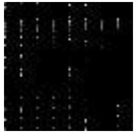
Analysis of Malware Classification

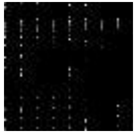
Vrinda Malhotra
Sushant Mane
Niraj Pandkar
Devinesh Singh
William Wang

Dataset

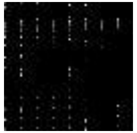
- 4096 Features
- 5 Families
- Principal Component Analysis (PCA)
  - 100 top features
  - Variance
- Clustering Using K-means accuracy: 63.4%
- Scale input to be within 0-1 (divide by 255)



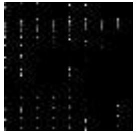
Family A



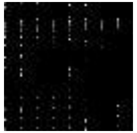
Family B



Family C

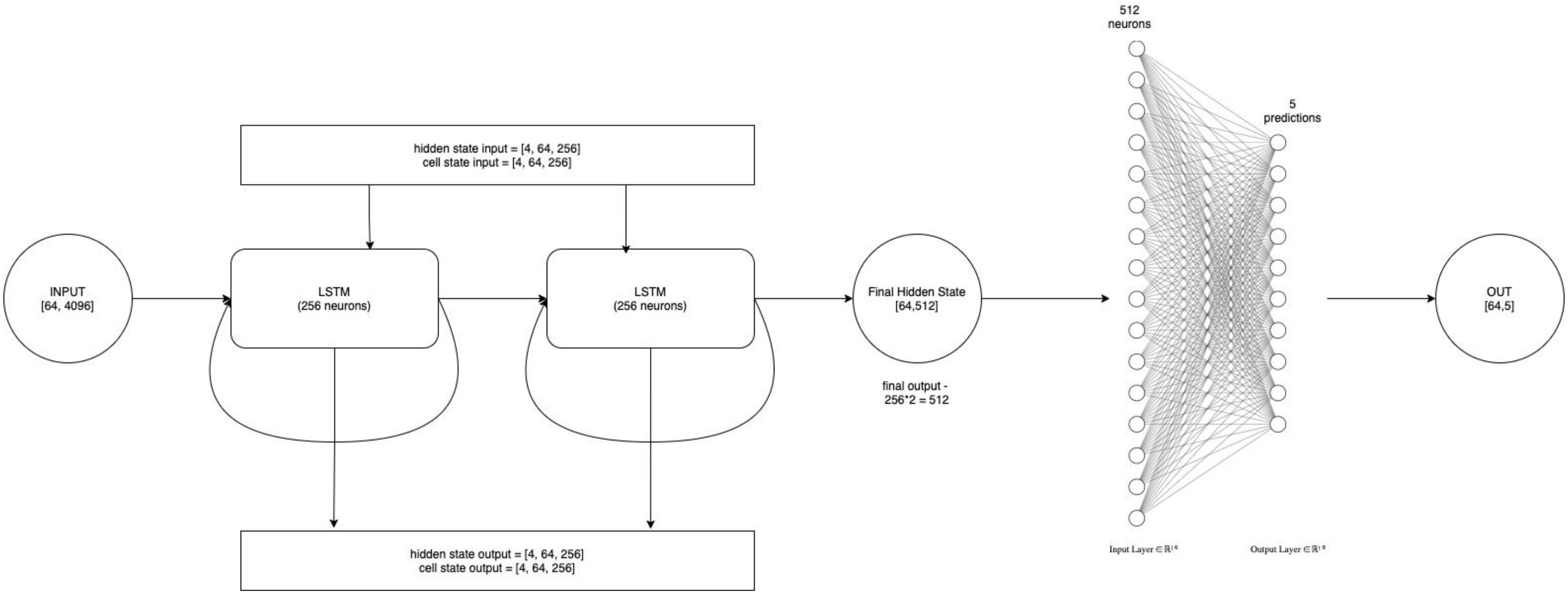


Family D



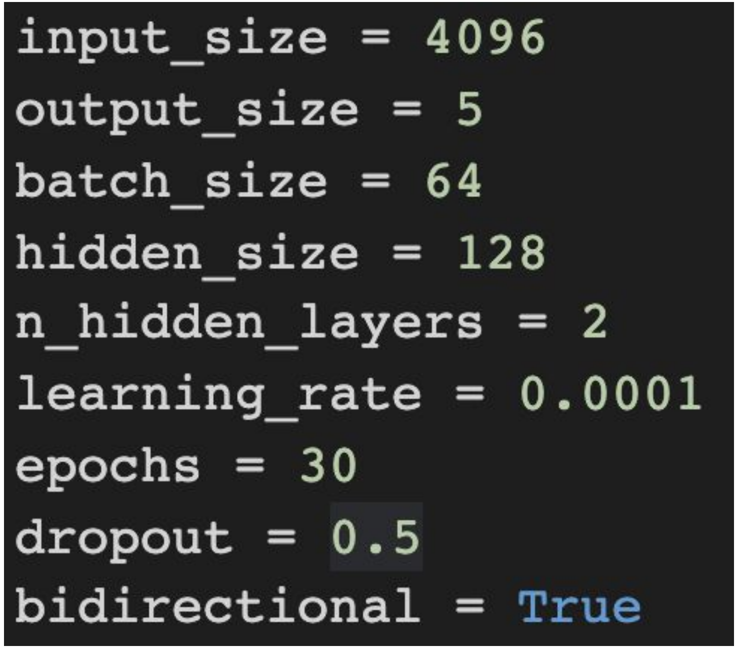
