Universidade de São Paulo
Instituto de Física de São Carlos

SCC00277-201-2021 Project 3

Éverton Luís Mendes da Silva (10728171)

Contents

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).

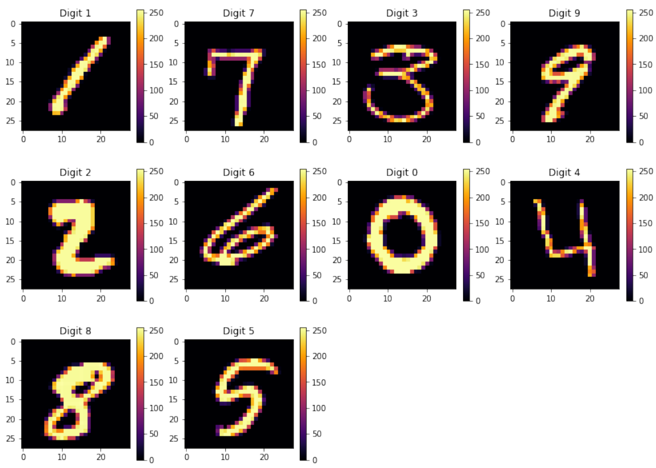

Figure 1: Digits

2.2 Item B

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

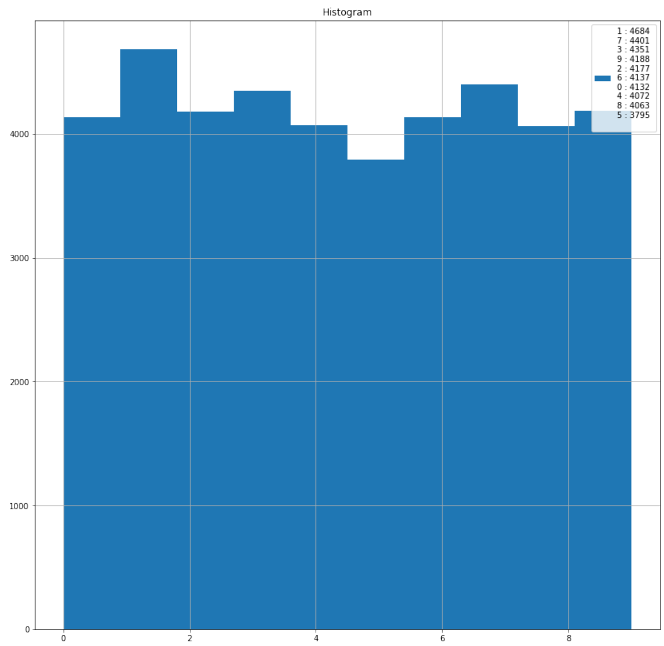

Figure 2: Histogram of Digits

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.

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

$$\sum_{i=1}^{m} p_i = 1 \tag{2}$$

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.

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

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

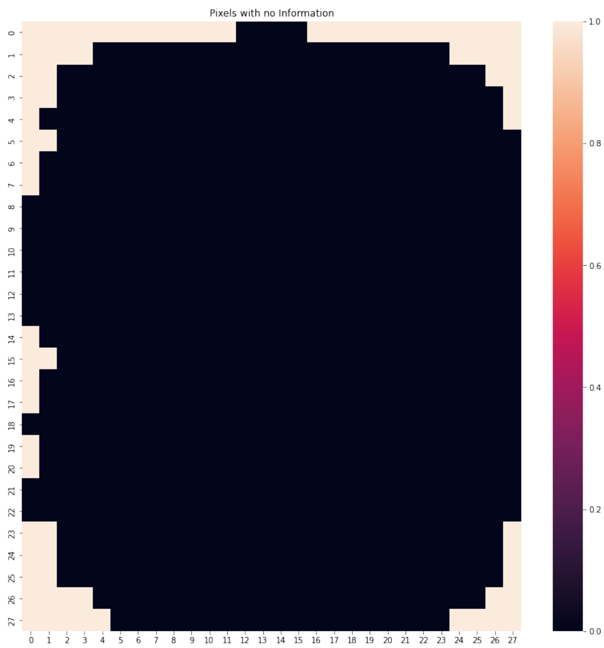

Figure 3: No information

```python
def drop_no_information(df):
    '''Drop columns with no information in a DataFrame
    Args:
        df, DataFrame
    Return:
        df[columns_to_keep], new DataFrame with relevant information
        columns_to_keep, columns with relevant information
    '''
    columns_to_keep=[column for column, boolean in (df.drop(columns='label').std()!=0).items() if boolean]

```

```
    sn.heatmap((df.drop(columns='label').std()==0).values.
    reshape(28, 28))
    plt.title("Pixels with no Information")
    plt.savefig("/content/drive/MyDrive/
    Data_Science_Competitions/Project3/Data/img/Digits/
    Pixels_no_inf.png")

    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 leaning 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)

```
from skopt.space import Real, Categorical, Integer

from sklearn.linear_model import RidgeClassifier
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.discriminant_analysis import
    QuadraticDiscriminantAnalysis
from sklearn.linear_model import PassiveAggressiveClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import NearestCentroid
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import HistGradientBoostingClassifier
from xgboost import XGBClassifier, XGBRFClassifier
from lightgbm import LGBMClassifier


lgbm_clf={
    'model':Categorical([LGBMClassifier()]),
    'model__boosting_type':Categorical(['gbdt', 'dart', 'goss
    ']),
    'model__num_leaves':Integer(8, 128, 'uniform'),
    'model__max_depth':Integer(4, 128, 'uniform'),
    'model__learning_rate':Real(0.001, 0.4, 'uniform'),
    'model__n_estimators':Integer(100, 150, 'uniform'),
    'model__subsample_for_bin':Integer(200000, 250000, '
    uniform'),
    'model__min_split_gain':Real(0, 0.1, 'uniform'),
```

```
    'model__min_child_weight':Real(0.000001, 0.001,  'uniform
    '),
    'model__min_child_samples':Integer(12, 40, 'uniform'),
    'model__subsample':Real(0.8, 1, 'uniform'),
    'model__colsample_bytree':Real(0.8, 1, 'uniform'),
}

xgbrf_gbtree_clf={
    'model':Categorical([XGBRFClassifier()]),
    'model__booster':Categorical(['gbtree']),
    'model__eta':Real(0.01, 0.99, 'uniform'),
    'model__gamma':Integer(0, 4, 'uniform'),
    'model__max_depth':Integer(4, 128, 'uniform'),
    'model__min_child_weight':Integer(0, 4, 'uniform'),
    'model__max_delta_step':Integer(0, 7, 'uniform'),
    'model__subsample':Real(0.01, 0.99, 'uniform'),
    'model__colsample_bytree':Real(0.01, 0.99, 'uniform'),
    'model__colsample_bylevel':Real(0.01, 0.99, 'uniform'),
    'model__colsample_bynode':Real(0.01, 0.99, 'uniform'),
    'model__n_estimators':Integer(100, 120, 'uniform'),
    'model__objective':Categorical(['reg:squarederror', 'reg:
    squaredlogerror', 'reg:logistic', 'reg:pseudohubererror'])
    ,
}






xgb_gbtree_clf={
    'model':Categorical([XGBClassifier()]),
    'model__booster':Categorical(['gbtree']),
    'model__eta':Real(0.01, 0.99, 'uniform'),
    'model__gamma':Integer(0, 4, 'uniform'),
    'model__max_depth':Integer(4, 128, 'uniform'),
    'model__min_child_weight':Integer(0, 4, 'uniform'),
    'model__max_delta_step':Integer(0, 7, 'uniform'),
    'model__subsample':Real(0.01, 0.99, 'uniform'),
    'model__colsample_bytree':Real(0.01, 0.99, 'uniform'),
    'model__colsample_bylevel':Real(0.01, 0.99, 'uniform'),
    'model__colsample_bynode':Real(0.01, 0.99, 'uniform'),
    'model__n_estimators':Integer(100, 120, 'uniform'),
    'model__objective':Categorical(['reg:squarederror', 'reg:
    squaredlogerror', 'reg:logistic', 'reg:pseudohubererror'])
    ,
}
```

```
gaussianNB_clf={
    'model':Categorical([GaussianNB()]),
    'model__var_smoothing':Real(0.0000000001, 0.0000001, 'uniform')
}

Decision_Tree_clf={
    'model':Categorical([DecisionTreeClassifier()]),
    'model__criterion':Categorical(['gini', 'entropy']),
    'model__splitter':Categorical(['best', 'random']),
    'model__min_samples_split':Integer(2, 8, 'uniform'),
    'model__max_features':Categorical(['sqrt', 'log2']),
    'model__max_depth':Integer(4, 128, 'uniform'),
    'model__min_samples_leaf':Integer(1,4,'uniform'),
}

