Data Analytics Project

Fan Jia, Jiawei Li, Strahinja Trenkic, Qiqi Zhou

Contents

1 Data Preparation

```
import pandas as pd
```

```
bank_mkt = pd.read_csv("data/BankMarketing.csv")
```

```
bank_mkt
```

```
# convert to categlory data type
```

```
# handle missing values
```

2 Exploratory Data Analysis

2.1 Import Data

The first step of data preparation is to import data. We use pandas's `read_csv()` to import data and take care of data types, true/false values and missing values.

```
import numpy as np
import pandas as pd

def import_dataset(filename):
    bank_mkt = pd.read_csv(filename,
                           na_values=["unknown", "nonexistent"],
                           true_values=["yes", "success"],
                           false_values=["no", "failure"])
    # Treat pdays = 999 as missing values
    bank_mkt["pdays"] = bank_mkt["pdays"].replace(999, pd.NA)
    # Convert types, "Int64" is nullable integer data type in pandas
    bank_mkt = bank_mkt.astype(dtype={"age": "Int64",
                                      "job": "category",
                                      "marital": "category",
                                      "education": "category",
                                      "default": "boolean",
                                      "housing": "boolean",
                                      "loan": "boolean",
                                      "contact": "category",
                                      "month": "category",
                                      "day_of_week": "category",
                                      "duration": "Int64",
                                      "campaign": "Int64",
                                      "pdays": "Int64",
```

```
                                                    "previous": "Int64",
                                                    "poutcome": "boolean",
                                                    "y": "boolean"})
    # reorder categorical data
    bank_mkt["education"] =
↪   bank_mkt["education"].cat.reorder_categories(["illiterate", "basic.4y",
↪   "basic.6y", "basic.9y", "high.school", "professional.course",
↪   "university.degree"], ordered=True)
    bank_mkt["month"] = bank_mkt["month"].cat.reorder_categories(["mar",
↪   "apr", "jun", "jul", "may", "aug", "sep", "oct", "nov", "dec"],
↪   ordered=True)
    bank_mkt["day_of_week"] =
↪   bank_mkt["day_of_week"].cat.reorder_categories(["mon", "tue", "wed",
↪   "thu", "fri"], ordered=True)
    return bank_mkt

bank_mkt = import_dataset("../data/BankMarketing.csv")
```

2.2 Exploratory Data Analysis

Exploratory Data Analysis is a process to explore the dataset with no assumptions or hypothesis. The objective is to give us enough insights for the future work.

There are many visualization libraries in Python. Pandas has its own plot API based on matplotlib and we will also use Seaborn and Altair. Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. Altair is a declarative statistical visualization library for Python, based on Vega and Vega-Lite. Both libraries provide easy to use APIs and produce beautiful graphs.

```
import altair as alt
import matplotlib.pyplot as plt
# cosmetic options for matplotlib
plt.style.use("seaborn")
plt.rcParams["figure.figsize"] = (6.4, 4.8)
plt.rcParams["figure.dpi"] = 300
plt.rcParams["axes.titleweight"] = "bold"
plt.rcParams["axes.titlepad"] = 10.0
plt.rcParams["axes.titlelocation"] = "left"
from IPython.display import set_matplotlib_formats
set_matplotlib_formats("svg")
import seaborn as sns
```

Let's first inpect the outcome distribution. As we can see below, the dataset is imbalanced. With 41188 rows of data, only 11.2% have positive outcome.

```
bank_mkt["y"].count()
```

```
bank_mkt["y"].sum()/bank_mkt["y"].count()
```

```
y_count = bank_mkt["y"].value_counts().plot(kind = "bar", title="Imbalanced
↪   Outcome")
```

Using `info()` we can see that most of features concerning the client are categorical/boolean type. And some fields such as `job`, `marital`, `education`, etc. are missing.

```
bank_mkt.info()
```

Missing values

By checking the number of missing values, we can see nearly all client do not have `pdays` and `poutcome`. 20% of the clients do not have information of `default`.

```
na = bank_mkt.isna().sum()
na_nonzero = na[na != 0]
na_perc = na_nonzero/bank_mkt.y.count()
na_bar = na_perc.plot.bar(title="Percentage of Missing Values")
```

Client Data

Let's start with basic client data.

Most of the clients's age are between 32 to 47 while there are some outlier cases beyond 70. This may imply that we should choose standardization for scaling since it's more tolerant for outliers.

```
age_hist = bank_mkt["age"].plot.hist(title="Age Histogram")
```

```
age_box = bank_mkt["age"].plot.box(vert=False, sym=".", title="Age
↪   Distribution")
```

From the graph below we can see that the age distribution in the true outcome group has lower median age but is more skewed toward an slightly older population.

```
age_y = bank_mkt[["age", "y"]].pivot(columns="y", values="age")
age_hist_outcome = age_y.plot.hist(alpha=0.9, legend=True, title="Age
↪   Histogram by Outcome")
```

```
age_box_outcome = age_y.plot.box(vert=False, sym=".", title="Age
↪   Distribution by Outcome")
```

We can also inspect the relationship between age and other categorical values.

```
age_job = bank_mkt[["age", "job"]].pivot(columns="job", values="age")
age_job_box = age_job.plot.box(vert=False, sym=".", title="Age Distribution
↪ by Job")

age_education = bank_mkt[["age", "education"]].pivot(columns="education",
↪ values="age")
age_education_box = age_education.plot.box(vert=False, sym=".", title="Age
↪ Distribution by Education")

age_marital = bank_mkt[["age", "marital"]].pivot(columns="marital",
↪ values="age")
age_marital_box = age_marital.plot.box(vert=False, sym=".", title="Age
↪ Distribution by Marital Status")

age_default = bank_mkt[["age", "default"]].pivot(columns="default",
↪ values="age")
age_default_box = age_default.plot.box(vert=False, sym=".", title="Age
↪ Distribution by Default")

age_housing = bank_mkt[["age", "housing"]].pivot(columns="housing",
↪ values="age")
age_housing_box = age_housing.plot.box(vert=False, sym=".", title="Age
↪ Distribution by Housing")

age_loan = bank_mkt[["age", "loan"]].pivot(columns="loan", values="age")
age_loan_box = age_loan.plot.box(vert=False, sym=".", title="Age
↪ Distribution by Loan")
```

