

Universidade de São Paulo
Instituto de Física de São Carlos

SCC00277-201-2021 Project 3

Éverton Luís Mendes da Silva (10728171)

Contents

1	Introduction	2
2	Question One	2
2.1	Item A	2
2.2	Item B	3
3	Question Two	3
4	Question Three	5
4.1	Scikit-Learn	5
4.1.1	LGBMClassifier	24
4.1.2	GaussianNB	25
4.1.3	DecisionTreeClassifier	26
4.1.4	RandomForestClassifier	26
4.1.5	AdaBoostClassifier	27
4.1.6	NearestCentroid	27
4.1.7	SVC	28
4.1.8	PassiveAggressiveClassifier	28
4.1.9	RidgeClassifierFalse	29
4.1.10	RidgeClassifierPositive	29
4.1.11	Perceptron	30
4.2	Keras in TensorFlow	30
4.3	Submissions	37
5	Question Four	38
6	Reference	39

1 Introduction

Undoubtedly, one of the fundamental factors in the evolution of societies was the use of writing as a form of communication. That is, with writing it was possible to keep the knowledge obtained during life for future generations. In this way, several areas of commerce and population management could be analyzed to make a counterpoint of the phenomena that occurred in the past with the current ones, for example we have the taxation of taxes by the Phoenicians. In view of this, currently, registered language is the basis of modern computers along with mathematical logic. Furthermore, we need to create computational tools that know how to analyze our language, differentiating letters or even Arabic numbers. In this project, the basic idea is to know how to differentiate numeric digits through optical character recognition (OCR).

2 Question One

2.1 Item A

First, we will need to visualize the example of each type of digit that will be provided for in this project. The data from the competition Digit Recognizer is in csv format with 785 columns, one for the target and the others representing a 28x28 image(784).



Figure 1: Digits

2.2 Item B

Second, let's look at the imbalance of the digits through the histogram shown below.

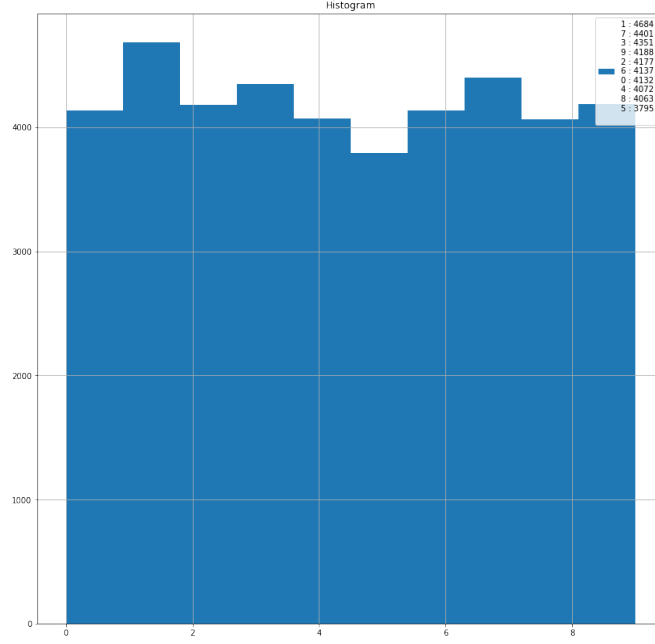


Figure 2: Histogram of Digits

3 Question Two

In this second section it is necessary to choose those pixels that will not be taken into account for the predictions. In this way, we will remove those columns that do not provide any information about the problem, that is, we will consider entropy, which is the key measure of information theory. Entropy gives us the degree of causality of a certain phenomenon, that is, the indeterminacy that a variable can have. For example, the greater the difficulty of predicting a certain variable, the greater its entropy and, respectively, the information contained in it.

$$H(X) = - \sum_{i=1}^m p_i \log_2(p_i) \quad (1)$$

$$\sum_{i=1}^m p_i = 1 \quad (2)$$

For this project, the only variables that will be removed are those that have information equal to zero, that is, during all samples they always have the same value. For this, we can consider the standard deviation of the variables, those with zero value have only one p_i with probability 1.

$$\sigma = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{N}} \quad (3)$$

$$\sigma = 0 \rightarrow p = 1 \rightarrow H = 0$$

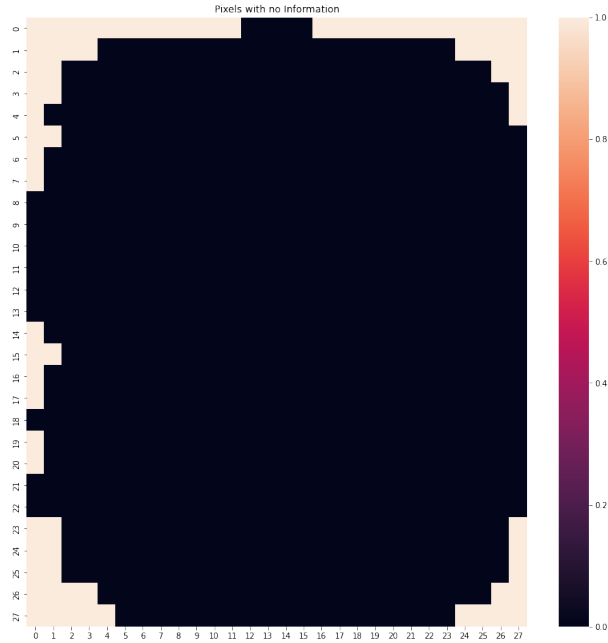


Figure 3: No information

```

1 def drop_no_information(df):
2     '''Drop columns with no information in a DataFrame
3     Args:
4         df, DataFrame
5     Return:
6         df[columns_to_keep], new DataFrame with relevant
information
7         columns_to_keep, columns with relevant information
8     '''
9     columns_to_keep=[column for column, boolean in (df.drop(
columns='label').std()!=0).items() if boolean]
10

```

```

11     sn.heatmap((df.drop(columns='label').std()==0).values.
reshape(28, 28))
12     plt.title("Pixels with no Information")
13     plt.savefig("/content/drive/MyDrive/
Data_Science_Competitions/Project3/Data/img/Digits/
Pixels_no_inf.png")
14
15     return df[columns_to_keep], columns_to_keep

```

Listing 1: Drop columns with no Information

4 Question Three

In this section we will choose to train the hyperparameters of several machine learning models and choose those with the highest accuracy value for digit classification. Hyperparameter searches were performed with two types of Bayesian optimizations, one using Scikit-Learn(Bayes Search CV) and another using TensorFlow(keras tuner).

4.1 Scikit-Learn

For the use of BayesSearchCV, the models below were chosen:

- LGBMClassifier
- GaussianNB
- DecisionTreeClassifier
- RandomForestClassifier
- AdaBoostClassifier
- NearestCentroid
- SVC
- PassiveAggressiveClassifier
- RidgeClassifierFalse
- RidgeClassifierPositive
- Perceptron

Before these models were trained, there was a feature engineering, in this process there was a removal of pixels that did not bring any information, as mentioned in the previous section.

The training of these models is divided into four different codes:

- 'model_classifiers'(contains the hyperparameters that will be fetched)
- 'data_preprocessing'(data normalization)
- 'Train_Test_model_Scikit'(trains the models)
- 'kaggle_submission'(create kaggle submission)
- 'Main_Scikit' (use the previous codes)

```
1 from skopt.space import Real, Categorical, Integer
2
3 from sklearn.linear_model import RidgeClassifier
4 from sklearn.linear_model import Perceptron
5 from sklearn.linear_model import SGDClassifier
6 from sklearn.discriminant_analysis import
   QuadraticDiscriminantAnalysis
7 from sklearn.linear_model import PassiveAggressiveClassifier
8 from sklearn.svm import SVC
9 from sklearn.neighbors import KNeighborsClassifier
10 from sklearn.neighbors import NearestCentroid
11 from sklearn.naive_bayes import GaussianNB
12 from sklearn.tree import DecisionTreeClassifier
13 from sklearn.ensemble import RandomForestClassifier
14 from sklearn.ensemble import AdaBoostClassifier
15 from sklearn.ensemble import GradientBoostingClassifier
16 from sklearn.ensemble import HistGradientBoostingClassifier
17 from xgboost import XGBClassifier, XGBRFClassifier
18 from lightgbm import LGBMClassifier
19
20
21 lgbm_clf={
22     'model':Categorical([LGBMClassifier()]),
23     'model__boosting_type':Categorical(['gbdt', 'dart', 'goss
24     ']),
25     'model__num_leaves':Integer(8, 128, 'uniform'),
26     'model__max_depth':Integer(4, 128, 'uniform'),
27     'model__learning_rate':Real(0.001, 0.4, 'uniform'),
28     'model__n_estimators':Integer(100, 150, 'uniform'),
29     'model__subsample_for_bin':Integer(200000, 250000, '
   uniform'),
30     'model__min_split_gain':Real(0, 0.1, 'uniform'),
```

```

30     'model__min_child_weight':Real(0.000001, 0.001, 'uniform
    '),
31     'model__min_child_samples':Integer(12, 40, 'uniform'),
32     'model__subsample':Real(0.8, 1, 'uniform'),
33     'model__colsample_bytree':Real(0.8, 1, 'uniform'),
34 }
35
36 xgbrf_gbtrees_clf={
37     'model':Categorical([XGBRFClassifier()]),
38     'model__booster':Categorical(['gbtree']),
39     'model__eta':Real(0.01, 0.99, 'uniform'),
40     'model__gamma':Integer(0, 4, 'uniform'),
41     'model__max_depth':Integer(4, 128, 'uniform'),
42     'model__min_child_weight':Integer(0, 4, 'uniform'),
43     'model__max_delta_step':Integer(0, 7, 'uniform'),
44     'model__subsample':Real(0.01, 0.99, 'uniform'),
45     'model__colsample_bytree':Real(0.01, 0.99, 'uniform'),
46     'model__colsample_bylevel':Real(0.01, 0.99, 'uniform'),
47     'model__colsample_bynode':Real(0.01, 0.99, 'uniform'),
48     'model__n_estimators':Integer(100, 120, 'uniform'),
49     'model__objective':Categorical(['reg:squarederror', 'reg:
    squaredlogerror', 'reg:logistic', 'reg:pseudohubererror'])
    ,
50 }
51
52
53
54
55
56
57 xgb_gbtrees_clf={
58     'model':Categorical([XGBClassifier()]),
59     'model__booster':Categorical(['gbtree']),
60     'model__eta':Real(0.01, 0.99, 'uniform'),
61     'model__gamma':Integer(0, 4, 'uniform'),
62     'model__max_depth':Integer(4, 128, 'uniform'),
63     'model__min_child_weight':Integer(0, 4, 'uniform'),
64     'model__max_delta_step':Integer(0, 7, 'uniform'),
65     'model__subsample':Real(0.01, 0.99, 'uniform'),
66     'model__colsample_bytree':Real(0.01, 0.99, 'uniform'),
67     'model__colsample_bylevel':Real(0.01, 0.99, 'uniform'),
68     'model__colsample_bynode':Real(0.01, 0.99, 'uniform'),
69     'model__n_estimators':Integer(100, 120, 'uniform'),
70     'model__objective':Categorical(['reg:squarederror', 'reg:
    squaredlogerror', 'reg:logistic', 'reg:pseudohubererror'])
    ,
71 }
72
73