Family E

Problem 1: LSTM - Model Architecture


hidden state input = [4, 64, 256]
cell state input = [4, 64, 256]
INPUT
[64, 4096]
LSTM
(256 neurons)
LSTM
(256 neurons)
Final Hidden State
[64,512]
final output -
256*2 = 512
512
neurons
5
predictions
OUT
[64,5]
hidden state output = [4, 64, 256]
cell state output = [4, 64, 256]
Input Layer $\in \mathbb{R}^{16}$
Output Layer $\in \mathbb{R}^{10}$

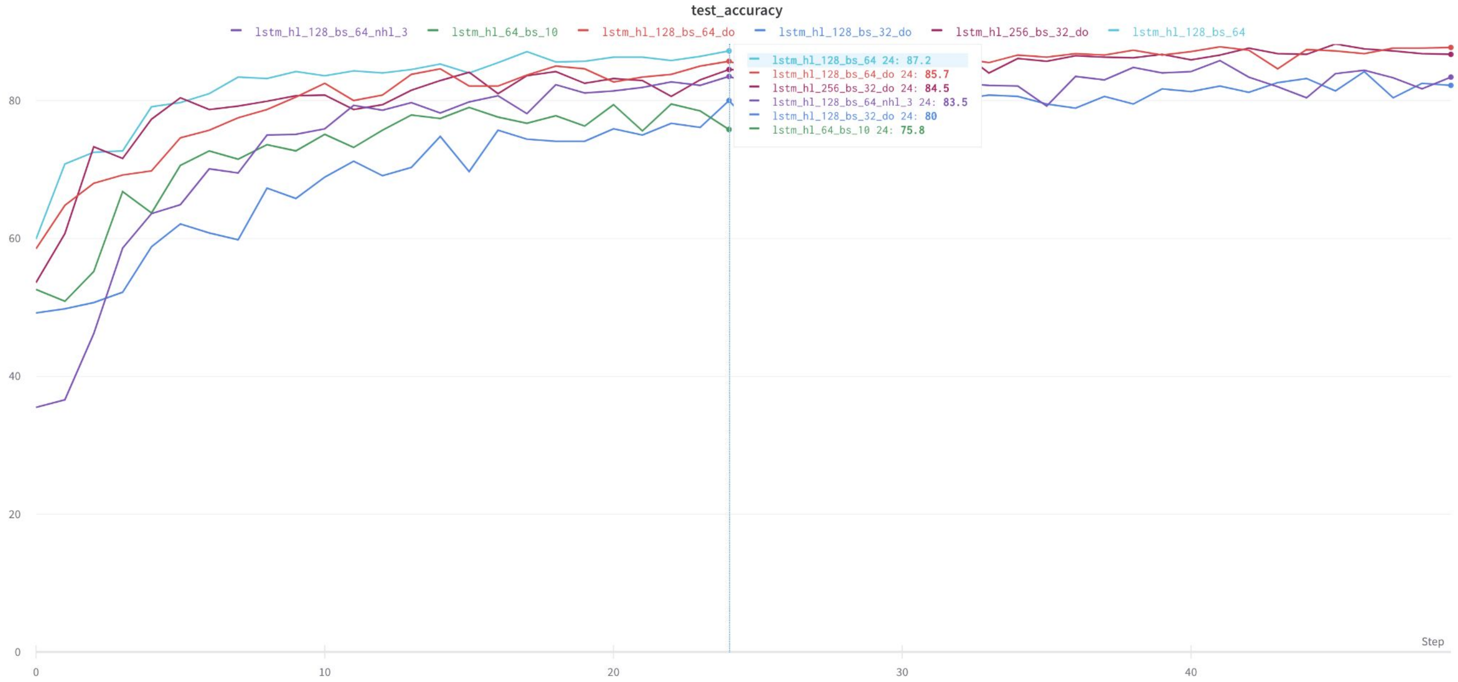
Problem 1: Hyperparameter Tuning

- Batch Size
- Learning Rate
- Bidirectionality
- Softmax Layer
- Number of hidden layers
- Number of neurons in hidden layers
- Dropout

