Attention: since I had a little time to write the codes I just used 1000 samples for training. So this results may differ with results of training over entire train set.

In this project, to choose the best possible classifier, the class of "classification" is defined. This class has predefined and almost all existing machine classifiers including KNN, linear and non-linear SVM, Logistic regression classifier, decision tree, random forest, stochastic gradient decent, Adaboost, MLP and etc.

```
self.models = {'Nearest
Neighbors':KNeighborsClassifier(n_neighbors=5),
              'Linear SVM':SVC(kernel="linear", C=0.5),
              'RBF SVM':SVC(gamma=2, C=1),

              'Decision Tree': DecisionTreeClassifier(max_depth=5),
              'Random Forest':RandomForestClassifier(max_depth=5,
n_estimators=10, max_features="auto"),
              'MLP':MLPClassifier(alpha=1, max_iter=1000),
              'AdaBoost':AdaBoostClassifier(),
              'Naive Bayes' :GaussianNB(),
              'QDA':QuadraticDiscriminantAnalysis(),
              'SGD':SGDClassifier(loss="hinge", penalty="l2",
max_iter=5),
              'NearestCenter': NearestCentroid(),
              'Gradient
Boosting':GradientBoostingClassifier(n_estimators=100,
learning_rate=0.1,max_depth=3,

random_state=0, loss='exponential'),
              'Logistic Regression': LogisticRegression(random_state=0,
solver='lbfgs',penalty='l2')}
```

Function

```
def proccess_train_data(self)
```

clears the given training set and prepares the data for machine learning classifiers. It also can give the option whether to drop Nan values or interpolate them (linear, quadratic, zero etc. interpolation). It also encodes string values to float values.

```
if self.nan_values==0:
    # Remove All Nan values
    data=pd.read_csv(self.file_name_train+".csv").dropna(axis=0,
how='any')
    # change the currency to value
    #see all object categorial data
    #data.select_dtypes(include=['object']).head(10)
    data["x41"]=data["x41"].replace('[\$,]', '',
regex=True).astype(float)
    data["x45"]=data["x45"].replace('[,\%]', '',
regex=True).astype(float)
    data["x34"]=data["x34"].astype('category').cat.codes
```

```
    data["x35"] = data["x35"].astype('category').cat.codes
    data["x68"]=data["x68"].astype('category').cat.codes
    data["x93"] = data["x93"].astype('category').cat.code
```

Function

```
def proccess_test_data(self)
```

clears the test data set and linearly interpolate the missing values.

```
data = pd.read_csv(self.file_name_test + ".csv")
data["x41"] = data["x41"].replace('[\$,]', '',
regex=True).astype(float)
data["x45"] = data["x45"].replace('[,\%]', '',
regex=True).astype(float)
data["x34"] = data["x34"].astype('category').cat.codes
data["x35"] = data["x35"].astype('category').cat.codes
data["x68"] = data["x68"].astype('category').cat.codes
data["x93"] = data["x93"].astype('category').cat.codes
data = data.interpolate(method=self.interpolate,
limit_direction='both')
```

Function

```
def model_evaluation(self,train_data,scores)
```

evaluates all the classifiers in our class based on different metrics. Since I saw lots of **multi-collinearity** warning for features, thus in the rest of the code, I used cross-entropy loss as decision making score for choosing the best models. However, the other scores including mean squared error, f1 score (which is basically R-squared for classifiers), accuracy and precisions are measured and plotted. All the results for all models are saved in **evaluation_results.csv**

```
X_train=train_data.iloc[0:100,0:train_data.shape[1]-1]
Y_label=train_data.iloc[0:100,train_data.shape[1]-1]
seed=10

eval_results =
pd.DataFrame(columns=scores.keys(),index=self.models.keys())
for key,value in self.models.items():
    # kfold = model_selection.KFold(n_splits=self.k_fold,
random_state=seed)
    result=cross_validate(value, X_train, Y_label,
                          scoring=scores, cv = self.k_fold,
                          return_train_score = True)
    for metric in scores.keys():
        tmp = 'test_' + metric
        eval_results.loc[key, metric] = result[tmp].mean()
        print('For "' + key + ' ' + metric + '" is {}
'.format(result[tmp].mean()))

eval_results.to_csv('evaluation_results.csv')
return eval_results
```

Function

```
def select_best_models(self,dataframe)
```

returns the best two models among all classifiers based on the out put of function **model_evaluation** .

```
"""selection is based on class given input"""
# for simplicity we already know that it is based on Entropy
dataframe.sort_values(by=self.select,ascending=False)
best_models = [dataframe[self.select].index[0],
dataframe[self.select].index[1]]
return best_models
```

