

Nama Akun Kaggle	:	Whisnumurty GA
Link Akun Kaggle	:	<a href="https://www.kaggle.com/whisnumurtyga">https://www.kaggle.com/whisnumurtyga</a>
Nama Competition	:	Steel Plate Defect Prediction
Deskripsi Challenge	:	Mengembangkan model prediktif yang dapat mengidentifikasi dan memprediksi keberadaan cacat pada pelat baja. Peserta diberikan kumpulan data yang berisi informasi tentang berbagai atribut fisik dan kimia dari pelat baja, serta label yang menunjukkan apakah pelat baja tersebut mengandung cacat atau tidak.

## Step 1 – Data Understanding

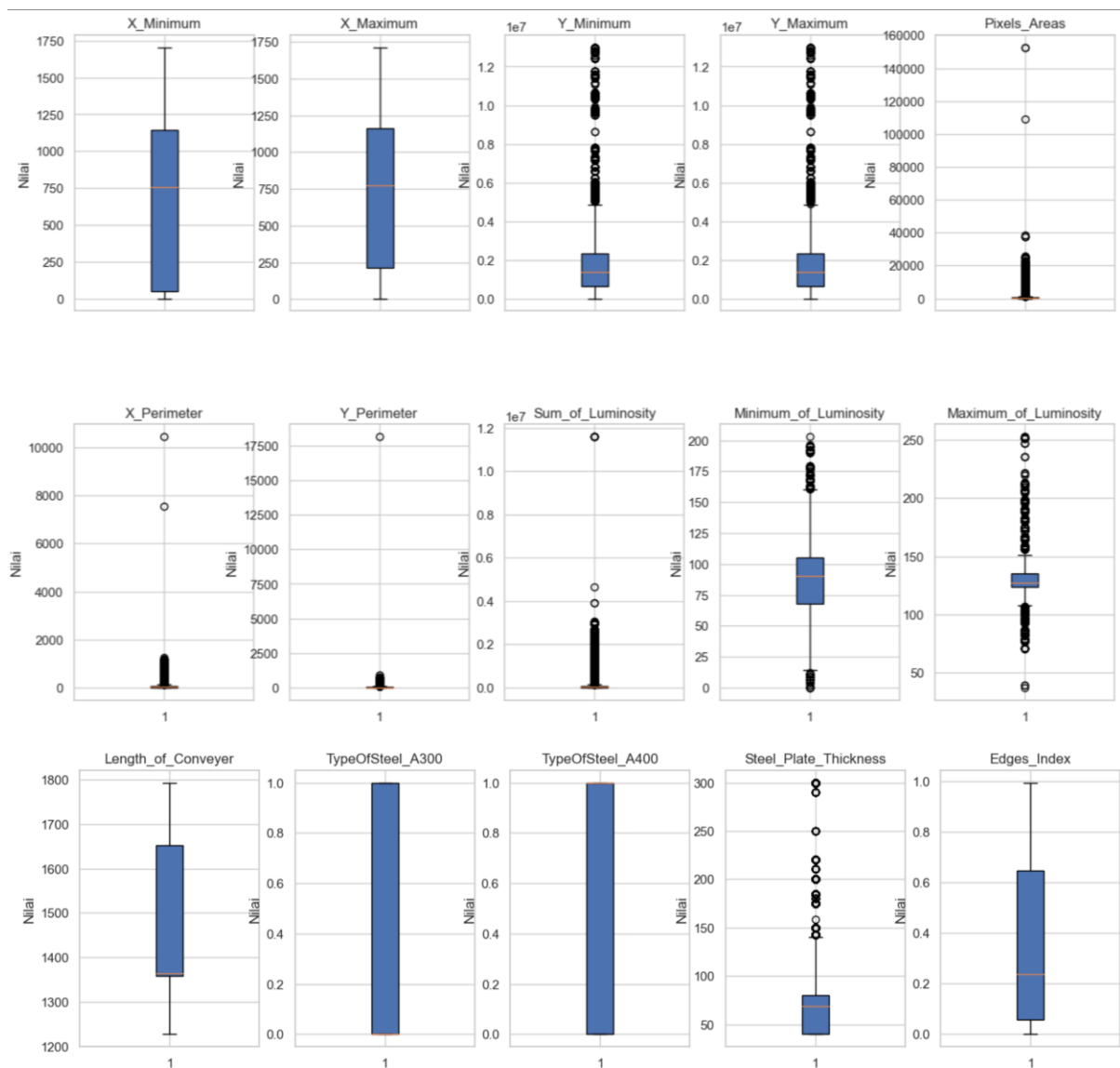
Nama Feature	Deskripsi
Id	ID unik untuk setiap sampel data.
X_Minimum, X_Maximum, Y_Minimum, Y_Maximum	Koordinat batas persegi panjang yang mengelilingi cacat pada pelat baja.
Pixel_Areas	Jumlah piksel yang tercakup oleh cacat tersebut.
X_Perimeter, Y_Perimeter	Panjang perimeter (keliling) dalam arah X dan Y dari cacat.
Sum_of_Luminosity, Minimum_of_Luminosity, Maximum_of_Luminosity	Informasi tentang luminositas cacat, termasuk jumlah total luminositas, luminositas minimum, dan maksimum.
Length_of_Conveyer	Panjang konveyor di mana pelat baja bergerak.
TypeOfSteel_A300, TypeOfSteel_A400	Variabel biner yang menunjukkan jenis baja (A300 atau A400).
Steel Plate Thickness	Ketebalan pelat baja.
Edges_Index, Empty_Index, Square_Index, Outside_X_Index, Edges_X_Index, Edges_Y_Index, Outside_Global_Index	Berbagai indeks yang mungkin dihitung dari citra pelat baja.
LogOfAreas, Log_X_Index, Log_Y_Index, Orientation_Index, Luminosity_Index, SigmoidOfAreas	Fitur-fitur lain yang dihitung dari citra pelat baja.
Pastry, Z_Scratch, K_Scratch, Stains, Dirtiness, Bumps, Other_Faults:	Label yang menunjukkan jenis cacat yang ada pada pelat baja. Misalnya, "Pastry" mungkin menunjukkan adanya cacat berbentuk pastry, dan seterusnya.