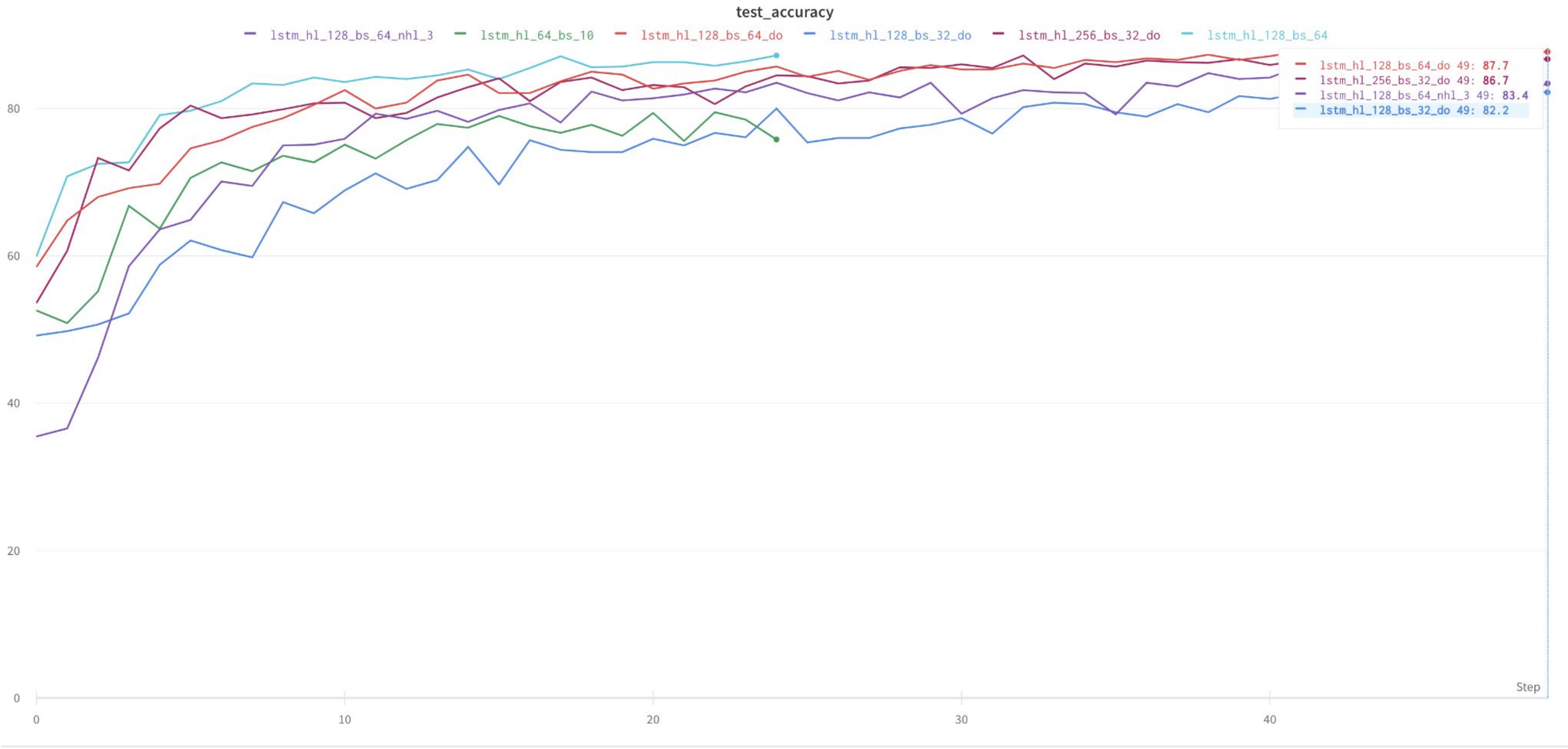


```
input_size = 4096
output_size = 5
batch_size = 64
hidden_size = 128
n_hidden_layers = 2
learning_rate = 0.0001
epochs = 30
dropout = 0.5
bidirectional = True
```