```



```

74
75 gaussianNB_clf={
76     'model':Categorical([GaussianNB()]),
77     'model__var_smoothing':Real(0.0000000001, 0.0000001, '
uniform')
78 }
79
80 Decision_Tree_clf={
81     'model':Categorical([DecisionTreeClassifier()]),
82     'model__criterion':Categorical(['gini', 'entropy']),
83     'model__splitter':Categorical(['best', 'random']),
84     'model__min_samples_split':Integer(2, 8, 'uniform'),
85     'model__max_features':Categorical(['sqrt', 'log2']),
86     'model__max_depth':Integer(4, 128, 'uniform'),
87     'model__min_samples_leaf':Integer(1,4,'uniform'),
88 }
89
90 Random_Forest_clf={
91     'model':Categorical([RandomForestClassifier()]),
92     'model__n_estimators':Integer(100, 120, 'uniform'),
93     'model__criterion':Categorical(['gini', 'entropy']),
94     'model__max_depth':Integer(4, 128, 'uniform'),
95     'model__min_samples_split':Integer(2, 8, 'uniform'),
96     'model__min_samples_leaf':Integer(1,4,'uniform'),
97     'model__max_features':Categorical(['sqrt', 'log2']),
98     'model__max_samples':Real(0.01, 0.99, 'uniform')
99 }
100 AdaBoost_clf={
101     'model':Categorical([AdaBoostClassifier()]),
102     'model__n_estimators':Integer(40, 100, 'uniform'),
103     'model__learning_rate':Real(0.8, 1.2, 'uniform'),
104     'model__algorithm':Categorical(['SAMME', 'SAMME.R']),
105 }
106
107 gradientBooster_clf={
108     'model':Categorical([GradientBoostingClassifier()]),
109     'model__loss':Categorical(['deviance']),
110     'model__learning_rate':Real(0.0001, 0.1, 'uniform'),
111     'model__n_estimators':Integer(80, 100, 'uniform'),
112     'model__subsample':Real(0.7, 1, 'uniform'),
113     'model__criterion':Categorical(['friedman_mse', '
squared_error']),
114     'model__min_samples_split':Integer(2, 8, 'uniform'),
115     'model__min_samples_leaf':Integer(1,4,'uniform'),
116     'model__max_depth':Integer(2, 64, 'uniform'),
117     'model__max_features':Categorical(['sqrt', 'log2']),
118     'model__tol':Real(0.000001, 0.01, 'uniform')
119 }
120

```

```

121 HistGradientBooster_clf={
122     'model':Categorical([HistGradientBoostingClassifier()]),
123     'model__loss':Categorical(['categorical_crossentropy']),
124     'model__learning_rate':Real(0.0001, 0.1, 'uniform'),
125     'model__max_iter':Integer(500, 2000, 'uniform'),
126     'model__max_leaf_nodes':Integer(20, 50, 'uniform'),
127     'model__max_depth':Integer(2, 64, 'uniform'),
128     'model__min_samples_leaf':Integer(1,20,'uniform'),
129     'model__tol':Real(0.00000001, 0.0001, 'uniform'),
130 }
131
132 knn_clf={
133     'model':Categorical([NearestCentroid()]),
134     'model__metric':Categorical(['euclidean', 'manhattan', '
chebyshev', 'minkowski'])
135 }
136
137 Knn_clf={
138     'model':Categorical([KNeighborsClassifier()]),
139     'model__weights':Categorical(['uniform', 'distance']),
140     'model__algorithm':Categorical(['kd_tree', 'brute']),
141     'model__leaf_size':Integer(10, 50, 'uniform'),
142     'model__p':Integer(1, 4, 'uniform'),
143     'model__metric':Categorical(['euclidean', 'manhattan', '
chebyshev', 'minkowski']),
144     'model__n_neighbors':Integer(3, 6, 'uniform')
145 }
146 #'ball_tree',
147
148 svc_clf={
149     'model':Categorical([SVC()]),
150     'model__kernel':Categorical(['linear', 'poly', 'rbf', '
sigmoid']),
151     'model__degree':Integer(2, 10, 'uniform'),
152     'model__gamma':Categorical(['scale', 'auto']),
153     'model__tol':Real(0.0000001, 0.01, 'uniform'),
154 }
155
156 QDA_clf={
157     'model':Categorical([QuadraticDiscriminantAnalysis()]),
158     'model__tol':Real(0.0000001, 0.01, 'uniform')
159 }
160
161 PassiveAggressive_clf={
162     'model': Categorical([PassiveAggressiveClassifier()]),
163     'model__max_iter':Integer(1000, 10000, 'uniform'),
164     'model__tol':Real(0.0000001, 0.01, 'uniform'),
165     'model__C': Real(0.01, 0.99, 'uniform'),
166     'model__loss': Categorical(['hinge', 'squared_hinge'])

```

```

167 }
168
169
170 ridge_clf_positive = {
171     'model': Categorical([RidgeClassifier(positive=True)]),
172     'model__tol': Real(0.0000001, 0.001, 'uniform'),
173     'model__alpha': Real(0.00001, 1, 'uniform'),
174     'model__max_iter': Integer(1000, 10000, 'uniform')
175 }
176
177 ridge_clf_false = {
178     'model': Categorical([RidgeClassifier(positive=False)]),
179     'model__solver': Categorical(['svd', 'cholesky', '
sparse_cg', 'sag', 'saga']),
180     'model__tol': Real(0.0000001, 0.001, 'uniform'),
181     'model__alpha': Real(0.00001, 1, 'uniform'),
182     'model__max_iter': Integer(1000, 10000, 'uniform')
183 }
184
185 perceptron_clf = {
186     'model': Categorical([Perceptron( fit_intercept=False)]),
187     'model__penalty': Categorical(['l2', 'l1', 'elasticnet'])
188     ,
189     'model__alpha': Real(0.00000001, 0.001, 'uniform'),
190     'model__l1_ratio': Real(0.01, 0.99, 'uniform'),
191     'model__max_iter': Integer(1000, 10000, 'uniform'),
192     'model__tol': Real(0.0000001, 0.001, 'uniform'),
193 }
194
195 sgd_clf = {
196     'model': Categorical([SGDClassifier(fit_intercept=False)
197 ]),
198     'model__loss': Categorical(['hinge', 'log', '
modified_huber', 'squared_hinge', 'perceptron', '
squared_error', 'huber', 'epsilon_insensitive', '
squared_epsilon_insensitive']),
199     'model__alpha': Real(0.00000001, 0.001, 'uniform'),
200     'model__max_iter': Integer(1000, 10000, 'uniform'),
201     'model__epsilon': Real(0.0000001, 0.001, 'uniform'),
202     'model__power_t': Real(0.01, 0.99, 'uniform'),
203     'model__eta0': Real(0.01, 0.99, 'uniform'),
204     'model__warm_start': Categorical([True, False]),
205     'model__tol': Real(0.0000001, 0.001, 'uniform'),
206     'model__penalty': Categorical(['l2', 'l1', 'elasticnet'])
207     ,
208     'model__l1_ratio': Real(0.01, 0.99, 'uniform'),
209     'model__learning_rate': Categorical(['constant', 'optimal
', 'invscaling', 'adaptive']),

```

```

208 }
209
210
211
212
213 '''
214
215 Decision_Tree_reg={
216     'model':Categorical([DecisionTreeRegressor()]),
217     'model_criterion':Categorical(['squared_error', '
friedman_mse', 'absolute_error', 'poisson']),
218     'model_splitter':Categorical(['best', 'random']),
219     'model_min_samples_split':Integer(2, 8, 'uniform'),
220     'model_max_features':Categorical(['sqrt', 'log2']),
221     'model_max_depth':Integer(4, 128, 'uniform'),
222     'model_min_samples_leaf':Integer(1,4,'uniform'),
223 }
224
225 Random_Forest_reg={
226     'model':Categorical([RandomForestRegressor()]),
227     'model_n_estimators':Integer(100, 300, 'uniform'),
228     'model_criterion':Categorical(['squared_error', '
absolute_error', 'poisson']),
229     'model_max_depth':Integer(4, 128, 'uniform'),
230     'model_min_samples_split':Integer(2, 8, 'uniform'),
231     'model_min_samples_leaf':Integer(1,4,'uniform'),
232     'model_max_features':Categorical(['sqrt', 'log2']),
233     'model_max_samples':Real(0.01, 0.99, 'uniform')
234 }
235
236
237 AdaBoost_reg={
238     'model':Categorical([AdaBoostRegressor()]),
239     'model_base_estimator_criterion':Categorical(['
squared_error', 'friedman_mse', 'absolute_error', 'poisson
']),
240     'model_base_estimator_splitter':Categorical(['best', '
random']),
241     'model_base_estimator_min_samples_split':Integer(2, 8, '
uniform'),
242     'model_base_estimator_max_features':Categorical(['sqrt',
'log2']),
243     'model_base_estimator_max_depth':Integer(4, 128, 'uniform
'),
244     'model_n_estimators':Integer(40, 70, 'uniform'),
245     'model_learning_rate':Real(1, 3, 'uniform'),
246     'model_loss':Categorical(['linear', 'square', '
exponential'])
247 }

