Machine Learning and Data Mining
Project Work

Alessandro Tutone

Alma Mater Studiorum, Università degli Studi di Bologna

2025

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Table of Contents

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Task Description

Objective: Identify which customers *will make a specific transaction in the future*, regardless of the amount of money transacted.

Data structure: Same structure used for the *real data*

Type of classification: *Binary*

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Dataset Description

Anonymized dataset containing *numeric feature* variables, the *binary target* column, and a *string ID_code* column.

The task is to predict the value of **target column** in the test set.

File descriptions:

- **train.csv** - the training set.
- **test.csv** - the test set.
- **sample_submission.csv** - a sample submission file.

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Training dataset

Lets take a look inside the file **train.csv**

In [4]:
```
train_df.head()
```

Out[4]:

	ID_code	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	...
0	train_0	0	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.6266	...
1	train_1	0	11.5006	-4.1473	13.8588	5.3890	12.3622	7.0433	5.6208	16.5338	...
2	train_2	0	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155	...
3	train_3	0	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.9250	...
4	train_4	0	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514	...

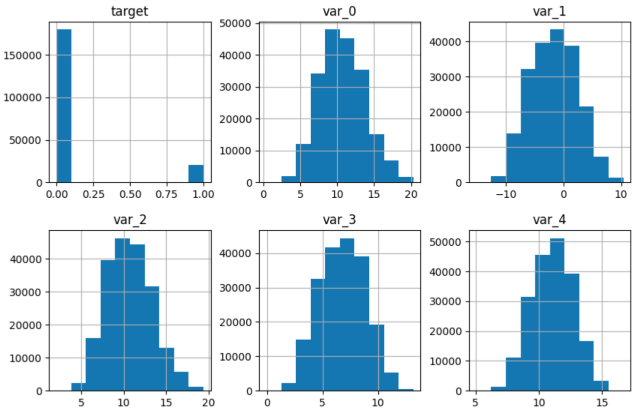
5 rows × 202 columns

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Data distribution

Considering the **distributions**, we can learn something new about the data.

In [6]:
```
train_df[train_df.columns[0:10]].hist(figsize=(10,10));
```



Data distribution

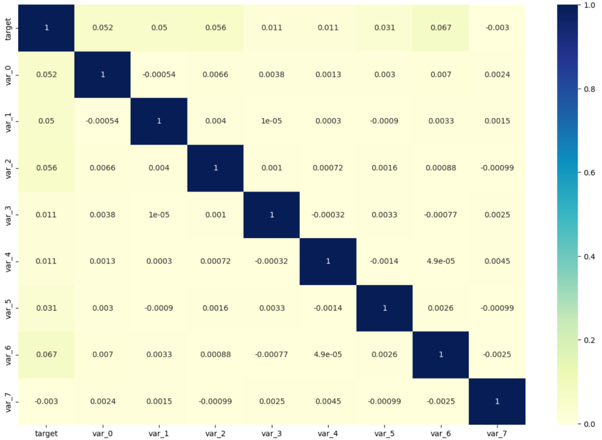
All features follow a **normal distribution**.

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Correlation matrix

What about the **correlations** between variables?

In [7]:
```
corr = train_df[train_df.columns[0:10]].corr(numeric_only=True)

plt.figure(figsize=(15,10)) # set X and Y size
sns.heatmap(corr, cmap="YlGnBu", annot=True);
```



ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA

Train-Test split

Since the only file that contains the **target column** is **train.csv**, this dataset is used for both *training* and *testing*

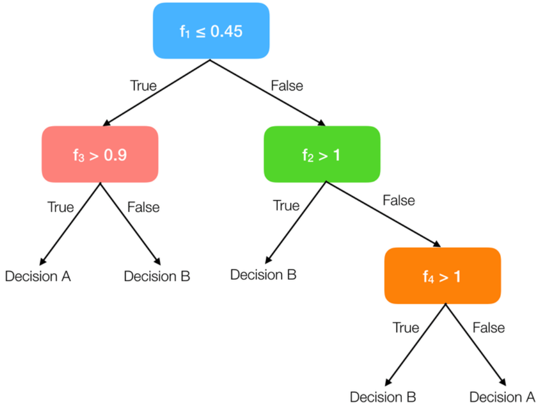
The data available are divided in the following way:

- 80% for **training**
- 20% for **testing**

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

First model: **DecisionTreeClassifier**

This model works by building a **tree-like structure** where each node represents a *decision* based on a *feature*. After traversing the **tree**, the **leaf node** reached represents the **final prediction** or **classification**.