We can then turn to job, eductaion and other categorical data to see their relationship to the outcome.

```
def explore_cat(df, feature):
    df = df.copy()
    if pd.api.types.is_categorical_dtype(df[feature]):
        df[feature] = df[feature].cat.add_categories('unknown')
        df[feature] = df[feature].fillna("unknown")
    feature_true = df[[feature,
↪   "y"]].groupby([feature]).sum().y.rename("True")
    feature_total = df[[feature,
↪   "y"]].groupby([feature]).count().y.rename("Total")
    feature_false = feature_total - feature_true
    feature_false = feature_false.rename("False")
    feature_true_rate = feature_true / feature_total
    feature_true_rate = feature_true_rate.rename("True Percentage")
```

```python
    explore_df = pd.concat([feature_true, feature_false, feature_total,
↪   feature_true_rate], axis=1).reset_index()
    return explore_df

def cat_outcome(df, feature):
    df = df.copy()
    if pd.api.types.is_categorical_dtype(df[feature]) and
    ↪   df[feature].isna().sum() > 0:
        df[feature] = df[feature].cat.add_categories("unknown")
        df[feature] = df[feature].fillna("unknown")
    title = feature.title().replace("_", " ").replace("Of", "of")
    f, axs = plt.subplots(1, 2, figsize=(8.6, 4.8), sharey=True,
↪   gridspec_kw=dict(wspace=0.04, width_ratios=[5, 2]))
    ax0 = df["y"].groupby(df[feature],
↪   dropna=False).value_counts(normalize=True).unstack().plot.barh(xlabel="",
↪   legend=False, stacked=True, ax=axs[0], title=f"Outcome Percentage and
↪   Total by {title}")
    ax1 = df["y"].groupby(df[feature],
↪   dropna=False).value_counts().unstack().plot.barh(xlabel="",
↪   legend=False, stacked=True, ax=axs[1])

job_outcome = cat_outcome(bank_mkt, "job")

marital_outcome = cat_outcome(bank_mkt, "marital")

education_outcome = cat_outcome(bank_mkt, "education")

default_outcome = cat_outcome(bank_mkt, "default")

housing_outcome = cat_outcome(bank_mkt, "housing")

loan_outcome = cat_outcome(bank_mkt, "loan")

job_marital_total = bank_mkt[["job", "marital", "y"]].groupby(["job",
↪   "marital"]).count().y.unstack()
job_marital_true = bank_mkt[["job", "marital", "y"]].groupby(["job",
↪   "marital"]).sum().y.unstack()
job_marital_rate = job_marital_true / job_marital_total
job_marital_rate = job_marital_rate.rename_axis(None,
↪   axis=0).rename_axis(None, axis=1)
job_marital_heatmap = sns.heatmap(data=job_marital_rate, vmin=0, vmax=0.5,
↪   annot=True).set_title("True Outcome Percentage by Job and Marital
↪   Status")

job_education_total = bank_mkt[["job", "education", "y"]].groupby(["job",
↪   "education"]).count().y.unstack()
```

```
job_education_true = bank_mkt[["job", "education", "y"]].groupby(["job", "education"]).sum().y.unstack()
job_education_rate = job_education_true / job_education_total
job_education_rate = job_education_rate.rename_axis(None, axis=0).rename_axis(None, axis=1)
job_education_heatmap = sns.heatmap(data=job_education_rate, vmin=0, vmax=0.5, annot=True).set_title("True Outcome Percentage by Job and Education")
```

```
education_marital_total = bank_mkt[["education", "marital", "y"]].groupby(["education", "marital"]).count().y.unstack()
education_marital_true = bank_mkt[["education", "marital", "y"]].groupby(["education", "marital"]).sum().y.unstack()
education_marital_rate = education_marital_true / education_marital_total
education_marital_rate = education_marital_rate.rename_axis(None, axis=0).rename_axis(None, axis=1)
education_marital_heatmap = sns.heatmap(data=education_marital_rate, vmin=0, vmax=0.5, annot=True).set_title("True Outcome Percentage by Education and Marital Status")
```

Current Campaign

```
contact_outcome = cat_outcome(bank_mkt, "contact")
```

```
month_outcome = cat_outcome(bank_mkt, "month")
```

```
day_outcome = cat_outcome(bank_mkt, "day_of_week")
```

Previous Campaign

We can plot the dirstribution of `pdays` and `previous`. As we can see, most of the client with `pdays` has been contacted 3 to 6 days before and peaked at 3 and 6 days.

```
pdays_hist = bank_mkt["pdays"].plot.hist(bins=27, title="Number of Days Since Last Contact Histogram")
```

Most of the client has never been contacted before.

```
previous_hist = bank_mkt["previous"].plot.hist(title="Number of Contacts Histogram")
```

If `pdays` is missing value, that means that the client was not previously contacted and therefore should not have `poutcome`. But `poutcome` column has less missing values than `pdays`.

```
previous_na = bank_mkt[["pdays", "poutcome"]].isna().sum()
previous_na_ax = previous_na.plot.bar(title="Number of Missing Values in
↪   pdays and poutcome")
```

