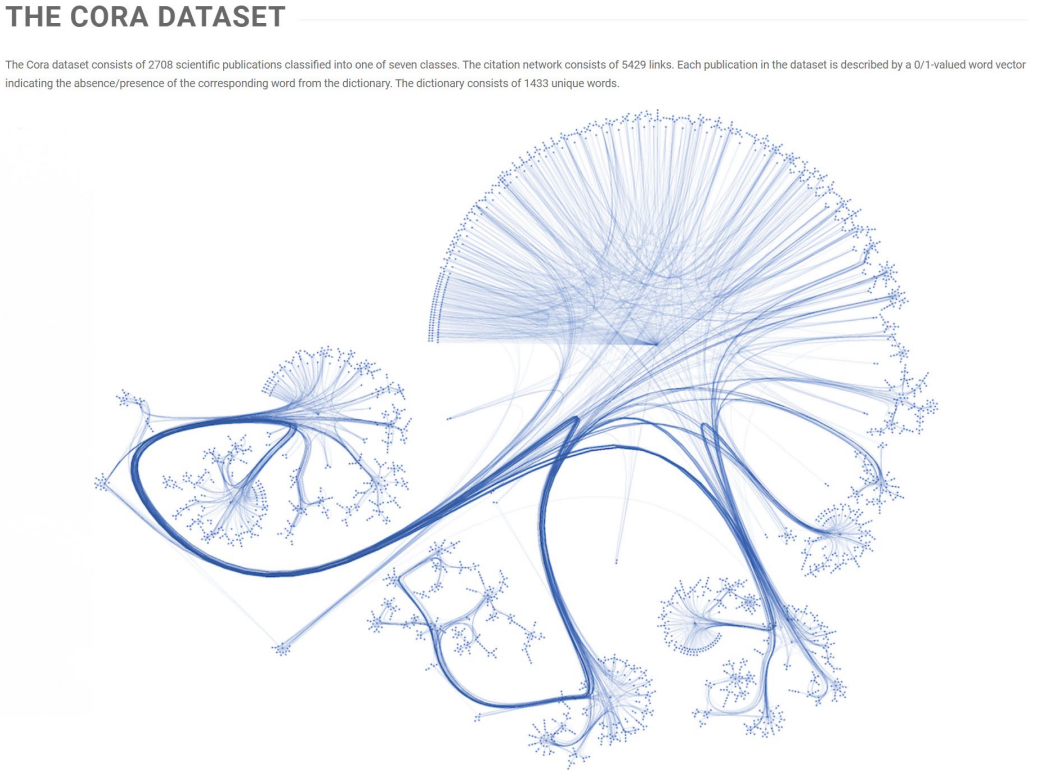
Mail : yykks3971@gmail.com

Github: https://github.com/reverse-sky

Dataset



Cora Dataset



The **Cora** dataset consists of **2708 scientific publications** classified into one of **seven classes**

The citation network consists of **5429 links**. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary

The dictionary consists of **1433 unique words**.

Dataset EDA

	target	source
0	35	1033
1	35	103482
2	35	103515
3	35	1050679
4	35	1103960
...	...	...
5424	853116	19621
5425	853116	853155
5426	853118	1140289
5427	853155	853118
5428	954315	1155073

5429 rows × 2 columns

Cora cites

	w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	...	w_1424	w_1425	w_1426	w_1427	w_1428	w_1429	w_1430	w_1431	w_1432	subject
31336	0	0	0	0	0	0	0	0	0	0	...	0	0	1	0	0	0	0	0	0	Neural_Networks
1061127	0	0	0	0	0	0	0	0	0	0	...	0	1	0	0	0	0	0	0	0	Rule_Learning
1106406	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	Reinforcement_Learning
13195	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	Reinforcement_Learning
37879	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	Probabilistic_Methods
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1128975	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	Genetic_Algorithms
1128977	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	Genetic_Algorithms
1128978	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	Genetic_Algorithms
117328	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	Case_Based
24043	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	Neural_Networks

2708 rows × 1434 columns

Seven labels

```
content["subject"].value_counts()
```

```
Neural_Networks           818
Probabilistic_Methods     426
Genetic_Algorithms        418
Theory                    351
Case_Based                298
Reinforcement_Learning    217
Rule_Learning             180
Name: subject, dtype: int64
```

Preprocessing

```
Ordinary_encode = {i:idx for idx,i in enumerate(label_list)}
content["subject"] = content["subject"].map(Ordinary_encode)
content['subject']
```

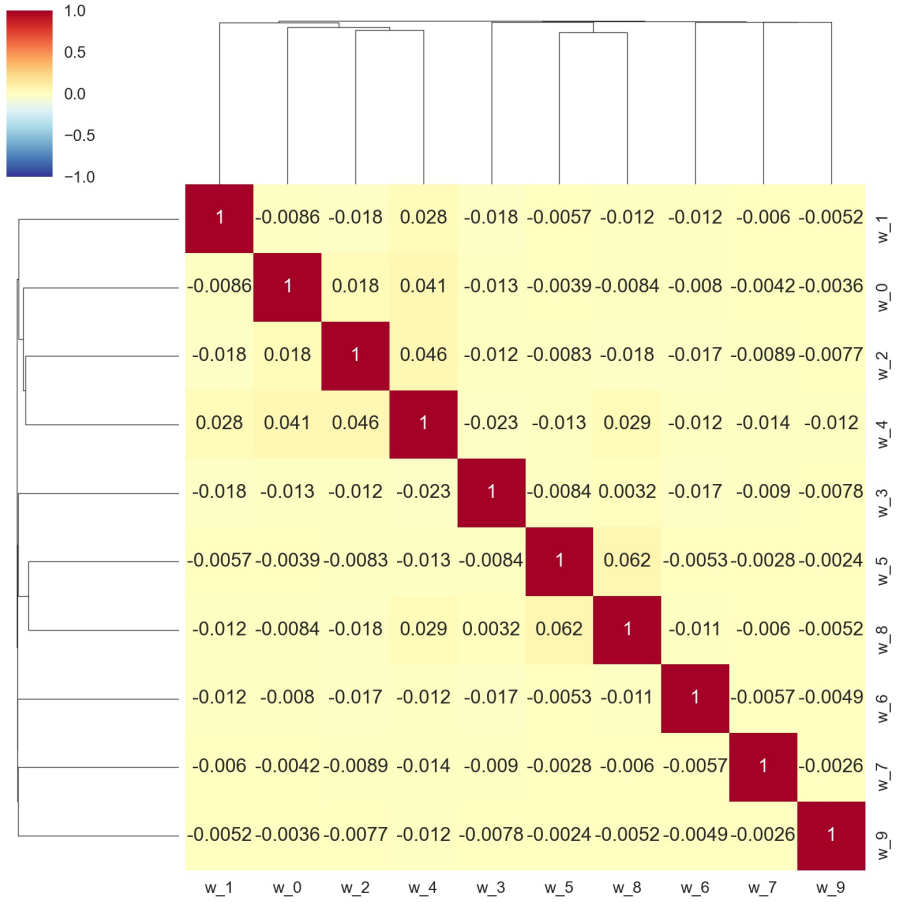
Ordinary encoding for categorical data

	w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	...	w_1424	w_1425	w_1426	w_1427	w_1428	w_1429	w_1430	w_1431	w_1432	subject
31336	0	0	0	0	0	0	0	0	0	0	...	0	0	1	0	0	0	0	0	0	Neural_Networks
1061127	0	0	0	0	0	0	0	0	0	0	...	0	1	0	0	0	0	0	0	0	Rule_Learning
1106406	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	Reinforcement_Learning
13195	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	Reinforcement_Learning
37879	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	Probabilistic_Methods
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1128975	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	Genetic_Algorithms
1128977	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	Genetic_Algorithms
1128978	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	Genetic_Algorithms
117328	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	Case_Based
24043	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	Neural_Networks