Random_Forest_clf={
    'model':Categorical([RandomForestClassifier()]),
    'model__n_estimators':Integer(100, 120, 'uniform'),
    'model__criterion':Categorical(['gini', 'entropy']),
    'model__max_depth':Integer(4, 128, 'uniform'),
    'model__min_samples_split':Integer(2, 8, 'uniform'),
    'model__min_samples_leaf':Integer(1,4,'uniform'),
    'model__max_features':Categorical(['sqrt', 'log2']),
    'model__max_samples':Real(0.01, 0.99, 'uniform')
}
AdaBoost_clf={
    'model':Categorical([AdaBoostClassifier()]),
    'model__n_estimators':Integer(40, 100, 'uniform'),
    'model__learning_rate':Real(0.8, 1.2, 'uniform'),
    'model__algorithm':Categorical(['SAMME', 'SAMME.R']),
}

gradientBooster_clf={
    'model':Categorical([GradientBoostingClassifier()]),
    'model__loss':Categorical(['deviance']),
    'model__learning_rate':Real(0.0001, 0.1, 'uniform'),
    'model__n_estimators':Integer(80, 100, 'uniform'),
    'model__subsample':Real(0.7, 1, 'uniform'),
    'model__criterion':Categorical(['friedman_mse', 'squared_error']),
    'model__min_samples_split':Integer(2, 8, 'uniform'),
    'model__min_samples_leaf':Integer(1,4,'uniform'),
    'model__max_depth':Integer(2, 64, 'uniform'),
    'model__max_features':Categorical(['sqrt', 'log2']),
    'model__tol':Real(0.000001, 0.01, 'uniform')
}

```

```
HistGradientBooster_clf={
    'model':Categorical([HistGradientBoostingClassifier()]),
    'model__loss':Categorical(['categorical_crossentropy']),
    'model__learning_rate':Real(0.0001, 0.1, 'uniform'),
    'model__max_iter':Integer(500, 2000, 'uniform'),
    'model__max_leaf_nodes':Integer(20, 50, 'uniform'),
    'model__max_depth':Integer(2, 64, 'uniform'),
    'model__min_samples_leaf':Integer(1,20,'uniform'),
    'model__tol':Real(0.00000001, 0.0001,  'uniform'),
}

knc_clf={
    'model':Categorical([NearestCentroid()]),
    'model__metric':Categorical(['euclidean', 'manhattan', '
   chebyshev', 'minkowski'])
}

Knn_clf={
    'model':Categorical([KNeighborsClassifier()]),
    'model__weights':Categorical(['uniform', 'distance']),
    'model__algorithm':Categorical([ 'kd_tree', 'brute']),
    'model__leaf_size':Integer(10, 50, 'uniform'),
    'model__p':Integer(1, 4, 'uniform'),
    'model__metric':Categorical(['euclidean', 'manhattan', '
   chebyshev', 'minkowski']),
    'model__n_neighbors':Integer(3, 6, 'uniform')
}
#'ball_tree',

svc_clf={
    'model':Categorical([SVC()]),
    'model__kernel':Categorical(['linear', 'poly', 'rbf', '
   sigmoid']),
    'model__degree':Integer(2, 10, 'uniform'),
    'model__gamma':Categorical(['scale', 'auto']),
    'model__tol':Real(0.0000001, 0.01,  'uniform'),
}

QDA_clf={
    'model':Categorical([QuadraticDiscriminantAnalysis()]),
    'model__tol':Real(0.0000001, 0.01,  'uniform')
}

PassiveAggressive_clf={
    'model': Categorical([PassiveAggressiveClassifier()]),
    'model__max_iter':Integer(1000, 10000, 'uniform'),
    'model__tol':Real(0.0000001, 0.01,  'uniform'),
    'model__C': Real(0.01, 0.99, 'uniform'),
    'model__loss': Categorical(['hinge', 'squared_hinge'])
```

```
}


ridge_clf_positive = {
    'model': Categorical([RidgeClassifier(positive=True)]),
    'model__tol': Real(0.0000001, 0.001,  'uniform'),
    'model__alpha':Real(0.00001, 1,  'uniform'),
    'model__max_iter':Integer(1000, 10000, 'uniform')
}

ridge_clf_false = {
    'model': Categorical([RidgeClassifier(positive=False)]),
    'model__solver': Categorical(['svd', 'cholesky', 'sparse_cg', 'sag', 'saga']),
    'model__tol': Real(0.0000001, 0.001,  'uniform'),
    'model__alpha':Real(0.00001, 1,  'uniform'),
    'model__max_iter':Integer(1000, 10000, 'uniform')
}

perceptron_clf = {
    'model': Categorical([Perceptron( fit_intercept=False)]),
    'model__penalty': Categorical(['l2', 'l1', 'elasticnet']),
    'model__alpha': Real(0.00000001, 0.001, 'uniform'),
    'model__l1_ratio': Real(0.01, 0.99, 'uniform'),
    'model__max_iter': Integer(1000, 10000, 'uniform'),
    'model__tol': Real(0.0000001, 0.001,  'uniform'),
}


sgd_clf = {
    'model': Categorical([SGDClassifier(fit_intercept=False)]),
    'model__loss': Categorical(['hinge', 'log', 'modified_huber', 'squared_hinge', 'perceptron', 'squared_error', 'huber', 'epsilon_insensitive', 'squared_epsilon_insensitive']),
    'model__alpha': Real(0.00000001, 0.001, 'uniform'),
    'model__max_iter': Integer(1000, 10000, 'uniform'),
    'model__epsilon': Real(0.0000001, 0.001,  'uniform'),
    'model__power_t': Real(0.01, 0.99, 'uniform'),
    'model__eta0': Real(0.01, 0.99, 'uniform'),
    'model__warm_start': Categorical([True, False]),
    'model__tol': Real(0.0000001, 0.001, 'uniform'),
    'model__penalty': Categorical(['l2', 'l1', 'elasticnet']),
    'model__l1_ratio': Real(0.01, 0.99, 'uniform'),
    'model__learning_rate': Categorical(['constant', 'optimal', 'invscaling', 'adaptive']),
```

```
}




'''

Decision_Tree_reg={
    'model':Categorical([DecisionTreeRegressor()]),
    'model_criterion':Categorical(['squared_error', 'friedman_mse', 'absolute_error', 'poisson']),
    'model_splitter':Categorical(['best', 'random']),
    'model_min_samples_split':Integer(2, 8, 'uniform'),
    'model_max_features':Categorical(['sqrt', 'log2']),
    'model_max_depth':Integer(4, 128, 'uniform'),
    'model_min_samples_leaf':Integer(1,4,'uniform'),
}

Random_Forest_reg={
    'model':Categorical([RandomForestRegressor()]),
    'model_n_estimators':Integer(100, 300, 'uniform'),
    'model_criterion':Categorical(['squared_error', 'absolute_error', 'poisson']),
    'model_max_depth':Integer(4, 128, 'uniform'),
    'model_min_samples_split':Integer(2, 8, 'uniform'),
    'model_min_samples_leaf':Integer(1,4,'uniform'),
    'model_max_features':Categorical(['sqrt', 'log2']),
    'model_max_samples':Real(0.01, 0.99, 'uniform')
}


AdaBoost_reg={
    'model':Categorical([AdaBoostRegressor()]),
    'model_base_estimator_criterion':Categorical(['squared_error', 'friedman_mse', 'absolute_error', 'poisson']),
    'model_base_estimator_splitter':Categorical(['best', 'random']),
    'model_base_estimator_min_samples_split':Integer(2, 8, 'uniform'),
    'model_base_estimator_max_features':Categorical(['sqrt', 'log2']),
    'model_base_estimator_max_depth':Integer(4, 128, 'uniform'),
    'model_n_estimators':Integer(40, 70, 'uniform'),
    'model_learning_rate':Real(1, 3, 'uniform'),
    'model_loss':Categorical(['linear', 'square', 'exponential'])
}
```