We can print out the 4110 rows where the client is not contacted but have `poutcome` and see how many times they have been contacted before. The figures suggest that maybe these clients has been actually contacted but it was more than 30 days ago so the contact date was not recorded. This leaves us plenty room for feature engineering.

```
previous = bank_mkt[["campaign", "pdays", "previous", "poutcome", "y"]]
previous = previous[previous["pdays"].isna() &
↪   previous["poutcome"].notna()].reset_index(drop=True)
previous
```

```
previous_ax = previous["previous"].plot.hist(bins=12, title="Number of Days
↪   Since Last Contact for Previous Client Histogram")
```

```
bank_mkt[bank_mkt["pdays"].isna() & bank_mkt["poutcome"].isna()]
```

Correlation Heatmap

```
corr_heatmap =
↪   sns.heatmap(data=bank_mkt.corr(method="pearson")).set_title("Correlation
↪   Heatmap")
```

3 Data Preparation

3.1 Import Data

```
import numpy as np
import pandas as pd
pd.set_option('max_columns', None)
pd.set_option("max_colwidth", None)
```

```
def import_dataset(filename):
    bank_mkt = pd.read_csv(filename,
                           na_values=["unknown", "nonexistent"],
                           true_values=["yes", "success"],
                           false_values=["no", "failure"])
    # Treat pdays = 999 as missing values
    bank_mkt["pdays"] = bank_mkt["pdays"].replace(999, pd.NA)
    # Convert types, "Int64" is nullable integer data type in pandas
    bank_mkt = bank_mkt.astype(dtype={"age": "Int64",
```

```
                                      "job": "category",
                                      "marital": "category",
                                      "education": "category",
                                      "default": "boolean",
                                      "housing": "boolean",
                                      "loan": "boolean",
                                      "contact": "category",
                                      "month": "category",
                                      "day_of_week": "category",
                                      "duration": "Int64",
                                      "campaign": "Int64",
                                      "pdays": "Int64",
                                      "previous": "Int64",
                                      "poutcome": "boolean",
                                      "y": "boolean"})
    # reorder categorical data
    bank_mkt["education"] =
↪   bank_mkt["education"].cat.reorder_categories(["illiterate", "basic.4y",
↪   "basic.6y", "basic.9y", "high.school", "professional.course",
↪   "university.degree"], ordered=True)
    bank_mkt["month"] = bank_mkt["month"].cat.reorder_categories(["mar",
↪   "apr", "jun", "jul", "may", "aug", "sep", "oct", "nov", "dec"],
↪   ordered=True)
    bank_mkt["day_of_week"] =
↪   bank_mkt["day_of_week"].cat.reorder_categories(["mon", "tue", "wed",
↪   "thu", "fri"], ordered=True)
    return bank_mkt

bank_mkt = import_dataset("../data/BankMarketing.csv")
```

3.2 Partition

We need to split the dataset into trainning set and test set, then we train models on the trainning set and only use test set for final validation purposes. However, simply sampling the dataset may lead to unrepresenatative partition given that our dataset is imbalanced and clients have different features. Luckily, `scikit-learn` provides a useful function to select representative data as test data.

```
from sklearn.model_selection import StratifiedShuffleSplit

train_test_split = StratifiedShuffleSplit(n_splits=1, test_size=0.2,
↪   random_state=42)
```

```
for train_index, test_index in train_test_split.split(bank_mkt.drop("y",
↪   axis=1), bank_mkt["y"]):
    bank_train_set = bank_mkt.loc[train_index].reset_index(drop=True)
    bank_test_set = bank_mkt.loc[test_index].reset_index(drop=True)
```

3.3 Handling Missing Data

We have several strategies to handle the missing values. For categorical data, we can either treat missing value as a different category or impute them as the most frequent value.

```
from sklearn.impute import SimpleImputer
```

```
cat_features = ["job", "marital", "education"]
X = bank_train_set.drop(["duration", "y"], axis=1)
X_cat = X[cat_features]
freq_imp = SimpleImputer(strategy="most_frequent")
freq_imp.fit_transform(X_cat)
```

```
X_cat = X[cat_features]
fill_imp = SimpleImputer(strategy="constant", fill_value="unknown")
fill_imp.fit_transform(X_cat)
```

Missing values in boolean data is more tricky and requires `pandas` to transform the data first because `SimpleImputer` can not fill nullable boolean data.

```
bool_features=["default", "housing", "loan"]
X_bool = X[bool_features].astype("category")
freq_imp.fit_transform(X_bool)
```

```
X_bool = X[bool_features].astype("category")
fill_imp.fit_transform(X_bool)
```

As discussed above, some clients do not have `pdays` but have `poutcome`, which implies that they may have been contacted before but the `pdays` is more than 30 days therefore not inluded. `pdays` can also be cut into different categories which is known as the discretization process.

```
X_pdays = X["pdays"]
X_pdays[X["pdays"].isna() & X["poutcome"].notna()] = 999
pd.cut(X_pdays, [0, 5, 10, 15, 30, 1000], labels=["pdays<=5", "pdays<=10",
↪   "pdays<=15", "pdays<=30", "pdays>30"], include_lowest=True)
```

3.4 Encoding

```
from sklearn.preprocessing import OneHotEncoder
```

education, month and day_of_week are ordinal data. We can say basic.6y is more "advanced" than basic.4y for example. Therefore, we should encode education into ordinal values or transform them into years of education. The same logic also goes for month and day_of_week. Even though sklearn has its own OrdinalEncoder, it is using alphabatical order therefore we use pandas instead.

```
ord_features = ["education", "month", "day_of_week"]
X_ord = X[ord_features]
X_ord.apply(lambda x: x.cat.codes)
```

We will also need OneHotEncoder to transform categorical data into multiple binary data.

```
one_hot_features = ["job", "marital", "default", "housing", "loan"]
one_hot_encoder = OneHotEncoder(drop="first")
X_one_hot = X[one_hot_features].astype("category")
X_one_hot = freq_imp.fit_transform(X_one_hot)
one_hot_encoder.fit_transform(X_one_hot)
one_hot_encoder.get_feature_names(one_hot_features)
```

This can also be done in pandas. The advantage of doing one hot encoding in pandas is that pd.get_dummies() can keep missing values as a row of 0.