```

```

248
249
250 GradientBoosting_reg={
251     'model':Categorical([GradientBoostingRegressor()]),
252     'model_loss':Categorical(['squared_error', '
absolute_error', 'huber', 'quantile']),
253     'model_learning_rate':Real(0.001, 0.4, 'uniform'),
254     'model_n_estimators':Integer(100, 250, 'uniform'),
255     'model_criterion':Categorical(['friedman_mse', '
squared_error', 'mse', 'mae']),
256     'model_min_samples_split':Integer(2, 8, 'uniform'),
257     'model_min_samples_leaf':Integer(1,4,'uniform'),
258     'model_max_depth':Integer(4, 128, 'uniform'),
259     'model_max_features':Categorical(['sqrt', 'log2']),
260     'model_alpha':Real(0.1, 0.9, 'uniform'),
261     'model_tol':Real(0.0000001, 0.001, 'uniform')
262 }
263
264
265 xgb_gbtreg={
266     'model':Categorical([XGBRegressor()]),
267     'model_booster':Categorical(['gbtree']),
268     'model_eta':Real(0.01, 0.99, 'uniform'),
269     'model_gamma':Integer(0, 4, 'uniform'),
270     'model_max_depth':Integer(4, 128, 'uniform'),
271     'model_min_child_weight':Integer(0, 4, 'uniform'),
272     'model_max_delta_step':Integer(0, 7, 'uniform'),
273     'model_subsample':Real(0.01, 0.99, 'uniform'),
274     'model_colsample_bytree':Real(0.01, 0.99, 'uniform'),
275     'model_colsample_bylevel':Real(0.01, 0.99, 'uniform'),
276     'model_colsample_bynode':Real(0.01, 0.99, 'uniform'),
277     'model_n_estimators':Integer(100, 300, 'uniform'),
278     'model_objective':Categorical(['reg:squarederror', 'reg:
squaredlogerror', 'reg:logistic', 'reg:pseudohubererror'])
279     ,
    'model_eval_metric':Categorical(['rmse', 'rmsle', 'mae',
'mape', 'mphe'])
280 }
281
282
283
284 xgb_dart_reg={
285     'model':Categorical([XGBRegressor()]),
286     'model_booster':Categorical(['dart']),
287     'model_eta':Real(0.01, 0.99, 'uniform'),
288     'model_gamma':Integer(0, 4, 'uniform'),
289     'model_max_depth':Integer(4, 128, 'uniform'),
290     'model_min_child_weight':Integer(0, 4, 'uniform'),
291     'model_max_delta_step':Integer(0, 7, 'uniform'),

```

```

292     'model_subsample':Real(0.01, 0.99, 'uniform'),
293     'model_colsample_bytree':Real(0.01, 0.99, 'uniform'),
294     'model_colsample_bylevel':Real(0.01, 0.99, 'uniform'),
295     'model_colsample_bynode':Real(0.01, 0.99, 'uniform'),
296     'model_n_estimators':Integer(100, 300, 'uniform'),
297     'model_sample_type':Categorical(['uniform', 'weighted']),
298     'model_normalize_type':Categorical(['tree', 'forest']),
299     'model_rate_drop':Real(0.01, 0.99, 'uniform'),
300     'model_skip_drop':Real(0.01, 0.99, 'uniform'),
301     'model_objective':Categorical(['reg:squarederror', 'reg:
squaredlogerror', 'reg:logistic', 'reg:pseudohubererror'])
302     ,
303     'model_eval_metric':Categorical(['rmse', 'rmsle', 'mae',
'mape', 'mphe'])
304 }
305 xgb_linear_reg={
306     'model':Categorical([XGBRegressor()]),
307     'model_booster':Categorical(['gblinear']),
308     'model_feature_selector':Categorical(['cyclic', 'shuffle
', 'random', 'greedy', 'thrifty', ]),
309     'model_updater':Categorical(['shotgun', 'coord_descent'])
310     ,
311     'model_objective':Categorical(['reg:squarederror', 'reg:
squaredlogerror', 'reg:logistic', 'reg:pseudohubererror'])
312     ,
313     'model_eval_metric':Categorical(['rmse', 'rmsle', 'mae',
'mape', 'mphe'])
314 }
315 lgbm_reg={
316     'model':Categorical([LGBMRegressor()]),
317     'model_boosting_type':Categorical(['gbdt', 'dart', 'goss
', 'rf']),
318     'model_num_leaves':Integer(8, 128, 'uniform'),
319     'model_max_depth':Integer(4, 128, 'uniform'),
320     'model_learning_rate':Real(0.001, 0.4, 'uniform'),
321     'model_n_estimators':Integer(100, 300, 'uniform'),
322     'model_subsample_for_bin':Integer(200000, 250000, '
uniform'),
323     'model_min_split_gain':Real(0, 0.1, 'uniform'),
324     'model_min_child_weight':Real(0.001, 0.000001, 'uniform')
325     ,
326     'model_min_child_samples':Integer(12, 40, 'uniform'),
327     'model_subsample':Real(0.8, 1, 'uniform'),
328     'model_colsample_bytree':Real(0.8, 1, 'uniform'),
329 }

```

```

330
331 xgbbrf_dart_clf={
332     'model':Categorical([XGBRFClassifier()]),
333     'model__booster':Categorical([ 'dart']),
334     'model__eta':Real(0.01, 0.99, 'uniform'),
335     'model__gamma':Integer(0, 4, 'uniform'),
336     'model__max_depth':Integer(4, 128, 'uniform'),
337     'model__min_child_weight':Integer(0, 4, 'uniform'),
338     'model__max_delta_step':Integer(0, 7, 'uniform'),
339     'model__subsample':Real(0.01, 0.99, 'uniform'),
340     'model__colsample_bytree':Real(0.01, 0.99, 'uniform'),
341     'model__colsample_bylevel':Real(0.01, 0.99, 'uniform'),
342     'model__colsample_bynode':Real(0.01, 0.99, 'uniform'),
343     'model__n_estimators':Integer(100, 120, 'uniform'),
344     'model__sample_type':Categorical(['uniform', 'weighted'])
345 ,
346     'model__normalize_type':Categorical(['tree', 'forest']),
347     'model__rate_drop':Real(0.01, 0.99, 'uniform'),
348     'model__skip_drop':Real(0.01, 0.99, 'uniform'),
349     'model__objective':Categorical(['reg:squarederror', 'reg:
squaredlogerror', 'reg:logistic', 'reg:pseudohubererror'])
350 ,
351 }
352
353 xgb_dart_clf={
354     'model':Categorical([XGBClassifier()]),
355     'model__booster':Categorical([ 'dart']),
356     'model__eta':Real(0.01, 0.99, 'uniform'),
357     'model__gamma':Integer(0, 4, 'uniform'),
358     'model__max_depth':Integer(4, 128, 'uniform'),
359     'model__min_child_weight':Integer(0, 4, 'uniform'),
360     'model__max_delta_step':Integer(0, 7, 'uniform'),
361     'model__subsample':Real(0.01, 0.99, 'uniform'),
362     'model__colsample_bytree':Real(0.01, 0.99, 'uniform'),
363     'model__colsample_bylevel':Real(0.01, 0.99, 'uniform'),
364     'model__colsample_bynode':Real(0.01, 0.99, 'uniform'),
365     'model__n_estimators':Integer(100, 120, 'uniform'),
366     'model__sample_type':Categorical(['uniform', 'weighted'])
367 ,
368     'model__normalize_type':Categorical(['tree', 'forest']),
369     'model__rate_drop':Real(0.01, 0.99, 'uniform'),
370     'model__skip_drop':Real(0.01, 0.99, 'uniform'),
371     'model__objective':Categorical(['reg:squarederror', 'reg:
squaredlogerror', 'reg:logistic', 'reg:pseudohubererror'])
372 ,
373 }

```

373 , , ,

Listing 2: contains the hyperparameters that will be fetched

```
1 from sklearn.pipeline import Pipeline
2 from sklearn.preprocessing import StandardScaler,
  MinMaxScaler
3 from sklearn.compose import ColumnTransformer
4 from sklearn.impute import SimpleImputer
5 from sklearn.preprocessing import OneHotEncoder,
  OrdinalEncoder
6
7
8 def preproc_normalize(X_train=None, X_test=None, y_train=None
  , y_test=None, scaler=None, scaler_trigger=False,
  X_test_trigger=True):
9     '''normilize the data with MinMaxScaler
10     Args:
11         X_train, X_test, y_train, y_test
12     Return:
13         X_train, X_test, y_train, y_test
14     '''
15
16     if scaler_trigger == False:
17         scaler = MinMaxScaler()
18         scaler.fit(X_train)
19
20     X_train = scaler.transform(X_train)
21     if X_test_trigger == True:
22         X_test = scaler.transform(X_test)
23
24     return X_train, X_test, y_train, y_test, scaler
25
26
27 def preprocess(X_train=None, X_test=None, y_train=None,
  y_test=None, categorical_features=None, numerical_features
  =None, normalize_fn=None, scaler_fn=None, scaler_trigger=
  False, X_test_trigger=True):
28     '''replace Nan values of categorical and numerical
  features. Moreover, transform categorical features in
  numerical data
29     Args:
30         X_train, X_test, y_train, y_test
31         categorical_features, list with the name of the
  categorical columns
32         numerical_features, list with the name of the numerical
  columns
33     Return:
34         X_train, X_test, y_train, y_test
35     '''
```



```

36
37 numerical_pipeline = Pipeline(steps=[
38     ('imputer', SimpleImputer(strategy='mean'))])
39
40 categorical_pipeline = Pipeline(steps=[
41     ('imputer', SimpleImputer(strategy='most_frequent')),
42     ('onehot', OneHotEncoder())])
43
44 transformation = ColumnTransformer(
45     transformers=[
46         ('numerical_transformation', numerical_pipeline,
numerical_features),
47         ('categorical_transformation',
48         categorical_pipeline, categorical_features),
49     ])
50
51 X_train = transformation.fit_transform(X_train)
52 if X_test_trigger == True:
53     X_test = transformation.transform(X_test)
54
55 if scaler_trigger == False and X_test_trigger == False:
56     X_train, X_test, y_train, y_test, scaler =
normalize_fn(
57         X_train=X_train, y_train=y_train, X_test_trigger=
False)
58
59 elif scaler_trigger == True and X_test_trigger == False:
60     X_train, X_test, y_train, y_test, scaler =
normalize_fn(
61         X_train=X_train, y_train=y_train, X_test_trigger=
False, scaler=scaler_fn, scaler_trigger=True)
62
63 elif scaler_trigger == False and X_test_trigger == True:
64     X_train, X_test, y_train, y_test, scaler =
normalize_fn(
65         X_train, X_test, y_train, y_test)
66 else:
67     X_train, X_test, y_train, y_test, scaler =
normalize_fn(
68         X_train=X_train, X_test=X_test, y_train=y_train,
y_test=y_test, scaler=scaler_fn, X_test_trigger=False,
scaler_trigger=True)
69
70 return X_train, X_test, y_train, y_test, scaler