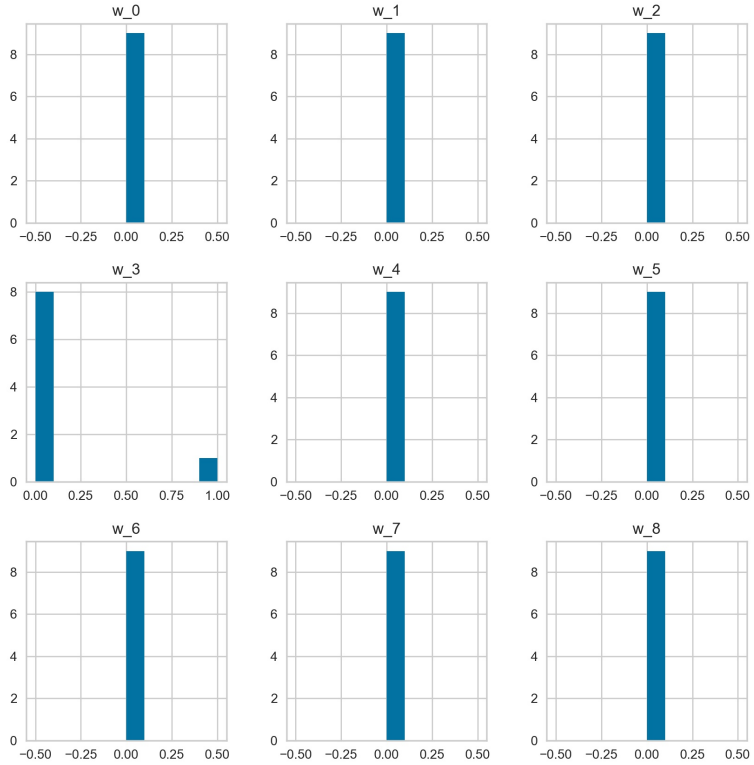
2708 rows × 1434 columns

subject
2
5
4
4
3
...
1
1
1
0
2

Dataset Correlation & Histogram

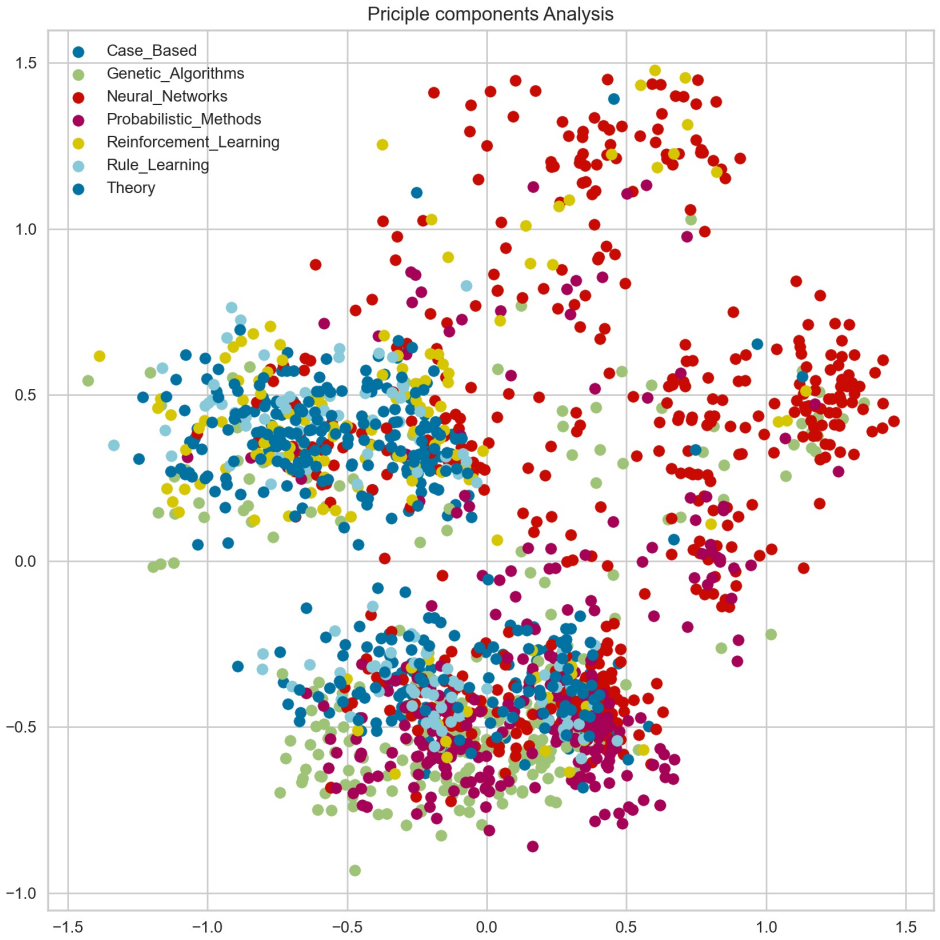


Correlation

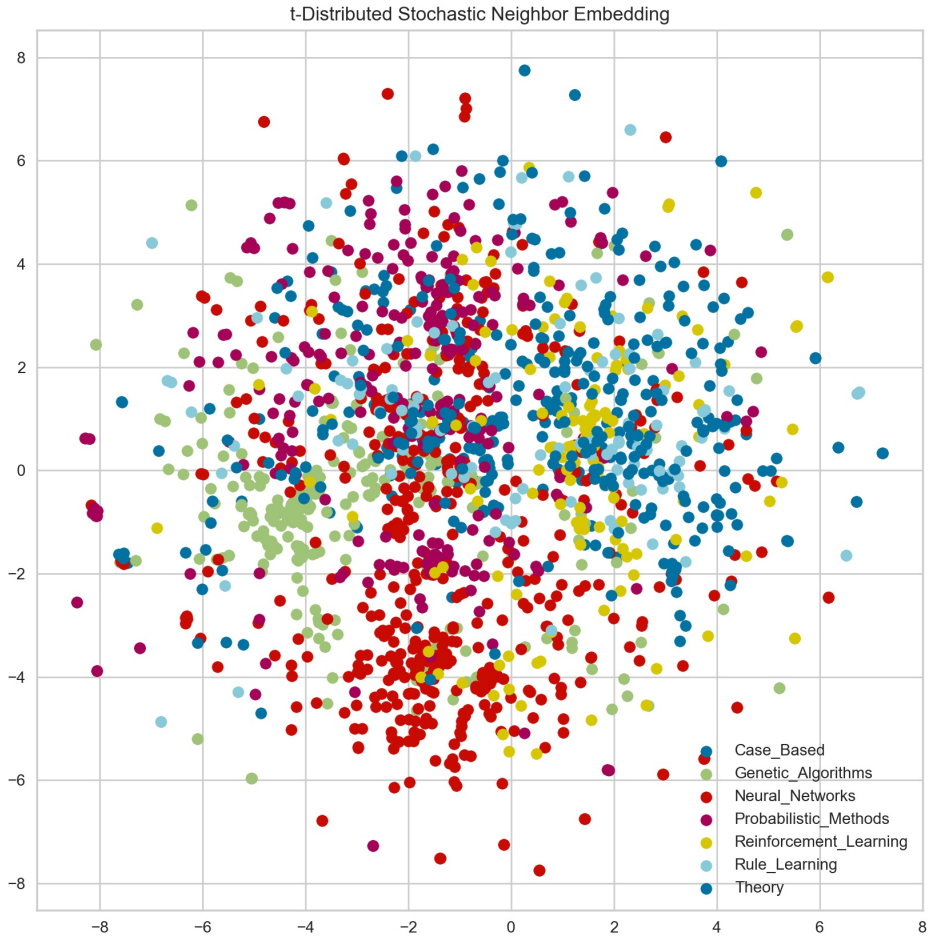


Histogram

High-dimensional data visualization

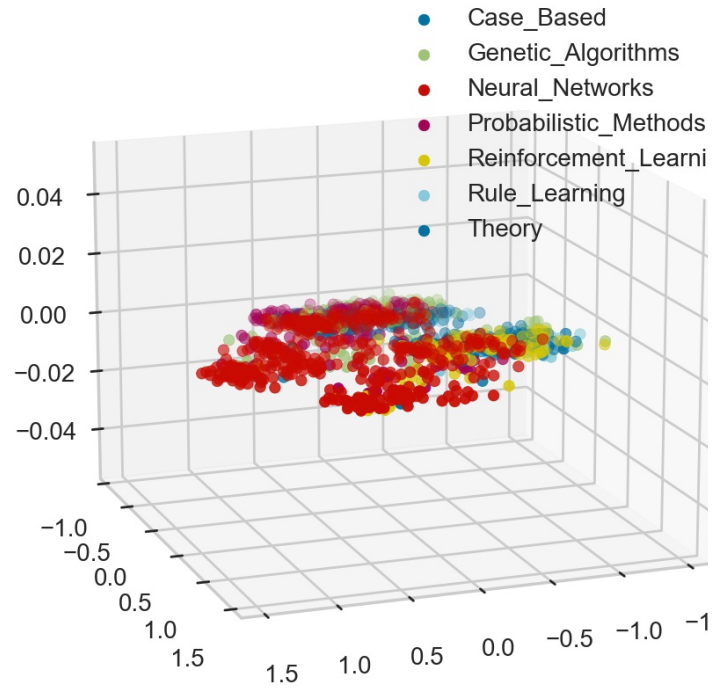


Principle Components Analysis

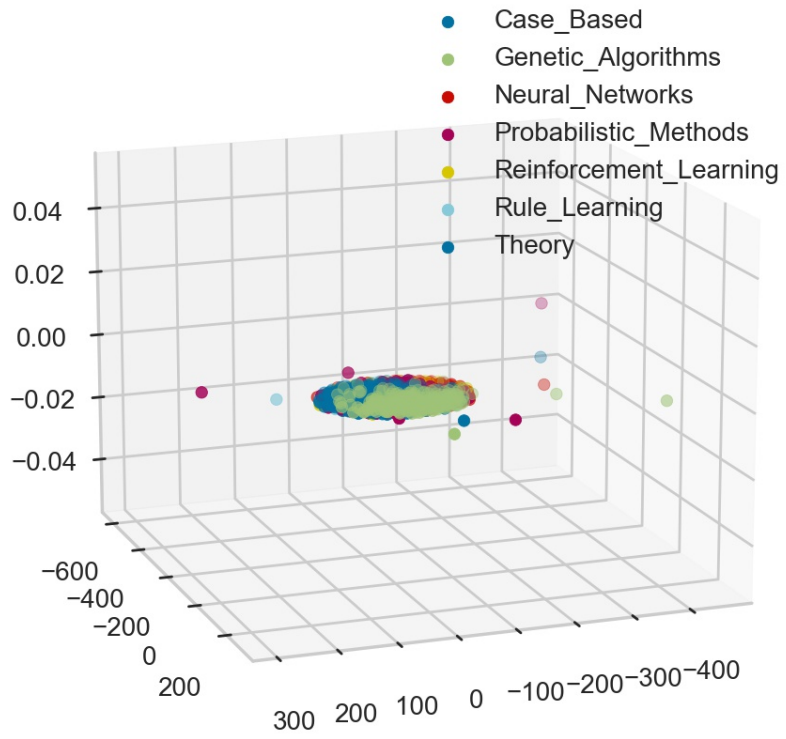


T-Distributed Stochastic Neighbor Embedding

High-dimensional data visualization - 3d



Principle Components Analysis



T-Distributed Stochastic Neighbor Embedding

Data set split

```
train_,test = train_test_split(content,test_size=0.2,random_state=42,stratify=content["subject"])
train, valid = train_test_split(train_,test_size=0.2 ,random_state=42,stratify=train_["subject"])
```

```
train_x = train.drop(columns = 'subject',axis=1)
train_y = train["subject"]
valid_x = valid.drop(columns = 'subject',axis=1)
valid_y = valid["subject"]
test_x = test.drop(columns = 'subject',axis=1)
test_y = test["subject"]

train_x  = np.array(train_x,dtype = np.float32)
train_y  = np.array(train_y,dtype = np.float32)
valid_x  = np.array(valid_x,dtype = np.float32)
valid_y  = np.array(valid_y,dtype = np.float32)
test_x   = np.array(test_x, dtype = np.float32)
test_y   = np.array(test_y, dtype = np.float32)
```

Dataset Split

Train : Valid : Test = 6 : 2 : 2

Model selection & Metric

Model
LogisticRegression
SGDClassifier
GradientBoostingClassifier
DecisionTreeClassifier
RandomForestClassifier

Model list in Lecture

$$K = \frac{P_A - P_C}{1 - P_C}$$

P_A : probability of a match between two people evaluation

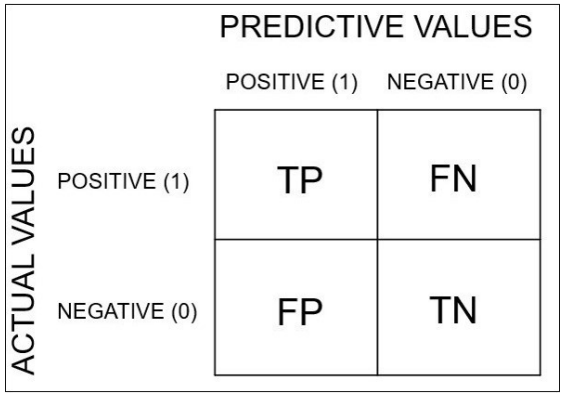
P_C : Probability by chance

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

: Metric of True false

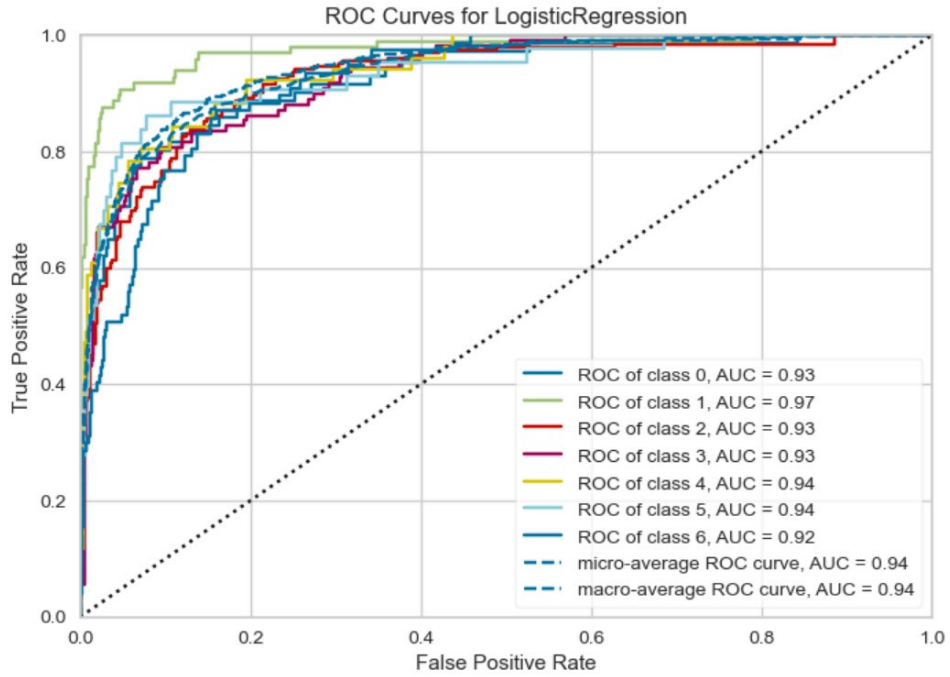
Accuracy **AUC** **Recall** **Prec.** **F1** **Kappa** **MCC**