3.5 Transformation Pipeline

We can then wrap all our transformations above into pipeline.

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import FunctionTransformer
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

def pdays_transformation(X):
    """Feature Engineering `pdays`."""
    X = X.copy()
    X.loc[X["pdays"].isna() & X["poutcome"].notna(), "pdays"] = 999
    X["pdays"] = pd.cut(X["pdays"], [0, 5, 10, 15, 30, 1000],
↪   labels=["<=5", "<=10", "<=15", "<=30", ">30"], include_lowest=True)
    return X

def ordinal_transformation(X, education=None):
    """Encode ordinal labels.
```

```
    education: if education is "year", education column will be encoded
↪   into years of eductaion.
    """
    X = X.copy()
    ordinal_features = ["education", "month", "day_of_week"]
    X[ordinal_features] = X[ordinal_features].apply(lambda x: x.cat.codes)
    if education=="year":
        education_map = { 0: 0, # illiterate
                          1: 4, # basic.4y
                          2: 6, # basic.6y
                          3: 9, # basic.9y
                          4: 12, # high.school
                          5: 15, # professional course
                          6: 16} # university
        X["education"] = X["education"].replace(education_map)
    return X

def bool_transformation(X):
    """Transform boolean data into categorical data."""
    X = X.copy()
    bool_features = ["default", "housing", "loan", "poutcome"]
    X[bool_features] = X[bool_features].astype("category")
    X[bool_features] = X[bool_features].replace({True: "true", False:
↪   "false"})
    return X

cut_transformer = FunctionTransformer(pdays_transformation)

ordinal_transformer = FunctionTransformer(ordinal_transformation)

bool_transformer = FunctionTransformer(bool_transformation)

freq_features = ["job", "marital", "education"]

fill_features = ["housing", "loan", "default", "pdays", "poutcome"]

one_hot_features = ["contact"]

freq_transformer = Pipeline([
    ("freq_imputer", SimpleImputer(strategy="most_frequent")),
    ("one_hot_encoder", OneHotEncoder(drop="first",
↪   handle_unknown="error"))
])
```

```
fill_transformer = Pipeline([
    ("freq_imputer", SimpleImputer(strategy="constant",
↪   fill_value="missing")),
    ("one_hot_encoder", OneHotEncoder(drop="first",
↪   handle_unknown="error"))
])

cat_transformer = ColumnTransformer([
    ("freq_imputer", freq_transformer, freq_features),
    ("fill_imputer", fill_transformer, fill_features),
    ("one_hot_encoder", OneHotEncoder(drop="first",
↪   handle_unknown="error"), one_hot_features)
], remainder="passthrough")

preprocessor = Pipeline([
    ("bool_transformer", bool_transformer),
    ("cut_transformer", cut_transformer),
    ("ordinal_transformer", ordinal_transformer),
    ("cat_transformer", cat_transformer),
    ("scaler", StandardScaler())
])

X_train = preprocessor.fit_transform(bank_train_set.drop(["duration", "y"],
↪   axis=1))
y_train = bank_train_set["y"].astype("int").to_numpy()
```

3.6 Baseline Benchmark

```
from sklearn.model_selection import cross_validate
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression

scoring = ["f1", "precision", "recall", "roc_auc"]
# Initialize Model
nb_model = GaussianNB()
logit_model = LogisticRegression(class_weight="balanced")
knn_model = KNeighborsClassifier(n_neighbors=5)
# Train model and get CV results
nb_cv = cross_validate(nb_model, X_train, y_train, scoring=scoring, cv = 5)
logit_cv = cross_validate(logit_model, X_train, y_train, scoring=scoring,
↪   cv = 5)
```

```
knn_cv = cross_validate(knn_model, X_train, y_train, scoring=scoring, cv =
↪  5)
# Calculate CV result mean
nb_result = pd.DataFrame(nb_cv).mean().rename("Naive Bayes")
logit_result = pd.DataFrame(logit_cv).mean().rename("Logistic Regression")
knn_result = pd.DataFrame(knn_cv).mean().rename("KNN")
# Store and output result
result = pd.concat([nb_result, logit_result, knn_result], axis=1)
result
```

```
from sklearn.dummy import DummyClassifier
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
```

```
X_test = preprocessor.transform(bank_test_set.drop(["duration", "y"],
↪  axis=1))
y_test = bank_test_set["y"].astype("int").to_numpy()
# Initialize and fit Model
dummy_model = DummyClassifier(strategy="prior").fit(X_train, y_train)
nb_model = GaussianNB().fit(X_train, y_train)
logit_model = LogisticRegression(class_weight="balanced").fit(X_train,
↪  y_train)
knn_model = KNeighborsClassifier(n_neighbors=5).fit(X_train, y_train)
# Predict and calculate score
dummy_predict = dummy_model.predict(X_test)
dummy_f1 = f1_score(y_test, dummy_predict)
dummy_precision = precision_score(y_test, dummy_predict)
dummy_recall = recall_score(y_test, dummy_predict)
dummy_roc_auc = roc_auc_score(y_test, dummy_predict)
nb_predict = nb_model.predict(X_test)
nb_f1 = f1_score(y_test, nb_predict)
nb_precision = precision_score(y_test, nb_predict)
nb_recall = recall_score(y_test, nb_predict)
nb_roc_auc = roc_auc_score(y_test, nb_predict)
logit_predict = logit_model.predict(X_test)
logit_f1 = f1_score(y_test, logit_predict)
logit_precision = precision_score(y_test, logit_predict)
logit_recall = recall_score(y_test, logit_predict)
logit_roc_auc = roc_auc_score(y_test, logit_predict)
knn_predict = knn_model.predict(X_test)
knn_f1 = f1_score(y_test, knn_predict)
knn_precision = precision_score(y_test, knn_predict)
```