```

Listing 3: Data normalizations

```

1 from sklearn.model_selection import train_test_split
2 import pandas as pd
3 import numpy as np

```

```

4 from drive.MyDrive.Data_Science_Competitions.Project3.Utils.
   preprocessing.data_treatment import undersample_bootstrap,
   feature_eng
5 from drive.MyDrive.Data_Science_Competitions.Project3.Utils.
   preprocessing.data_preprocessing import preproc_normalize,
   preprocess
6 from skopt import BayesSearchCV
7 from skopt.plots import plot_objective
8 from drive.MyDrive.Data_Science_Competitions.Project3.Utils.
   model_params.model_classifiers import *
9 from sklearn.pipeline import Pipeline
10 import joblib
11 from skopt.plots import plot_objective
12 import matplotlib.pyplot as plt
13 from skopt.plots import plot_convergence
14 from sklearn import metrics
15 import seaborn as sn
16 from sklearn.svm import SVC
17
18
19 def Train_model(lgbm_clf_n_calls, xgb_rf_dart_clf_n_calls,
   xgb_rf_gbtree_clf_n_calls, xgb_gbtree_clf_n_calls,
   xgb_dart_clf_n_calls, gaussianNB_clf_n_calls,
   Decision_Tree_clf_n_calls, Random_Forest_clf_n_calls,
   AdaBoost_clf_n_calls, gradientBooster_clf_n_calls,
   HistGradientBooster_clf_n_calls, knn_clf_n_calls,
   Knn_clf_n_calls, svc_clf_n_calls,
   PassiveAggressive_clf_n_calls, sgd_n_calls,
   ridge_false_n_calls, ridge_positive_n_calls,
   perceptron_n_calls, QDA_clf_n_calls, plot=False):
20     '''Apply feature engineering(data_treatment) and
   preprocessing to the train data. Furthermore, do
   optimization of hyperparameters by pipeline
21     and BayesSearch CV. At the end, save results and creat
   plots for analysis
22     Args:
23         model_n_calls, number of calls for bayesian
   optimization
24         plot, if plot is True, the function create plot of
   dependes, convergence, ROC and Confusion Matrix
25     Return:
26         dropped_columns, columns dropped from train data
27         columns_to_keep, columns with relevant information
28     '''
29
30     ''' '''
31     # reading the training file
32     train_df = pd.read_csv('/content/drive/MyDrive/
   Data_Science_Competitions/Project3/Data/train.csv')

```

```

33
34     # concatening two dataframe and aplying feature
engineeering
35     new_train_df, dropped_columns, columns_to_keep=
feature_eng(train_df)
36
37     # creating the features and targets for this study
38     X = new_train_df
39     y = train_df['label']
40     X_train, X_test, y_train, y_test = train_test_split(X, y,
train_size=0.7, shuffle=False)
41
42     # undersample data(there is to many no Fraud in samples)
43     X_train, y_train = undersample_bootstrap(X_train, y_train,
0)
44
45     # preprocessing(replacing Nan values, Hot enconder and
normalization)
46     numerical_features = X._get_numeric_data().columns.tolist
()
47     categorical_features = [attribute for attribute in X.
columns.tolist() if attribute not in X._get_numeric_data()
.columns.tolist()]
48     X_train, X_test, y_train, y_test, scaler = preprocess(
X_train, X_test, y_train, y_test, categorical_features,
numerical_features, preproc_normalize)
49
50     #pipeline for many models
51     pipeline = Pipeline([
52         ('model', SVC())
53     ])
54
55     #models for pipeline
56     Search_spaces=[
57         (lgbm_clf, lgbm_clf_n_calls),
58         (xgbrf_gbtrees_clf, xgbrf_gbtrees_clf_n_calls),
59         (xgb_gbtrees_clf, xgb_gbtrees_clf_n_calls),
60         (gaussianNB_clf, gaussianNB_clf_n_calls),
61         (Decision_Tree_clf, Decision_Tree_clf_n_calls),
62         (Random_Forest_clf, Random_Forest_clf_n_calls),
63         (AdaBoost_clf, AdaBoost_clf_n_calls),
64         (knc_clf, knc_clf_n_calls),
65         (Knn_clf, Knn_clf_n_calls),
66         (svc_clf, svc_clf_n_calls),
67         (PassiveAggressive_clf, PassiveAggressive_clf_n_calls)
68     ],
69     (sgd_clf, sgd_n_calls),
70     (ridge_clf_false, ridge_false_n_calls),
71     (ridge_clf_positive, ridge_positive_n_calls),

```

```

71         (perceptron_clf, perceptron_n_calls,),
72         (QDA_clf, QDA_clf_n_calls)
73     ]
74
75     '''
76     (xgbrf_dart_clf, xgbrf_dart_clf_n_calls),
77     (xgb_dart_clf, xgb_dart_clf_n_calls),
78     (HistGradientBooster_clf, HistGradientBooster_clf_n_calls
79 ),
80     (gradientBooster_clf, gradientBooster_clf_n_calls),
81     '''
82
83     optimizer = BayesSearchCV(estimator=pipeline,
84 search_spaces=Search_spaces, cv=3, scoring='accuracy',
85 verbose=3)
86
87     #fitting models and cross-validation
88     optimizer.fit(X_train, np.ravel(y_train))
89
90     print("val. score: %s" % optimizer.best_score_)
91     print("test score: %s" % optimizer.score(X_test, y_test))
92     print("best params: %s" % str(optimizer.best_params_))
93
94     # saving best model and results
95     joblib.dump(optimizer.best_estimator_, 'best_estimator.
96 pkl')
97     np.save('my_results.npy', optimizer.cv_results_)
98
99     #---IMAGES FOR ANALYSIS --- #
100
101     classifiers = [
102         'LGBMClassifier',
103         'XGBRFClassifier_gbtrees',
104         'XGBClassifier_gbtrees',
105         'GaussianNB',
106         'DecisionTreeClassifier',
107         'RandomForestClassifier',
108         'AdaBoostClassifier',
109         'KNeighborsClassifier',
110         'NearestCentroid',
111         'SVC',
112         'PassiveAggressiveClassifier',
113         'SGDClassifier',
114         'RidgeClassifierFalse',
115         'RidgeClassifierPositive',
116         'Perceptron',
117         'QuadraticDiscriminantAnalysis'
118     ]

```

```

116
117     '''
118     'XGBRFClassifier_dart',
119     'XGBClassifier_dart',
120     'HistGradientBoostingClassifier',
121     'GradientBoostingClassifier',
122     '''
123
124     if plot == True:
125
126         for i in range(len(optimizer.optimizer_results_)):
127             plt.title(classifiers[i])
128             _ = plot_objective(optimizer.optimizer_results_[i
129 ])
130             plt.savefig(classifiers[i]+'_dependence.png')
131             plt.clf()
132
133             plt.rcParams['figure.figsize'] = [15, 15]
134             plt.title("Convergence of models")
135             clf_plot = ((classifiers[index], optimizer.
136 optimizer_results_[
137 index]) for index in range(len(
138 classifiers)))
139             plot = plot_convergence(*clf_plot)
140             plot.legend(loc="best", prop={'size': 6}, numpoints
141 =1)
142             plt.savefig('Convergence.png')
143             plt.clf()
144
145             plt.title("Confusion Matrix Best Score")
146             metrics.plot_confusion_matrix(optimizer.
147 best_estimator_, X_test, y_test, cmap="inferno")
148             plt.savefig("Confusion_Matrix.png")
149             plt.clf()
150
151             try:
152                 print(optimizer.best_estimator_.predict_proba(
153 X_test))
154             except:
155                 print("predict_proba does not exist for this
156 classifier")
157
158
159     # save scaler from of the training
160     X = new_train_df
161     y = train_df['label']
162     X_train, X_test, y_train, y_test, scaler = preprocess(
163 X_train=X, y_train=y, categorical_features=

```

```

    categorical_features, numerical_features=
    numerical_features, normalize_fn=preproc_normalize,
    X_test_trigger=False)
158     joblib.dump(scaler, 'scaler.save')
159
160     return dropped_columns, columns_to_keep

```

Listing 4: training the models

```

1  import joblib
2  import pandas as pd
3  from drive.MyDrive.Data_Science_Competitions.Project3.Utils.
    preprocessing.data_preprocessing import preproc_normalize,
    preprocess
4  import copy
5
6  def submission(dropped_columns, columns_to_keep):
7      '''Read best model and predict the file Test of kaggle,
    at the end creates kaggle_submission.csv with predictions
8      Args:
9          dropped_columns, columns to be dropped from Test file(
    obtained in training)
10         columns_to_keep, columns with relevant information
11      Return:
12          None
13      '''
14
15      # load best model
16      best_model = joblib.load('best_estimator.pkl')
17      print(dropped_columns)
18
19      # reading test file
20      test_df = pd.read_csv('/content/drive/MyDrive/
    Data_Science_Competitions/Project3/Data/test.csv')
21
22      submission_df = pd.DataFrame(data=[index+1 for index in
    test_df.index], columns=['ImageId'])
23
24      #applying drop to the columns founded in training and
    reliability of some columns
25      test_df.drop(columns=dropped_columns, inplace=True)
26
27
28      # creating the features and targets for this study
29      X = test_df[columns_to_keep]
30
31      # getting the name of numerical and categorical columns
32      numerical_features = X._get_numeric_data().columns.tolist
    ()