Metric



Logistic Regression

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Logistic Regression	0.7200	0.9348	0.7026	0.7248	0.7209	0.6591	0.6597

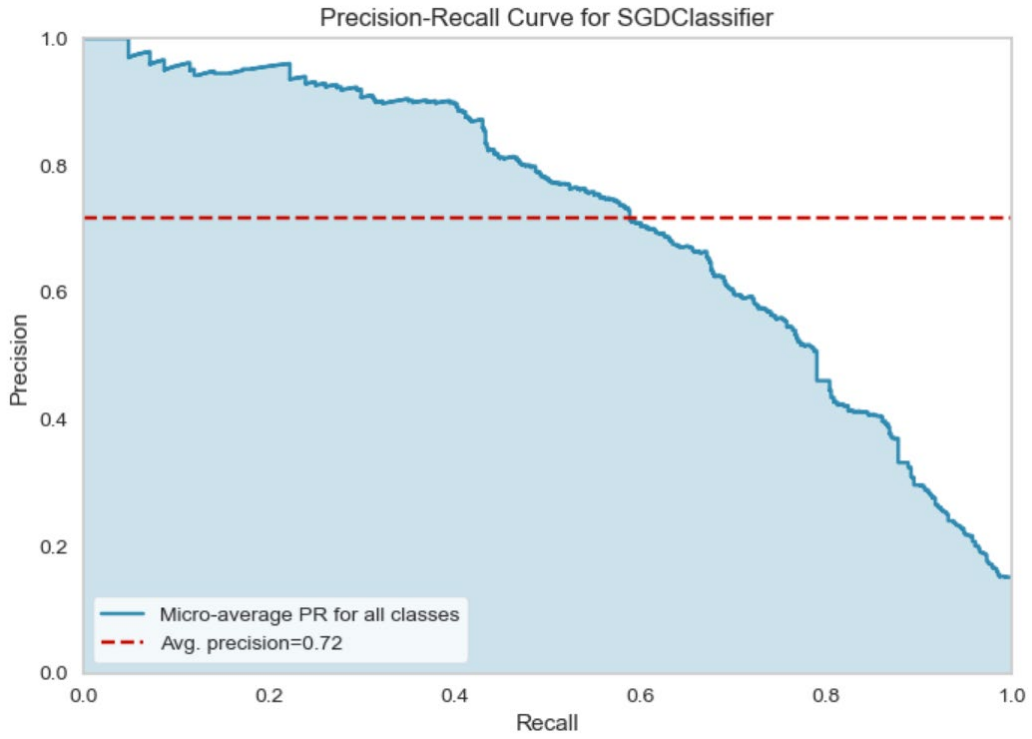


Fold	Accuracy	AUC	Recall	Prec.	F1
0	0.7368	0.9378	0.6899	0.7362	0.7336
1	0.7368	0.9504	0.6898	0.7444	0.7338
2	0.7961	0.9561	0.7616	0.8102	0.7908
3	0.6974	0.9179	0.6608	0.7095	0.6980
4	0.7368	0.9434	0.6828	0.7531	0.7308
5	0.7368	0.9470	0.6877	0.7451	0.7345
6	0.7682	0.9537	0.7000	0.7584	0.7585
7	0.6954	0.9269	0.6559	0.7008	0.6899
8	0.7417	0.9446	0.7192	0.7422	0.7401
9	0.7483	0.9351	0.7353	0.7528	0.7485
Mean	0.7394	0.9413	0.6983	0.7453	0.7359
Std	0.0280	0.0114	0.0309	0.0280	0.0270

SVM

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	SVM - Linear Kernel	0.6831	0	0.6097	0.6922	0.6741	0.6014	0.6068

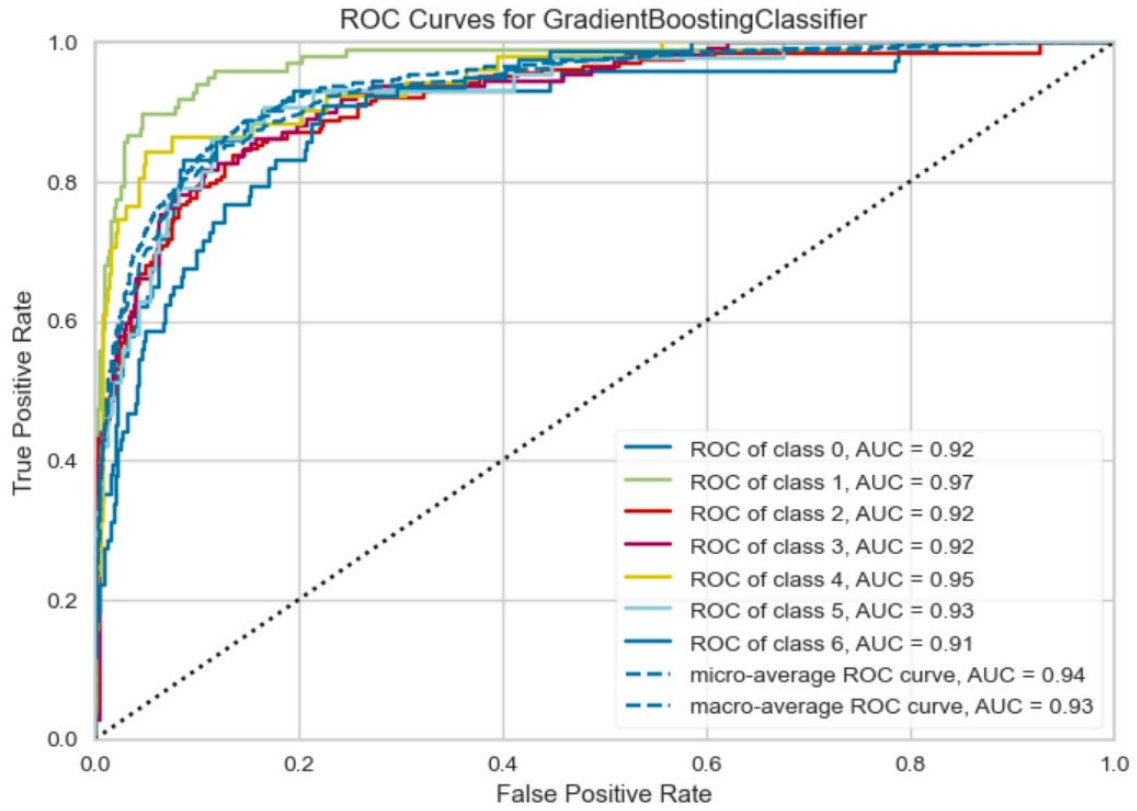
```
SGDClassifier(alpha=0.005, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.1, fit_intercept=False,
              l1_ratio=0.5500000001, learning_rate='invscaling', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=-1, penalty='l1',
              power_t=0.5, random_state=1212, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
```



Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7171	0.0000	0.6547	0.7416	0.7128	0.6487	0.6536
1	0.7039	0.0000	0.6464	0.7140	0.6988	0.6319	0.6369
2	0.7039	0.0000	0.6242	0.7564	0.6969	0.6278	0.6380
3	0.6447	0.0000	0.5648	0.6932	0.6331	0.5529	0.5650
4	0.6908	0.0000	0.6284	0.7248	0.6867	0.6173	0.6214
5	0.6842	0.0000	0.5976	0.6862	0.6706	0.6044	0.6112
6	0.6954	0.0000	0.6428	0.7038	0.6901	0.6265	0.6292
7	0.6159	0.0000	0.5507	0.6546	0.6077	0.5203	0.5287
8	0.6887	0.0000	0.6167	0.7008	0.6786	0.6113	0.6168
9	0.6556	0.0000	0.6453	0.6818	0.6552	0.5734	0.5788
Mean	0.6800	0.0000	0.6172	0.7057	0.6731	0.6014	0.6079
Std	0.0299	0.0000	0.0338	0.0284	0.0309	0.0382	0.0368

Gradient Boosting Classifier

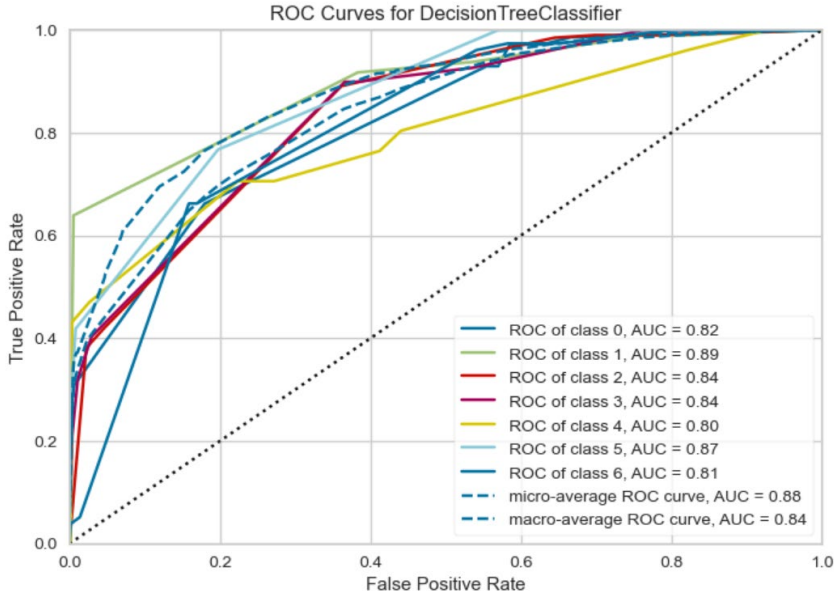
	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Gradient Boosting Classifier	0.7123	0.9284	0.6625	0.7092	0.7076	0.6433	0.6445

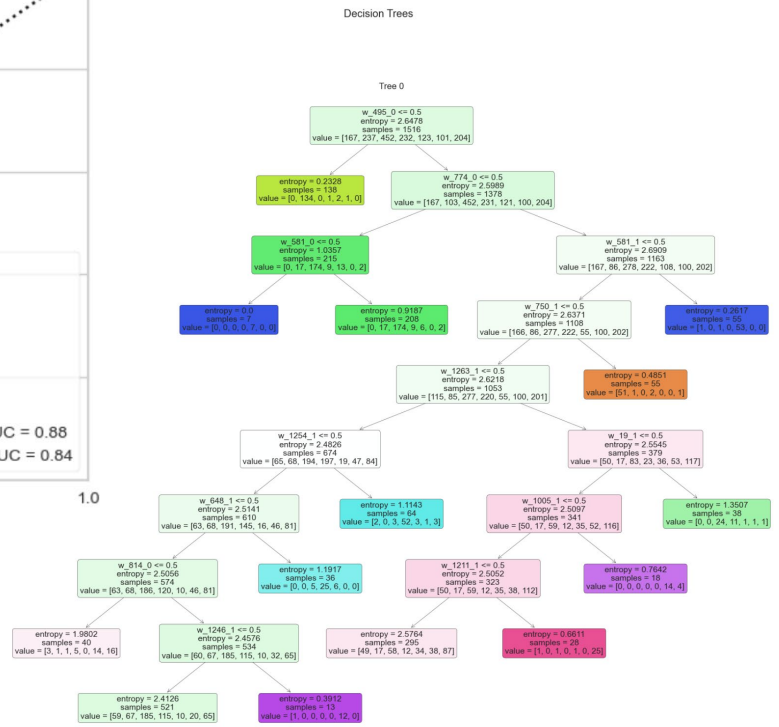


Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7237	0.9476	0.6868	0.7226	0.7207	0.6627	0.6635
1	0.7566	0.9529	0.6987	0.7598	0.7534	0.7001	0.7024
2	0.7566	0.9394	0.7223	0.7587	0.7539	0.7009	0.7023
3	0.6711	0.9122	0.6269	0.6818	0.6682	0.5924	0.5965
4	0.6842	0.9403	0.6123	0.6708	0.6743	0.6117	0.6128
5	0.7105	0.9403	0.6549	0.7193	0.7041	0.6401	0.6444
6	0.7483	0.9489	0.6863	0.7519	0.7407	0.6883	0.6923
7	0.7020	0.9172	0.6561	0.7178	0.6979	0.6318	0.6343
8	0.6755	0.9293	0.6416	0.6749	0.6721	0.6041	0.6048
9	0.7351	0.9265	0.7243	0.7443	0.7359	0.6775	0.6780
Mean	0.7164	0.9355	0.6710	0.7202	0.7121	0.6509	0.6531
Std	0.0311	0.0130	0.0367	0.0326	0.0318	0.0385	0.0383