Function

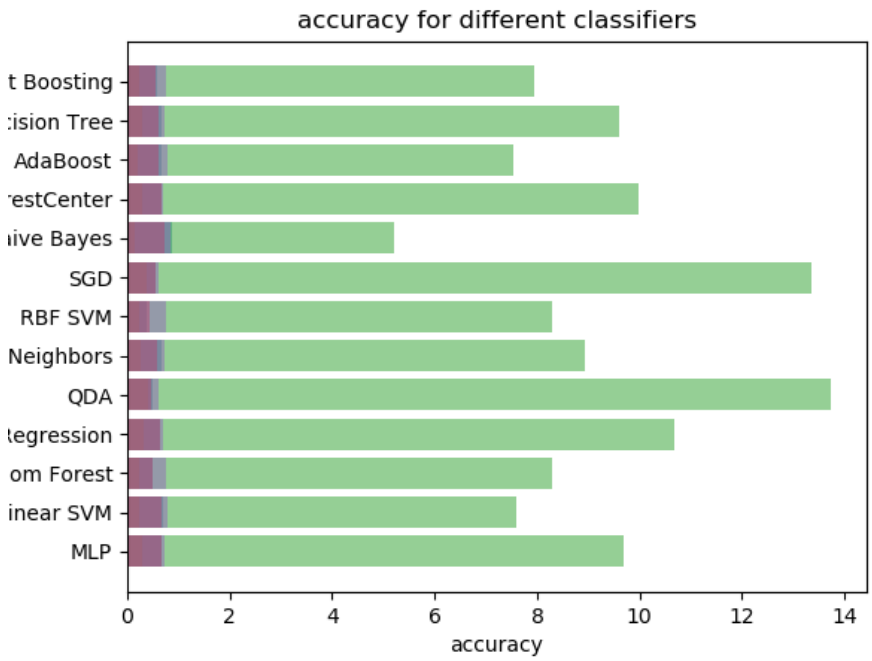
```
def optimizehyperparameters(self,bestmodels,train_data)
```

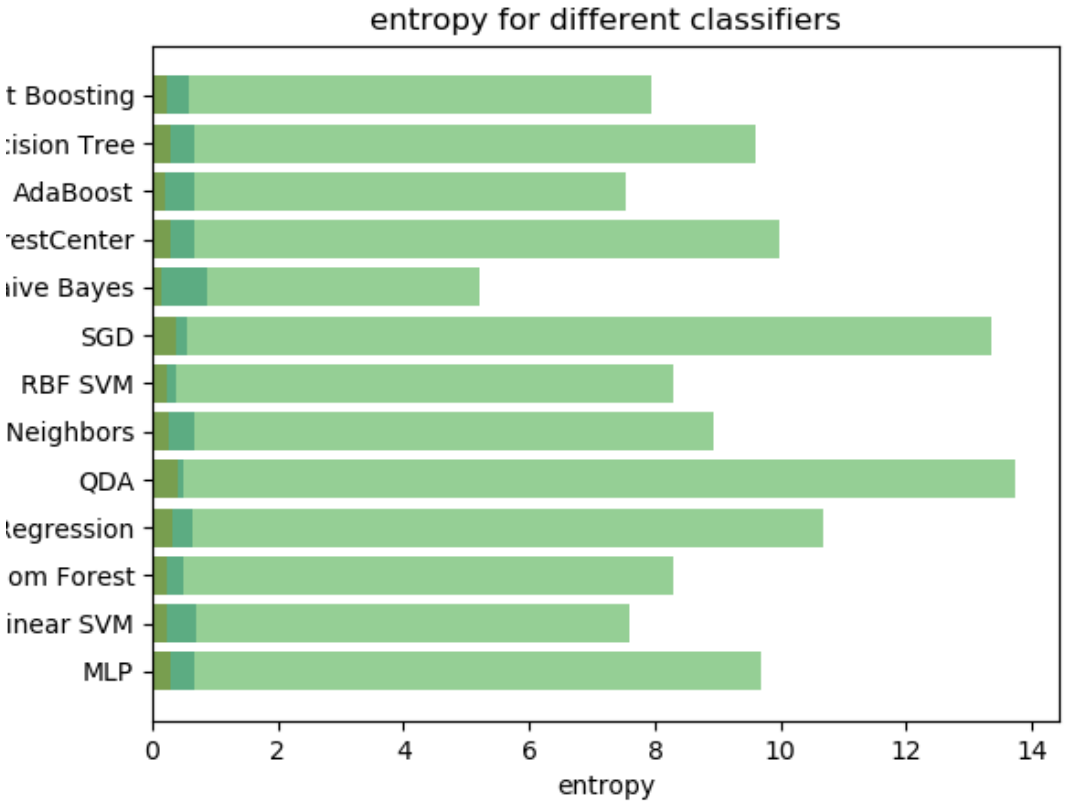
train the best two models and in the mean time it optimizes the networks parameter using brute force search by applying GridSearchCV function. In fact, This function tune the hyper parameters for the best two selected models (loss function for evaluation is cross entropy) and run the predictions. The results are saved in excel files. Since in this example stochastic gradient decent and decision tree are the best models (lowest log of error), therefore, the excele files are **SGD_result1.csv** and **Decision Tree_result1.csv**. The test data set has the linear interpolation for missing values.

