

Eksperimen :

Kita memiliki data train dan test yang sudah terpisah dari Kaggle

Pertama kita memiliki data yang sudah terpisah, dengan metode pembagian seperti gambar dibawah ini

```
[ ] x = merge_df.drop(columns = ['y'])  
    y = merge_df['y']
```

```
[ ] y_train = dftrain['y']
```

```
[ ] y_test = dftest['y']
```

```
[ ] x_train = dftrain.drop('y', axis = 1)
```

```
[ ] x_test = dftest.drop('y', axis = 1)
```

Kita melakukan pengujian menggunakan algoritma :

- Decision Tree
- Random Forest
- Logistic Regression
- Support Vector Machine

Setelah dilakukan pengujian kita mendapatkan hasil seperti dibawah ini

Angka dibawah ini adalah angka dari metrics precision

```
array([0.92820513, 0.10037999, 0.07530454, 0.09356081, 0.11885057])
```

Terlihat pada iterasi pertama score terlihat baik, tetapi pada iterasi selanjutnya score menurun secara drastic

Hal ini kemungkinan bermasalah dengan cara pembagian dataset nya. Maka dari itu kita akan menggunakan `train_test_split` untuk data train kita.


Dan untuk kedepan nya data testing yang berasal dari Kaggle akan kita gunakan untuk prediksi pada tahap paling akhir setelah model sudah optimal

Models :

```
models = [  
    ('Logistic Regression', LogisticRegression(class_weight='balanced', random_state=42)),  
    ('Random Forest', RandomForestClassifier(class_weight='balanced', random_state=42)),  
    ('Decision Tree', DecisionTreeClassifier(class_weight='balanced', random_state=42)),  
    ('SVM', SVC(class_weight='balanced', random_state=42))  
]
```

Kita menggunakan algoritma logistic regression, random forest, decision tree, dan svm. Mengapa kita memutuskan untuk memilih algoritma tersebut?

- Kita memiliki label yang berupa kategorikal (yes/no) dengan tipe data imbalance, maka dari itu kita perlu menggunakan algoritma klasifikasi



	total	percentage
0	33316	88.205237
1	4455	11.794763

- Algoritma tersebut termasuk kedalam algoritma klasifikasi
- Algoritma tersebut yang dapat diberikan parameter `class_weight = balanced`, dan `random state`
- Kita menggunakan parameter `class_weight = balanced` sebagai metode untuk mengatasi data imbalance yang kita punya

Metrics :

- Metrics yang akan kita gunakan yaitu precision, hal ini dikarenakan kita ingin mengurangi False Positive

Penjelasan :

Mengapa False positive yang ingin di kurangi? Karena kita ingin meningkatkan efisiensi marketing

Jika false positive tinggi ini berarti banyak orang yang di prediksi akan deposit tetapi kenyataan nya tidak, hal ini akan berdampak kepada tingginya biaya campaign yang dilakukan dikarenakan banyak dari yang di prediksi ternyata tidak akan melakukan deposit

- Hyper parameter

Hyper parameter yang digunakan adalah `randomizedsearchcv`, hal ini kita lakukan dikarenakan kita tidak memiliki waktu yang cukup untuk melakukan `gridsearchcv`

Pengujian final :

Kita melakukan `train_test_split`

```
[ ] #split data
    from sklearn.model_selection import train_test_split

    X = dftrain.drop(columns="y")
    y = dftrain["y"]

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)
    X_train.shape, X_test.shape, y_train.shape, y_test.shape

((27649, 43), (6913, 43), (27649,), (6913,))
```

Kita membuat function def agar dapat langsung menguji semua algoritma yang kita tentukan

```
models = [
    ('Logistic Regression', LogisticRegression(class_weight='balanced', random_state=42)),
    ('Random Forest', RandomForestClassifier(class_weight='balanced', random_state=42)),
    ('Decision Tree', DecisionTreeClassifier(class_weight='balanced', random_state=42)),
    ('SVM', SVC(class_weight='balanced', random_state=42))
]

def train_eval_cv_model(models, X_train, y_train):
    results_cv = []
    for name, model in models:
        model.fit(X_train, y_train)
        # Cross validation (tidak pakai data test)
        stratified_kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
        score = cross_validate(model, X_train, y_train, cv=stratified_kfold, scoring=['accuracy', 'precision', 'recall', 'f1', 'roc_auc'], return_train_score=True)

        result_cv = {
            'Model': name,
            'Accuracy_CV_train': score['train_accuracy'].mean(),
            'Accuracy_CV_test': score['test_accuracy'].mean(),
            'Accuracy_CV_std_test': score['test_accuracy'].std(),
            'Precision_CV_train': score['train_precision'].mean(),
            'Precision_CV_test': score['test_precision'].mean(),
            'Precision_CV_std_test': score['test_precision'].std(),
            'Recall_CV_train': score['train_recall'].mean(),
            'Recall_CV_test': score['test_recall'].mean(),
            'Recall_CV_std_test': score['test_recall'].std(),
            'F1_CV_train': score['train_f1'].mean(),
            'F1_CV_test': score['test_f1'].mean(),
            'F1_CV_std_test': score['test_f1'].std(),
            'Roc_Auc_CV_train': score['train_roc_auc'].mean(),
            'Roc_Auc_CV_test': score['test_roc_auc'].mean(),
            'Roc_Auc_CV_std_test': score['test_roc_auc'].std()
        }

        results_cv.append(result_cv)

    eval_cv = pd.DataFrame(results_cv)
    return eval_cv
```

Setelah kita memperbaiki syntax yang kita punya kita mendapatkan hasil seperti dibawah ini :

```
n_iter_i = _check_optimize_result(
    Model Accuracy_CV_train Accuracy_CV_test \
0 Logistic Regression 0.837010 0.836052
1 Random Forest 0.999991 0.927954
2 Decision Tree 1.000000 0.908315
3 SVM 0.863349 0.852942

Accuracy_CV_std_test Precision_CV_train Precision_CV_test \
0 0.003064 0.329967 0.327878
1 0.001035 1.000000 0.714233
2 0.001603 1.000000 0.465092
3 0.002348 0.381579 0.357654
```

Kita dapat melihat metrics precision yang paling baik dari seluruhnya adalah random forest dengan nilai :

- Precision_train = 1
- Precision_test = 0.71

Dari hasil tersebut terlihat model ini terindikasi overfit

maka kita akan menggunakan algoritma itu untuk dilanjutkan ke tahap selanjutnya yaitu hyper parameter tuning

link github :

https://github.com/jundanaa/PROJECT_RAKAMIN