Decision Tree Classifier

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Decision Tree Classifier	0.5908	0.8414	0.5062	0.7301	0.5789	0.4734	0.5017

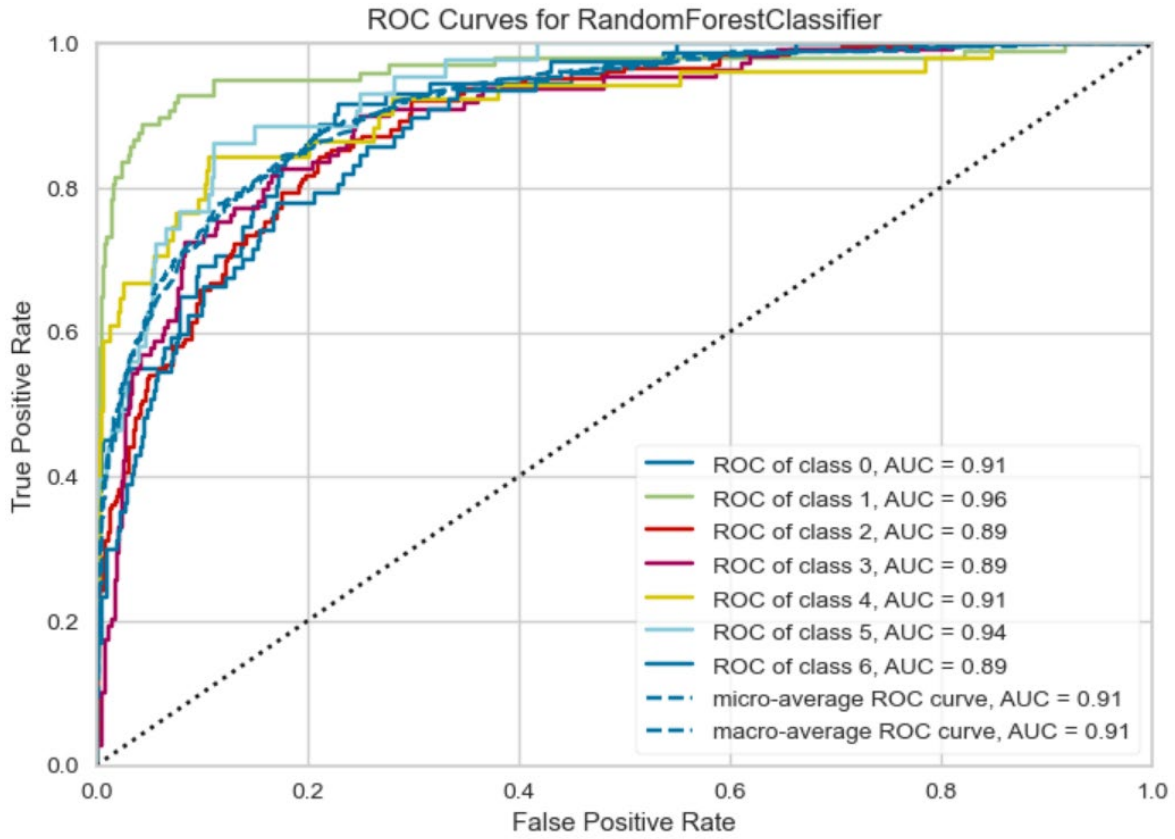




Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.5461	0.7866	0.4536	0.6900	0.5197	0.4193	0.4472
1	0.5855	0.8561	0.4953	0.7183	0.5873	0.4689	0.4960
2	0.5592	0.8456	0.4688	0.6650	0.5506	0.4408	0.4610
3	0.4868	0.7908	0.4102	0.6490	0.4790	0.3482	0.3717
4	0.5263	0.8041	0.4475	0.5971	0.5097	0.3975	0.4153
5	0.5789	0.8505	0.4827	0.6817	0.5661	0.4616	0.4851
6	0.5960	0.8518	0.5162	0.7464	0.5906	0.4830	0.5134
7	0.5563	0.8046	0.4697	0.7296	0.5473	0.4282	0.4590
8	0.5232	0.8081	0.4378	0.6357	0.5166	0.3883	0.4109
9	0.5364	0.7939	0.4620	0.6682	0.5245	0.4102	0.4349
Mean	0.5495	0.8192	0.4644	0.6781	0.5391	0.4246	0.4494
Std	0.0313	0.0268	0.0284	0.0431	0.0339	0.0389	0.0408

Random Forest Classifier

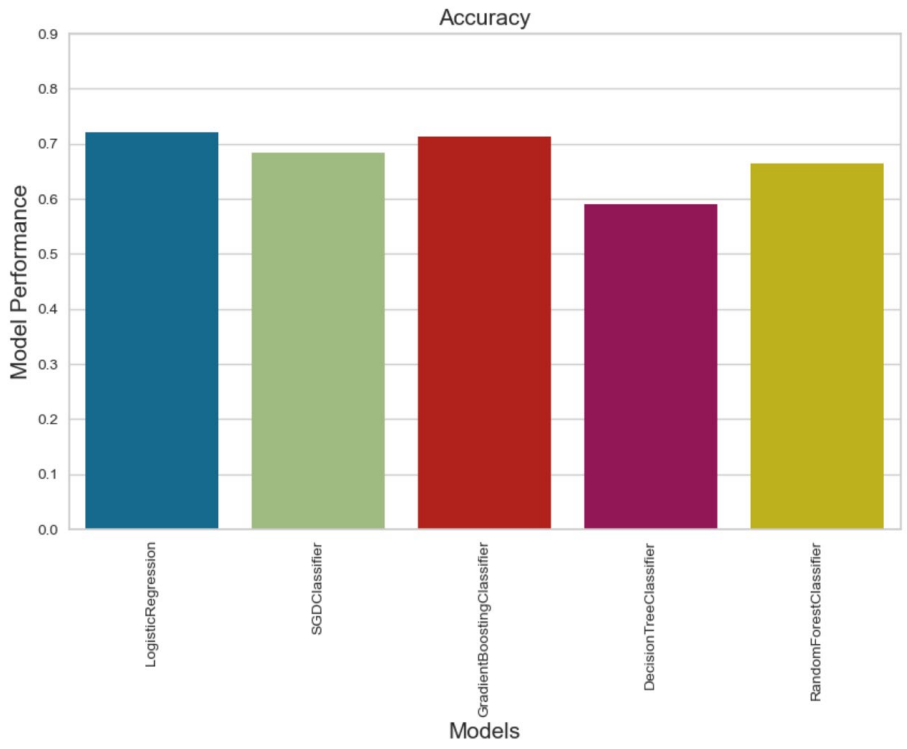
	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Random Forest Classifier	0.6646	0.9080	0.6621	0.6839	0.6678	0.5960	0.5987



Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7039	0.9110	0.7076	0.7200	0.7059	0.6440	0.6464
1	0.7829	0.9451	0.7724	0.7941	0.7855	0.7371	0.7383
2	0.7303	0.9219	0.7274	0.7424	0.7330	0.6745	0.6756
3	0.7237	0.9119	0.7442	0.7422	0.7272	0.6686	0.6710
4	0.6645	0.9230	0.6805	0.7074	0.6663	0.6018	0.6093
5	0.7105	0.9243	0.6790	0.7228	0.7131	0.6513	0.6528
6	0.6821	0.9218	0.6705	0.7190	0.6896	0.6190	0.6230
7	0.6887	0.9024	0.6915	0.7079	0.6913	0.6239	0.6259
8	0.7483	0.9351	0.7394	0.7636	0.7470	0.6981	0.7020
9	0.6291	0.9034	0.6689	0.6669	0.6281	0.5604	0.5675
Mean	0.7064	0.9200	0.7081	0.7286	0.7087	0.6479	0.6512
Std	0.0414	0.0128	0.0341	0.0328	0.0416	0.0478	0.0462

Ensemble

	Model	Score
0	LogisticRegression	0.720000
1	SGDClassifier	0.683077
2	GradientBoostingClassifier	0.712308
3	DecisionTreeClassifier	0.590769
4	RandomForestClassifier	0.664615



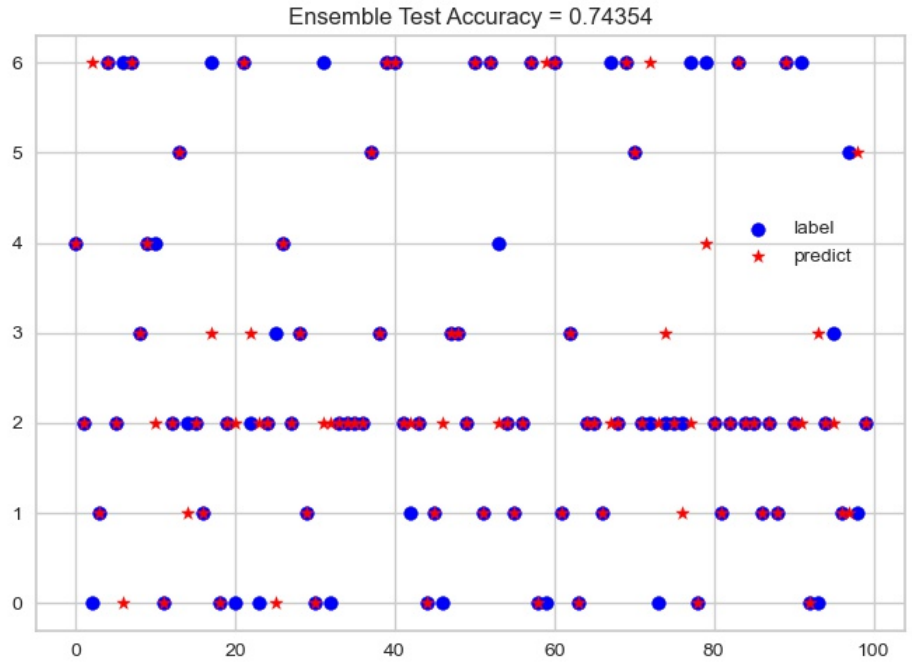

```
model_list.pop(-1)
model_list.pop(-1)
```