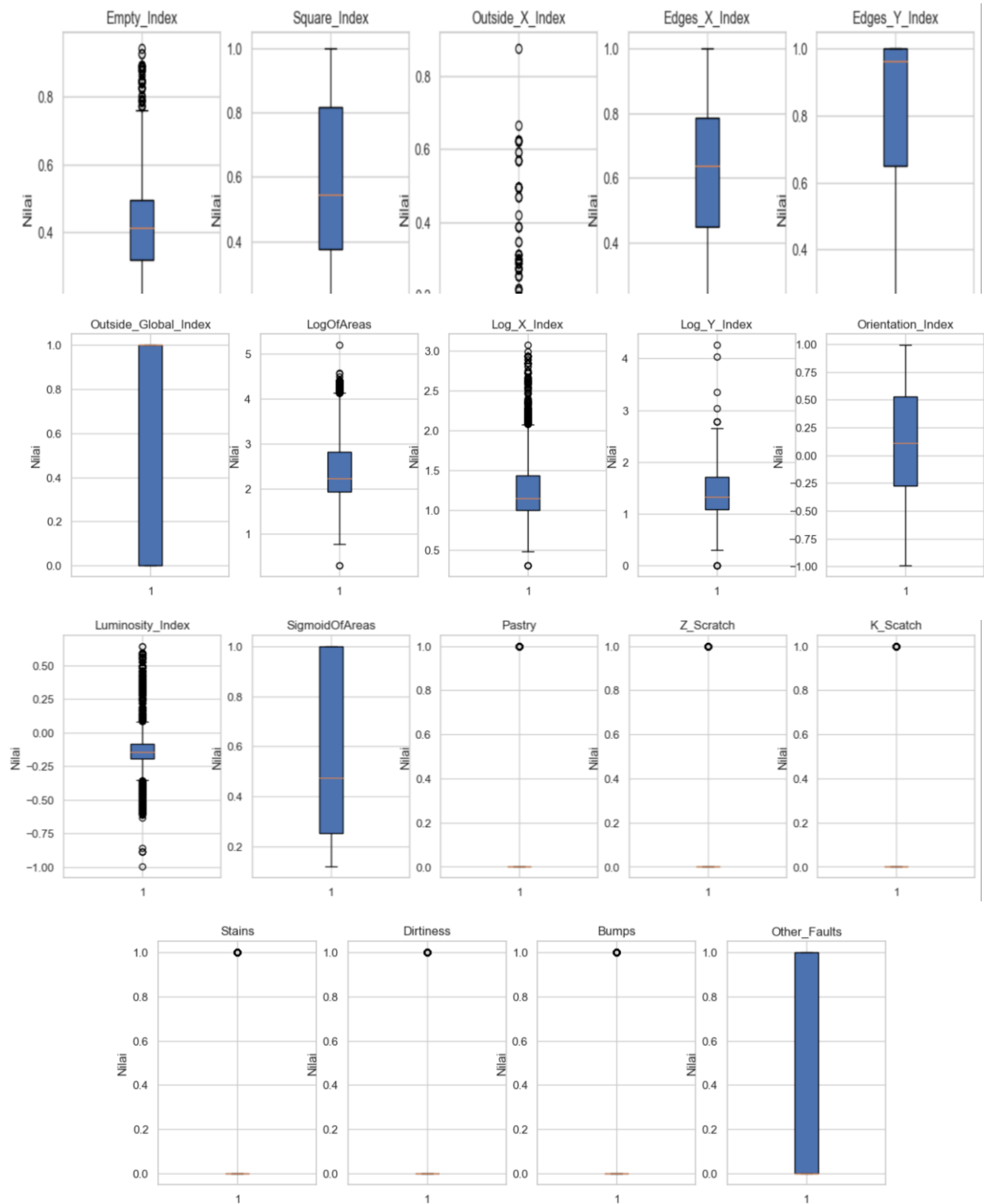
## Step 2 – Set Requirements

Library	Fungsi
NumPy	melakukan operasi numerik untuk bekerja dengan array dan matriks
Pandas	Manipulasi dan analisis data
Pretty Table	Membuat table rapi dan mudah dibaca
Matplotlib & Seaborn	Visualisasi Data
DeepCopy	Mencopy model
Partial	Untuk membuat fungsi baru dengan beberapa argumen dari fungsi yang ada.

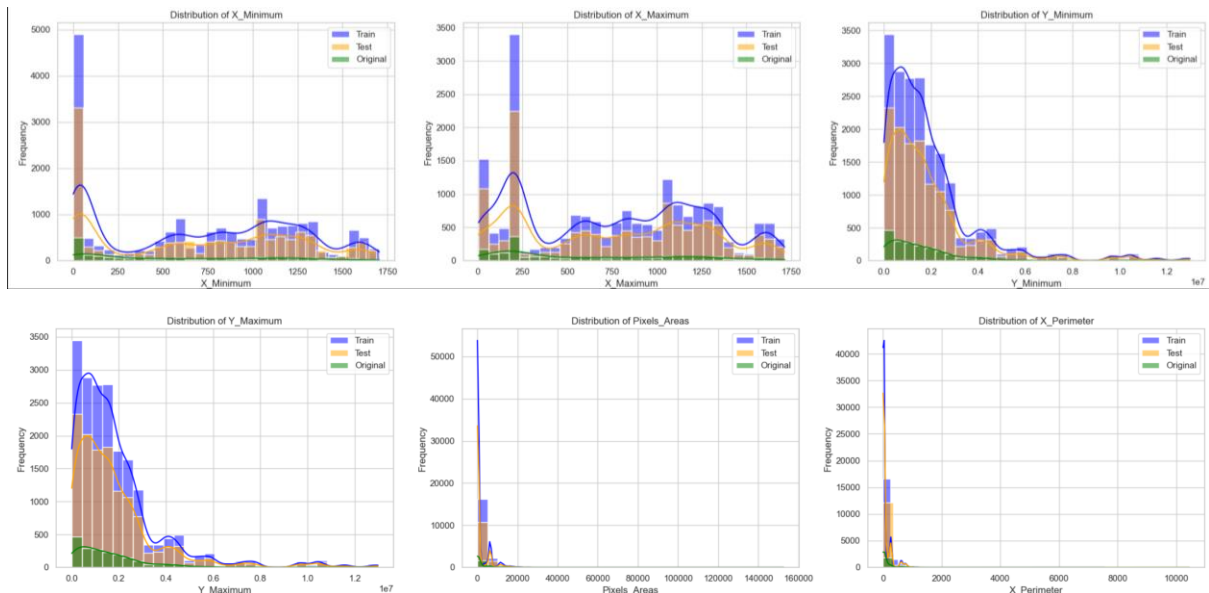
gc	Garbage Collector untuk memerikan control atas pemulihan memori
Scikit Learn	Untuk machine learning dan analisis data. Serta menyediakan berbagai algoritma machine learning, evaluasi model, dan alat-alat pra-pemrosesan data.
Optuna	Mengoptimasi parameter secara otomatis
XGBoost	Library untuk pemodelan prediktif decision tree based sebagai model dasar
CatBoost	Library untuk menangani data kategorikal