Problem 1: Experiments and Results

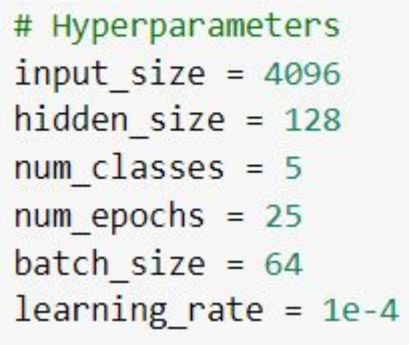


Problem 1: Experiments and Results (cntd.)

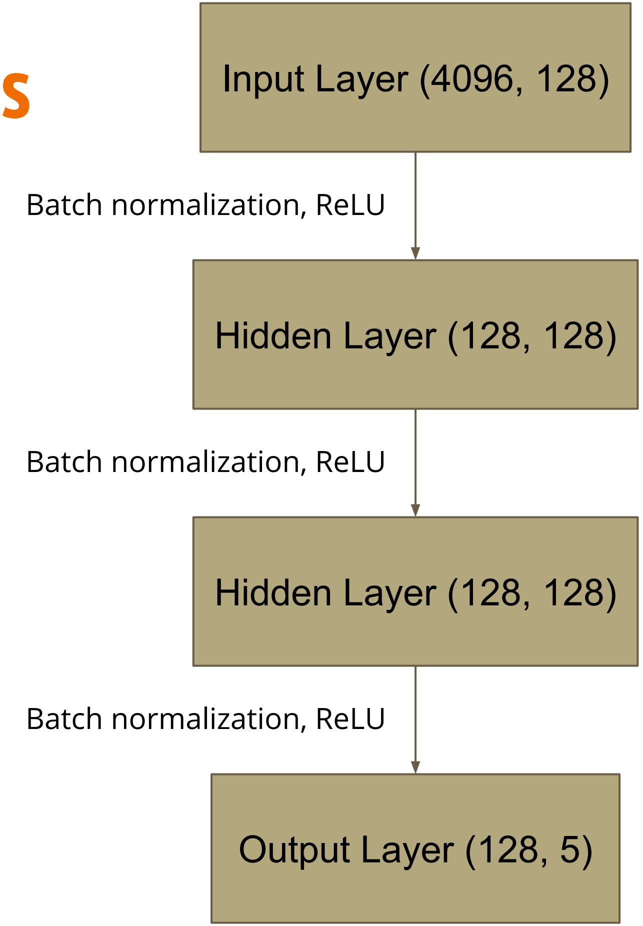


Problem 2: MLP - Structure/Results

- 1 input layer, 2 hidden layers, 1 output layer
- Batch normalization, rectified linear unit (ReLU) used between linear layers
- Best accuracy: 95.4%



```
# Hyperparameters
input_size = 4096
hidden_size = 128
num_classes = 5
num_epochs = 25
batch_size = 64
learning_rate = 1e-4
```



Input Layer (4096, 128)

Batch normalization, ReLU

Hidden Layer (128, 128)

Batch normalization, ReLU

Hidden Layer (128, 128)

Batch normalization, ReLU

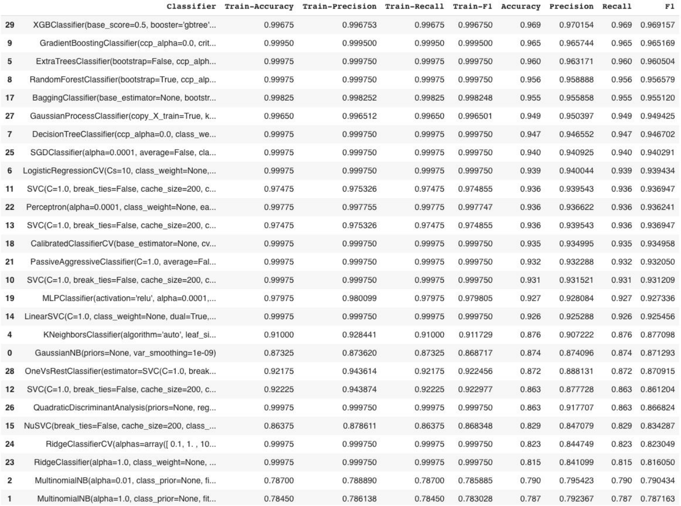
Output Layer (128, 5)

