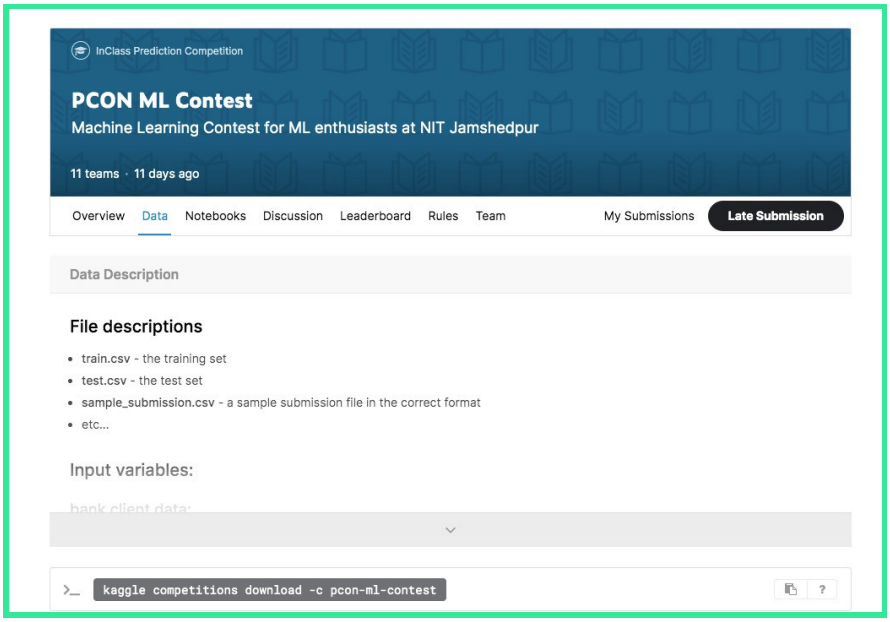
PCON ML CONTEST

Explaining Approach towards the given classification problem.



Here I am describing my work including all the steps performed during building this machine learning project/contest from scratch.

- **Importing** the given packages that installs the environment, libraries, csv files required for the codes.
- **Collecting data**- loading and exploring the data with the following commands:

```
saving training filepath to local variable
reading the training data and storing in DataFrame titled
train_data
(Repeating same with test datasets)
```

- **Interpreting Data Description**- further I visualised my data, preprocessed it, checked whether it contains missing values, null values, categorical variables or not through several pandas functionalities like count(), describe(), mean(), info() etc.

Moving on…

- **Selecting The Prediction Target**- I pulled target variable (single column) with **dot-notation**.
- By convention, the prediction target is called **y**. So I extracted y to save the response whether the client has subscribed a term deposit or not (binary: 'yes', 'no') in the dataset.

```
y = train_data.y
```

- **Choosing features-** The columns that are inputted into our model (and later will use to make predictions) . Here I chose all the columns excluding y. By convention, this data is called **X**.

```
featured_data = ['age', 'job', 'marital', 'education',
'default', 'housing', 'loan', 'contact', 'month',
'day_of_week', 'duration', 'campaign', 'pdays','previous',
'poutcome', 'emp.var.rate',
'cons.price.idx','cons.conf.idx', 'euribor3m',
'nr.employed', 'ID']


X=train_data[featured_data]
```

- During **data preprocessing**, I found categorical variables and so I added encoders to my code to convert the labels into numeric form so as to convert it into the machine-readable form.. So I used label encoder here imported from sklearn library and then the encoder trnasforms all the categorical features from object to numeric form.

```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
categorical_values = X.select_dtypes(include =
object).columns
for x in categorical_values:
    X[x] = label_encoder.fit_transform(X[x])
```

- As mentioned in the contest, **evaluation** has to be done on f1_score that's why I preferred f1_score to be my loss function and imported it from sklearn.metrics.
- Splitting dataset- For training model initially I splited the data into 3 three sections which are '**Training data**' ,'**Validation data**' and '**Testing data**'. Separating **data into training and testing sets** is an important part of evaluating **data** mining models. … By using similar **data** for **training and testing**, minimizes the effects of **data** discrepancies and better understand the characteristics of the model.
- After splitting 80/20 i.e.. 80% of data for training and 20% for validating, initializing the internal **random** number generator with one, which will decide the splitting of data into train and vaild indices while validating process over multiple runs of the code.

```
X_train, X_valid, y_train, y_valid = train_test_split(X, y,
train_size=0.8, random_state=1)
```

Now turning to the most important step i.e.. model selection. Model selection is crucial step and so I tried many variety of models eg.. Decision tree, Random forest, XgBoost, AdaBoost, CatBoost , ensemble methods without tuning and made submissions to test which goes for better performance.