### Step 3 – Exploratory Data Analysis





Dari hasil boxplot tersebut dapat kita ketahui bahwa terdapat kolom kategorik yang sudah dijadikan bentuk numerik seperti (Pastry, Z\_Scratch, K\_Scratch, Strains, Dirtiness, Bumps, Other Faults, Outside\_Global Index), dari box plot tersebut kita juga dapat melihat beberapa feature terutama pada y\_minimum dan y\_maximum (artinya Panjang baja secara vertical sangat bervariasi)



Dari grafik histogram tersebut bis akita lihat bahwa distribusi data numerik memiliki pola kurva distribusi yang agak berbeda dari data original. Hal tersebut menandakan bahwa data sintesis yang dihasilkan dari data original memiliki pola yang berbeda.

## Step 4 - Feature Engineering

### 4.1 One Hot Encoding

```

1  def OHE(train_df,test_df,cols,target):
2      combined = pd.concat([train_df, test_df], axis=0)
3      for col in cols:
4          one_hot = pd.get_dummies(combined[col]).astype(int)
5          counts = combined[col].value_counts()
6          min_count_category = counts.idxmin()
7          one_hot = one_hot.drop(min_count_category, axis=1)
8          one_hot.columns=[str(f) + col for f in one_hot.columns]
9          combined = pd.concat([combined, one_hot], axis="columns")
10         combined = combined.loc[:, ~combined.columns.duplicated()]
11
12     # split back to train and test dataframes
13     train_ohe = combined[:len(train_df)]
14     test_ohe = combined[len(train_df):]
15     test_ohe.reset_index(inplace=True,drop=True)
16     test_ohe.drop(columns=[target],inplace=True)
17     return train_ohe, test_ohe

```

Melakukan proses one-hot encoding (OHE) pada kolom-kolom tertentu dari data latih (train\_df) dan data uji (test\_df), dengan menghapus kategori yang memiliki frekuensi terendah. Proses ini bertujuan untuk mengubah variabel kategorikal menjadi representasi biner yang dapat digunakan oleh model pembelajaran mesin untuk meningkatkan kinerja dalam melakukan prediksi.

### 4.2 Stabilize Categorical Columns

```

1 cat_cols = [f for f in test.columns if test[f].nunique() / test.shape[0] *
2 test[cat_cols].nunique()
3
4 def nearest_val(target):
5     return min(common, key=lambda x: abs(x - target))
6
7 global cat_cols_updated
8 cat_cols_updated = []
9 for col in cat_cols:
10     train[f"{col}_cat"] = train[col]
11     test[f"{col}_cat"] = test[col]
12     cat_cols_updated.append(f"{col}_cat")
13     uncommon = list((set(test[col].unique()) | set(train[col].unique())) -
14                     if uncommon:
15                         common = list(set(test[col].unique()) & set(train[col].unique()))
16                         train[f"{col}_cat"] = train[col].apply(nearest_val)
17                         test[f"{col}_cat"] = test[col].apply(nearest_val)