Problem 2: MLP - Observations

- Accuracy did not change significantly after around 25 epochs
- Increase accuracy
  - Include batch normalization (slightly)
  - Increase batch size (slightly)
  - Add additional layer (slightly)
- Decrease accuracy
  - Decrease hidden size (slightly)
  - Increase learning rate (significantly)
  - Include softmax function (significantly)

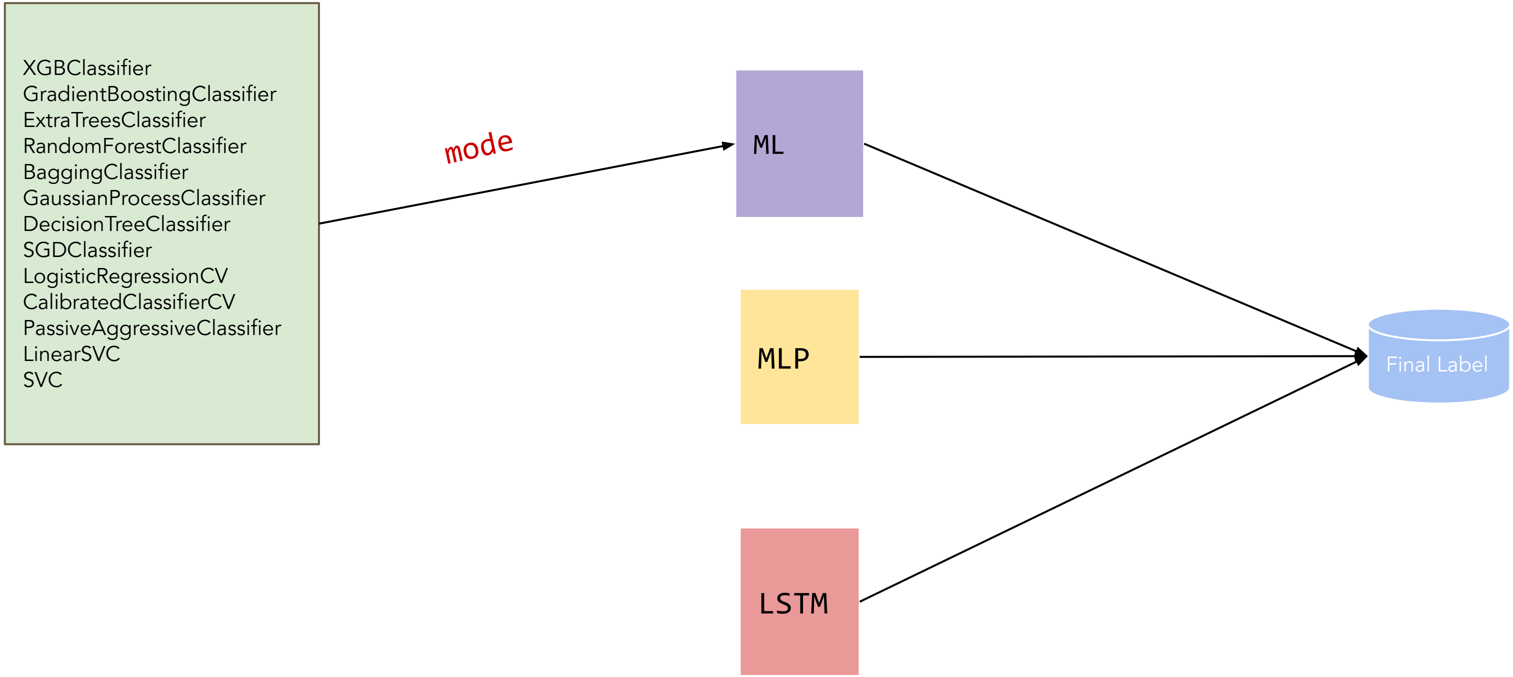
Problem 2: An Ensemble of Classic ML Techniques

- Experimented with a bunch of classifiers
- Used grid search to find optimal parameters
- However, this was causing overfitting
- We got best results with following classifiers
    - XGBClassifier: 96.9%
    - GradientBoostingClassifier: 96.5%
    - ExtraTreesClassifier: 96.0%
    - RandomForestClassifier:95.6%
- Tried UMAP but didn't see any improvements in accuracy except for KNN (90% → 92%)