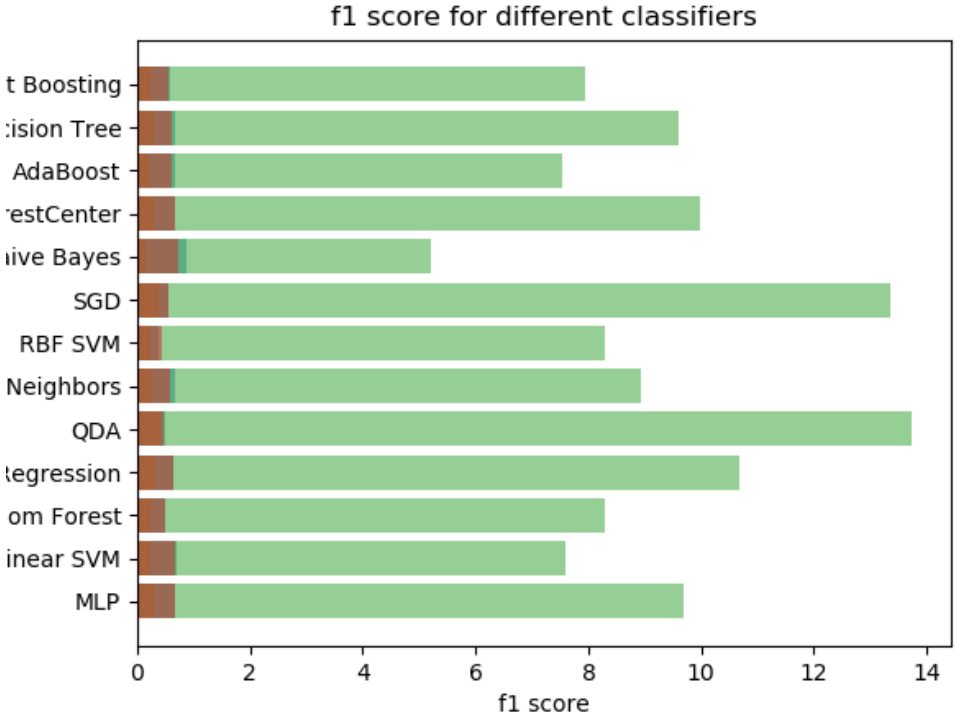
```
"""In this part we want to optimize the hyper parameters for models"""
model_parameters={'Nearest
Neighbors':[{'n_neighbors':np.linspace(1,10,10)}],
              'Linear
SVM':[{'kernel':['linear'],'C':np.linspace(0.1,1,10)}],
              'RBF SVM':[{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],'C':
[1, 10, 100, 1000]}],

              'Decision Tree':
[{'max_depth':np.linspace(1,10,10),'min_samples_split':np.linspace(2,8
,4).astype(int)}],
              'Random
Forest':[{'max_depth':np.linspace(1,10,10),'n_estimators':np.linspace(
10,100,10)}],

'MLP':[{'alpha':np.linspace(0.1,1,10),'max_iter':np.linspace(500,1000,
2)}],
              'AdaBoost':[{'n_estimators
':np.linspace(50,100,6),'learning_rate':[1e-3,1e-2,.5,1]}],
              'Naive Bayes' :[{'var_smoothing':[1e-09,1e-07,1e-05]}],
              'QDA':[{'reg_param':[0, 0.05,0.1,0.5,1]}],
              'SGD':[{'loss':['hinge', 'log', 'squared_hinge',
'perceptron'],
                      'penalty':['l2','l1','elasticnet']}],
```