```

Mengelompokkan variabel kategorikal yang memiliki jumlah kategori yang sedikit dan jarang muncul dalam data menjadi kategori baru yang lebih umum, berdasarkan nilai terdekat dari kategori yang umum di setiap data latih dan data uji. Hal ini dilakukan dengan mengecek variabel kategorikal dalam data uji, kemudian mencari nilai unik yang kurang dari 5% dari total baris dan lebih dari 2 nilai unik. Setelah itu, dilakukan proses pembuatan kategori baru dengan memilih nilai terdekat dari nilai-nilai umum yang terdapat dalam data latih dan data uji untuk setiap variabel kategorikal yang diproses. Variabel-variabel kategorikal yang telah diperbarui kemudian disimpan dalam list `cat_cols_updated` untuk penggunaan selanjutnya.

### 4.3 High Freq OHE

```

1 def high_freq_ohe(train, test, extra_cols, target, n_limit=50):
2     ...
3     If you wish to apply one hot encoding on a feature with so many unique values, then this can be
4     where it takes a maximum of n categories and drops the rest of them treating as rare categories
5     ...
6     train_copy = train.copy()
7     test_copy = test.copy()
8     ohe_cols = []
9     for col in extra_cols:
10         dict1 = train_copy[col].value_counts().to_dict()
11         ordered = dict(sorted(dict1.items(), key=lambda x: x[1], reverse=True))
12         rare_keys = list(*ordered.keys())[n_limit:]
13         # ext_keys=[f[0] for f in ordered.items() if f[1]<50]
14         rare_key_map = dict(zip(rare_keys, np.full(len(rare_keys), 9999)))
15
16         train_copy[col] = train_copy[col].replace(rare_key_map)
17         test_copy[col] = test_copy[col].replace(rare_key_map)
18
19     train_copy, test_copy = OHE(train_copy, test_copy, extra_cols, target)
20     drop_cols = [f for f in train_copy.columns if "9999" in f or train_copy[f].nunique() == 1]
21     train_copy = train_copy.drop(columns=drop_cols)
22     test_copy = test_copy.drop(columns=drop_cols)
23
24     return train_copy, test_copy
25

```

Menerapkan one-hot encoding pada fitur dengan jumlah nilai unik yang banyak dengan batasan tertentu. Ketika fitur memiliki jumlah kategori yang melebihi batas yang ditentukan (`n_limit`), fitur akan dianggap memiliki kategori langka. Fitur-fitur dengan kategori langka ini akan diubah menjadi nilai yang disesuaikan (misalnya, 9999) dalam kedua dataset train dan test. Setelah itu, fungsi OHE dipanggil untuk melakukan one-hot encoding pada fitur-fitur yang telah dimodifikasi. Kemudian, kolom-kolom yang memiliki nilai 9999 atau memiliki satu nilai unik saja dihapus dari kedua dataset untuk mengurangi dimensi data yang dihasilkan. Setelah itu, dataset train dan test yang telah dimodifikasi kembali dikembalikan sebagai output dari fungsi ini.

#### **4.4 Category Encoding**

Melakukan encoding pada variabel kategorikal dalam dataset train dan test. Pertama-tama, fungsi ini membuat salinan dari dataset train dan test. Kemudian, untuk setiap variabel kategorikal yang telah diperbarui (`cat_cols_updated`), fungsi ini menghitung frekuensi masing-masing nilai, membuat kolom baru yang berisi jumlah kemunculan setiap nilai kategorikal (`feature + "_count"`) serta memberikan peringkat pada setiap nilai kategorikal berdasarkan jumlah kemunculannya (`feature + "_count_label"`). Jika jumlah nilai unik dari variabel kategorikal kurang dari atau sama dengan 5, variabel tersebut akan dianggap kategorikal dan akan diterapkan one-hot encoding. Jika tidak, variabel tersebut akan diubah menggunakan fungsi `high_freq_ohe`. Selanjutnya, model Machine Learning akan dilatih menggunakan fitur-fitur yang telah dihasilkan untuk memprediksi target, dan diukur kinerjanya menggunakan AUC (Area Under the ROC Curve). Fitur dengan kinerja terbaik akan dipilih dan kolom-kolom yang berkorelasi kuat dengan fitur terbaik akan dihapus untuk menghindari multikolinearitas. Informasi mengenai fitur terbaik beserta kinerjanya akan ditambahkan ke dalam sebuah tabel yang ditampilkan, dan dataset train dan test yang telah dimodifikasi akan dikembalikan sebagai output dari fungsi ini.