```
knn_recall = recall_score(y_test, knn_predict)
knn_roc_auc = roc_auc_score(y_test, knn_predict)
# Store and output result
result = pd.DataFrame(data={"Dummy Classifier": [dummy_f1, dummy_precision,
↪  dummy_recall, dummy_roc_auc],
                            "Naive Bayes": [nb_f1, nb_precision, nb_recall,
                             ↪  nb_roc_auc],
                            "Logistic Regression": [logit_f1,
                             ↪  logit_precision, logit_recall,
                             ↪  logit_roc_auc],
                            "KNN": [knn_f1, knn_precision, knn_recall,
                             ↪  knn_roc_auc]},
                       index=["F1 Score", "Precision Score", "Recall
↪  Score", "ROC AUC Score"])
result
```

4 Exploratory Data Analysis

This is exploratory data analysis part.

You can write LaTeX, which is a nice tool for generating mathematical formulas like this:

$$y = \beta_0 + \beta_1 X$$

```
# Insert code here.
```

5 Tree-based Models

5.1 Data Preparation

```
import numpy as np
import pandas as pd
pd.set_option('max_columns', None)
pd.set_option("max_colwidth", None)
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
```

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder

# Features where the missing values will be imputed as the most frequent
↪  value
freq_features = ["job", "marital", "education"]

# Features where the missing values will be filled as a distinct value
fill_features = ["housing", "loan", "default", "pdays", "poutcome"]

# Features that are not in freq_features or fill_features but need to be
↪  one hot encoded
one_hot_features = ["contact"]

def import_dataset(filename):
    bank_mkt = pd.read_csv(filename,
                           na_values=["unknown", "nonexistent"],
                           true_values=["yes", "success"],
                           false_values=["no", "failure"])
    # Treat pdays = 999 as missing values
    bank_mkt["pdays"] = bank_mkt["pdays"].replace(999, pd.NA)
    # Convert types, "Int64" is nullable integer data type in pandas
    bank_mkt = bank_mkt.astype(dtype={"age": "Int64",
                                      "job": "category",
                                      "marital": "category",
                                      "education": "category",
                                      "default": "boolean",
                                      "housing": "boolean",
                                      "loan": "boolean",
                                      "contact": "category",
                                      "month": "category",
                                      "day_of_week": "category",
                                      "duration": "Int64",
                                      "campaign": "Int64",
                                      "pdays": "Int64",
                                      "previous": "Int64",
                                      "poutcome": "boolean",
                                      "y": "boolean"})
    # reorder categorical data
    bank_mkt["education"] =
↪   bank_mkt["education"].cat.reorder_categories(["illiterate", "basic.4y",
↪   "basic.6y", "basic.9y", "high.school", "professional.course",
↪   "university.degree"], ordered=True)
```

```
    bank_mkt["month"] = bank_mkt["month"].cat.reorder_categories(["mar",
↪   "apr", "jun", "jul", "may", "aug", "sep", "oct", "nov", "dec"],
↪   ordered=True)
    bank_mkt["day_of_week"] =
↪   bank_mkt["day_of_week"].cat.reorder_categories(["mon", "tue", "wed",
↪   "thu", "fri"], ordered=True)
    return bank_mkt

def pdays_transformation(X):
    """Feature Engineering `pdays`."""
    X = X.copy()
    X.loc[X["pdays"].isna() & X["poutcome"].notna(), "pdays"] = 999
    X["pdays"] = pd.cut(X["pdays"], [0, 5, 10, 15, 30, 1000],
↪   labels=["<=5", "<=10", "<=15", "<=30", ">30"], include_lowest=True)
    return X

def ordinal_transformation(X, education=None):
    """Encode ordinal labels.

    education: if education is "year", education column will be encoded
↪   into years of eductaion.
    """
    X = X.copy()
    ordinal_features = ["education", "month", "day_of_week"]
    X[ordinal_features] = X[ordinal_features].apply(lambda x: x.cat.codes)
    if education=="year":
        education_map = { 0: 0, # illiterate
                          1: 4, # basic.4y
                          2: 6, # basic.6y
                          3: 9, # basic.9y
                          4: 12, # high.school
                          5: 15, # professional course
                          6: 16} # university
        X["education"] = X["education"].replace(education_map)
    return X

def bool_transformation(X):
    """Transform boolean data into categorical data."""
    X = X.copy()
    bool_features = ["default", "housing", "loan", "poutcome"]
    X[bool_features] = X[bool_features].astype("category")
    X[bool_features] = X[bool_features].replace({True: "true", False:
↪   "false"})
```

```
    return X

cut_transformer = FunctionTransformer(pdays_transformation)

ordinal_transformer = FunctionTransformer(ordinal_transformation)

bool_transformer = FunctionTransformer(bool_transformation)

freq_transformer = Pipeline([
    ("freq_imputer", SimpleImputer(strategy="most_frequent")),
    ("one_hot_encoder", OneHotEncoder(drop="first",
↪   handle_unknown="error"))
])

fill_transformer = Pipeline([
    ("freq_imputer", SimpleImputer(strategy="constant",
↪   fill_value="missing")),
    ("one_hot_encoder", OneHotEncoder(drop="first",
↪   handle_unknown="error"))
])

cat_transformer = ColumnTransformer([
    ("freq_imputer", freq_transformer, freq_features),
    ("fill_imputer", fill_transformer, fill_features),
    ("one_hot_encoder", OneHotEncoder(drop="first",
↪   handle_unknown="error"), one_hot_features)
], remainder="passthrough")

preprocessor = Pipeline([
    ("bool_transformer", bool_transformer),
    ("cut_transformer", cut_transformer),
    ("ordinal_transformer", ordinal_transformer),
    ("cat_transformer", cat_transformer),
    ("scaler", StandardScaler())
])

bank_mkt = import_dataset("../data/BankMarketing.csv")

train_test_split = StratifiedShuffleSplit(n_splits=1, test_size=0.2,
↪   random_state=42)

for train_index, test_index in train_test_split.split(bank_mkt.drop("y",
↪   axis=1), bank_mkt["y"]):
```

```
    bank_train_set = bank_mkt.loc[train_index].reset_index(drop=True)
    bank_test_set = bank_mkt.loc[test_index].reset_index(drop=True)

X_train = preprocessor.fit_transform(bank_train_set.drop(["duration", "y"],
↪   axis=1))
y_train = bank_train_set["y"].astype("int").to_numpy()
```

