

## DATA PREPARATION

In order to get training data for model training, we first started with its preparation. For this, we read both the train.csv and feamat.csv using the pandas dataframe. While reading the train.csv, we made sure that we use both ; and , as the separators to get three columns corresponding to chemical Id, assay Id and target variable. Remaining features for training were present in the feamat.csv, so we matched the chemical id column in both these datasets, to get the final combined dataset which can be utilized for training. For merging, we used left join. After merging both train and feamat, we get a combined dataframe which has (77413, 1077) shape meaning that there are 77413 rows and 1077 columns.

```
In [10]: combined_df = train_df.merge(feamat_df, on = ['V1'], how = 'left')
         combined_df.head()
```

Out[10]:

	V1	Id	Expected	V2	V3	V4	V5	V6	V7	V8	...	V10
0	2971-36-0	1644	2	76302	315.982463	4.592	40.46	0.0	0.0	0.000000	...	0
1	693-54-9	2451	2	12741	156.151415	3.852	17.07	0.0	0.0	0.000000	...	0
2	7173-51-5	1384	2	23558	361.347528	9.912	0.00	0.0	0.0	0.000000	...	0
3	138261-41-3	16	2	86418	255.052302	2.294	83.66	0.0	0.0	0.117851	...	0
4	7681-82-5	1856	2	5238	149.894242	1.050	0.00	0.0	0.0	0.000000	...	0

```
In [11]: #check its shape
         combined_df.shape
```

Out[11]:

```
(77413, 1077)
```

Before this data is used for training, it is important to **pre-process** it. The following pre-processing steps have been implemented:

- Dealing INF values: There are infinity values present in the dataset which needs to be taken care of. We firstly tried finding already built made in functions like isnull() and fillna() for infinity values. But then later, we replaced the inf values with nan values which can be dealt with in the next step.

```
In [12]: #replacing inf with nan value
         combined_df = combined_df.replace([np.inf, -np.inf], np.nan)
```

- Dealing with nan values: Firstly, we counted the total number of nan values present in the dataset. Then later we filled these nan values with zero using fillna(0) function. In addition to this fillna function, we also used simpleImputer class from sklearn and filled the nan values with mean.

#### 2. Checking null and nan values

contains 1 null value & nan value

```
In [13]: combined_df.isnull().any().sum()
```

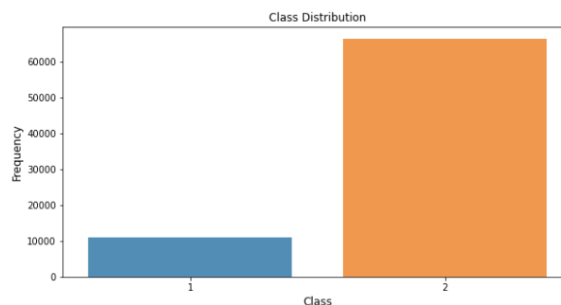
```
Out[13]:
```

1

#### 3. Fill nan value

```
In [14]: combined_df = combined_df.fillna(0)
```

- Class distributions were plotted: Total number of samples present where each class were plotted and visualized using the matplotlib library. It can be seen that class 2 has a greater number of samples than class 1, so it is a clear case of an unbalanced dataset. In order to solve this, we used SMOTE from imblearn and RandomUnderSample to make the dataset balanced using undersampling and undersampling



```
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline

# define pipeline
over = SMOTE()
under = RandomUnderSampler()
steps = [('o', over)], ('u', under)]
pipeline = Pipeline(steps=steps)

X_kbest, y = pipeline.fit_resample(X_kbest, y)
```

- Encoding Object data type: On checking the data types using `df.dtypes`, the first V1 column had the object data type which was encoded using `LabelEncoder()` from sklearn.
- X and y datasets were extracted from the `combined_df` where X had all the features required for training and y had the target variable.
- Internal Split: Using `train_test_split` available in sklearn library, training data was internally divided into training and test split with 80-20 split, that is 20 percent of the samples for the testing data and 80% of the samples for the training data.

```
In [21]:
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Normalization: Since all the features had different scales, we did normalization of features. We tried using both the `StandardScaler()` and `MinMaxScaler()` from the sklearn library. Scaler was fitted on the training data and used for transforming both the training and testing data.