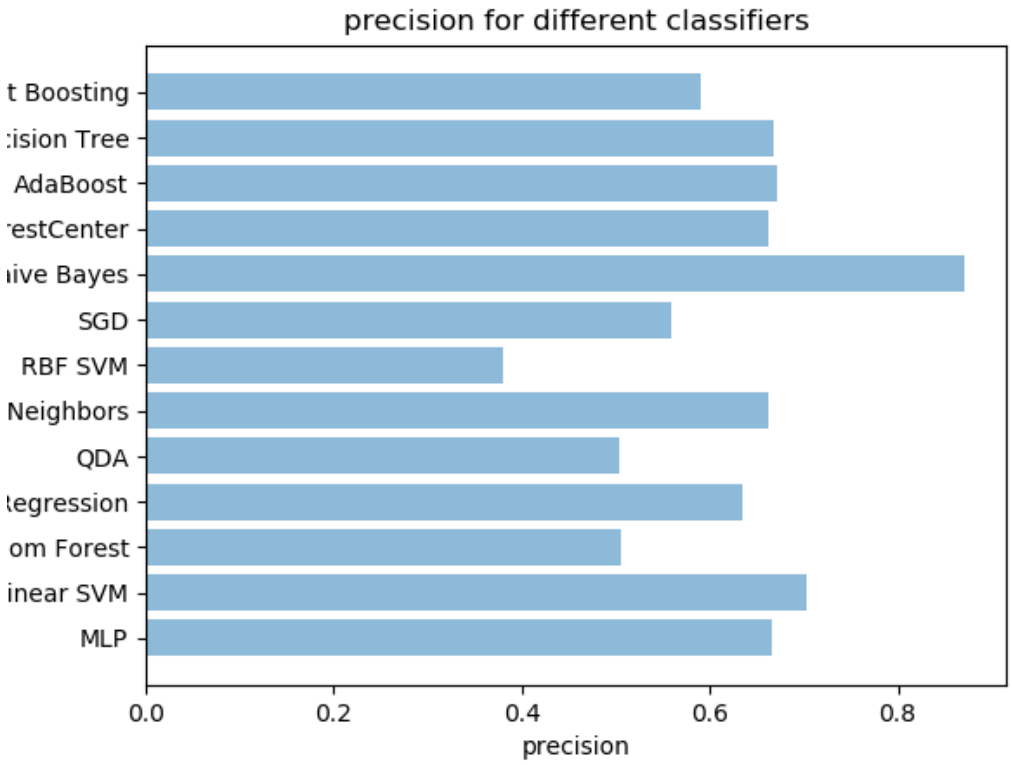
```
            'NearestCenter': [{'metric':'euclidean'}] ,
            'Gradient Boosting': [{'loss':['deviance',
'exponential'],'learning_rate':[0.01,0.05,0.1,0.5]}],
            'Logistic Regression':[{'solver':['newton-
cg','lbfgs','liblinear'],
                                    'penalty': ['l2', 'l1',
'elasticnet'],'C':[0.1,.5,1]}]}
X_train = train_data.iloc[0:100, 0:train_data.shape[1] - 1]
Y_label = train_data.iloc[0:100, train_data.shape[1]-1 ]

trained_model={}
test_data = self.proccess_test_data()
for model in bestmodels:
    trained_model[model]=GridSearchCV(self.models[model],
model_parameters[model], cv=self.k_fold,
                 scoring=make_scorer(log_loss)).fit(X_train,Y_label)

    test_data['Y_prediction']=
trained_model[model].predict(self.proccess_test_data())
    test_data.to_csv(model+'_result1'+'.csv')
    print(trained_model[model].best_params_)

return trained_model
```

In following the results for evaluation of models based on different scores are depicted.




entropy for different classifiers
t Boosting
ision Tree
AdaBoost
estCenter
ive Bayes
SGD
RBF SVM
Neighbors
QDA
egression
om Forest
inear SVM
MLP
0
2
4
6
8
10
12
14
entropy


f1 score for different classifiers
t Boosting
ision Tree
AdaBoost
estCenter
ive Bayes
SGD
RBF SVM
Neighbors
QDA
egression
om Forest
inear SVM
MLP
0
2
4
6
8
10
12
14
f1 score


precision for different classifiers
t Boosting
ision Tree
AdaBoost
estCenter
ive Bayes
SGD
RBF SVM
Neighbors
QDA
egression
om Forest
inear SVM
MLP
0.0
0.2
0.4
0.6
0.8
precision