5.2 Methods

6 Neural Network

6.1 Data Preparation

```
import numpy as np
import pandas as pd
pd.set_option('max_columns', None)
pd.set_option("max_colwidth", None)
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder

# Features where the missing values will be imputed as the most frequent
↪   value
freq_features = ["job", "marital", "education"]

# Features where the missing values will be filled as a distinct value
fill_features = ["housing", "loan", "default", "pdays", "poutcome"]

# Features that are not in freq_features or fill_features but need to be
↪   one hot encoded
one_hot_features = ["contact"]

def import_dataset(filename):
    bank_mkt = pd.read_csv(filename,
```

```
                               na_values=["unknown", "nonexistent"],
                               true_values=["yes", "success"],
                               false_values=["no", "failure"])
    # Treat pdays = 999 as missing values
    bank_mkt["pdays"] = bank_mkt["pdays"].replace(999, pd.NA)
    # Convert types, "Int64" is nullable integer data type in pandas
    bank_mkt = bank_mkt.astype(dtype={"age": "Int64",
                                      "job": "category",
                                      "marital": "category",
                                      "education": "category",
                                      "default": "boolean",
                                      "housing": "boolean",
                                      "loan": "boolean",
                                      "contact": "category",
                                      "month": "category",
                                      "day_of_week": "category",
                                      "duration": "Int64",
                                      "campaign": "Int64",
                                      "pdays": "Int64",
                                      "previous": "Int64",
                                      "poutcome": "boolean",
                                      "y": "boolean"})
    # reorder categorical data
    bank_mkt["education"] =
↪   bank_mkt["education"].cat.reorder_categories(["illiterate", "basic.4y",
↪   "basic.6y", "basic.9y", "high.school", "professional.course",
↪   "university.degree"], ordered=True)
    bank_mkt["month"] = bank_mkt["month"].cat.reorder_categories(["mar",
↪   "apr", "jun", "jul", "may", "aug", "sep", "oct", "nov", "dec"],
↪   ordered=True)
    bank_mkt["day_of_week"] =
↪   bank_mkt["day_of_week"].cat.reorder_categories(["mon", "tue", "wed",
↪   "thu", "fri"], ordered=True)
    return bank_mkt

def pdays_transformation(X):
    """Feature Engineering `pdays`."""
    X = X.copy()
    X.loc[X["pdays"].isna() & X["poutcome"].notna(), "pdays"] = 999
    X["pdays"] = pd.cut(X["pdays"], [0, 5, 10, 15, 30, 1000],
↪   labels=["<=5", "<=10", "<=15", "<=30", ">30"], include_lowest=True)
    return X
```

```
def ordinal_transformation(X, education=None):
    """Encode ordinal labels.

    education: if education is "year", education column will be encoded
↪   into years of eductaion.
    """
    X = X.copy()
    ordinal_features = ["education", "month", "day_of_week"]
    X[ordinal_features] = X[ordinal_features].apply(lambda x: x.cat.codes)
    if education=="year":
        education_map = { 0: 0, # illiterate
                          1: 4, # basic.4y
                          2: 6, # basic.6y
                          3: 9, # basic.9y
                          4: 12, # high.school
                          5: 15, # professional course
                          6: 16} # university
        X["education"] = X["education"].replace(education_map)
    return X

def bool_transformation(X):
    """Transform boolean data into categorical data."""
    X = X.copy()
    bool_features = ["default", "housing", "loan", "poutcome"]
    X[bool_features] = X[bool_features].astype("category")
    X[bool_features] = X[bool_features].replace({True: "true", False:
↪   "false"})
    return X

cut_transformer = FunctionTransformer(pdays_transformation)

ordinal_transformer = FunctionTransformer(ordinal_transformation)

bool_transformer = FunctionTransformer(bool_transformation)

freq_transformer = Pipeline([
    ("freq_imputer", SimpleImputer(strategy="most_frequent")),
    ("one_hot_encoder", OneHotEncoder(drop="first",
↪   handle_unknown="error"))
])

fill_transformer = Pipeline([
```

```
    ("freq_imputer", SimpleImputer(strategy="constant",
↪   fill_value="missing")),
    ("one_hot_encoder", OneHotEncoder(drop="first",
↪   handle_unknown="error"))
])

cat_transformer = ColumnTransformer([
    ("freq_imputer", freq_transformer, freq_features),
    ("fill_imputer", fill_transformer, fill_features),
    ("one_hot_encoder", OneHotEncoder(drop="first",
↪   handle_unknown="error"), one_hot_features)
], remainder="passthrough")

preprocessor = Pipeline([
    ("bool_transformer", bool_transformer),
    ("cut_transformer", cut_transformer),
    ("ordinal_transformer", ordinal_transformer),
    ("cat_transformer", cat_transformer),
    ("scaler", StandardScaler())
])

bank_mkt = import_dataset("../data/BankMarketing.csv")

train_test_split = StratifiedShuffleSplit(n_splits=1, test_size=0.2,
↪   random_state=42)

for train_index, test_index in train_test_split.split(bank_mkt.drop("y",
↪   axis=1), bank_mkt["y"]):
    bank_train_set = bank_mkt.loc[train_index].reset_index(drop=True)
    bank_test_set = bank_mkt.loc[test_index].reset_index(drop=True)

X_train = preprocessor.fit_transform(bank_train_set.drop(["duration", "y"],
↪   axis=1))
y_train = bank_train_set["y"].astype("int").to_numpy()
```

6.2 Methods

7 Support Vector Machine

This is SVM part.