```
In [22]:
from sklearn.preprocessing import MinMaxScaler, StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
scaler.transform(X_train)
scaler.transform(X_test)
```

## **MODEL TRAINING AND TESTING**

- Logistic Regression, Decision Trees and Random Forest Classifiers are some of the models using which we did the training of the data. From these models, the best score we have obtained on the unseen data is 0.75361 using the Decision Tree classifier with an internal evaluation of 0.876.

```
#testing accuracy
from sklearn.metrics import accuracy_score
y_pred = model.predict(X_test)
print(accuracy_score(y_test, y_pred))
```

```
0.8759930246076342
```

- We then calculate the feature importances using `model.feature_importances_` function.

- Using SelectFromModel from sklearn.feature\_selection, the best model is selected with a threshold value of 0.01. This threshold value helps us select the features whose importances are more than 0.01.
- After selecting the important features, the model is retrained, but this time only using the important features in contrast to all features. On evaluating this model, we got internal score of 0.873.

```
In [31]:
model = DecisionTreeClassifier()
model.fit(X_important_train, y_train)

print("Training Score {}".format(model.score(X_important_train, y_train)))
print("Testing Score {}".format(model.score(X_important_test, y_test)))
```

```
Training Score 1.0
Testing Score 0.8730220241555254
```

- Also, K-fold cross validation was performed. Here, we performed 5-fold cross validation and then calculated scores using cross\_val\_score from sklearn.model\_selection. Mean score of 0.87 was obtained.

```
#cross validation
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
cv = KFold(n_splits=5, random_state=1, shuffle=True)

model = DecisionTreeClassifier(max_depth=20)
model.fit(X_important_train, y_train)

scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)

# report performance
print('Accuracy: %.3f' % (np.mean(scores)))
```

```
Accuracy: 0.877
```

- In addition to the Decision Tree, we also used a Bagging classifier and XGboost Classifier. Furthermore, different feature selection methods were also used like using model's inbuilt feature importances, recursive feature elimination and using ANOVA. Parameter tuning was done using GridSearchCV
- cross val score using cv =5

```

from sklearn.metrics import f1_score
y_pred = search.predict(X_test)
f1_score(y_test, y_pred)





















0.9464449191333536

from sklearn.model_selection import cross_val_score
print(cross_val_score(best_model, X_train, y_train, cv=5))

```

LEADERBOARD SCORE

Below is the screenshot of the leaderboard:

#	Δpub	Team Name	Notebook	Team Members	Score ?	Entries	Last
1	▲2	NextGen		  	0.80775	43	4d
2	▲5	x2020fvt			0.80478	29	7d
3	▼2	Maple Squad		  	0.80294	74	4d
4	▲1	Orion		  	0.80243	52	4d
5	▲6	Bit and Byte		 	0.80201	68	3d
6	▼2	FALCONS		 	0.80153	42	4d
7	▲1	x2020fjw			0.79869	39	8d
8	▲1	Team Scorpion		  	0.79867	51	8d
9	▲4	AlphaX		 	0.79766	48	10d

EVALUATION METRICS

Classification model	Cross_val_score	F1_Score	Leaderboard score
XG boost Classifier with parameter tuning and feature selection using ANOVA	Using cv=5 [0.94440035 0.94205865 0.94590191 0.94469045 0.94502465]	0.949	0..80416
Bagging Tree Classifier (Ensemble with base as Decision Tree) with parameter tuning and feature selection using SelectFromModel	0.771	0.768	0.77416
Decision Tree Classifier with parameter tuning and feature selection using SelectFromModel	0.738	0.7475	0.75361