```
GradientBoosting_reg={
    'model':Categorical([GradientBoostingRegressor()]),
    'model_loss':Categorical(['squared_error', 'absolute_error', 'huber', 'quantile']),
    'model_learning_rate':Real(0.001, 0.4, 'uniform'),
    'model_n_estimators':Integer(100, 250, 'uniform'),
    'model_criterion':Categorical(['friedman_mse', 'squared_error', 'mse', 'mae']),
    'model_min_samples_split':Integer(2, 8, 'uniform'),
    'model_min_samples_leaf':Integer(1,4,'uniform'),
    'model_max_depth':Integer(4, 128, 'uniform'),
    'model_max_features':Categorical(['sqrt', 'log2']),
    'model_alpha':Real(0.1, 0.9, 'uniform'),
    'model_tol':Real(0.0000001, 0.001,  'uniform')
}


xgb_gbtree_reg={
    'model':Categorical([XGBRegressor()]),
    'model_booster':Categorical(['gbtree']),
    'model_eta':Real(0.01, 0.99, 'uniform'),
    'model_gamma':Integer(0, 4, 'uniform'),
    'model_max_depth':Integer(4, 128, 'uniform'),
    'model_min_child_weight':Integer(0, 4, 'uniform'),
    'model_max_delta_step':Integer(0, 7, 'uniform'),
    'model_subsample':Real(0.01, 0.99, 'uniform'),
    'model_colsample_bytree':Real(0.01, 0.99, 'uniform'),
    'model_colsample_bylevel':Real(0.01, 0.99, 'uniform'),
    'model_colsample_bynode':Real(0.01, 0.99, 'uniform'),
    'model_n_estimators':Integer(100, 300, 'uniform'),
    'model_objective':Categorical(['reg:squarederror', 'reg:squaredlogerror', 'reg:logistic', 'reg:pseudohubererror']),
    'model_eval_metric':Categorical(['rmse', 'rmsle', 'mae', 'mape', 'mphe'])
}



xgb_dart_reg={
    'model':Categorical([XGBRegressor()]),
    'model_booster':Categorical([ 'dart']),
    'model_eta':Real(0.01, 0.99, 'uniform'),
    'model_gamma':Integer(0, 4, 'uniform'),
    'model_max_depth':Integer(4, 128, 'uniform'),
    'model_min_child_weight':Integer(0, 4, 'uniform'),
    'model_max_delta_step':Integer(0, 7, 'uniform'),
```

```
    'model_subsample':Real(0.01, 0.99, 'uniform'),
    'model_colsample_bytree':Real(0.01, 0.99, 'uniform'),
    'model_colsample_bylevel':Real(0.01, 0.99, 'uniform'),
    'model_colsample_bynode':Real(0.01, 0.99, 'uniform'),
    'model_n_estimators':Integer(100, 300, 'uniform'),
    'model_sample_type':Categorical(['uniform', 'weighted']),
    'model_normalize_type':Categorical(['tree', 'forest']),
    'model_rate_drop':Real(0.01, 0.99, 'uniform'),
    'model_skip_drop':Real(0.01, 0.99, 'uniform'),
    'model_objective':Categorical(['reg:squarederror', 'reg:squaredlogerror', 'reg:logistic', 'reg:pseudohubererror']),
    'model_eval_metric':Categorical(['rmse', 'rmsle', 'mae', 'mape', 'mphe'])
}

xgb_linear_reg={
    'model':Categorical([XGBRegressor()]),
    'model_booster':Categorical(['gblinear']),
    'model_feature_selector':Categorical(['cyclic', 'shuffle', 'random', 'greedy', 'thrifty', ]),
    'model_updater':Categorical(['shotgun', 'coord_descent']),
    'model_objective':Categorical(['reg:squarederror', 'reg:squaredlogerror', 'reg:logistic', 'reg:pseudohubererror']),
    'model_eval_metric':Categorical(['rmse', 'rmsle', 'mae', 'mape', 'mphe'])
}


lgbm_reg={
    'model':Categorical([LGBMRegressor()]),
    'model_boosting_type':Categorical(['gbdt', 'dart', 'goss', 'rf']),
    'model_num_leaves':Integer(8, 128, 'uniform'),
    'model_max_depth':Integer(4, 128, 'uniform'),
    'model_learning_rate':Real(0.001, 0.4, 'uniform'),
    'model_n_estimators':Integer(100, 300, 'uniform'),
    'model_subsample_for_bin':Integer(200000, 250000, 'uniform'),
    'model_min_split_gain':Real(0, 0.1, 'uniform'),
    'model_min_child_weight':Real(0.001, 0.000001, 'uniform'),
    'model_min_child_samples':Integer(12, 40, 'uniform'),
    'model_subsample':Real(0.8, 1, 'uniform'),
    'model_colsample_bytree':Real(0.8, 1, 'uniform'),

}
```

```
xgbrf_dart_clf={
    'model':Categorical([XGBRFClassifier()]),
    'model__booster':Categorical([ 'dart']),
    'model__eta':Real(0.01, 0.99, 'uniform'),
    'model__gamma':Integer(0, 4, 'uniform'),
    'model__max_depth':Integer(4, 128, 'uniform'),
    'model__min_child_weight':Integer(0, 4, 'uniform'),
    'model__max_delta_step':Integer(0, 7, 'uniform'),
    'model__subsample':Real(0.01, 0.99, 'uniform'),
    'model__colsample_bytree':Real(0.01, 0.99, 'uniform'),
    'model__colsample_bylevel':Real(0.01, 0.99, 'uniform'),
    'model__colsample_bynode':Real(0.01, 0.99, 'uniform'),
    'model__n_estimators':Integer(100, 120, 'uniform'),
    'model__sample_type':Categorical(['uniform', 'weighted']),
    'model__normalize_type':Categorical(['tree', 'forest']),
    'model__rate_drop':Real(0.01, 0.99, 'uniform'),
    'model__skip_drop':Real(0.01, 0.99, 'uniform'),
    'model__objective':Categorical(['reg:squarederror', 'reg:squaredlogerror', 'reg:logistic', 'reg:pseudohubererror']),
}


xgb_dart_clf={
    'model':Categorical([XGBClassifier()]),
    'model__booster':Categorical([ 'dart']),
    'model__eta':Real(0.01, 0.99, 'uniform'),
    'model__gamma':Integer(0, 4, 'uniform'),
    'model__max_depth':Integer(4, 128, 'uniform'),
    'model__min_child_weight':Integer(0, 4, 'uniform'),
    'model__max_delta_step':Integer(0, 7, 'uniform'),
    'model__subsample':Real(0.01, 0.99, 'uniform'),
    'model__colsample_bytree':Real(0.01, 0.99, 'uniform'),
    'model__colsample_bylevel':Real(0.01, 0.99, 'uniform'),
    'model__colsample_bynode':Real(0.01, 0.99, 'uniform'),
    'model__n_estimators':Integer(100, 120, 'uniform'),
    'model__sample_type':Categorical(['uniform', 'weighted']),
    'model__normalize_type':Categorical(['tree', 'forest']),
    'model__rate_drop':Real(0.01, 0.99, 'uniform'),
    'model__skip_drop':Real(0.01, 0.99, 'uniform'),
    'model__objective':Categorical(['reg:squarederror', 'reg:squaredlogerror', 'reg:logistic', 'reg:pseudohubererror']),
}
```

```
'''
```

Listing 2: contains the hyperparameters that will be fetched

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder


def preproc_normalize(X_train=None, X_test=None, y_train=None, y_test=None, scaler=None, scaler_trigger=False, X_test_trigger=True):
    '''normilize the data with MinMaxScaler
    Args:
      X_train, X_test, y_train, y_test
    Return:
      X_train, X_test, y_train, y_test
    '''

    if scaler_trigger == False:
        scaler = MinMaxScaler()
        scaler.fit(X_train)

    X_train = scaler.transform(X_train)
    if X_test_trigger == True:
        X_test = scaler.transform(X_test)

    return X_train, X_test, y_train, y_test, scaler


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):
    '''replace Nan values of categorical and numerical features. Moreover, transform categorical features in numerical data
    Args:
      X_train, X_test, y_train, y_test
      categorical_features, list with the name of the categorical columns
      numerical_features, list with the name of the numerical columns
    Return:
      X_train, X_test, y_train, y_test
    '''
```

```

    numerical_pipeline = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='mean'))])

    categorical_pipeline = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('onehot', OneHotEncoder())])

    transformation = ColumnTransformer(
        transformers=[
            ('numerical transformation', numerical_pipeline, numerical_features),
            ('categorical transformation',
             categorical_pipeline, categorical_features),
        ])

    X_train = transformation.fit_transform(X_train)
    if X_test_trigger == True:
        X_test = transformation.transform(X_test)

    if scaler_trigger == False and X_test_trigger == False:
        X_train, X_test, y_train, y_test, scaler = normalize_fn(
            X_train=X_train, y_train=y_train, X_test_trigger=False)

    elif scaler_trigger == True and X_test_trigger == False:
        X_train, X_test, y_train, y_test, scaler = normalize_fn(
            X_train=X_train, y_train=y_train, X_test_trigger=False, scaler=scaler_fn, scaler_trigger=True)

    elif scaler_trigger == False and X_test_trigger == True:
        X_train, X_test, y_train, y_test, scaler = normalize_fn(
            X_train, X_test, y_train, y_test)
    else:
        X_train, X_test, y_train, y_test, scaler = normalize_fn(
            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)

    return X_train, X_test, y_train, y_test, scaler
```