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

First model: **DecisionTreeClassifier**

A first training, using the **default parameters**, has been done.

In [10]:
```
model = DecisionTreeClassifier(criterion = 'entropy')

model.fit(X_train, y_train)

y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)
```

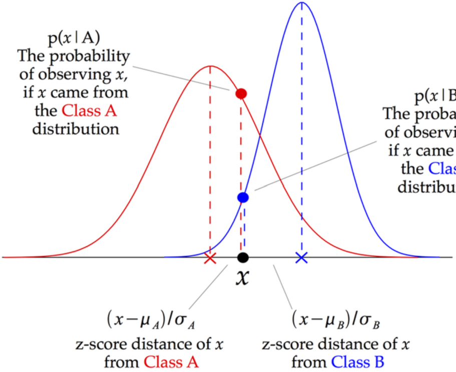
In [11]:
```
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.91      0.90      0.91      9047
           1       0.17      0.19      0.18       953

    accuracy                           0.84     10000
   macro avg       0.54      0.55      0.54     10000
weighted avg       0.84      0.84      0.84     10000
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Second model: **GaussianNB**

This is a classification algorithm that assumes *features* are **normally distributed** and **independent**. It calculates the probability of a *feature value* given a *class*, then uses **Bayes' theorem** to determine the most likely *class* for a new data point.



Second model: **GaussianNB**

A first training, using the **default parameters**, has been done.

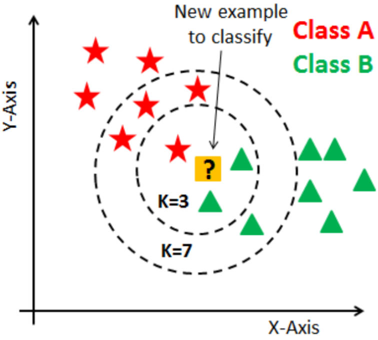
In [13]:
```
model = GaussianNB()

model.fit(X_train, y_train)

y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)
```

In [14]:
```
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.94      0.98      0.96      9047
           1       0.70      0.37      0.48       953

    accuracy                           0.92     10000
   macro avg       0.82      0.68      0.72     10000
weighted avg       0.91      0.92      0.91     10000
```

Third model: **KNeighborsClassifier**

This algorithm is a non-parametric, supervised learning classifier, which uses **proximity** to make classifications or predictions about the *grouping* of an individual data point.
This algorithm assumes that *similar things* exist in *close* **proximity**



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Third model: **KNeighborsClassifier**

A first training, using the **default parameters**, has been done.

In [16]:
```
model = KNeighborsClassifier()

model.fit(X_train, y_train)

y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)
```

In [17]:
```
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.90      1.00      0.95      9047
           1       0.30      0.00      0.01       953

    accuracy                           0.90     10000
   macro avg       0.60      0.50      0.48     10000
weighted avg       0.85      0.90      0.86     10000
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Comparing all models

The code used for comparing all the **models** (with different *parameters*):

In [19]:
```
model_lbls = ['decision_tree', 'gaussian_nb','knn']

models = {
    'decision_tree': {'name': 'Decision Tree',
           'estimator': DecisionTreeClassifier(random_state=random_state),
           'param': [{'max_depth': [*range(1,20)],'class_weight':[None,'balanced']}],
          },
    'gaussian_nb': {'name': 'Gaussian Naive Bayes',
           'estimator': GaussianNB(),
           'param': [{'var_smoothing': [10**exp for exp in range(-3,-13,-1)]}]
          },
    'knn':{'name': 'K Nearest Neighbor',
           'estimator': KNeighborsClassifier(),
           'param': [{'n_neighbors': list(range(1,7))}]
           },
}

scorings = ['accuracy','precision_macro','recall_macro','f1_macro']
clfs = []
```