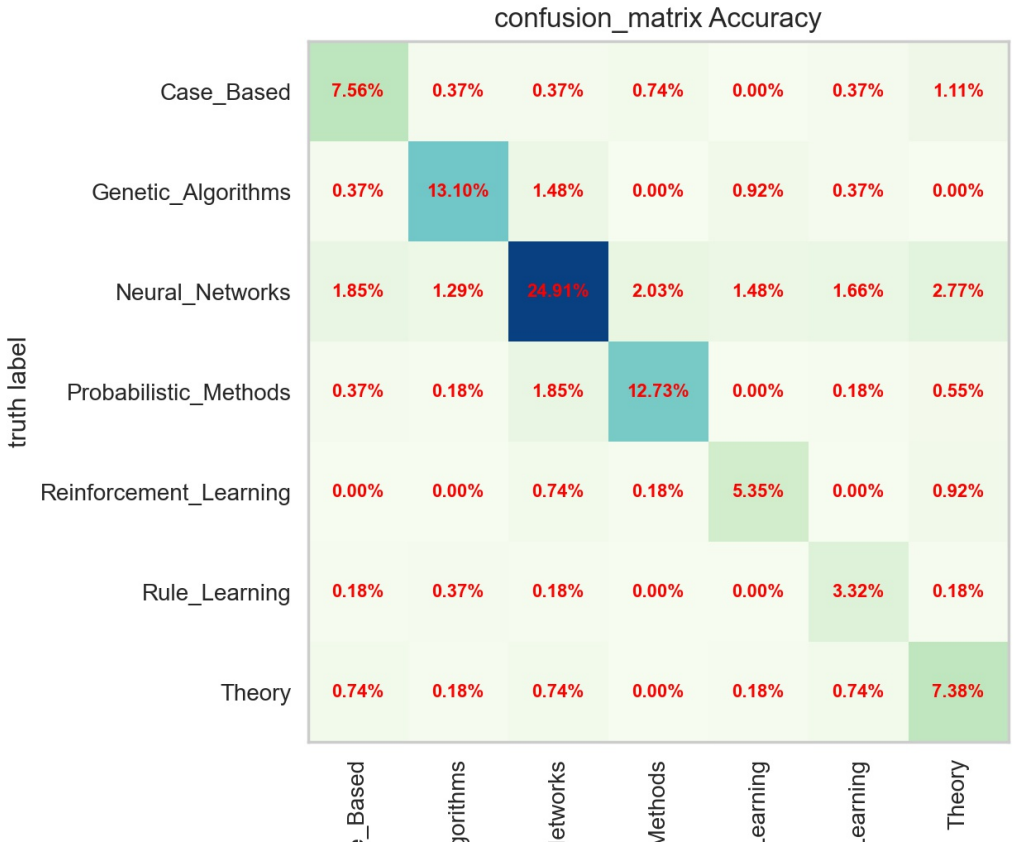
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=9, max_features=1.0, max_leaf_nodes=None,
                       min_impurity_decrease=0.02, min_impurity_split=None,
                       min_samples_leaf=5, min_samples_split=10,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=1212, splitter='best')
```

```
model_list
```

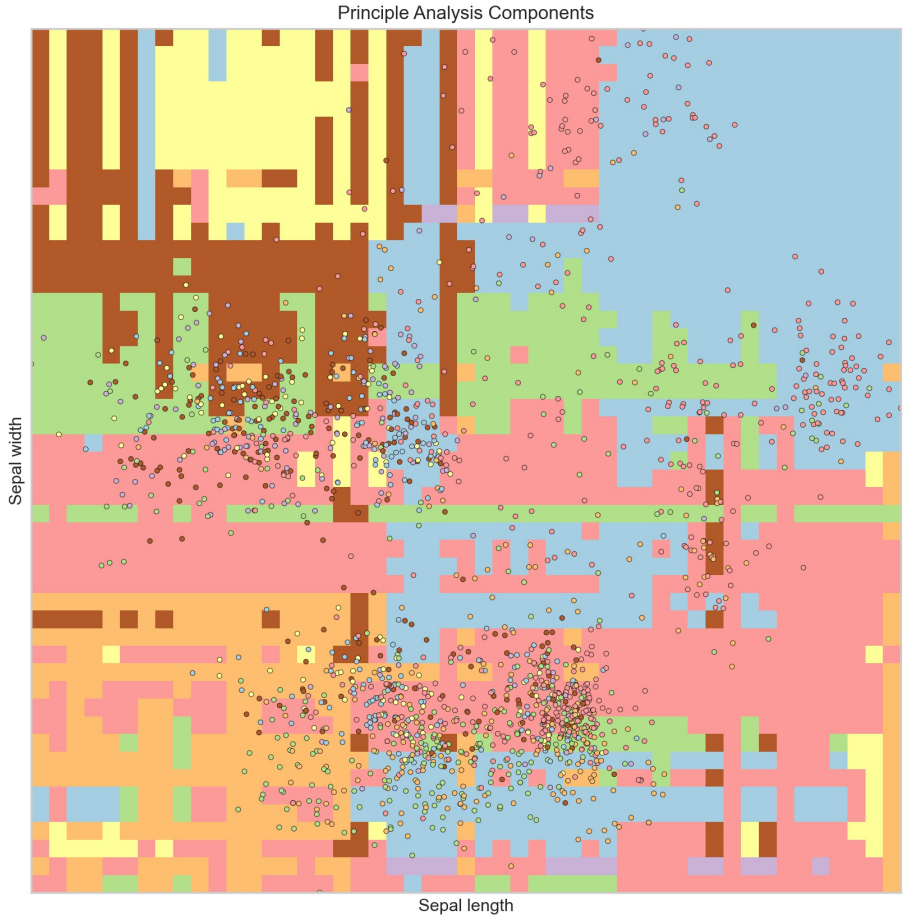
```
[LogisticRegression(C=0.846, class_weight='balanced', dual=False,
                    fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                    max_iter=1000, multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=1212, solver='lbfgs', tol=0.0001, verbose=0,
                    warm_start=False),
 SGDClassifier(alpha=0.005, average=False, class_weight=None,
               early_stopping=False, epsilon=0.1, eta0=0.1, fit_intercept=False,
               l1_ratio=0.5500000001, learning_rate='invscaling', loss='hinge',
               max_iter=1000, n_iter_no_change=5, n_jobs=-1, penalty='l1',
               power_t=0.5, random_state=1212, shuffle=True, tol=0.001,
               validation_fraction=0.1, verbose=0, warm_start=False),
 GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                            learning_rate=0.2, loss='deviance', max_depth=4,
                            max_features='log2', max_leaf_nodes=None,
                            min_impurity_decrease=0.01, min_impurity_split=None,
                            min_samples_leaf=4, min_samples_split=2,
                            min_weight_fraction_leaf=0.0, n_estimators=230,
                            n_iter_no_change=None, presort='deprecated',
                            random_state=1212, subsample=0.5, tol=0.0001,
                            validation_fraction=0.1, verbose=0,
                            warm_start=False)]
```

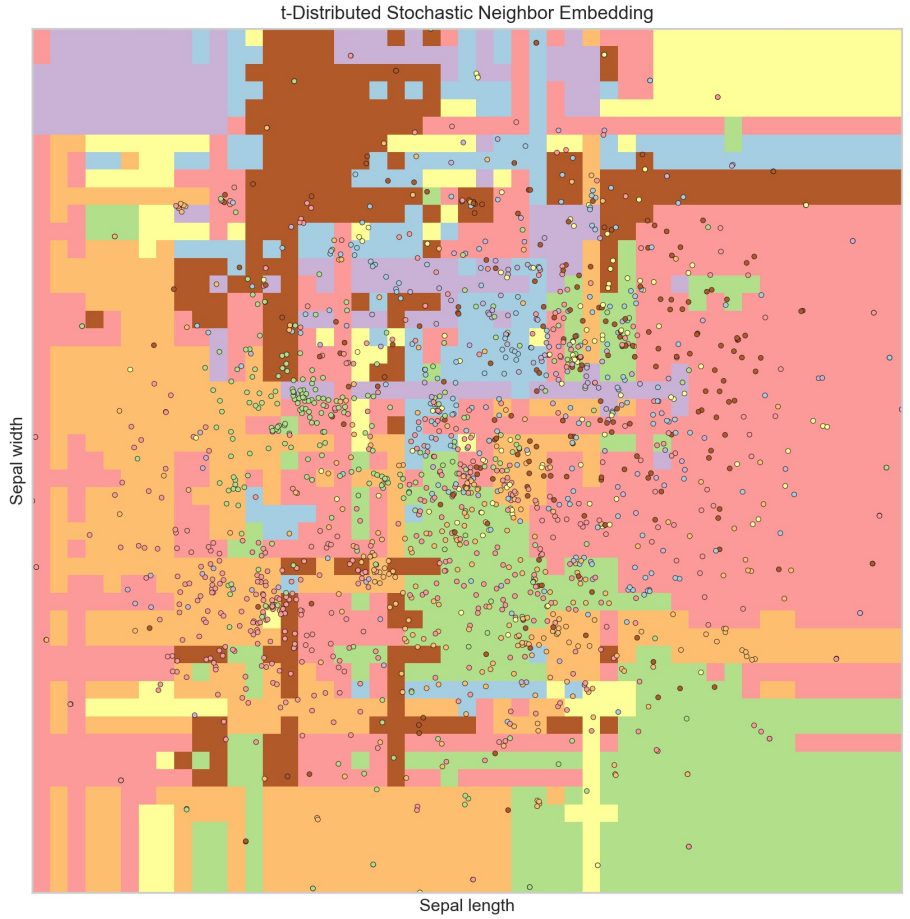
Ensemble Result





Ensemble Decision boundary

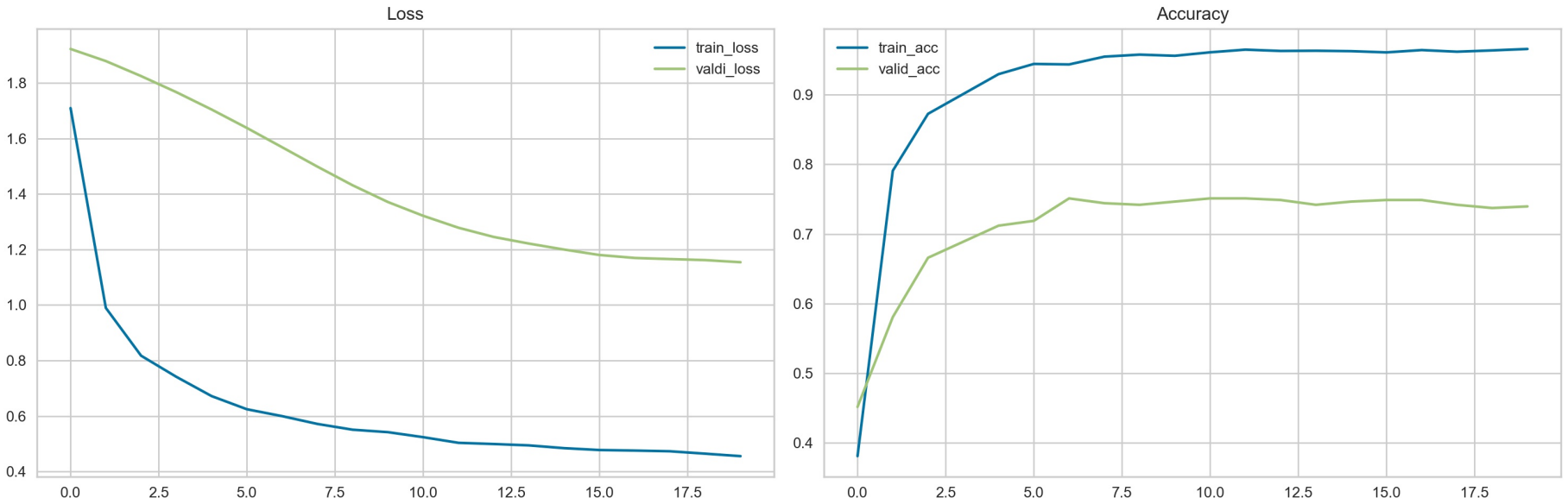




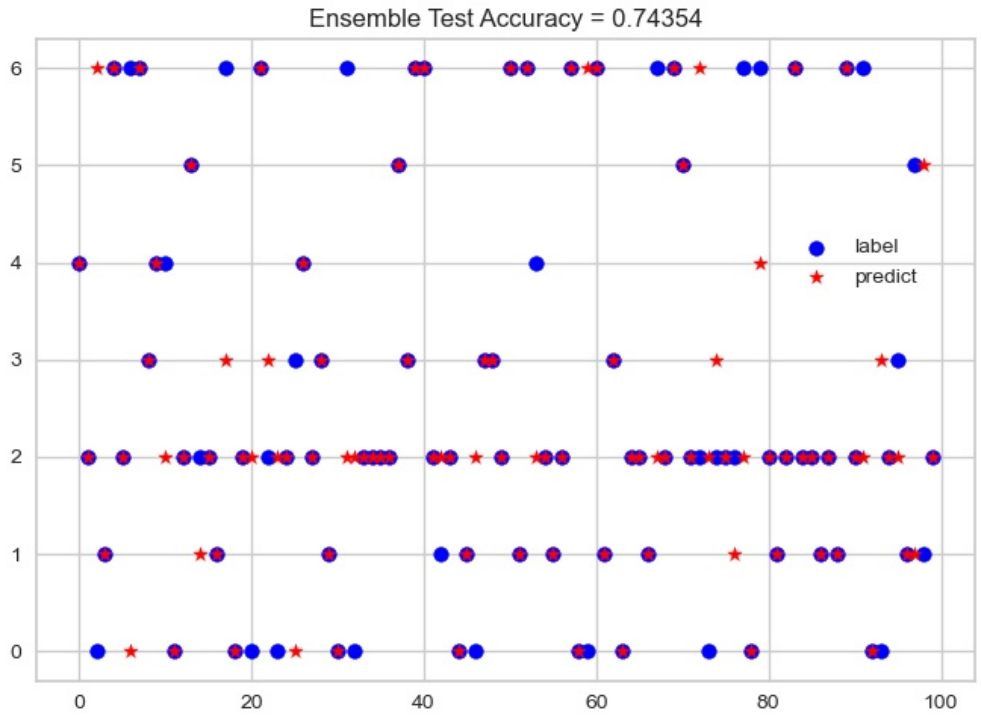
DNN(Fully Connected)

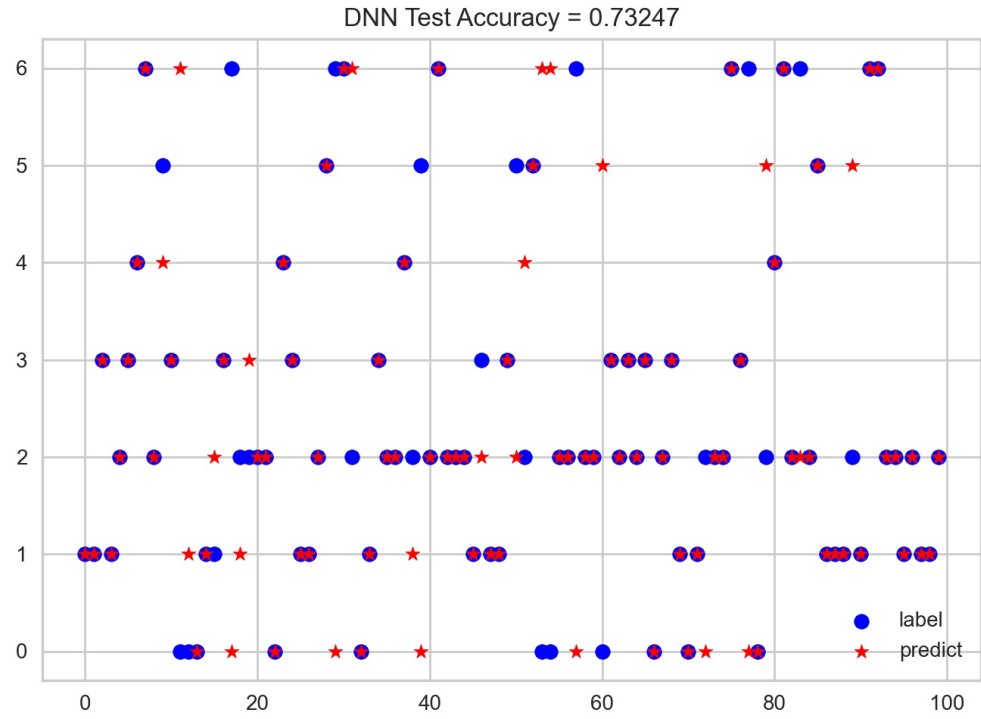
```
def linblock(in_dim,out_dim):
    return nn.Sequential(nn.Linear(in_dim,out_dim,bias=False),
                         nn.BatchNorm1d(out_dim),
                         nn.ReLU())
simple_dnn = nn.Sequential(linblock(1433,512),
                           linblock(512,128),
                           linblock(128,7))
simple_dnn.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(simple_dnn.parameters(),lr=1e-3)
```

Deep learning model implementation with pytorch

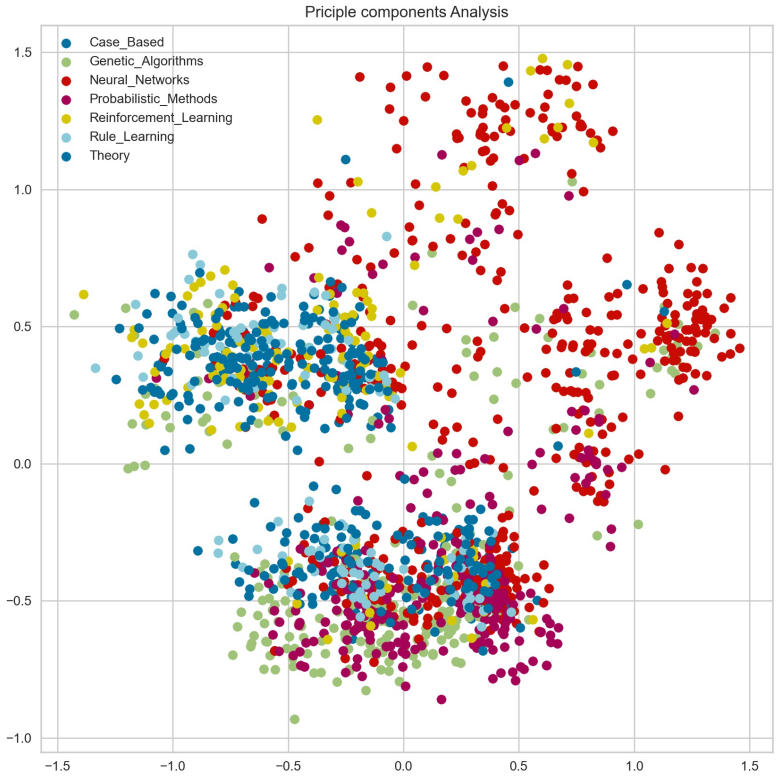


DNN(Fully Connected)



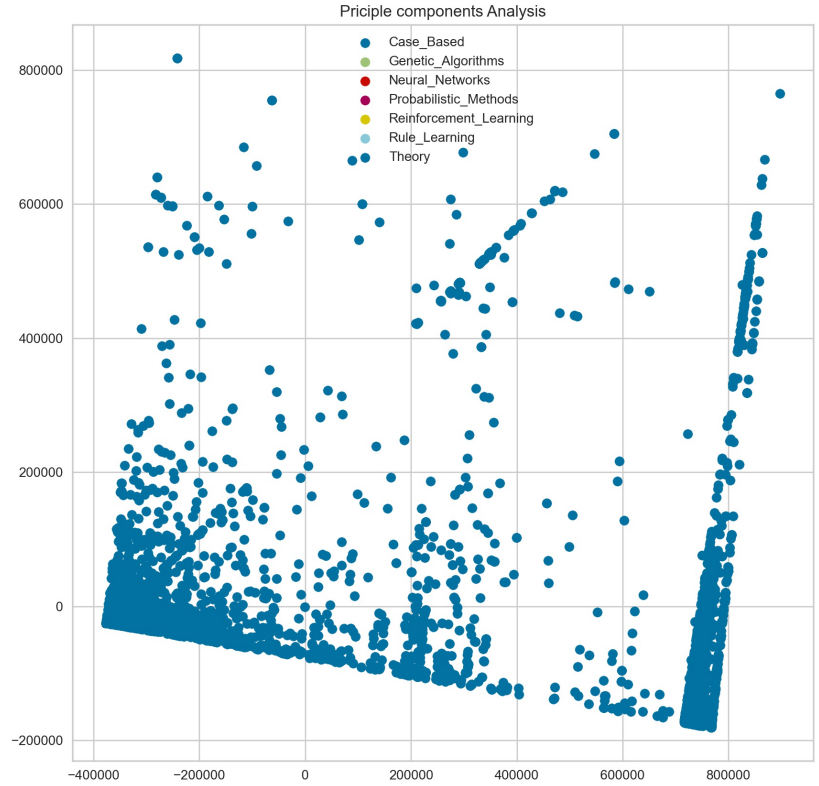


Edge Information



	target	source
0	35	1033
1	35	103482
2	35	103515
3	35	1050679
4	35	1103960
...	...	...
5424	853116	19621
5425	853116	853155
5426	853118	1140289
5427	853155	853118
5428	954315	1155073

5429 rows × 2 columns



Graph structure in data structure

$\mathrm{G}= (\mathrm{V},\mathrm{E})$

V : Vertex set $\mathrm{V} \in R^n$

E : Edge set $\mathrm{E} \in R^n$

$N(v)$: Neighborhood set of node v

$A \in R^{N \times N}$ Adjacency Matrix

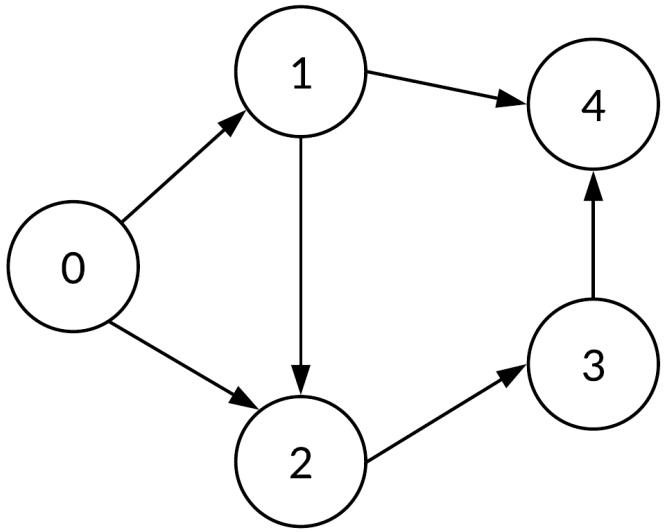
For example,

$\mathrm{V} = \{0,1,2,3,4\}$

$\mathrm{E} = \{(0,1),\ (0,2)\ ,(1,2)\ ,\ (1,4)\ ,(2,3)\ ,(3,4)\}$

$In\ N(1) = \{0\}$

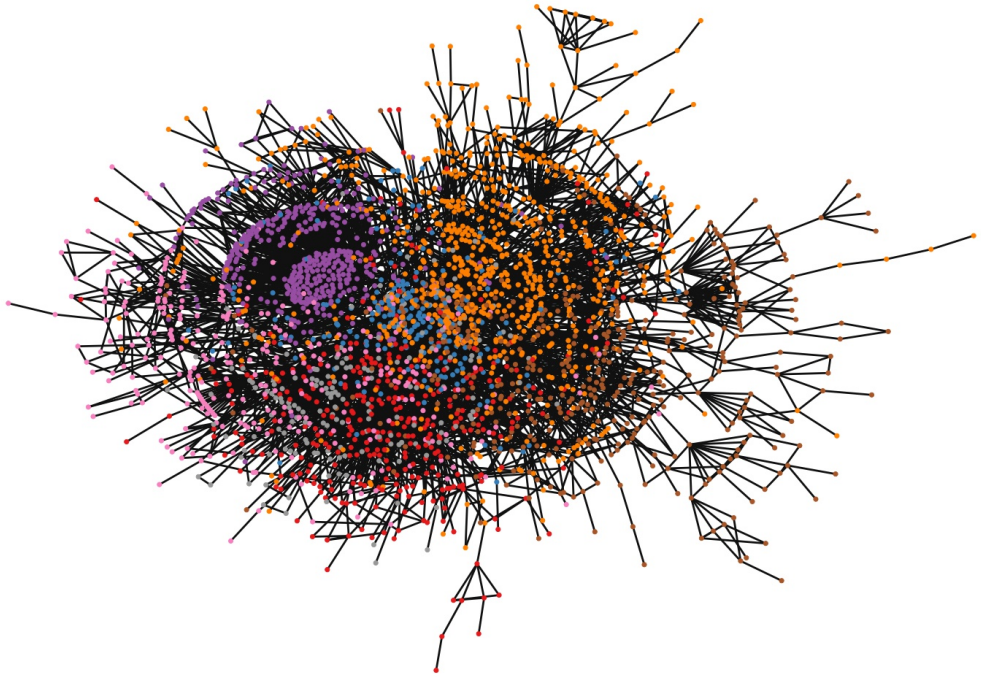
$Out\ N(1) = \{2, 4\}$



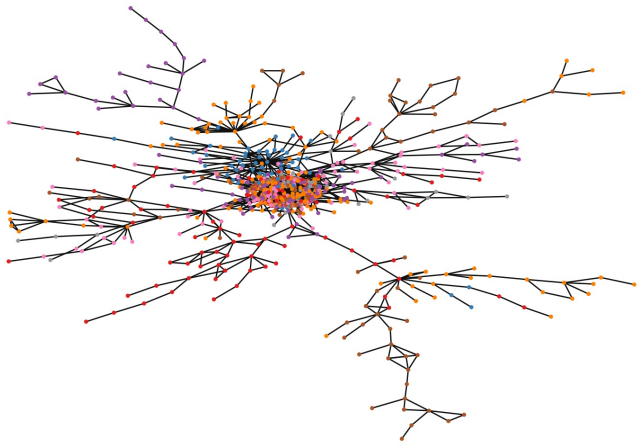
Adjacency Matrix

	0	1	2	3	4
0	0	1	1	0	0
1	0	0	1	0	1
2	0	0	0	1	0
3	0	0	0	0	1
4	0	0	0	0	0

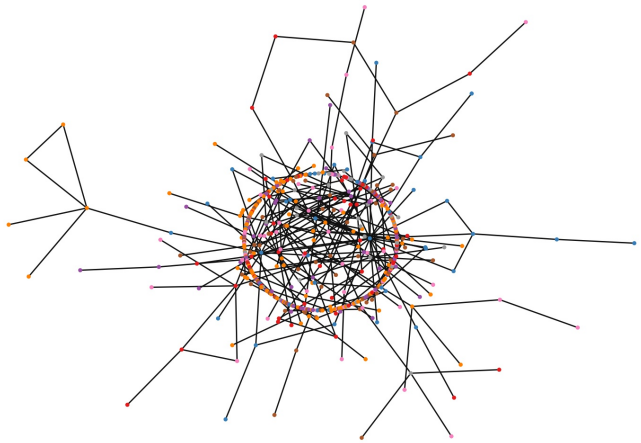
Graph plot in Cora Dataset



Total 2,708 node



Subgraph 1,000 node



Subgraph 500 node

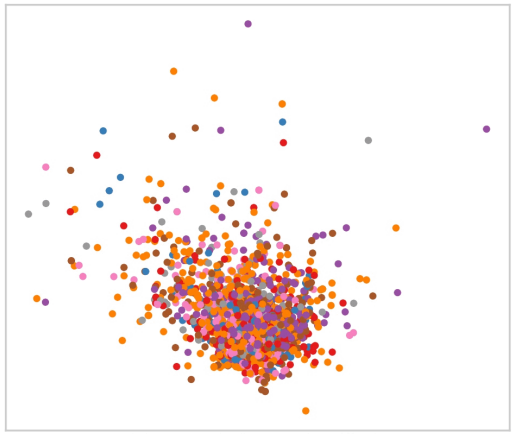
GCN