Listing 3: Data normalizations

```
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
```

```
from drive.MyDrive.Data_Science_Competitions.Project3.Utils.
    preprocessing.data_treatment import undersample_boostrap,
    feature_eng
from drive.MyDrive.Data_Science_Competitions.Project3.Utils.
    preprocessing.data_preprocessing import preproc_normalize,
     preprocess
from skopt import BayesSearchCV
from skopt.plots import plot_objective
from drive.MyDrive.Data_Science_Competitions.Project3.Utils.
    model_params.model_classifiers import *
from sklearn.pipeline import Pipeline
import joblib
from skopt.plots import plot_objective
import matplotlib.pyplot as plt
from skopt.plots import plot_convergence
from sklearn import metrics
import seaborn as sn
from sklearn.svm import SVC


def Train_model(lgbm_clf_n_calls, xgbrf_dart_clf_n_calls,
    xgbrf_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, knc_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):
     '''Apply feature engineering(data_treatment) and
    preprocessing to the train data. Furthermore, do
    optimization of hyperparameters by pipeline
        and BayesSearch CV. At the end, save results and creat
     plots for analysis
     Args:
       model_n_calls,  number of calls for bayesian
    optimization
       plot, if plot is True, the function create plot of
    dependes, convergence, ROC and Confusion Matrix
     Return:
       dropped_columns, columns dropped from train data
       columns_to_keep, columns with relevant information
     '''

     ''' '''
     # reading the training file
     train_df = pd.read_csv('/content/drive/MyDrive/
    Data_Science_Competitions/Project3/Data/train.csv')
```

```

    # concatening two datframe and aplying feature engeneering
    new_train_df, dropped_columns, columns_to_keep= feature_eng(train_df)

    # creating the features and targets for this study
    X = new_train_df
    y = train_df['label']
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, shuffle=False)

    # undersample data(there is to many no Fraud in samples)
    X_train, y_train = undersample_boostrap(X_train, y_train, 0)

    # preprocessing(replacing Nan values, Hot enconder and normalization)
    numerical_features = X._get_numeric_data().columns.tolist()
    categorical_features = [attribute for attribute in X.columns.tolist() if attribute not in X._get_numeric_data().columns.tolist()]
    X_train, X_test, y_train, y_test, scaler = preprocess(X_train, X_test, y_train, y_test, categorical_features, numerical_features, preproc_normalize)

    #pipeline for many models
    pipeline = Pipeline([
        ('model', SVC())
    ])

    #models for pipeline
    Search_spaces=[
        (lgbm_clf, lgbm_clf_n_calls),
        (xgbrf_gbtree_clf, xgbrf_gbtree_clf_n_calls),
        (xgb_gbtree_clf, xgb_gbtree_clf_n_calls),
        (gaussianNB_clf, gaussianNB_clf_n_calls),
        (Decision_Tree_clf, Decision_Tree_clf_n_calls),
        (Random_Forest_clf, Random_Forest_clf_n_calls),
        (AdaBoost_clf, AdaBoost_clf_n_calls),
        (knc_clf, knc_clf_n_calls),
        (Knn_clf, Knn_clf_n_calls),
        (svc_clf, svc_clf_n_calls),
        (PassiveAggressive_clf,PassiveAggressive_clf_n_calls),
        (sgd_clf, sgd_n_calls),
        (ridge_clf_false, ridge_false_n_calls),
        (ridge_clf_positive, ridge_positive_n_calls),
```

```
        (perceptron_clf, perceptron_n_calls,),
        (QDA_clf, QDA_clf_n_calls)
    ]

    '''
    (xgbrf_dart_clf, xgbrf_dart_clf_n_calls),
    (xgb_dart_clf, xgb_dart_clf_n_calls),
    (HistGradientBooster_clf, HistGradientBooster_clf_n_calls),
    (gradientBooster_clf, gradientBooster_clf_n_calls),
        '''

    optimizer = BayesSearchCV(estimator=pipeline, search_spaces=Search_spaces, cv=3, scoring='accuracy', verbose=3)

    #fittimg models and cross-validation
    optimizer.fit(X_train, np.ravel(y_train))

    print("val. score: %s" % optimizer.best_score_)
    print("test score: %s" % optimizer.score(X_test, y_test))
    print("best params: %s" % str(optimizer.best_params_))

    # saving best model and results
    joblib.dump(optimizer.best_estimator_, 'best_estimator.pkl')
    np.save('my_results.npy', optimizer.cv_results_)


    #---IMAGES FOR ANALYSIS --- #

    classifiers = [
        'LGBMClassifier',
        'XGBRFClassifier_gbtree',
        'XGBClassifier_gbtree',
        'GaussianNB',
        'DecisionTreeClassifier',
        'RandomForestClassifier',
        'AdaBoostClassifier',
        'KNeighborsClassifier',
        'NearestCentroid',
        'SVC',
        'PassiveAggressiveClassifier',
        'SGDClassifier',
        'RidgeClassifierFalse',
        'RidgeClassifierPositive',
        'Perceptron',
        'QuadraticDiscriminantAnalysis'
        ]
```

```
    '''
    'XGBRFClassifier_dart',
    'XGBClassifier_dart',
    'HistGradientBoostingClassifier',
    'GradientBoostingClassifier',
        '''

    if plot == True:

        for i in range(len(optimizer.optimizer_results_)):
            plt.title(classifiers[i])
            _ = plot_objective(optimizer.optimizer_results_[i
])
            plt.savefig(classifiers[i]+'_dependence.png')
            plt.clf()

        plt.rcParams['figure.figsize'] = [15, 15]
        plt.title("Convergence of models")
        clf_plot = ((classifiers[index], optimizer.
optimizer_results_[
                    index]) for index in range(len(
classifiers)))
        plot = plot_convergence(*clf_plot)
        plot.legend(loc="best", prop={'size': 6}, numpoints
=1)
        plt.savefig('Convergence.png')
        plt.clf()


        plt.title("Confusion Matrix Best Score")
        metrics.plot_confusion_matrix(optimizer.
best_estimator_, X_test, y_test, cmap="inferno")
        plt.savefig("Confusion_Matrix.png")
        plt.clf()

        try:
          print(optimizer.best_estimator_.predict_proba(
X_test))
        except:
          print("predict_proba does not exist for this
classifier")


    # save scaler from of the training
    X = new_train_df
    y = train_df['label']
    X_train, X_test, y_train, y_test, scaler = preprocess(
        X_train=X, y_train=y, categorical_features=
```

```
    categorical_features, numerical_features=
    numerical_features, normalize_fn=preproc_normalize,
    X_test_trigger=False)
    joblib.dump(scaler, 'scaler.save')

    return dropped_columns, columns_to_keep
```

Listing 4: training the models

```
import joblib
import pandas as pd
from drive.MyDrive.Data_Science_Competitions.Project3.Utils.
    preprocessing.data_preprocessing import preproc_normalize,
     preprocess
import copy

def submission(dropped_columns, columns_to_keep):
    '''Read best model and predict the file Test of kaggle,
    at the end creates kaggle_submission.csv with predictions
    Args:
      dropped_columns, columns to be dropped from Test file(
    obtained in training)
      columns_to_keep, columns with relevant information
    Return:
      None
    '''

    # load best model
    best_model = joblib.load('best_estimator.pkl')
    print(dropped_columns)

    # reading test file
    test_df = pd.read_csv('/content/drive/MyDrive/
    Data_Science_Competitions/Project3/Data/test.csv')

    submission_df = pd.DataFrame(data=[index+1 for index in
    test_df.index], columns=['ImageId'])

    #applying drop to the columns founded in training and
    reliability of some columns
    test_df.drop(columns=dropped_columns, inplace=True)


    # creating the features and targets for this study
    X = test_df[columns_to_keep]

    # getting the name of numerical and categorical columns
    numerical_features = X._get_numeric_data().columns.tolist
    ()
```

```
    categorical_features = [attribute for attribute in X.columns.tolist() if attribute not in X._get_numeric_data().columns.tolist()]

    # applying the preprocessing using the scaler obtained from Training
    scaler = joblib.load('scaler.save')
    X_train, X_test, y_train, y_test, scaler = preprocess(X_train=X, categorical_features=categorical_features, numerical_features=numerical_features,
                                                          normalize_fn=preproc_normalize, scaler_fn=scaler, scaler_trigger=True, X_test_trigger=False)

    # prediction
    y_pred = best_model.predict(X_train)

    # kaggle submission
    submission_df['Label'] = y_pred
    submission_df.to_csv('kaggle_submission.csv', index=False)
```