- And finally i selected my model i.e.. CatBoost classifier model for my predictions. **CatBoost** is an **algorithm** based on gradient boosting on decision trees. Its specification is it can handle missing values internally.
- Tuning parameters- Now it was the turn to increase the accuracy, i.e.. need of tuning parameters and hyper parameters. **Parameters and hyperparameter** are key to machine learning algorithms . And their tuning yield a better model, better predictions.
- So, I tuned count of iterations, learning_rate for proper training, early_stopping_rounds to get control over overfitting and

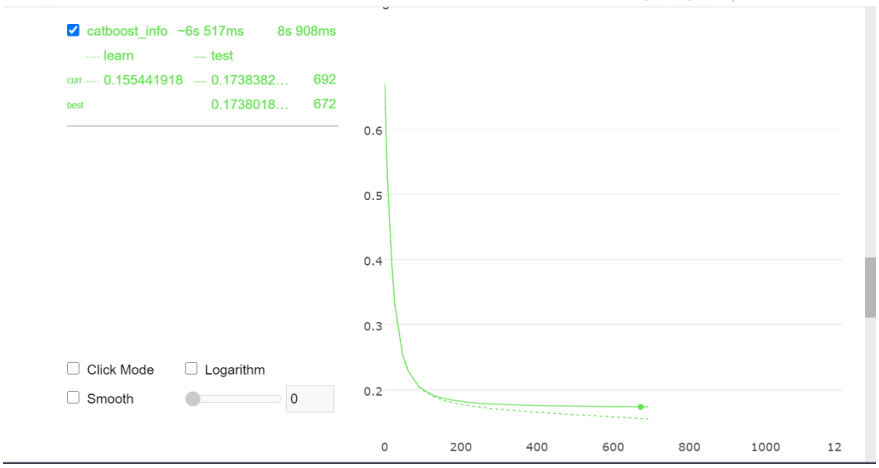
underfitting values. and random_seed for considering the same set again and again for iterations.

```
model = CatBoostClassifier(
    iterations=1200,
    learning_rate=0.015,
    random_seed=0,
    early_stopping_rounds=20
)
```

- **Training the model**- After building and setting those hypertuning parameters I fitted the model that involves providing an **ML** algorithm (that is, the learning algorithm) with training data.

```
model.fit(
    X_train, y_train,
    verbose=False,
    eval_set=(X_valid,y_valid),
    plot=True
)
```

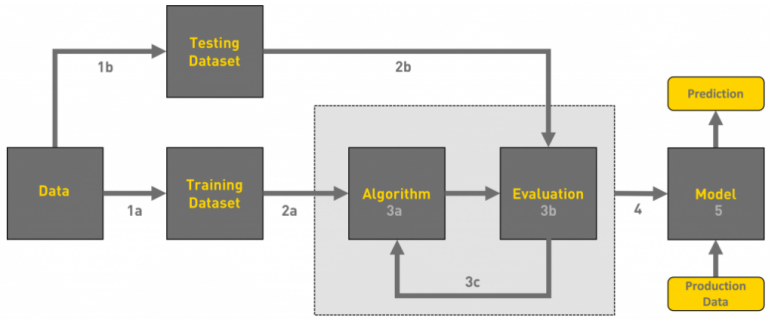
- Here i got this plot-



- Further I made predictions from my validation dataset and compared it with validation target variable (**y**) and calculate the loss function ***f1_score*** to check my accuracy level.

- **Final step**- And then summarizing I made my prediction on testdata set and stored in a variable. By analyzing the columnar structure of given submission file I replicated the same and finally made submission.

```
my_submission = pd.DataFrame({'Id':Id ,
'Predicted':prediction})
my_submission.to_csv('Submissn.csv', index=False)
```



And submitted the file to contest….