7.1 Data Preparation

```python
import numpy as np
import pandas as pd
pd.set_option('max_columns', None)
pd.set_option("max_colwidth", None)
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder

# Features where the missing values will be imputed as the most frequent
↪   value
freq_features = ["job", "marital", "education"]

# Features where the missing values will be filled as a distinct value
fill_features = ["housing", "loan", "default", "pdays", "poutcome"]

# Features that are not in freq_features or fill_features but need to be
↪   one hot encoded
one_hot_features = ["contact"]

def import_dataset(filename):
    bank_mkt = pd.read_csv(filename,
                           na_values=["unknown", "nonexistent"],
                           true_values=["yes", "success"],
                           false_values=["no", "failure"])
    # Treat pdays = 999 as missing values
    bank_mkt["pdays"] = bank_mkt["pdays"].replace(999, pd.NA)
    # Convert types, "Int64" is nullable integer data type in pandas
    bank_mkt = bank_mkt.astype(dtype={"age": "Int64",
                                      "job": "category",
                                      "marital": "category",
                                      "education": "category",
                                      "default": "boolean",
```

```
                                            "housing": "boolean",
                                            "loan": "boolean",
                                            "contact": "category",
                                            "month": "category",
                                            "day_of_week": "category",
                                            "duration": "Int64",
                                            "campaign": "Int64",
                                            "pdays": "Int64",
                                            "previous": "Int64",
                                            "poutcome": "boolean",
                                            "y": "boolean"})
    # reorder categorical data
    bank_mkt["education"] =
↪   bank_mkt["education"].cat.reorder_categories(["illiterate", "basic.4y",
↪   "basic.6y", "basic.9y", "high.school", "professional.course",
↪   "university.degree"], ordered=True)
    bank_mkt["month"] = bank_mkt["month"].cat.reorder_categories(["mar",
↪   "apr", "jun", "jul", "may", "aug", "sep", "oct", "nov", "dec"],
↪   ordered=True)
    bank_mkt["day_of_week"] =
↪   bank_mkt["day_of_week"].cat.reorder_categories(["mon", "tue", "wed",
↪   "thu", "fri"], ordered=True)
    return bank_mkt

def pdays_transformation(X):
    """Feature Engineering `pdays`."""
    X = X.copy()
    X.loc[X["pdays"].isna() & X["poutcome"].notna(), "pdays"] = 999
    X["pdays"] = pd.cut(X["pdays"], [0, 5, 10, 15, 30, 1000],
↪   labels=["<=5", "<=10", "<=15", "<=30", ">30"], include_lowest=True)
    return X

def ordinal_transformation(X, education=None):
    """Encode ordinal labels.

    education: if education is "year", education column will be encoded
↪   into years of eductaion.
    """
    X = X.copy()
    ordinal_features = ["education", "month", "day_of_week"]
    X[ordinal_features] = X[ordinal_features].apply(lambda x: x.cat.codes)
    if education=="year":
        education_map = { 0: 0, # illiterate
```

```python
                                  1: 4, # basic.4y
                                  2: 6, # basic.6y
                                  3: 9, # basic.9y
                                  4: 12, # high.school
                                  5: 15, # professional course
                                  6: 16} # university
        X["education"] = X["education"].replace(education_map)
    return X

def bool_transformation(X):
    """Transform boolean data into categorical data."""
    X = X.copy()
    bool_features = ["default", "housing", "loan", "poutcome"]
    X[bool_features] = X[bool_features].astype("category")
    X[bool_features] = X[bool_features].replace({True: "true", False:
↪   "false"})
    return X

cut_transformer = FunctionTransformer(pdays_transformation)

ordinal_transformer = FunctionTransformer(ordinal_transformation)

bool_transformer = FunctionTransformer(bool_transformation)

freq_transformer = Pipeline([
    ("freq_imputer", SimpleImputer(strategy="most_frequent")),
    ("one_hot_encoder", OneHotEncoder(drop="first",
↪   handle_unknown="error"))
])

fill_transformer = Pipeline([
    ("freq_imputer", SimpleImputer(strategy="constant",
↪   fill_value="missing")),
    ("one_hot_encoder", OneHotEncoder(drop="first",
↪   handle_unknown="error"))
])

cat_transformer = ColumnTransformer([
    ("freq_imputer", freq_transformer, freq_features),
    ("fill_imputer", fill_transformer, fill_features),
    ("one_hot_encoder", OneHotEncoder(drop="first",
↪   handle_unknown="error"), one_hot_features)
], remainder="passthrough")
```

```
preprocessor = Pipeline([
    ("bool_transformer", bool_transformer),
    ("cut_transformer", cut_transformer),
    ("ordinal_transformer", ordinal_transformer),
    ("cat_transformer", cat_transformer),
    ("scaler", StandardScaler())
])

bank_mkt = import_dataset("../data/BankMarketing.csv")

train_test_split = StratifiedShuffleSplit(n_splits=1, test_size=0.2,
↪   random_state=42)

for train_index, test_index in train_test_split.split(bank_mkt.drop("y",
↪   axis=1), bank_mkt["y"]):
    bank_train_set = bank_mkt.loc[train_index].reset_index(drop=True)
    bank_test_set = bank_mkt.loc[test_index].reset_index(drop=True)

X_train = preprocessor.fit_transform(bank_train_set.drop(["duration", "y"],
↪   axis=1))
y_train = bank_train_set["y"].astype("int").to_numpy()
```