Listing 5: Create kaggle submission

```
from drive.MyDrive.Data_Science_Competitions.Project3.Train_Test_model_Scikit import Train_model
from drive.MyDrive.Data_Science_Competitions.Project3.kaggle_submission import submission

def Main(dict_of_calls, trigger_plot=False):
    '''Train models and create kaggle submission
    Args:
      list_of_calls, list with the number of calls for each model
      trigger_plot, if True, create plots of dependence, convergence, ROC and confusion Matrix
    Return:
      None
    '''

    '''
    classifiers = [
        'lgbm_clf_n_calls',
        'xgbrf_dart_clf_n_calls',
        'xgbrf_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',
        'knc_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']

    '''
    #classifiers_calls=[30, 30, 40, 60, 30, 20]


    print(dict_of_calls)

    print("Training Model")
    dropped_columns, columns_to_keep=Train_model(**dict_of_calls, plot=trigger_plot)

    print("Creating kaggle submission")
    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:

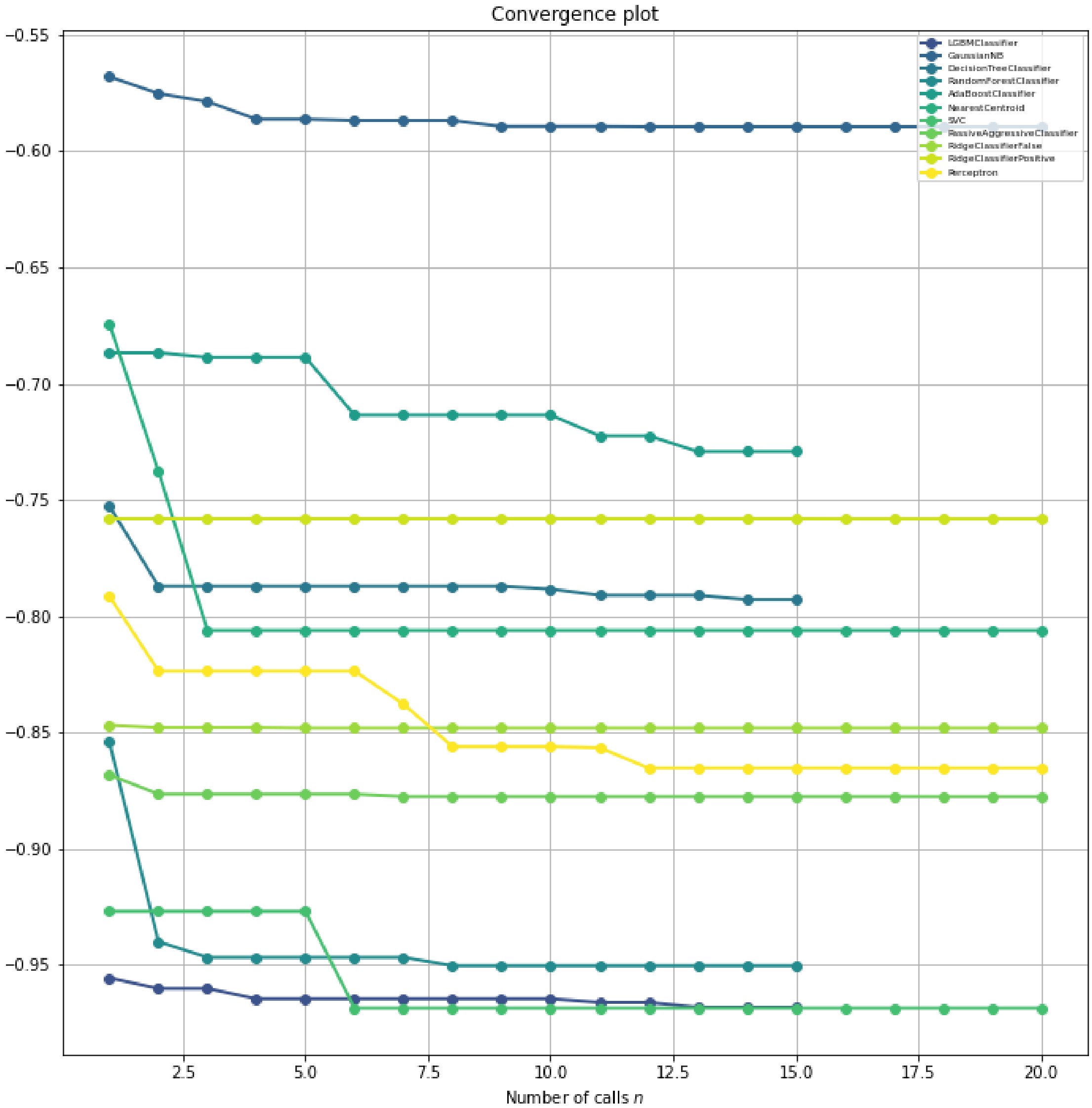

Figure 4: Convergence

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

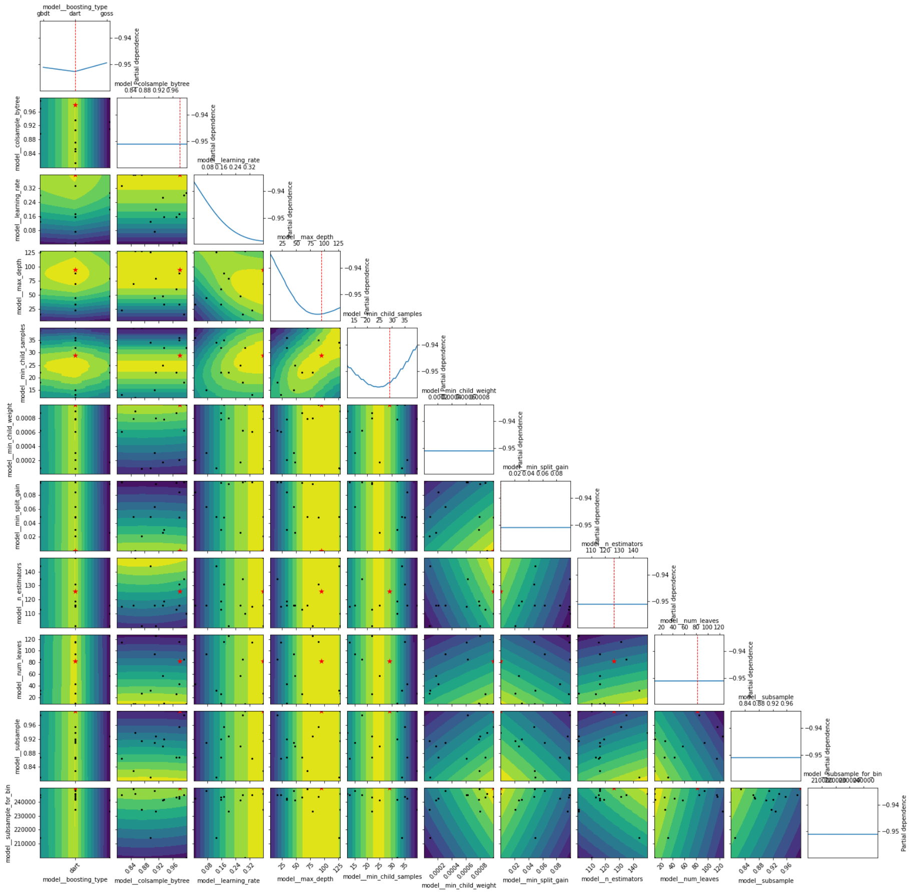

Figure 5: LGBMClassifier

4.1.2 GaussianNB

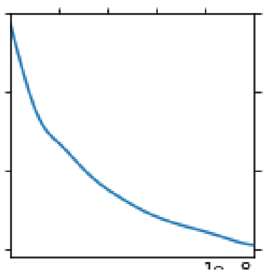

Figure 6: GaussianNB

4.1.3 DecisionTreeClassifier

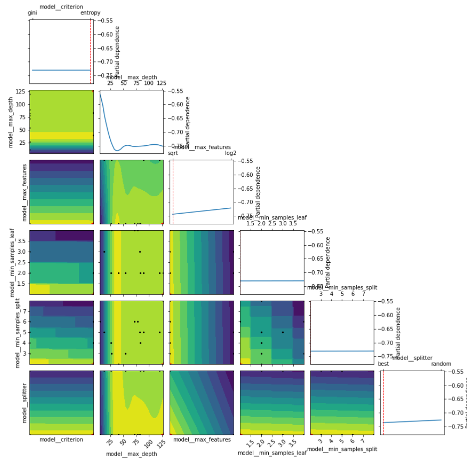

Figure 7: DecisionTreeClassifier

4.1.4 RandomForestClassifier

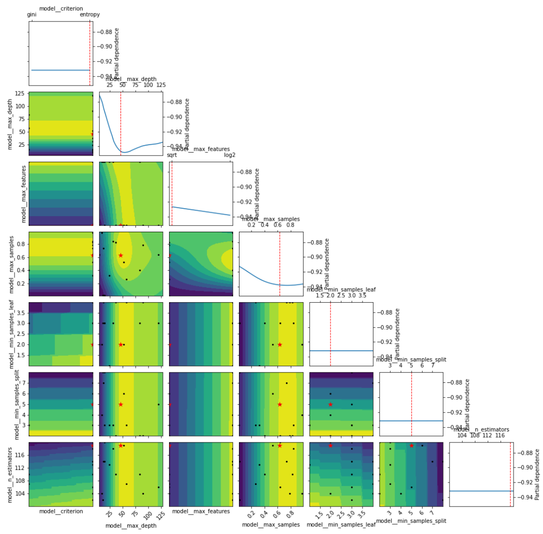

Figure 8: RandomForestClassifier

4.1.5 AdaBoostClassifier

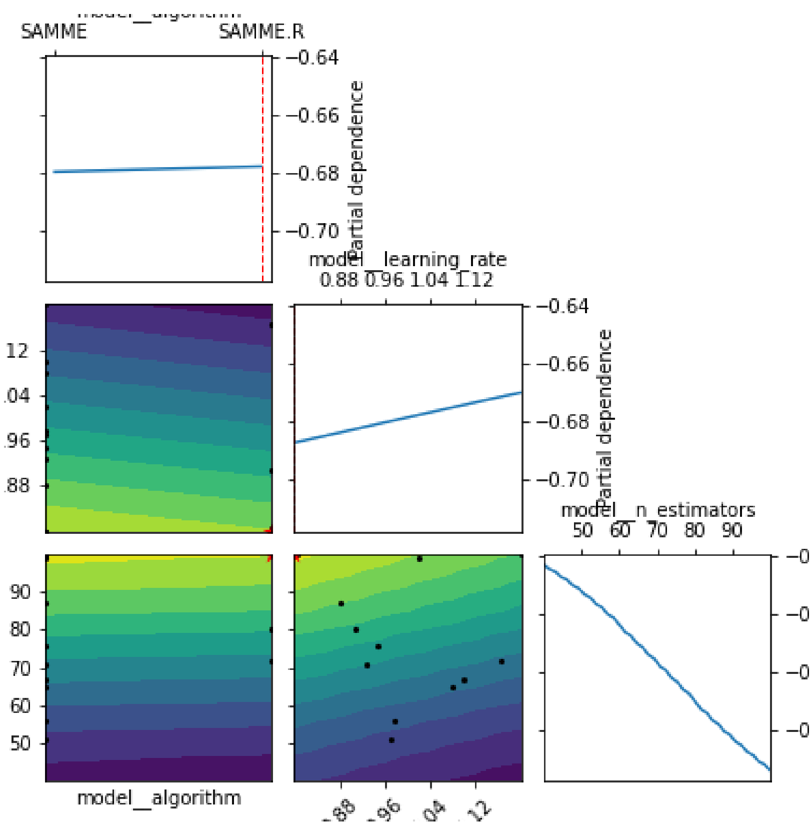

Figure 9: AdaBoostClassifier

4.1.6 NearestCentroid

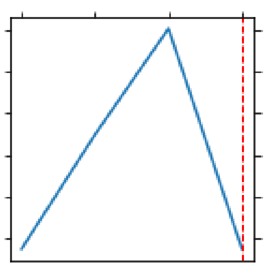

Figure 10: NearestCentroid

4.1.7 SVC

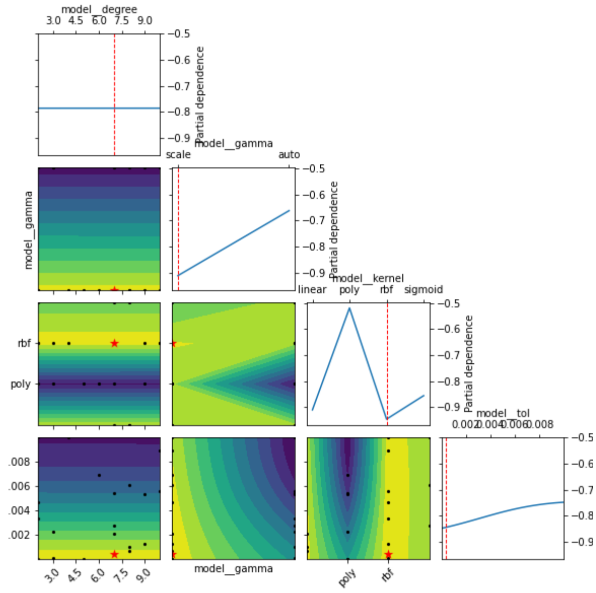

Figure 11: SVC

4.1.8 PassiveAggressiveClassifier

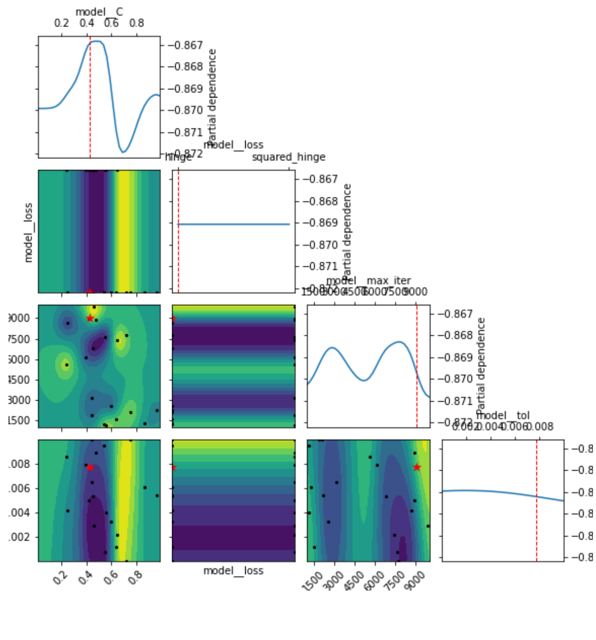

Figure 12: PassiveAggressiveClassifier

4.1.9 RidgeClassifierFalse

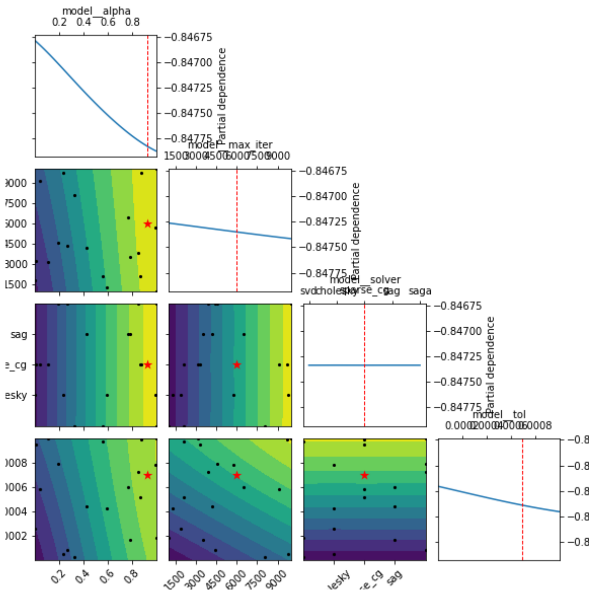

Figure 13: RidgeClassifierFalse

4.1.10 RidgeClassifierPositive

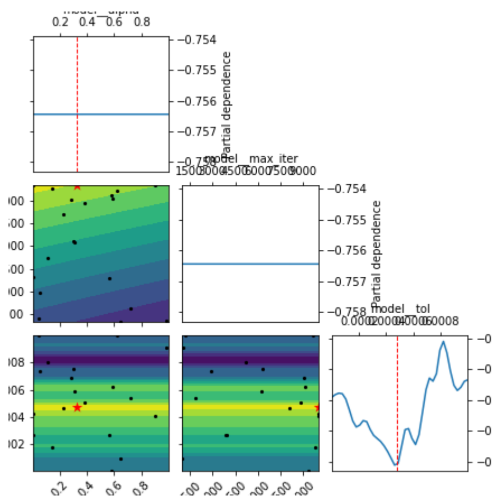

Figure 14: RidgeClassifierPositive

4.1.11 Perceptron

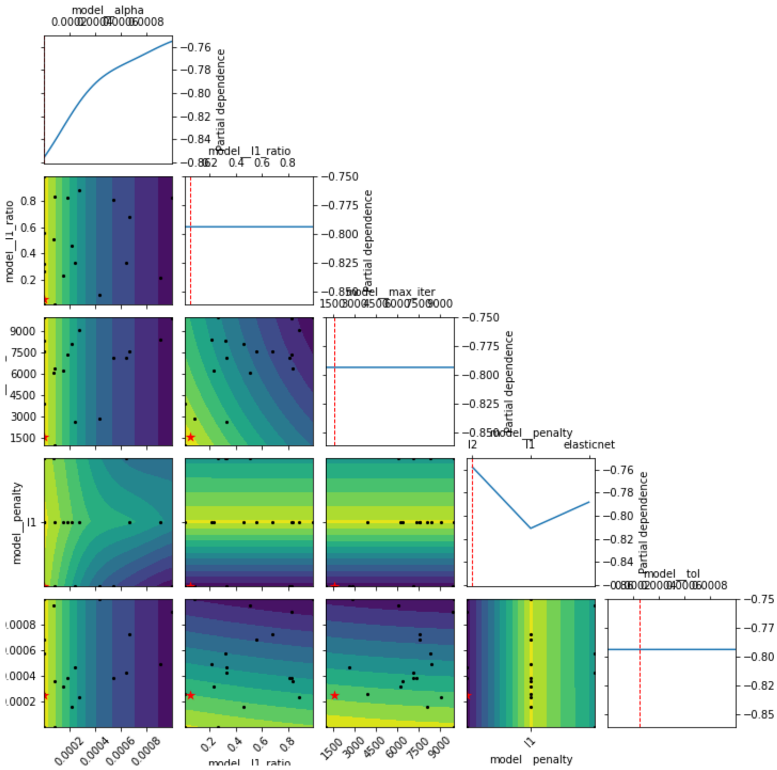

Figure 15: Perceptron

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).

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

```
#from attention import Attention
from tensorflow.keras.models import Model
#from keras_self_attention import SeqSelfAttention


class Models_Constructor():

    def __init__(self, num_classes, input_shape = (28, 28, 1)):

        self.num_classes=num_classes
        self.input_shape=input_shape


    def ConvNet_builder(self, hp):

        # defining a set of hyperparametrs for tuning
        conv_filters=hp.Int(name = 'filters', min_value = 1, max_value = 256, step = 1)
        conv_kernel=hp.Int(name = 'kernel_size', min_value = 4, max_value = 8, step = 1)
        conv_pool_size=hp.Int(name = 'pool_size', min_value = 1, max_value = 2, step = 1)
        conv_activation = hp.Choice(name='activation', values = ['tanh', 'relu','sigmoid', 'softmax', 'softplus'], ordered = False)
        #dropout layer
        dropout = hp.Float(name = 'dropout', min_value=0, max_value=.5, step=0.05)
        #Dense layers
        activation = hp.Choice(name='activation', values = ['tanh', 'relu','sigmoid', 'softmax', 'softplus'], ordered = False)
        activation1 = hp.Choice(name='activation1', values = ['tanh', 'relu','sigmoid', 'softmax', 'softplus'], ordered = False)
        activation2 = hp.Choice(name='activation2', values = ['tanh', 'relu','sigmoid', 'softmax', 'softplus'], ordered = False)
        units=hp.Int(name = 'units', min_value = 1, max_value = 256, step = 1)
        #learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4, 1e-5, 1e-6])
        optimizer=hp.Choice('optimizer', ['sgd', 'adam', 'rmsprop', 'adadelta', 'adagrad', 'adamax', 'ftrl'])

        #model
        model=Sequential()
```

```
        model.add(Conv2D(2*conv_filters, kernel_size=(
    conv_kernel, conv_kernel), activation=conv_activation,
    input_shape=self.input_shape))
        model.add(MaxPooling2D(pool_size=(conv_pool_size,
    conv_pool_size)))
        model.add(Conv2D(conv_filters, kernel_size=(
    conv_kernel, conv_kernel), activation=conv_activation))
        model.add(MaxPooling2D(pool_size=(conv_pool_size,
    conv_pool_size)))
        model.add(Flatten())
        model.add(Dropout(dropout))
        model.add(Dense(units=units, activation=activation1))
        model.add(Dense(self.num_classes, activation=
    activation2))

        model.compile(loss="categorical_crossentropy",
    optimizer=optimizer, metrics=["accuracy"])

        return model
```

Listing 7: Neural Architectures

```

import keras_tuner as kt
from sklearn.model_selection import train_test_split
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.metrics import confusion_matrix
from drive.MyDrive.Data_Science_Competitions.Project3.Utils.
    model_params.Deep_architectures import Models_Constructor



def Train_Test_keras():


    PROJECT_PATH="/content/drive/MyDrive/
    Data_Science_Competitions/Project3/"
    train_df=pd.read_csv(PROJECT_PATH+"Data/train.csv")

    features_df = train_df.drop(columns=['label'])
    targets_df = train_df['label']

    num_classes = len(targets_df.value_counts().index)
    input_shape = (28, 28, 1)

    X_train, X_test, y_train, y_test = train_test_split(
    features_df, targets_df, train_size=0.8)

```

```
    X_train = X_train.values.reshape(X_train.shape[0], 28, 28, 1)
    X_test = X_test.values.reshape(X_test.shape[0], 28, 28, 1)
    X_train = X_train.astype("float32") / 255
    X_test = X_test.astype("float32") / 255

    y_train = keras.utils.to_categorical(y_train, num_classes)
    y_test = keras.utils.to_categorical(y_test, num_classes)

    train_callbacks = [
        keras.callbacks.EarlyStopping(
            monitor="val_accuracy", patience=10, restore_best_weights=True
        )
    ]

    Builders=Models_Constructor(num_classes=num_classes)
    bayesian_tuner=kt.BayesianOptimization(Builders.ConvNet_builder,
                                           objective='val_accuracy',
                                           max_trials=40,
                                           executions_per_trial=3,
                                           overwrite=False,
                                           beta=3.6,
                                           directory="My_Dir",
                                           project_name="DigitRecognizer")

    bayesian_tuner.search(X_train, y_train, epochs=100, callbacks=train_callbacks, validation_split=0.3)

    best_model=bayesian_tuner.get_best_models()[0]
    y_pred = best_model.predict(X_test)
    y_pred=np.argmax(y_pred, axis=1, out=None)
    y_test=np.argmax(y_test, axis=1, out=None)

    from sklearn.metrics import confusion_matrix

    plt.rcParams['figure.figsize'] = [20, 15]
    plt.title("Confusion Matrix Best Score")
    conf_matrix=confusion_matrix(y_test, y_pred)
    #sn.heatmap(conf_matrix, annot=True, cmap="inferno")
    sn.heatmap(conf_matrix/np.sum(conf_matrix), annot=True, fmt='.3%', cmap="inferno")
```

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

Listing 8: Training the model

```

import keras_tuner as kt
from sklearn.model_selection import train_test_split
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.metrics import confusion_matrix
from drive.MyDrive.Data_Science_Competitions.Project3.Utils.model_params.Deep_architectures import Models_Constructor


def create_kaggle():

    test_df = pd.read_csv('/content/drive/MyDrive/Data_Science_Competitions/Project3/Data/test.csv')

    submission_df = pd.DataFrame(data=[index+1 for index in test_df.index], columns=['ImageId'])

    kaggle_test=test_df.values
    kaggle_test=kaggle_test.astype("float32") / 255
    kaggle_test = kaggle_test.reshape(kaggle_test.shape[0], 28, 28, 1)

    bayesian_tuner=kt.BayesianOptimization(Builders.ConvNet_builder,
                                           objective='val_accuracy',
                                           max_trials=80,
                                           executions_per_trial=2,
                                           overwrite=False,
                                           beta=3.6,
                                           directory="/content/drive/MyDrive/Data_Science_Competitions/Project3/My_Dir",
                                           project_name="DigitRecognizer")

    best_model=bayesian_tuner.get_best_models()[0]
    kaggle_pred = best_model.predict(kaggle_test)

    submission_df = pd.DataFrame(data=[index+1 for index in test_df.index], columns=['ImageId'])
    submission_df['Label'] = np.argmax(kaggle_pred, axis=1, out=None)
    submission_df.to_csv('kaggle_submission.csv', index=False
```

```
    )
```

Listing 9: Create kaggle submission

```
from drive.MyDrive.Data_Science_Competitions.Project3.DeepBayesianSearch import Train_Test_keras
from drive.MyDrive.Data_Science_Competitions.Project3.kaggle_submission_keras import create_kaggle

def Main():

    Train_Test_keras()
    create_kaggle()


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.