### **Step 5 – Modelling**

#### **5.1 Create Model**

```

1 class Splitter:
2     def __init__(self, test_size=0.2, kfold=True, n_splits=5):
3         self.test_size = test_size
4         self.kfold = kfold
5         self.n_splits = n_splits
6
7     def split_data(self, X, y, random_state_list):
8         if self.kfold:
9             for random_state in random_state_list:
10                 kf = KFold(n_splits=self.n_splits, random_state=random_state, shuffle=True)
11                 for train_index, val_index in kf.split(X, y):
12                     X_train, X_val = X.iloc[train_index], X.iloc[val_index]
13                     y_train, y_val = y.iloc[train_index], y.iloc[val_index]
14                     yield X_train, X_val, y_train, y_val
15
16 class Classifier:
17     def __init__(self, n_estimators=100, device="cpu", random_state=0):
18         self.n_estimators = n_estimators
19         self.device = device
20         self.random_state = random_state
21         self.models = self._define_model()
22         self.len_models = len(self.models)
23
24     def _define_model(self):
25         xgb_params = {
26             'n_estimators': self.n_estimators,
27             'learning_rate': 0.1,
28             'max_depth': 4,
29             'subsample': 0.8,
30             'colsample_bytree': 0.1,
31             'n_jobs': -1,
32             'eval_metric': 'logloss',
33             'objective': 'binary:logistic',
34             'tree_method': 'hist',
35             'verbosity': 0,
36             'random_state': self.random_state,
37

```

Kelas `Splitter` bertujuan untuk memisahkan data menjadi set pelatihan dan validasi menggunakan teknik pengacakan K-Fold Cross Validation. Pengguna dapat menentukan ukuran data uji (`test\_size`) dan jumlah lipatan (`n\_splits`) untuk validasi silang. Metode `split\_data()` digunakan untuk membagi data menjadi set pelatihan dan validasi.

Kelas `Classifier` bertujuan untuk mendefinisikan beberapa model klasifikasi, yaitu XGBoost (`xgb`) dan CatBoost (`cat` dan `cat\_sym`). Pengguna dapat menentukan jumlah estimator (`n\_estimators`) dan perangkat (`device`) yang ingin digunakan. Pada bagian ini, parameter-parameter untuk setiap model didefinisikan, seperti jumlah estimator, kedalaman maksimum, tingkat pembelajaran, dan lain-lain. Model-model ini disimpan dalam atribut `models` dari objek `Classifier`.

## 5.2 Optimize Ensemble Weight

```

1 class OptunaWeights:
2     def __init__(self, random_state, n_trials=5000):
3         self.study = None
4         self.weights = None
5         self.random_state = random_state
6         self.n_trials = n_trials
7
8     def _objective(self, trial, y_true, y_preds):
9         weights = [trial.suggest_float(f"weight{n}", 0, 1) for n in range(len(y_preds))]
10        weighted_pred = np.average(np.array(y_preds).T, axis=1, weights=weights)
11
12        auc_score = roc_auc_score(y_true, weighted_pred)
13        log_loss_score = log_loss(y_true, weighted_pred)
14        return auc_score
15
16    def fit(self, y_true, y_preds):
17        optuna.logging.set_verbosity(optuna.logging.ERROR)
18        sampler = optuna.samplers.CmaEsSampler(seed=self.random_state)
19        pruner = optuna.pruners.HyperbandPruner()
20        self.study = optuna.create_study(sampler=sampler, pruner=pruner, study_name="OptunaWeights")
21        objective_partial = partial(self._objective, y_true=y_true, y_preds=y_preds)
22        self.study.optimize(objective_partial, n_trials=self.n_trials)
23        self.weights = [self.study.best_params[f"weight{n}"] for n in range(len(y_preds))]
24
25    def predict(self, y_preds):
26        assert self.weights is not None, 'OptunaWeights error, must be fitted before predict'
27        weighted_pred = np.average(np.array(y_preds).T, axis=1, weights=self.weights)
28        return weighted_pred
29
30    def fit_predict(self, y_true, y_preds):
31        self.fit(y_true, y_preds)
32        return self.predict(y_preds)
33
34    def weights(self):
35        return self.weights