	Classifier	Train-Accuracy	Train-Precision	Train-Recall	Train-F1	Accuracy	Precision	Recall	F1
29	XGBClassifier(base_score=0.5, booster='gbtree'...	0.99675	0.996753	0.99675	0.996750	0.969	0.970154	0.969	0.969157
9	GradientBoostingClassifier(ccp_alpha=0.0, crit...	0.99950	0.999500	0.99950	0.999500	0.965	0.965744	0.965	0.965169
5	ExtraTreesClassifier(bootstrap=False, ccp_alph...	0.99975	0.999750	0.99975	0.999750	0.960	0.963171	0.960	0.960504
8	RandomForestClassifier(bootstrap=True, ccp_alp...	0.99975	0.999750	0.99975	0.999750	0.956	0.958888	0.956	0.956579
17	BaggingClassifier(base_estimator=None, bootstr...	0.99825	0.998252	0.99825	0.998248	0.955	0.955858	0.955	0.955120
27	GaussianProcessClassifier(copy_X_train=True, k...	0.99650	0.996512	0.99650	0.996501	0.949	0.950397	0.949	0.949425
7	DecisionTreeClassifier(ccp_alpha=0.0, class_we...	0.99975	0.999750	0.99975	0.999750	0.947	0.946552	0.947	0.946702
25	SGDClassifier(alpha=0.0001, average=False, cla...	0.99975	0.999750	0.99975	0.999750	0.940	0.940925	0.940	0.940291
6	LogisticRegressionCV(Cs=10, class_weight=None,...	0.99975	0.999750	0.99975	0.999750	0.939	0.940044	0.939	0.939434
11	SVC(C=1.0, break_ties=False, cache_size=200, c...	0.97475	0.975326	0.97475	0.974855	0.936	0.939543	0.936	0.936947
22	Perceptron(alpha=0.0001, class_weight=None, ea...	0.99775	0.997755	0.99775	0.997747	0.936	0.936622	0.936	0.936241
13	SVC(C=1.0, break_ties=False, cache_size=200, c...	0.97475	0.975326	0.97475	0.974855	0.936	0.939543	0.936	0.936947
18	CalibratedClassifierCV(base_estimator=None, cv...	0.99975	0.999750	0.99975	0.999750	0.935	0.934995	0.935	0.934958
21	PassiveAggressiveClassifier(C=1.0, average=Fal...	0.99975	0.999750	0.99975	0.999750	0.932	0.932288	0.932	0.932050
10	SVC(C=1.0, break_ties=False, cache_size=200, c...	0.99975	0.999750	0.99975	0.999750	0.931	0.931521	0.931	0.931209
19	MLPClassifier(activation='relu', alpha=0.0001,...	0.97975	0.980099	0.97975	0.979805	0.927	0.928084	0.927	0.927336
14	LinearSVC(C=1.0, class_weight=None, dual=True,...	0.99975	0.999750	0.99975	0.999750	0.926	0.925288	0.926	0.925456
4	KNeighborsClassifier(algorithm='auto', leaf_si...	0.91000	0.928441	0.91000	0.911729	0.876	0.907222	0.876	0.877098
0	GaussianNB(priors=None, var_smoothing=1e-09)	0.87325	0.873620	0.87325	0.868717	0.874	0.874096	0.874	0.871293
28	OneVsRestClassifier(estimator=SVC(C=1.0, break...	0.92175	0.943614	0.92175	0.922456	0.872	0.888131	0.872	0.870915
12	SVC(C=1.0, break_ties=False, cache_size=200, c...	0.92225	0.943874	0.92225	0.922977	0.863	0.877728	0.863	0.861204
26	QuadraticDiscriminantAnalysis(priors=None, reg...	0.99975	0.999750	0.99975	0.999750	0.863	0.917707	0.863	0.866824
15	NuSVC(break_ties=False, cache_size=200, class_...	0.86375	0.878611	0.86375	0.868348	0.829	0.847079	0.829	0.834287
24	RidgeClassifierCV(alphas=array([0.1, 1. , 10...	0.99975	0.999750	0.99975	0.999750	0.823	0.844749	0.823	0.823049
23	RidgeClassifier(alpha=1.0, class_weight=None, ...	0.99975	0.999750	0.99975	0.999750	0.815	0.841099	0.815	0.816050
2	MultinomialNB(alpha=0.01, class_prior=None, fi...	0.78700	0.788890	0.78700	0.785885	0.790	0.795423	0.790	0.790434
1	MultinomialNB(alpha=1.0, class_prior=None, fit...	0.78450	0.786138	0.78450	0.783028	0.787	0.792367	0.787	0.787163

Problem 3: Challenge Results



THANK YOU