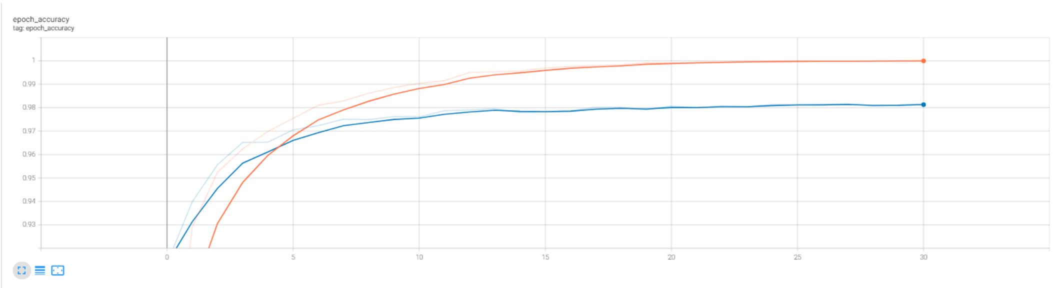

Figure 16: Accuracy in each epoch

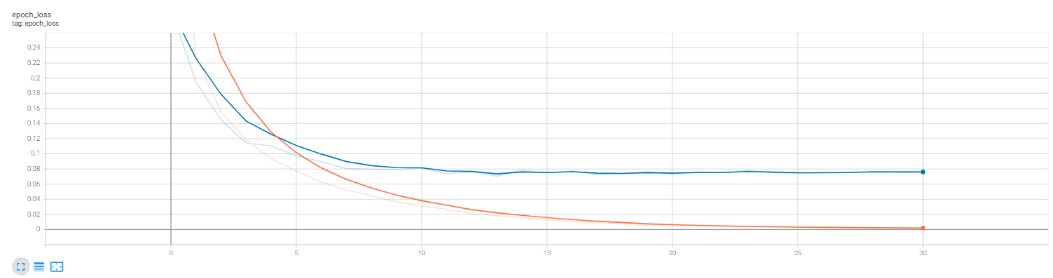

Figure 17: Loss in each epoch

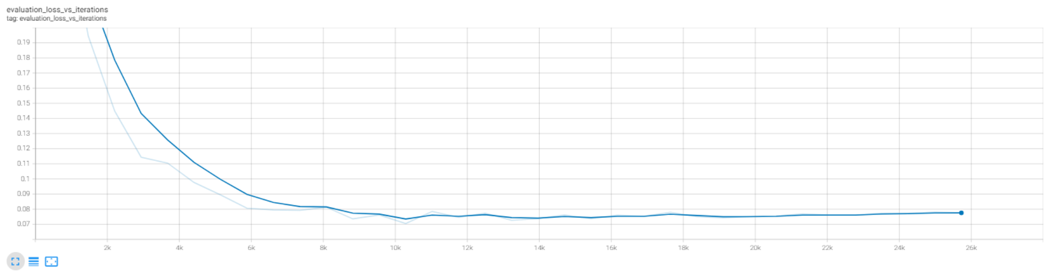

Figure 18: Evaluation x Loss

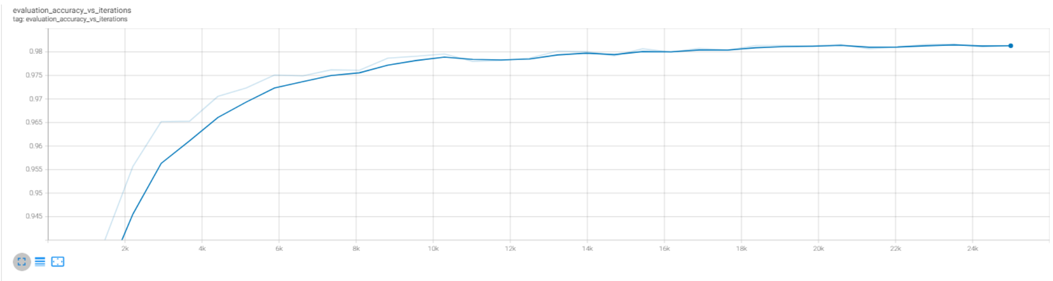

Figure 19: Evaluation x Iteration

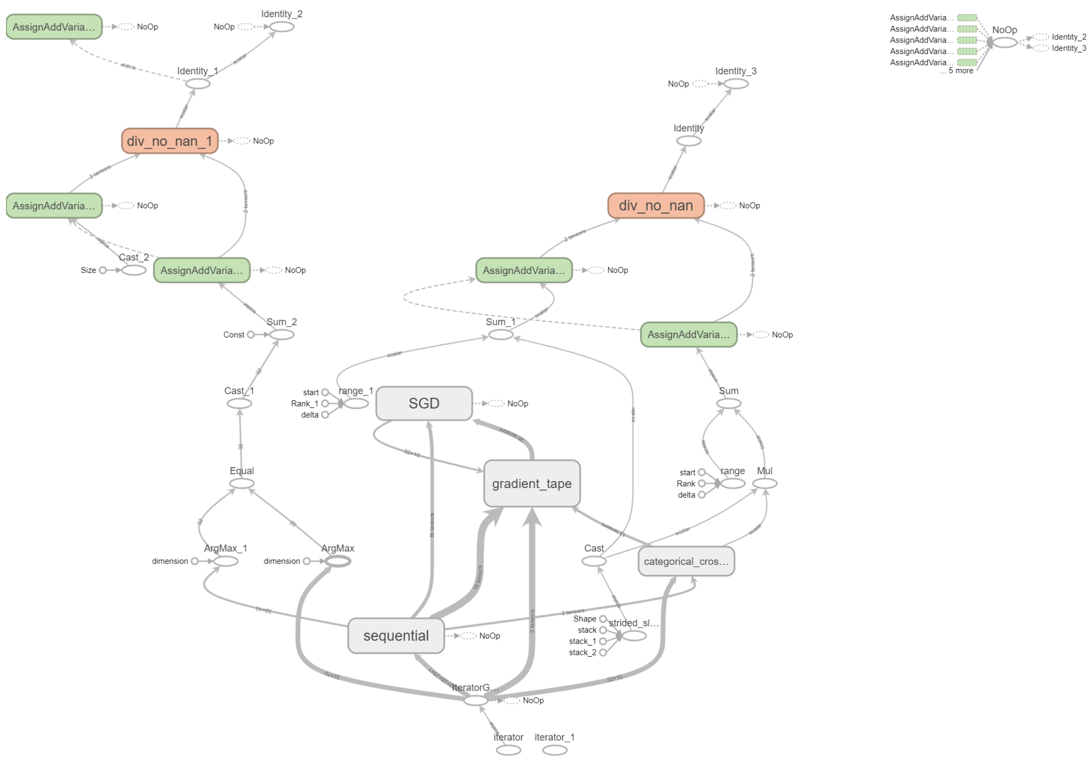

Figure 20: Training Schema

4.3 Submissions

Figure 21: Scikit Submission

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.

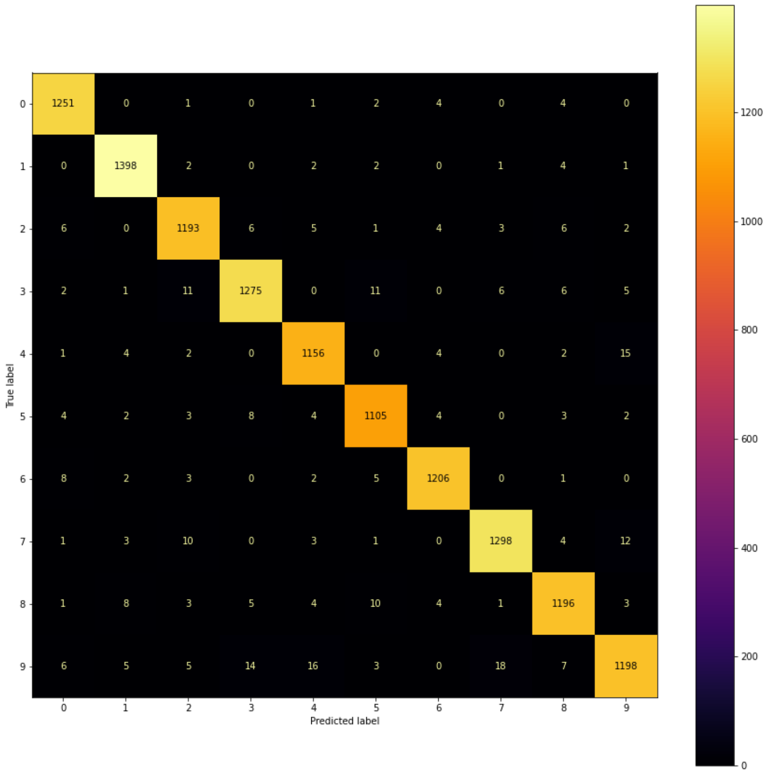

Figure 23: Confusion Matrix in Scikit Learn

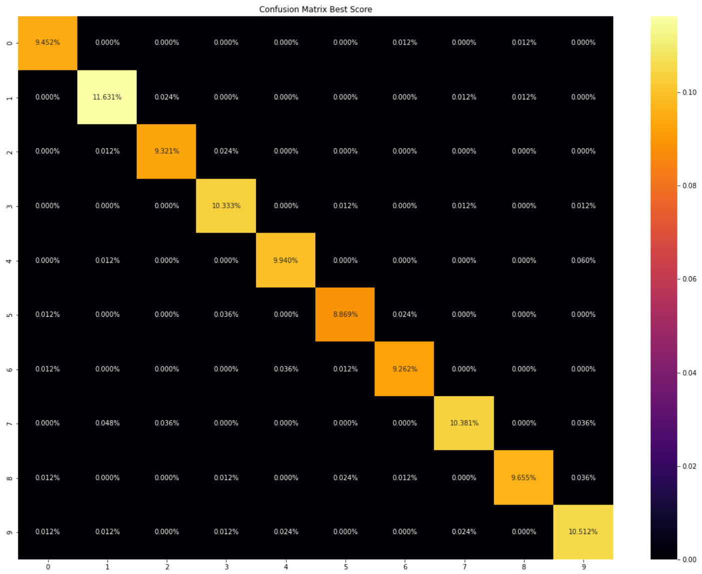

Figure 24: Confusion Matrix in Keras

6 Reference

[1] da Silva, Éverton Luís Mendes. Codes from this project

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