```

Kelas `OptunaWeights` bertujuan untuk mencari bobot terbaik yang diberikan kepada prediksi dari beberapa model. Ini berguna ketika Anda memiliki beberapa model yang menghasilkan prediksi untuk tugas klasifikasi atau regresi, dan Anda ingin menggabungkan prediksi-prediksi ini menjadi satu prediksi tunggal dengan bobot yang dioptimalkan. Kelas ini menggunakan `Optuna`, sebuah library untuk optimasi hiperparameter, untuk mencari bobot terbaik yang memberikan hasil optimal berdasarkan metrik evaluasi yang dipilih (dalam hal ini, Area Under the ROC Curve atau AUC).

- Metode `fit()` digunakan untuk melakukan pencarian bobot terbaik berdasarkan prediksi dari beberapa model.
- Metode `predict()` digunakan untuk menghasilkan prediksi tunggal menggunakan bobot terbaik yang telah ditemukan.
- Metode `fit_predict()` adalah gabungan dari `fit()` dan `predict()`, yang digunakan untuk melakukan pencarian bobot terbaik dan menghasilkan prediksi tunggal dalam satu langkah.
- Metode `weights()` digunakan untuk mengembalikan bobot yang telah ditemukan setelah proses pelatihan.

### 5.3 Fit the Model



```

1  def fit_model(X_train,X_test,y_train):
2      kfold = True
3      n_splits = 1 if not kfold else 5
4      random_state = 2023
5      random_state_list = [42] # used by split_data [71]
6      n_estimators = 9999 # 9999
7      early_stopping_rounds = 300
8      verbose = False
9
10     splitter = Splitter(kfold=kfold, n_splits=n_splits)
11
12     # Initialize an array for storing test predictions
13     test_preds = np.zeros(X_test.shape[0])
14     y_train_pred = y_train.copy()
15
16     ensemble_score = []
17     weights = []
18     trained_models = {'xgb':[], 'lgb':[]}
19
20
21     for i, (X_train_, X_val, y_train_, y_val) in enumerate(splitter.split_data(X_train, y_train, n
22         n = i % n_splits
23         m = i // n_splits
24
25         # Get a set of Regressor models
26         classifier = Classifier(n_estimators, device, random_state)
27         models = classifier.models
28
29         # Initialize lists to store oof and test predictions for each base model
30         oof_preds = []
31         test_preds = []
32
33         # Loop over each base model and fit it to the training data, evaluate on validation data, e
34         for name, model in models.items():
35             if ('cat' in name) or ("lgb" in name) or ("xgb" in name):
36                 if 'lgb' in name: #categorical_feature=cat_features
37                     model.fit(X_train_, y_train_, eval_set=[(X_val, y_val)],#,categorical_feature=

```

Fungsi `fit_model` bertujuan untuk melatih model ensemble menggunakan beberapa model dasar seperti XGBoost dan CatBoost, serta mengevaluasi kinerjanya menggunakan metrik ROC AUC. Berikut adalah penjelasan singkat tentang bagaimana fungsi ini bekerja:

- Fungsi ini membagi data latih menjadi set pelatihan dan validasi menggunakan teknik K-Fold Cross Validation jika `kfold` diatur sebagai `True`, dan menggunakan satu lipatan jika tidak. Ini dilakukan menggunakan objek `Splitter`.
- Untuk setiap lipatan, fungsi akan melatih setiap model dasar (XGBoost, CatBoost, dll.) menggunakan data pelatihan dan mengevaluasinya menggunakan data validasi.
- Setelah model-model dasar dilatih, fungsi akan menggunakan Optuna untuk menemukan bobot terbaik yang akan diberikan kepada prediksi dari model-model tersebut, sehingga menghasilkan prediksi ensemble yang optimal.
- Fungsi akan menggabungkan prediksi dari model-model dasar dengan bobot yang dioptimalkan untuk menghasilkan prediksi ensemble akhir.
- Kinerja ensemble diukur menggunakan metrik ROC AUC dan dicetak.
- Bobot rata-rata dan standar deviasi bobot dari setiap model dasar dicetak.
- Prediksi ensemble yang dihasilkan dikembalikan sebagai output dari fungsi ini.