7.2 Methods

8 Ensemble

8.1 Data Preparation

```
import numpy as np
import pandas as pd
pd.set_option('max_columns', None)
pd.set_option("max_colwidth", None)
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
```

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder

# Features where the missing values will be imputed as the most frequent
↪  value
freq_features = ["job", "marital", "education"]

# Features where the missing values will be filled as a distinct value
fill_features = ["housing", "loan", "default", "pdays", "poutcome"]

# Features that are not in freq_features or fill_features but need to be
↪  one hot encoded
one_hot_features = ["contact"]

def import_dataset(filename):
    bank_mkt = pd.read_csv(filename,
                           na_values=["unknown", "nonexistent"],
                           true_values=["yes", "success"],
                           false_values=["no", "failure"])
    # Treat pdays = 999 as missing values
    bank_mkt["pdays"] = bank_mkt["pdays"].replace(999, pd.NA)
    # Convert types, "Int64" is nullable integer data type in pandas
    bank_mkt = bank_mkt.astype(dtype={"age": "Int64",
                                      "job": "category",
                                      "marital": "category",
                                      "education": "category",
                                      "default": "boolean",
                                      "housing": "boolean",
                                      "loan": "boolean",
                                      "contact": "category",
                                      "month": "category",
                                      "day_of_week": "category",
                                      "duration": "Int64",
                                      "campaign": "Int64",
                                      "pdays": "Int64",
                                      "previous": "Int64",
                                      "poutcome": "boolean",
                                      "y": "boolean"})
    # reorder categorical data
    bank_mkt["education"] =
↪   bank_mkt["education"].cat.reorder_categories(["illiterate", "basic.4y",
↪   "basic.6y", "basic.9y", "high.school", "professional.course",
↪   "university.degree"], ordered=True)
```

```
    bank_mkt["month"] = bank_mkt["month"].cat.reorder_categories(["mar",
↪   "apr", "jun", "jul", "may", "aug", "sep", "oct", "nov", "dec"],
↪   ordered=True)
    bank_mkt["day_of_week"] =
↪   bank_mkt["day_of_week"].cat.reorder_categories(["mon", "tue", "wed",
↪   "thu", "fri"], ordered=True)
    return bank_mkt

def pdays_transformation(X):
    """Feature Engineering `pdays`."""
    X = X.copy()
    X.loc[X["pdays"].isna() & X["poutcome"].notna(), "pdays"] = 999
    X["pdays"] = pd.cut(X["pdays"], [0, 5, 10, 15, 30, 1000],
↪   labels=["<=5", "<=10", "<=15", "<=30", ">30"], include_lowest=True)
    return X

def ordinal_transformation(X, education=None):
    """Encode ordinal labels.

    education: if education is "year", education column will be encoded
↪   into years of eductaion.
    """
    X = X.copy()
    ordinal_features = ["education", "month", "day_of_week"]
    X[ordinal_features] = X[ordinal_features].apply(lambda x: x.cat.codes)
    if education=="year":
        education_map = { 0: 0, # illiterate
                          1: 4, # basic.4y
                          2: 6, # basic.6y
                          3: 9, # basic.9y
                          4: 12, # high.school
                          5: 15, # professional course
                          6: 16} # university
        X["education"] = X["education"].replace(education_map)
    return X

def bool_transformation(X):
    """Transform boolean data into categorical data."""
    X = X.copy()
    bool_features = ["default", "housing", "loan", "poutcome"]
    X[bool_features] = X[bool_features].astype("category")
    X[bool_features] = X[bool_features].replace({True: "true", False:
↪   "false"})
```

```
    return X

cut_transformer = FunctionTransformer(pdays_transformation)

ordinal_transformer = FunctionTransformer(ordinal_transformation)

bool_transformer = FunctionTransformer(bool_transformation)

freq_transformer = Pipeline([
    ("freq_imputer", SimpleImputer(strategy="most_frequent")),
    ("one_hot_encoder", OneHotEncoder(drop="first",
↪   handle_unknown="error"))
])

fill_transformer = Pipeline([
    ("freq_imputer", SimpleImputer(strategy="constant",
↪   fill_value="missing")),
    ("one_hot_encoder", OneHotEncoder(drop="first",
↪   handle_unknown="error"))
])

cat_transformer = ColumnTransformer([
    ("freq_imputer", freq_transformer, freq_features),
    ("fill_imputer", fill_transformer, fill_features),
    ("one_hot_encoder", OneHotEncoder(drop="first",
↪   handle_unknown="error"), one_hot_features)
], remainder="passthrough")

preprocessor = Pipeline([
    ("bool_transformer", bool_transformer),
    ("cut_transformer", cut_transformer),
    ("ordinal_transformer", ordinal_transformer),
    ("cat_transformer", cat_transformer),
    ("scaler", StandardScaler())
])

bank_mkt = import_dataset("../data/BankMarketing.csv")

train_test_split = StratifiedShuffleSplit(n_splits=1, test_size=0.2,
↪   random_state=42)

for train_index, test_index in train_test_split.split(bank_mkt.drop("y",
↪   axis=1), bank_mkt["y"]):
```

```
    bank_train_set = bank_mkt.loc[train_index].reset_index(drop=True)
    bank_test_set = bank_mkt.loc[test_index].reset_index(drop=True)

X_train = preprocessor.fit_transform(bank_train_set.drop(["duration", "y"],
↪   axis=1))
y_train = bank_train_set["y"].astype("int").to_numpy()
```

8.2 Methods

```
# Insert code here.
```