```

```

33     categorical_features = [attribute for attribute in X.
columns.tolist() if attribute not in X._get_numeric_data()
.columns.tolist()]
34
35     # applying the preprocessing using the scaler obtained
from Training
36     scaler = joblib.load('scaler.save')
37     X_train, X_test, y_train, y_test, scaler = preprocess(
X_train=X, categorical_features=categorical_features,
numerical_features=numerical_features,
38
normalize_fn=preproc_normalize, scaler_fn=scaler,
scaler_trigger=True, X_test_trigger=False)
39
40     # prediction
41     y_pred = best_model.predict(X_train)
42
43     # kaggle submission
44     submission_df['Label'] = y_pred
45     submission_df.to_csv('kaggle_submission.csv', index=False
)

```

Listing 5: Create kaggle submission

```

1 from drive.MyDrive.Data_Science_Competitions.Project3.
Train_Test_model_Scikit import Train_model
2 from drive.MyDrive.Data_Science_Competitions.Project3.
kaggle_submission import submission
3
4 def Main(dict_of_calls, trigger_plot=False):
5     '''Train models and create kaggle submission
6     Args:
7         list_of_calls, list with the number of calls for each
model
8         trigger_plot, if True, create plots of dependence,
convergence, ROC and confusion Matrix
9     Return:
10         None
11     '''
12
13
14     classifiers = [
15         'lgbm_clf_n_calls',
16         'xgbrf_dart_clf_n_calls',
17         'xgbrf_gbtrees_clf_n_calls',
18         'xgb_gbtrees_clf_n_calls',
19         'xgb_dart_clf_n_calls',
20         'gaussianNB_clf_n_calls',
21         'Decision_Tree_clf_n_calls',
22         'Random_Forest_clf_n_calls',

```

```

23         'AdaBoost_clf_n_calls',
24         'gradientBooster_clf_n_calls',
25         'HistGradientBooster_clf_n_calls',
26         'knc_clf_n_calls',
27         'Knn_clf_n_calls',
28         'svc_clf_n_calls',
29         'PassiveAggressive_clf_n_calls',
30         'sgd_n_calls',
31         'ridge_false_n_calls',
32         'ridge_positive_n_calls',
33         'perceptron_n_calls',
34         'QDA_clf_n_calls']
35
36     '''
37     #classifiers_calls=[30, 30, 40, 60, 30, 20]
38
39
40     print(dict_of_calls)
41
42     print("Training Model")
43     dropped_columns, columns_to_keep=Train_model(**
44 dict_of_calls, plot=trigger_plot)
45
46     print("Creating kaggle submission")
47     submission(dropped_columns, columns_to_keep)

```

Listing 6: Main File

Finally, after several training sessions, the best model was chosen based on the convergence of each one of them, as shown in the photo below:

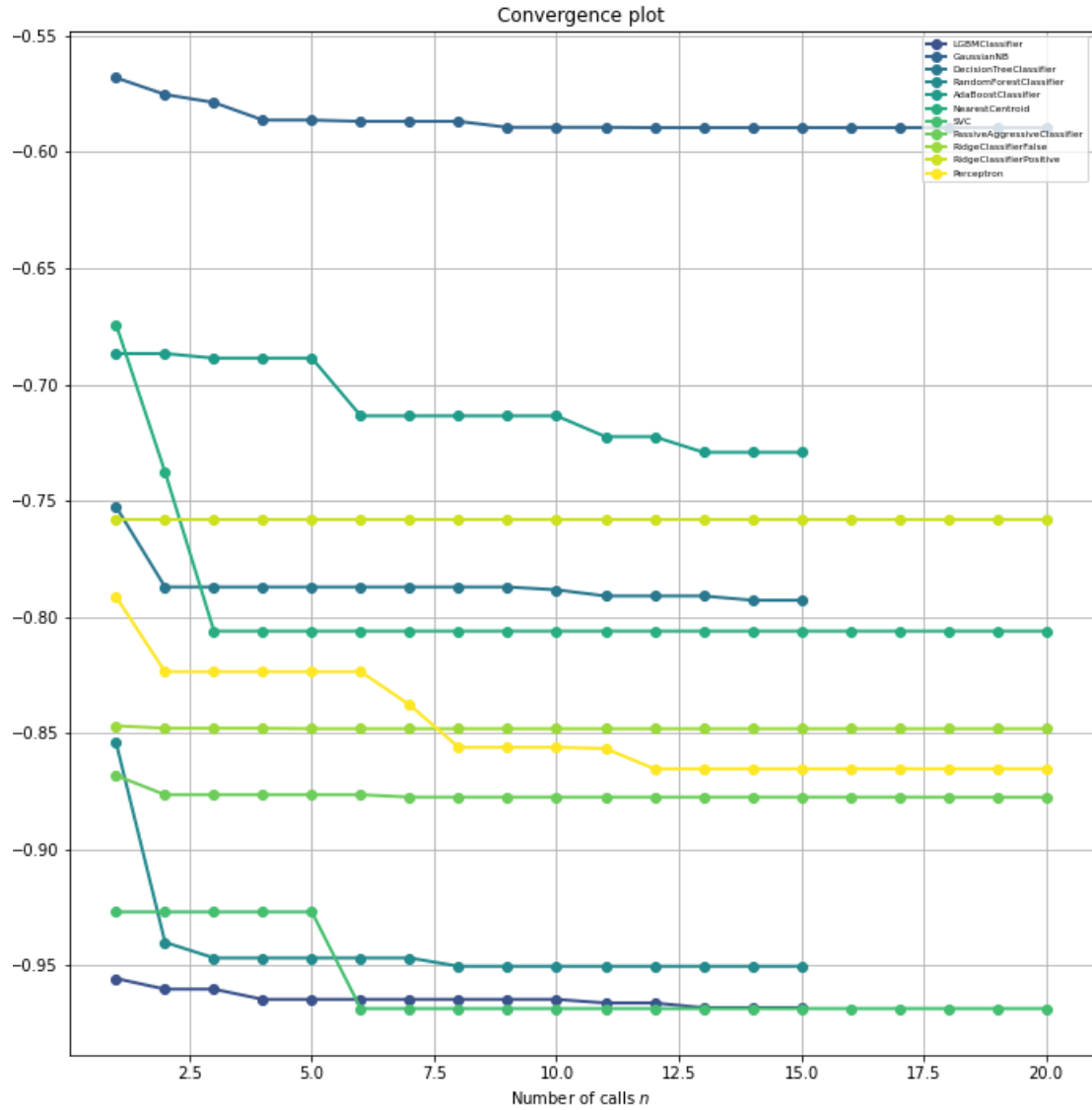


Figure 4: Convergence

With this in mind, we can see below how this search for hyperparameters took place through the relationships between a pair of variables and the value of the accuracy (level curves).

4.1.1 LGBMClassifier

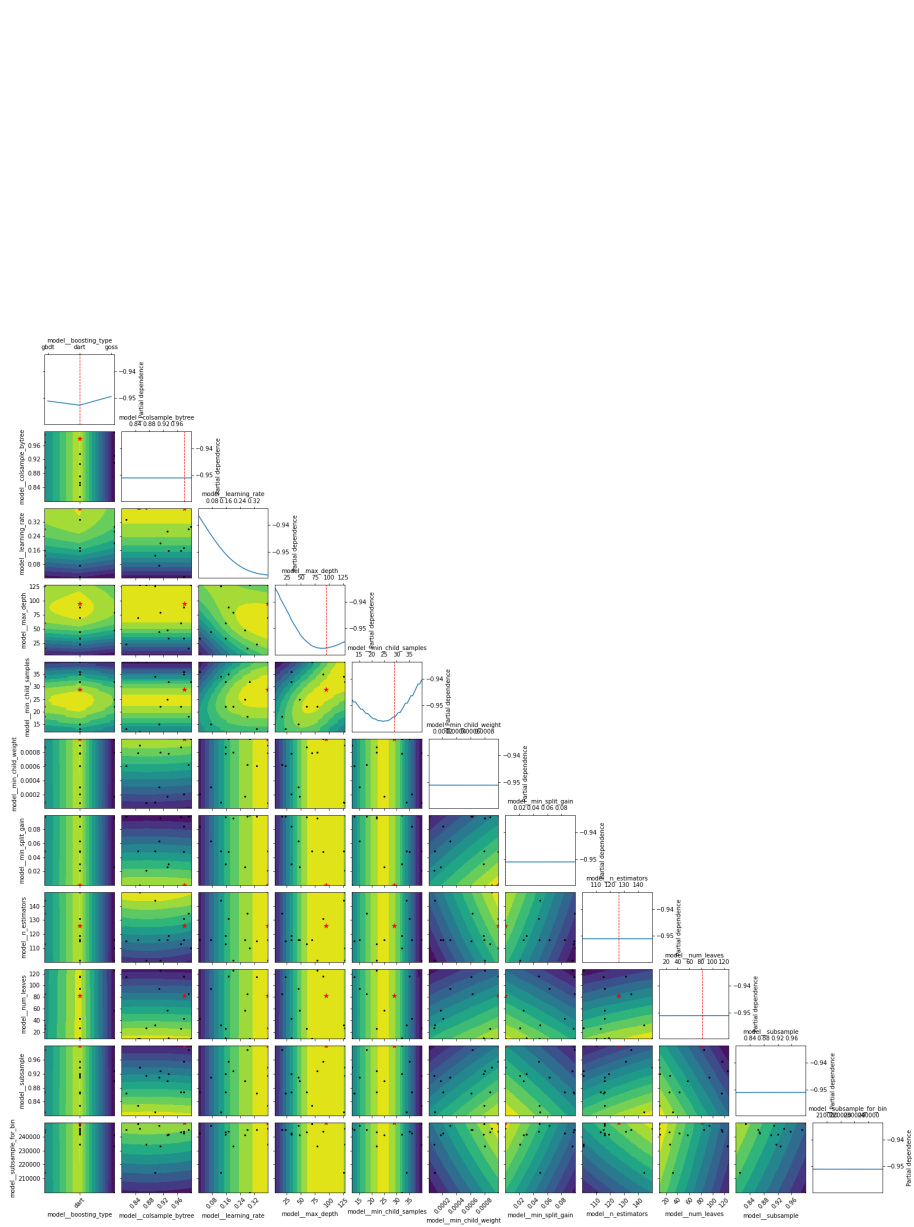


Figure 5: LGBMClassifier

4.1.2 GaussianNB

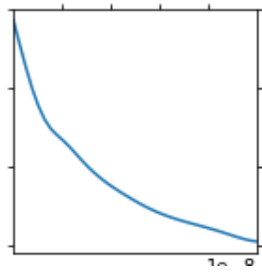


Figure 6: GaussianNB

4.1.3 DecisionTreeClassifier

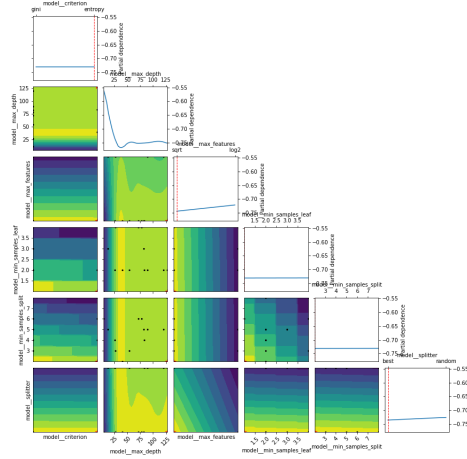


Figure 7: DecisionTreeClassifier

4.1.4 RandomForestClassifier

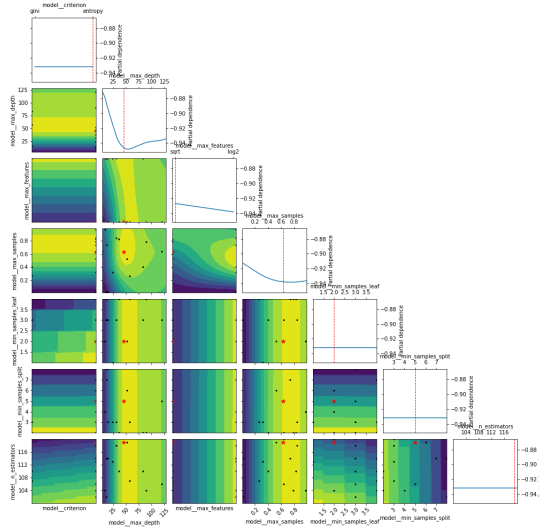


Figure 8: RandomForestClassifier

4.1.5 AdaBoostClassifier

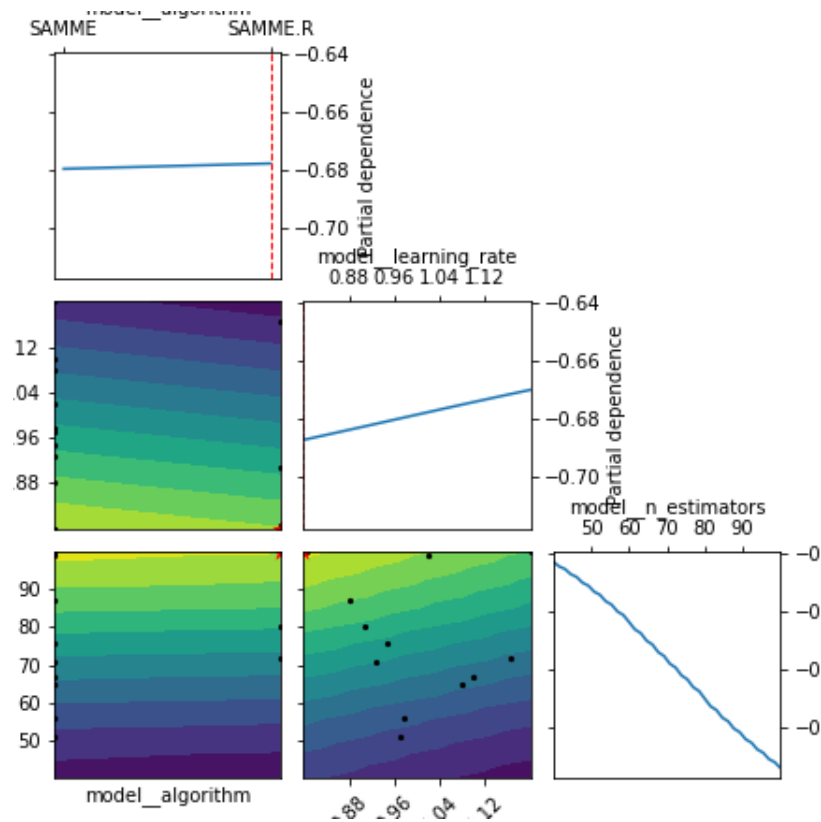


Figure 9: AdaBoostClassifier

4.1.6 NearestCentroid

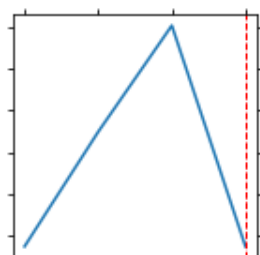


Figure 10: NearestCentroid

4.1.7 SVC

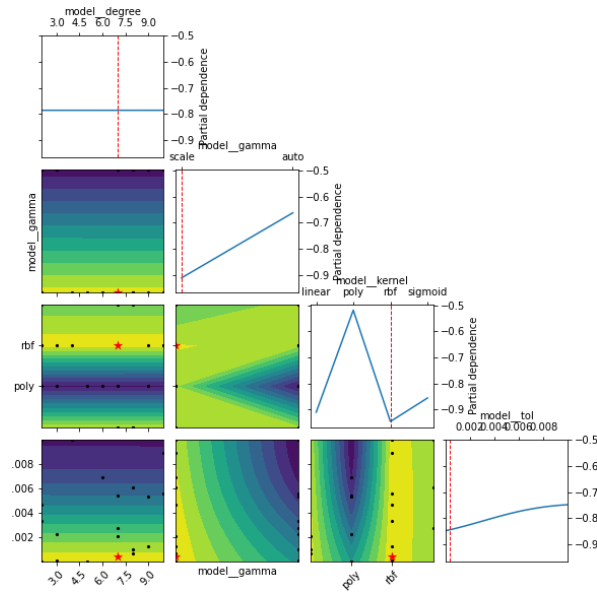


Figure 11: SVC

4.1.8 PassiveAggressiveClassifier

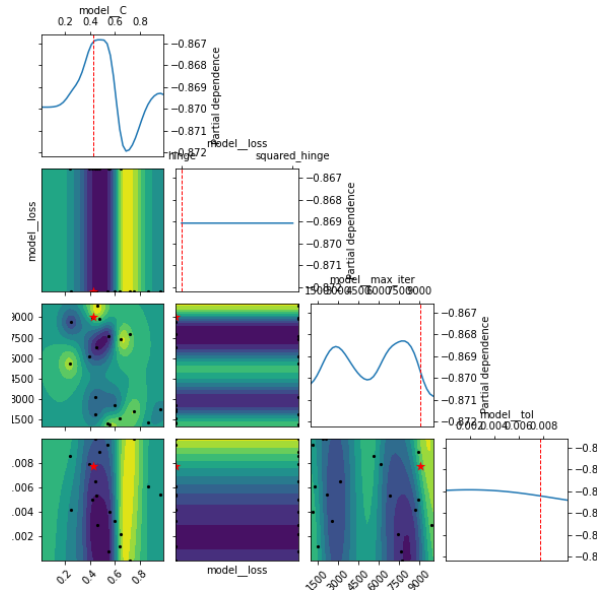


Figure 12: PassiveAggressiveClassifier

4.1.9 RidgeClassifierFalse

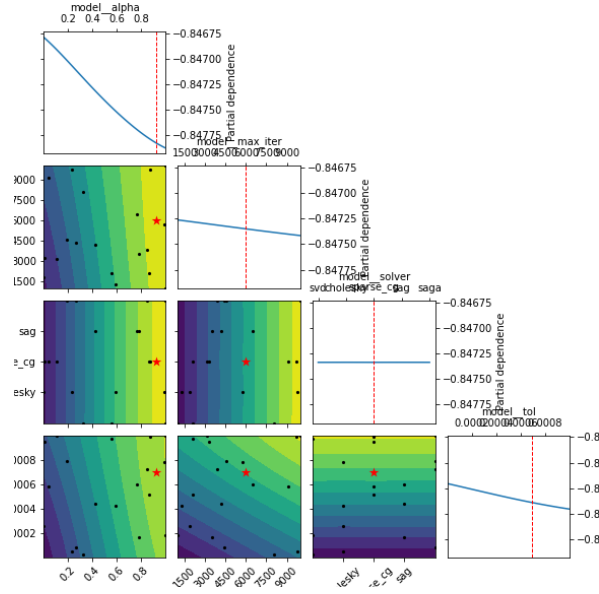


Figure 13: *RidgeClassifierFalse*

4.1.10 RidgeClassifierPositive

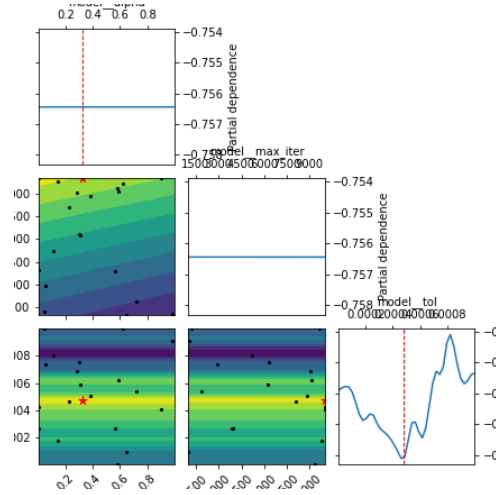


Figure 14: *RidgeClassifierPositive*

4.1.11 Perceptron

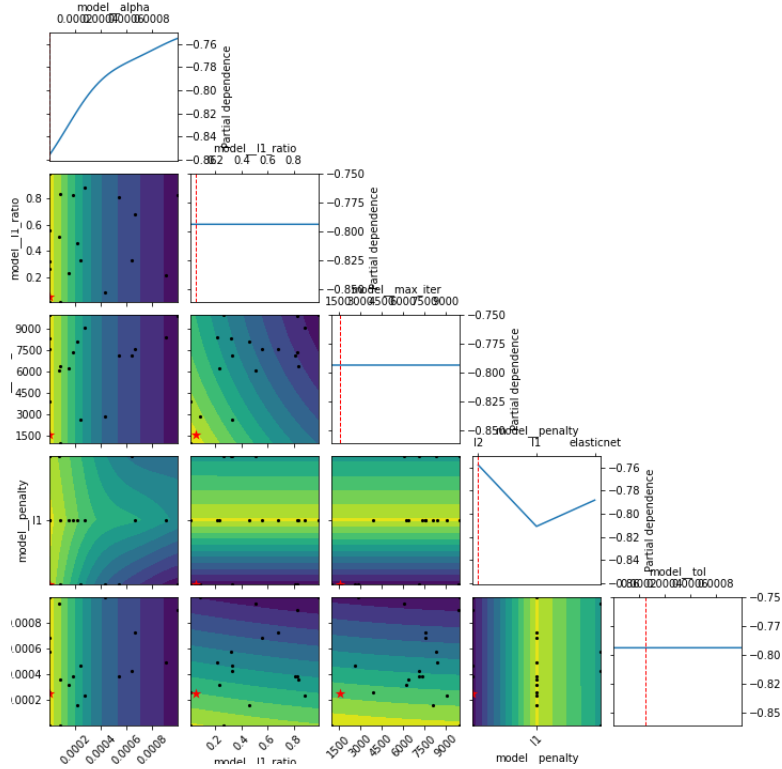


Figure 15: Perceptron

4.2 Keras in TensorFlow

In the Bayesian optimization of TensorFlow, only a simple architecture of convolucinal networks was used. With this in mind, only three codes were used: 'Deep_architectures' (contains the convolutional network model), 'DeepBayesianSearch' (trains the models) and 'Main_Keras' (uses the other codes and submits the kaggle).

```

1 #from keras_self_attention import seq_self_attention
2 from tensorflow.keras.layers import Dense, LSTM,
   Bidirectional, Flatten, TimeDistributed, Concatenate,
   Dropout
3 from tensorflow.keras.layers import Conv1D, MaxPooling1D,
   Conv2D, MaxPooling2D
4 from tensorflow.keras.models import Sequential
5 #from tensorflow.python.keras.layers.dense_attention import
   Attention

```

```

6 #from attention import Attention
7 from tensorflow.keras.models import Model
8 #from keras_self_attention import SeqSelfAttention
9
10
11 class Models_Constructor():
12
13     def __init__(self, num_classes, input_shape = (28, 28, 1)
14 ):
15
16         self.num_classes=num_classes
17         self.input_shape=input_shape
18
19     def ConvNet_builder(self, hp):
20
21         # defining a set of hyperparametrts for tuning
22         conv_filters=hp.Int(name = 'filters', min_value = 1,
23 max_value = 256, step = 1)
24         conv_kernel=hp.Int(name = 'kernel_size', min_value =
25 4, max_value = 8, step = 1)
26         conv_pool_size=hp.Int(name = 'pool_size', min_value =
27 1, max_value = 2, step = 1)
28         conv_activation = hp.Choice(name='activation', values
29 = ['tanh', 'relu','sigmoid', 'softmax', 'softplus'],
30 ordered = False)
31         #dropout layer
32         dropout = hp.Float(name = 'dropout', min_value=0,
33 max_value=.5, step=0.05)
34         #Dense layers
35         activation = hp.Choice(name='activation', values = ['
36 tanh', 'relu','sigmoid', 'softmax', 'softplus'], ordered =
37 False)
38         activation1 = hp.Choice(name='activation1', values =
39 ['tanh', 'relu','sigmoid', 'softmax', 'softplus'], ordered
40 = False)
41         activation2 = hp.Choice(name='activation2', values =
42 ['tanh', 'relu','sigmoid', 'softmax', 'softplus'], ordered
43 = False)
44         units=hp.Int(name = 'units', min_value = 1, max_value
45 = 256, step = 1)
46         #learning_rate = hp.Choice('learning_rate', values=[1
47 e-2, 1e-3, 1e-4, 1e-5, 1e-6])
48         optimizer=hp.Choice('optimizer', ['sgd', 'adam', '
49 rmsprop', 'adadelata', 'adagrad', 'adamax', 'ftrl'])
50
51         #model
52         model=Sequential()

```



```

38         model.add(Conv2D(2*conv_filters, kernel_size=(
conv_kernel, conv_kernel), activation=conv_activation,
input_shape=self.input_shape))
39         model.add(MaxPooling2D(pool_size=(conv_pool_size,
conv_pool_size)))
40         model.add(Conv2D(conv_filters, kernel_size=(
conv_kernel, conv_kernel), activation=conv_activation))
41         model.add(MaxPooling2D(pool_size=(conv_pool_size,
conv_pool_size)))
42         model.add(Flatten())
43         model.add(Dropout(dropout))
44         model.add(Dense(units=units, activation=activation1))
45         model.add(Dense(self.num_classes, activation=
activation2))
46
47         model.compile(loss="categorical_crossentropy",
optimizer=optimizer, metrics=["accuracy"])
48
49         return model

```

Listing 7: Neural Architectures

```

1
2 import keras_tuner as kt
3 from sklearn.model_selection import train_test_split
4 from tensorflow import keras
5 from tensorflow.keras import layers
6 from sklearn.metrics import confusion_matrix
7 from drive.MyDrive.Data_Science_Competitions.Project3.Utils.
model_params.Deep_architectures import Models_Constructor
8
9
10
11 def Train_Test_keras():
12
13
14     PROJECT_PATH="/content/drive/MyDrive/
Data_Science_Competitions/Project3/"
15     train_df=pd.read_csv(PROJECT_PATH+"Data/train.csv")
16
17     features_df = train_df.drop(columns=['label'])
18     targets_df = train_df['label']
19
20     num_classes = len(targets_df.value_counts().index)
21     input_shape = (28, 28, 1)
22
23     X_train, X_test, y_train, y_test = train_test_split(
features_df, targets_df, train_size=0.8)
24

```

```

25     X_train = X_train.values.reshape(X_train.shape[0], 28,
26     28, 1)
27     X_test = X_test.values.reshape(X_test.shape[0], 28, 28,
28     1)
29     X_train = X_train.astype("float32") / 255
30     X_test = X_test.astype("float32") / 255
31
32     y_train = keras.utils.to_categorical(y_train, num_classes)
33     y_test = keras.utils.to_categorical(y_test, num_classes)
34
35     train_callbacks = [
36         keras.callbacks.EarlyStopping(
37             monitor="val_accuracy", patience=10,
38             restore_best_weights=True
39         )
40     ]
41
42     Builders=Models_Constructor(num_classes=num_classes)
43     bayesian_tuner=kt.BayesianOptimization(Builders.
44     ConvNet_builder,
45
46                                     objective='
47     val_accuracy',
48
49                                     max_trials=40,
50                                     executions_per_trial
51     =3,
52
53                                     overwrite=False,
54                                     beta=3.6,
55                                     directory="My_Dir",
56                                     project_name="
57     DigitRecognizer")
58
59     bayesian_tuner.search(X_train, y_train, epochs=100,
60     callbacks=train_callbacks, validation_split=0.3)
61
62     best_model=bayesian_tuner.get_best_models()[0]
63     y_pred = best_model.predict(X_test)
64     y_pred=np.argmax(y_pred, axis=1, out=None)
65     y_test=np.argmax(y_test, axis=1, out=None)
66
67     from sklearn.metrics import confusion_matrix
68
69     plt.rcParams['figure.figsize'] = [20, 15]
70     plt.title("Confusion Matrix Best Score")
71     conf_matrix=confusion_matrix(y_test, y_pred)
72     #sn.heatmap(conf_matrix, annot=True, cmap="inferno")
73     sn.heatmap(conf_matrix/np.sum(conf_matrix), annot=True,
74     fmt='.3%', cmap="inferno")

```

```
63 plt.savefig("Confusion_Matrix.png")
```

Listing 8: Training the model

```
1
2 import keras_tuner as kt
3 from sklearn.model_selection import train_test_split
4 from tensorflow import keras
5 from tensorflow.keras import layers
6 from sklearn.metrics import confusion_matrix
7 from drive.MyDrive.Data_Science_Competitions.Project3.Utils.
  model_params.Deep_architectures import Models_Constructor
8
9
10 def create_kaggle():
11
12     test_df = pd.read_csv('/content/drive/MyDrive/
Data_Science_Competitions/Project3/Data/test.csv')
13
14     submission_df = pd.DataFrame(data=[index+1 for index in
test_df.index], columns=['ImageId'])
15
16     kaggle_test=test_df.values
17     kaggle_test=kaggle_test.astype("float32") / 255
18     kaggle_test = kaggle_test.reshape(kaggle_test.shape[0],
28, 28, 1)
19
20     bayesian_tuner=kt.BayesianOptimization(Builders.
ConvNet_builder,
21
22                                     objective='
val_accuracy',
23                                     max_trials=80,
24                                     executions_per_trial
=2,
25                                     overwrite=False,
26                                     beta=3.6,
27                                     directory="/content/
drive/MyDrive/Data_Science_Competitions/Project3/My_Dir",
28                                     project_name="
DigitRecognizer")
29
30     best_model=bayesian_tuner.get_best_models()[0]
31     kaggle_pred = best_model.predict(kaggle_test)
32
33     submission_df = pd.DataFrame(data=[index+1 for index in
test_df.index], columns=['ImageId'])
34     submission_df['Label'] = np.argmax(kaggle_pred, axis=1,
out=None)
35     submission_df.to_csv('kaggle_submission.csv', index=False
```

```
)
```

Listing 9: Create kaggle submission

```
1 from drive.MyDrive.Data_Science_Competitions.Project3.
   DeepBayesianSearch import Train_Test_keras
2 from drive.MyDrive.Data_Science_Competitions.Project3.
   kaggle_submission_keras import create_kaggle
3
4 def Main():
5
6     Train_Test_keras()
7     create_kaggle()
8
9
10 Main()
```

Listing 10: Main File

Below we can the images that contain the information of the search for the best accuracy, these graphs can be obtained by the Log of google colab.

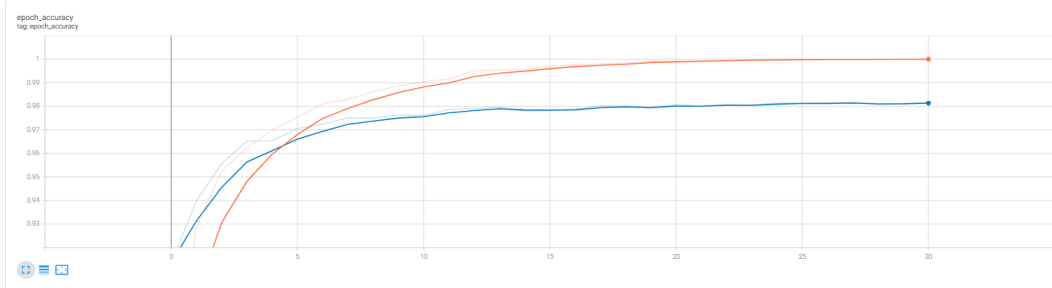


Figure 16: Accuracy in each epoch

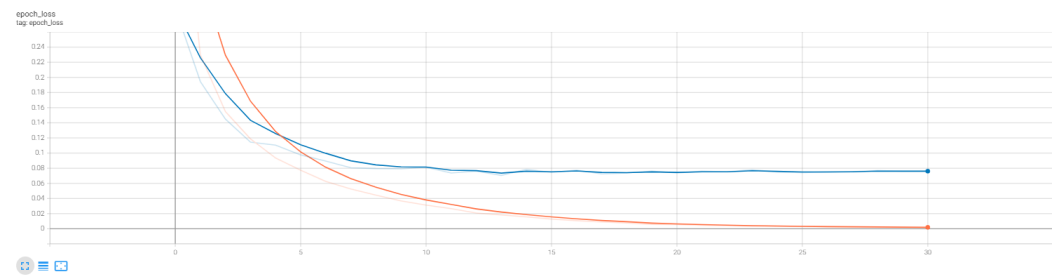


Figure 17: Loss in each epoch

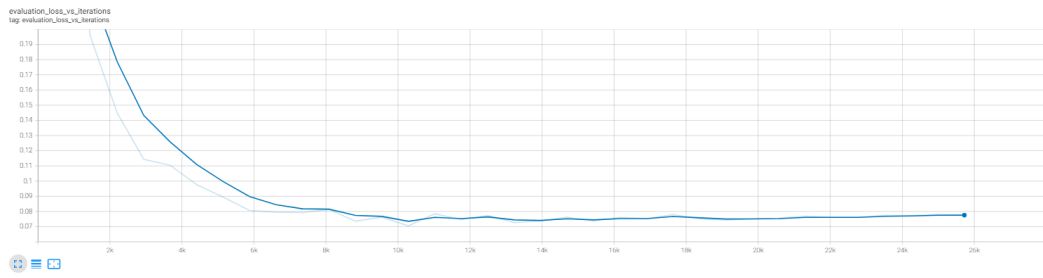


Figure 18: Evaluation x Loss

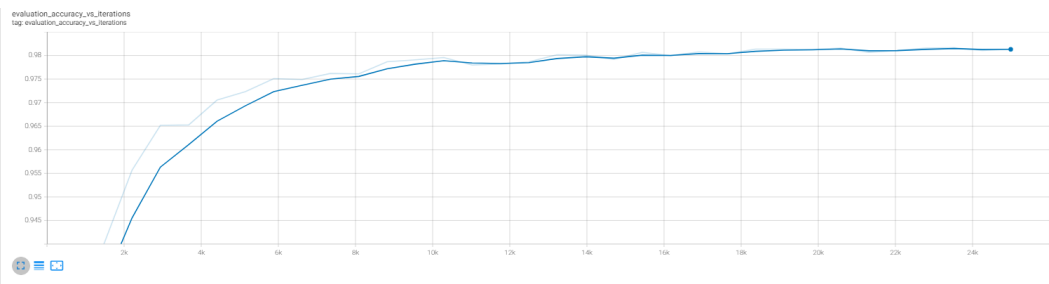


Figure 19: Evaluation x Iteration

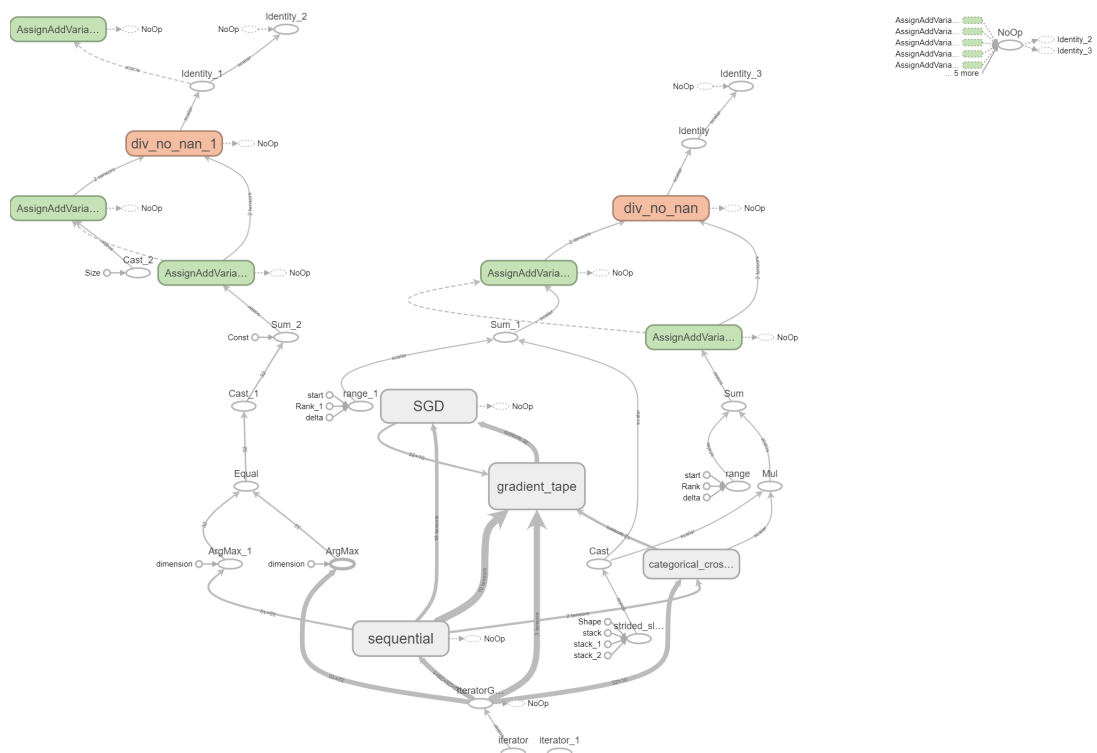


Figure 20: Training Schema

4.3 Submissions

kaggle_submission (8).csv
a minute ago by evertonmendes73

0.97178

Figure 21: Scikit Submission

kaggle_submission (6).csv
2 days ago by evertonmendes73
add submission details

0.99039

Figure 22: Keras Submission

5 Question Four

For the two models obtained from the optimizers above, we can visualize their confusion matrices in order to better understand the predictions and their errors.

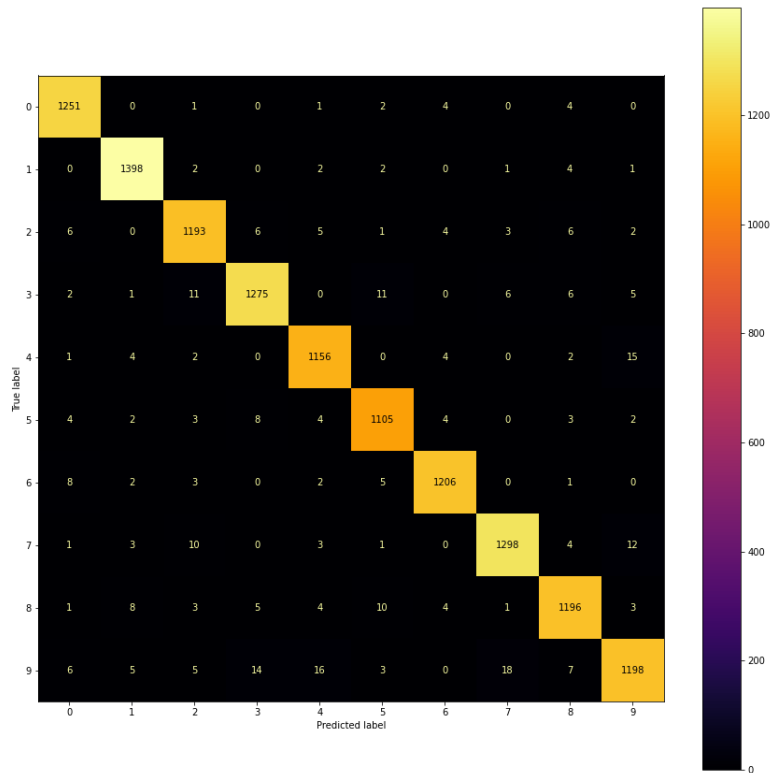


Figure 23: Confusion Matrix in Scikit Learn

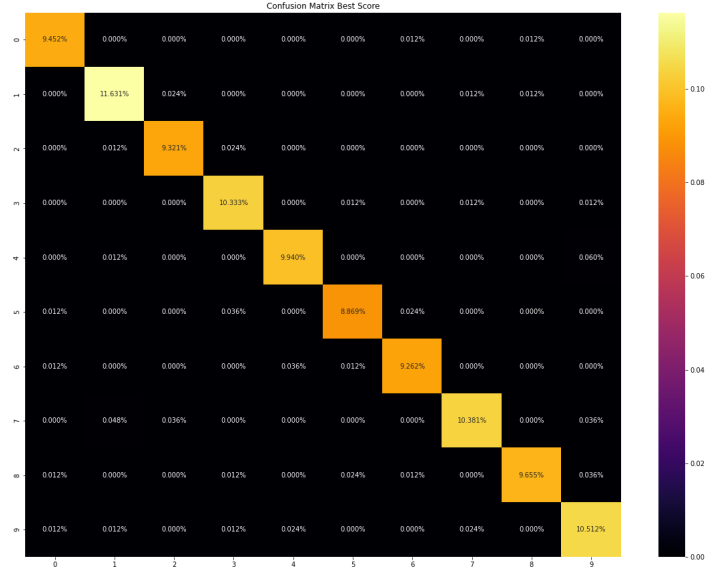


Figure 24: Confusion Matrix in Keras

6 Reference

- [1] da Silva, Éverton Luís Mendes. Codes from this project
- [2] da Silva, Éverton Luis Mendes. Codes from this project in Drive