## Step 6 – Submission

```

1  count=0
2  for col in target:
3      train_temp = train[test.columns.tolist() + [col]]
4      test_temp = test.copy()
5      train_temp, test_temp = cat_encoding(train_temp, test_temp, col)
6
7      final_features = test.columns.tolist()
8      sc = StandardScaler()
9
10     train_scaled = train_temp.copy()
11     test_scaled = test_temp.copy()
12
13     train_scaled[final_features] = sc.fit_transform(train[final_features])
14     test_scaled[final_features] = sc.transform(test[final_features])
15
16     # train_cop, test_cop= post_processor(train_scaled, test_scaled)
17     train_cop, test_cop = train_scaled, test_scaled
18     X_train = train_cop.drop(columns=[col])
19     y_train = train_cop[col]
20
21     X_test = test_cop.copy()
22
23     test_predss = fit_model(X_train,X_test,y_train)
24     submission[col] = test_predss
25
26     count+=1
27     print(f'Column {col}, loop # {count}')

```

Dalam percobaan ini, dilakukan iterasi untuk setiap kolom target dalam dataset. Setiap iterasi dimulai dengan memodifikasi dataset latih dan uji menggunakan fungsi `cat_encoding`, yang mengkodekan variabel kategorikal, dan menghasilkan set fitur final yang akan digunakan untuk pelatihan model. Selanjutnya, fitur-fitur ini dinormalisasi menggunakan `StandardScaler` untuk memastikan distribusi yang seragam, penting untuk beberapa jenis model seperti SVM. Proses pelatihan model kemudian dilakukan menggunakan fungsi `fit_model`, yang melibatkan pemilihan model ensemble dan pencarian bobot terbaik untuk prediksi yang dioptimalkan. Prediksi akhir dihasilkan untuk setiap kolom target dalam dataset uji, dan hasilnya disimpan dalam dataframe `submission`. Proses ini diulang untuk setiap kolom target, dan setiap langkah dalam iterasi diberi nomor untuk memantau kemajuan. Ini menyajikan suatu pendekatan sistematis dalam menghasilkan prediksi yang akurat untuk setiap kolom target dalam dataset.



## Step 7 – Generalization Ensemble

Kode ini digunakan untuk menggabungkan hasil dari beberapa file `submission` dengan memberikan bobot tertentu pada masing-masingnya. Pertama, dua file `submission` (`sub1` dan `submission`) dibaca. Selanjutnya, bobot diberikan pada setiap file `submission` dan file-file `submission` ini digabungkan menjadi satu dalam sebuah list. Fungsi `ensemble_mean` kemudian digunakan untuk menghitung ensemble dari file `submission` dalam list tersebut, dengan mempertimbangkan tiga jenis rata-rata: Arithmetic Mean (AM), Geometric Mean (GM), dan Harmonic Mean (HM). Dalam kasus ini, digunakan AM. Hasil ensemble ini kemudian disimpan dalam sebuah file CSV baru dengan nama `submission-ensemble.csv` untuk digunakan lebih lanjut. Ini memberikan cara efisien untuk menggabungkan hasil prediksi dari berbagai model atau strategi yang telah dikembangkan.

# Result

Akurasi Public : 0.89598

Akurasi Private : 0.88877

Submission and Description		Private Score ⓘ	Public Score ⓘ	Selected
	<b>submission-ensemble.csv</b> Complete (after deadline) · 32m ago	<b>0.88877</b>	<b>0.89598</b>	<input type="checkbox"/>
	<b>submission_pure.csv</b> Complete (after deadline) · 2h ago	<b>0.88579</b>	<b>0.89376</b>	<input type="checkbox"/>