```python
from torch_geometric.nn import GCNConv,GATConv ,GATv2Conv
class GCN(torch.nn.Module):
    def __init__(self,feature_shape, num_classes):
        super(GCN,self).__init__()
        self.relu = nn.ReLU()
        self.tanh = nn.Tanh()
        self.gcn1 = GCNConv(feature_shape,14) #GNN은 Layer가 얕을 수록 학습 효과가 좋음, 3,4layer 이후는 Residual block필수
        # self.bn   = nn.BatchNorm1d(14)
        self.gcn2 = GCNConv(14,num_classes)


        self.dropout = nn.Dropout(0.5)
        self.fc = nn.Linear(num_classes,2) #
    def forward(self,x,edge_index):
        x = self.gcn1(x,edge_index)
        # x = self.bn(x)
        x = self.relu(x)
        # x = self.tanh(x)
        x = self.dropout(x)
        out = self.gcn2(x,edge_index)
        embeddings =self.tanh(self.fc(self.tanh(out))) #why? maybe mapping the real euclidean space

        return out, embeddings

model = GCN(data.x.shape[-1],cora.num_classes)
_, h = model(data.x, data.edge_index)
```

$$\mathbf{h}_v^{(l)} = \sigma\left(\mathbf{W}^{(l)} \sum_{u \in N(v)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}\right)$$

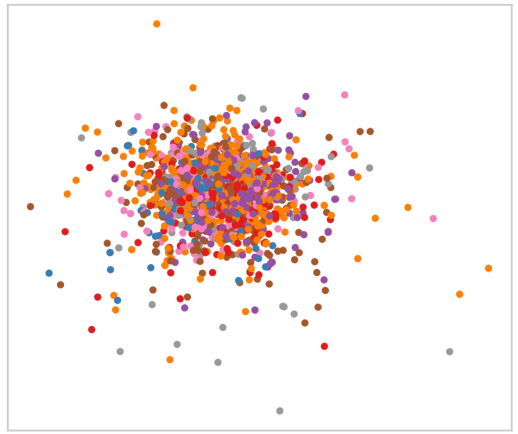
GCN formula

Graph train embedding



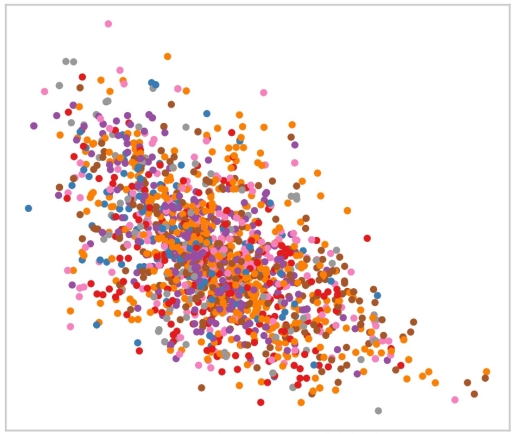
Epoch: 0, Loss: 1.9732
Training Accuracy: 9.24%
Validation Accuracy: 10.09%

GAT_RELU



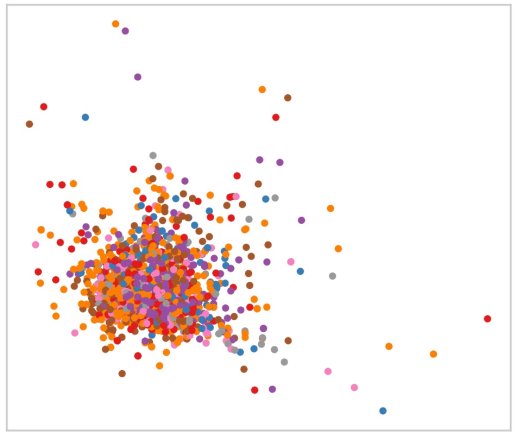
Epoch: 0, Loss: 1.9534
Training Accuracy: 16.86%
Validation Accuracy: 24.50%

GAT_tanh



Epoch: 0, Loss: 2.3233
Training Accuracy: 9.18%
Validation Accuracy: 13.26%

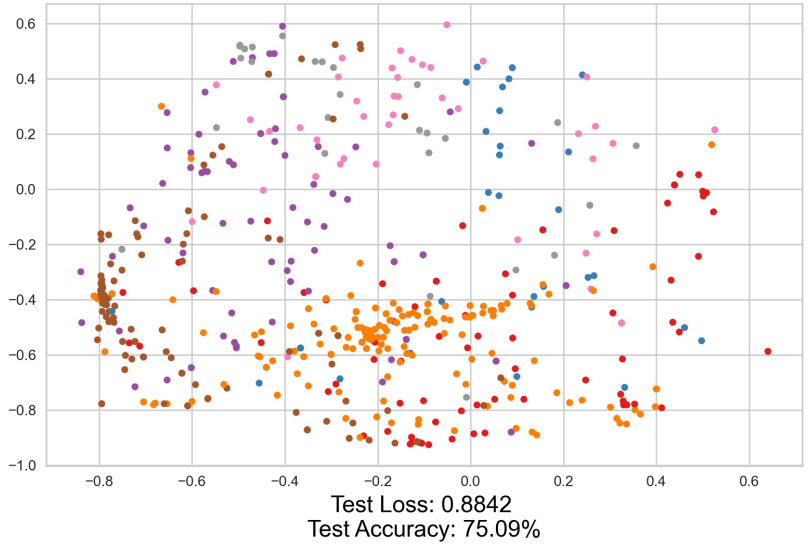
GCN_Relu_Batchnorm



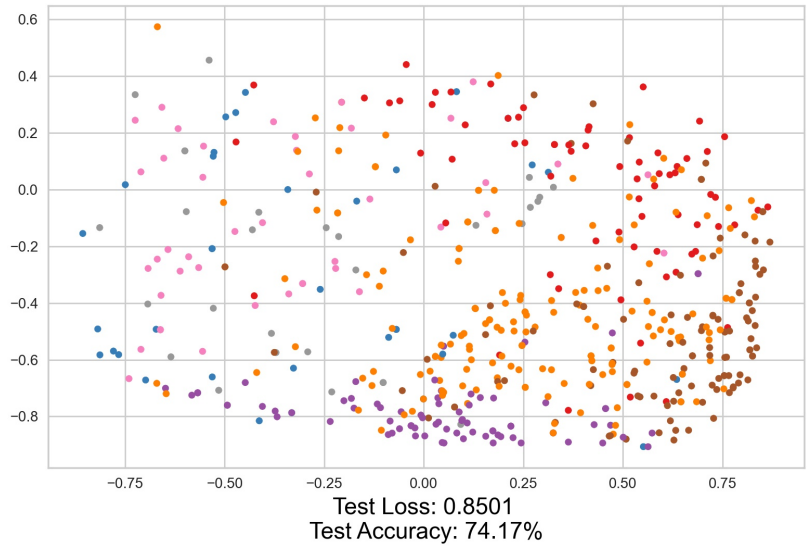
Epoch: 0, Loss: 1.9540
Training Accuracy: 13.68%
Validation Accuracy: 14.70%

GCN_relu

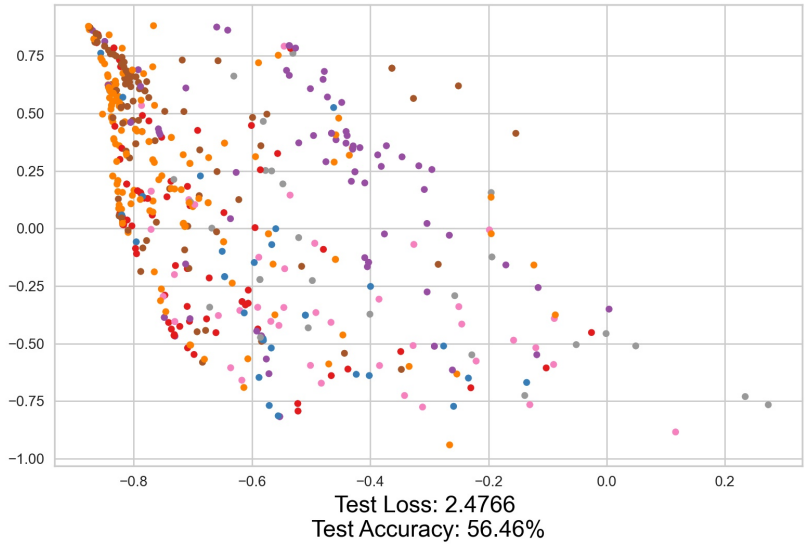
GCN Test Result



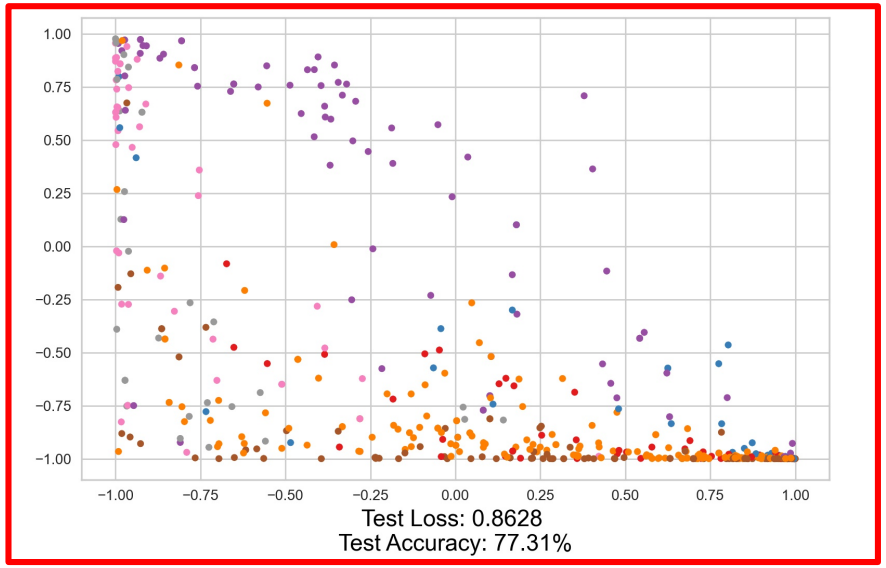
GAT_RELU



GAT_tanh

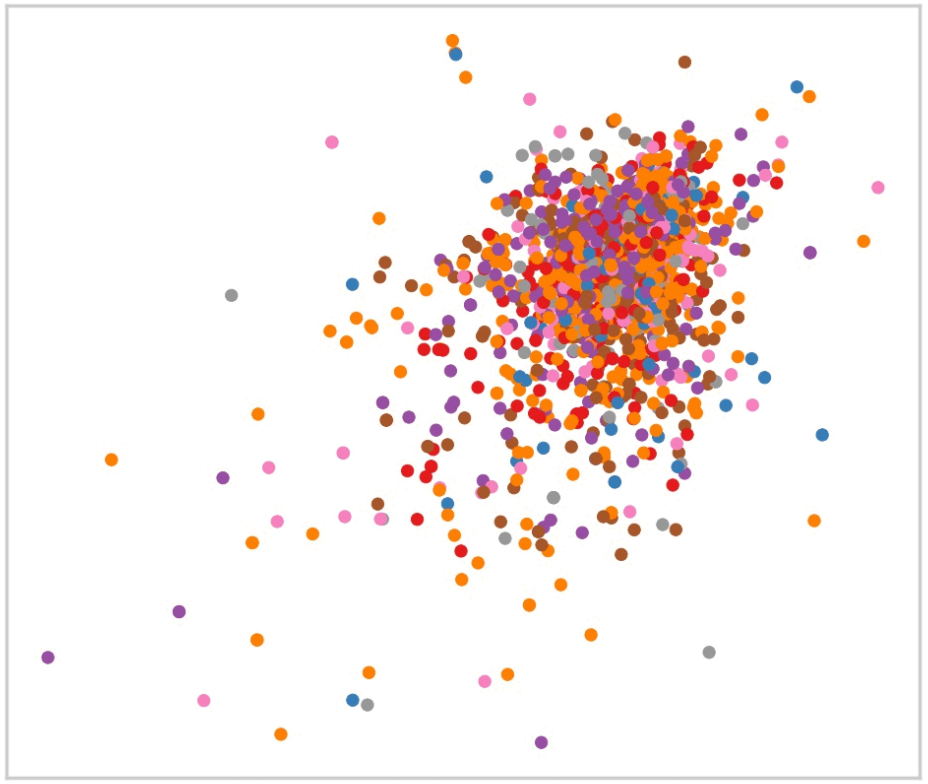


GCN_Relu_Batchnorm

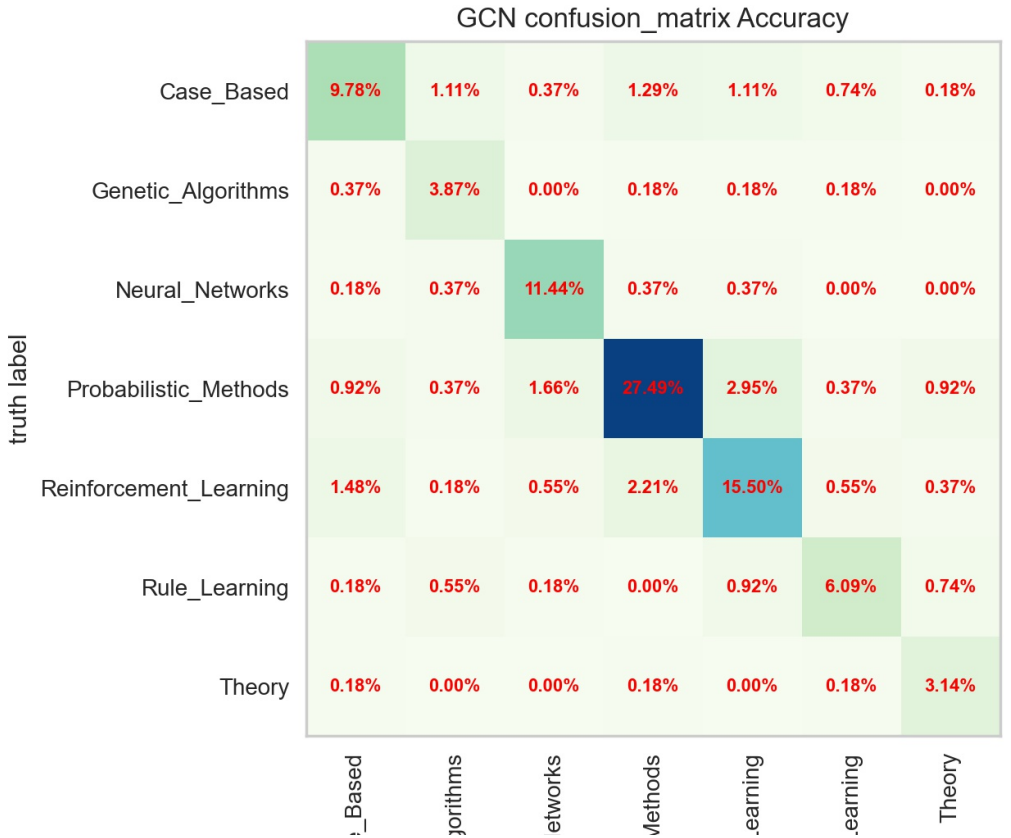


GCN_relu

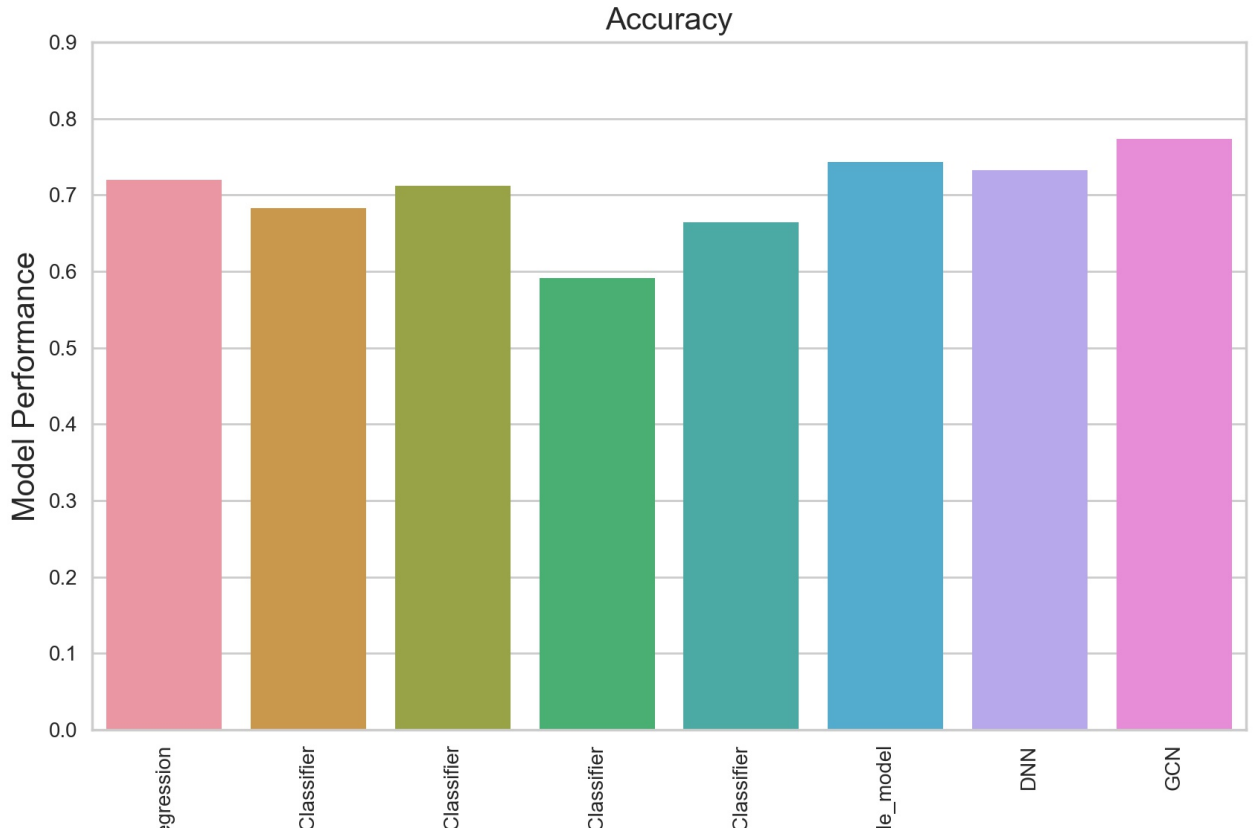
Best Model



Epoch: 0, Loss: 1.9602
Training Accuracy: 12.41%
Validation Accuracy: 34.29%



Conclusion



	Model	Score
0	LogisticRegression	0.720000
1	SGDClassifier	0.683077
2	GradientBoostingClassifier	0.712308
3	DecisionTreeClassifier	0.590769
4	RandomForestClassifier	0.664615
5	ensemble_model	0.743542
6	DNN	0.745387
7	GCN	0.780300