In [20]:
```
for scoring in scorings:
    for m in model_lbls:
        clf = GridSearchCV(models[m]['estimator'], models[m]['param'], cv = cv, scoring = scoring)
        clf.fit(X_train, y_train)
        clfs.append(clf)

        y_true, y_pred = y_test, clf.predict(X_test)

        cr = classification_report(y_true,y_pred, output_dict=True, zero_division=1)

        comparison_df.loc[len(comparison_df)] = [scoring, models[m]['name'], clf.best_params_,
                                                 cr['accuracy'],
                                                 cr['macro avg']['precision'],
                                                 cr['macro avg']['recall'],
                                                 cr['macro avg']['f1-score']]
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Comparing all models

Operating a **grid search** with all *models*, and comparing them using different **metrics**, it is possible to find the model that stands out.

```
The best models, considering <accuracy>, are the following:
```

	Model name	Parameters	accuracy	precision_macro	recall_macro	f1_macro
1	Gaussian Naive Bayes	{'var_smoothing': 1e-08}	0.9244	0.815485	0.676114	0.720708
0	Decision Tree	{'class_weight': None, 'max_depth': 1}	0.9047	0.952350	0.500000	0.474983
2	K Nearest Neighbor	{'n_neighbors': 6}	0.9047	0.702390	0.500469	0.476027

```
The best models, considering <precision_macro>, are the following:
```

	Model name	Parameters	accuracy	precision_macro	recall_macro	f1_macro
4	Gaussian Naive Bayes	{'var_smoothing': 0.001}	0.9189	0.847935	0.602196	0.644121
5	K Nearest Neighbor	{'n_neighbors': 5}	0.9043	0.602452	0.501187	0.477980
3	Decision Tree	{'class_weight': None, 'max_depth': 3}	0.9038	0.584089	0.501850	0.479865

```
The best models, considering <recall_macro>, are the following:
```

	Model name	Parameters	accuracy	precision_macro	recall_macro	f1_macro
7	Gaussian Naive Bayes	{'var_smoothing': 1e-09}	0.9246	0.816524	0.676694	0.721446
6	Decision Tree	{'class_weight': 'balanced', 'max_depth': 4}	0.6377	0.529828	0.580092	0.488019
8	K Nearest Neighbor	{'n_neighbors': 1}	0.8749	0.519650	0.508878	0.506180

```
The best models, considering <f1_macro>, are the following:
```

	Model name	Parameters	accuracy	precision_macro	recall_macro	f1_macro
10	Gaussian Naive Bayes	{'var_smoothing': 1e-09}	0.9246	0.816524	0.676694	0.721446
9	Decision Tree	{'class_weight': None, 'max_depth': 19}	0.8723	0.562244	0.538421	0.544558
11	K Nearest Neighbor	{'n_neighbors': 1}	0.8749	0.519650	0.508878	0.506180

```
The best model is: GaussianNB()
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

The best model

After all the comparisons, the best model has been found: the metric used to pick the best one is the **F1-Score**.

Best model: *GaussianNB(var_smoothing = 1e-09)*

To speed up the computations, all the **trainings** before have been done with a smaller number of samples with respect to the all available samples for **training**.

Now it is possible to operate a **full training** on the best model.

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Final results

After a complete training (using all the **160.000** samples) the results obtained with the **best model** are the following:

In [26]:
```
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.93      0.98      0.96     35903
           1       0.72      0.37      0.49      4097

    accuracy                           0.92     40000
   macro avg       0.83      0.68      0.72     40000
weighted avg       0.91      0.92      0.91     40000
```

ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA