

**TỔNG CÔNG TY VIỄN THÔNG VIETTEL**

**KHỐI CÔNG NGHỆ THÔNG TIN**

**XÂY DỰNG MÔ HÌNH DỰ ĐOÁN BANKING POTENTIAL CHO NHÓM MERCHANT**

**Mã hiệu dự án:** **MyViettel\_0009**

**Mã hiệu tài liệu: MyViettel\_0009**

**<Địa điểm, Thời gian>**

**BẢNG GHI NHẬN TIẾN ĐỘ**

\*A - Tạo mới, M - Sửa đổi, D - Xóa bỏ

| **Ngày**  **bàn giao** | **Yêu cầu** | **A\***  **M, D** | **Trạng thái** | **Đầu mối KH** | **Mô tả** | **Ghi chú** |
| --- | --- | --- | --- | --- | --- | --- |
| 20/5/2023 | xây dựng mô hình dự đoán banking potential cho nhóm merchant |  | Hoàn thành | PTC, PKH | * xây dựng mô hình dự đoán độ phù hợp banking potential cho nhóm merchant |  |
|  |  |  |  |  |  |  |

**MỤC LỤC**

[**1. Tìm Hiểu Dữ Liệu 6**](#_heading=h.4ekebi7s4l4y)

[1.1. Phân tích dữ liệu 6](#_heading=h.z2e8u9lu4qb1)

[1.2. Sinh dữ liệu từ meta data 6](#_heading=h.eh2b0738579d)

[1.3. Kiểm tra missing value 7](#_heading=h.ulta7nmxxtu1)

[1.4. Xử lý missing value 7](#_heading=h.f1lxdx3kjilz)

[1.5. Chuẩn hóa dữ liệu 7](#_heading=h.c2jj3miqc0rl)

[1.6. Selection features 7](#_heading=h.i0yps1ib7pnw)

[1.7. Chia tập test set 8](#_heading=h.pu3dnsrro5i3)

[**2. Fine-Tune và đánh giá 8**](#_heading=h.bbfba0mzzvtu)

[2.1. SVM 8](#_heading=h.4p9bz0k59t26)

[2.1.1. Training model base 8](#_heading=h.3fldyvc50f61)

[2.1.2. Fine-tune hyperparameters lần 2 8](#_heading=h.80euquu3jpbu)

[2.1.3. Fine-tune hyperparameters lần 3 9](#_heading=h.iw3lp02oends)

[2.1.4. Fine-tune hyperparameters lần 4 9](#_heading=h.8c0xrfp6duxn)

[2.2. XGBoost 9](#_heading=h.daima08lvgf)

[2.2.1. Training model base 9](#_heading=h.ccwkqpan8i79)

[2.2.2. Fine-tune hyperparameters lần 2 9](#_heading=h.a41709b9r2fz)

[2.2.3. Fine-tune hyperparameters lần 3 9](#_heading=h.on9tobpn3y7a)

[2.2.4. Fine-tune hyperparameters lần 4 9](#_heading=h.fgd0t8kagmwh)

[2.3. RandomForest 9](#_heading=h.ucfuigj1oih8)

[2.3.1. Training model base 9](#_heading=h.c1p3czfyng2c)

[2.3.3. Fine-tune hyperparameters lần 3 10](#_heading=h.xx2rsiotewzg)

[2.3.4. Fine-tune hyperparameters lần 4 10](#_heading=h.2kt2dsqo9wo6)

[2.4. MLP 10](#_heading=h.vr55alvasfwe)

[2.4.1. Training model base 11](#_heading=h.kpjxppda701m)

[2.4.2. Fine-tune hyperparameters lần 2 11](#_heading=h.mv09xp3x0yqr)

[2.4.3. Fine-tune hyperparameters lần 3 11](#_heading=h.xvpmxjaotvc8)

[2.4.4. Fine-tune hyperparameters lần 4 11](#_heading=h.76kmab4rsb08)

**TRANG KÝ**

Người lập: <Ngày>

<Chức danh>

Người xem xét: <Ngày>

<Chức danh>

Người xem xét: <Ngày>

<Chức danh>

Người phê duyệt: <Ngày>

<Chức danh>

# **Tìm Hiểu Dữ Liệu**

## Phân tích dữ liệu

df.head(15)

df.describe()

values\_count = []

**for** col **in** label:

values = df[col].value\_counts().to\_dict()[1]

values\_count.append(values)

df\_plot = pd.DataFrame({'name':label,'count':values\_count})

**import** plotly.express as px

fig = px.pie(df\_plot, values='count', names='name', title='Pie plot for ratio label in data')

fig.show()

## Sinh dữ liệu từ meta data

**def** make\_dataset(num\_samples, num\_features):

df = pd.DataFrame()

types = ["float", "int", "string"]

**for** i **in** range(num\_features):

data\_type = random.choice(types)

**if** data\_type == "int":

df['feature\_' + str(i)] = random.sample(range(0, 9999999), num\_samples)

**elif** data\_type == "float":

df['feature\_' + str(i)] = np.random.uniform(low=-10, high=10, size=(num\_samples,)).tolist()

**elif** data\_type == "string":

string\_list = []

**for** j **in** range(num\_samples):

string\_list.append(''.join(random.choice(string.ascii\_letters + string.digits + string.punctuation)

**for** j **in** range(5)))

df['feature\_' + str(i)] = string\_list

df['output'] = np.random.uniform(low=-10, high=10, size=(num\_samples,)).tolist()

**return** df

my\_data = make\_dataset(2000, 901)

my\_data.to\_csv('data\_path/data\_name.csv', index=False)

## Kiểm tra missing value

# sum total null values

**print**(df.isna().sum().sum())

**for** i **in** df :

count\_miss = df[i].isna().sum().sum()

ratio\_miss = count\_miss/len(df)

**print**(i,':',round(ratio\_miss,3)," % miss values")

## Xử lý missing value

#drop sample all null values

df.dropna(how='all',inplace=True)

**for** i **in** df :

count\_miss = df[i].isna().sum().sum()

ratio\_miss = count\_miss/len(df)

**print**(i,':',round(ratio\_miss,3)," % miss values")

**if** ratio\_miss > 0.45:

df.drop(columns = [i],inplace=True)

df.dropna(thresh=int(len(df.columns)\*0.75),inplace=True)

**for** i **in** df.columns:

**if** df[i].dtypes =='O':

df[i].fillna('unknow',inplace=True)

df[i] = LabelEncoder().fit\_transform(df[i])

**else**:

df[i] = df[i].interpolate(method='polynomial', order=2,inplace=False)

df.dropna(inplace=True)

## Chuẩn hóa dữ liệu

norm = StandardScaler()

X\_train\_norm = norm.fit\_transform(X\_train)

X\_test\_norm = norm.fit\_transform(X\_test)

## Selection features

**def** ranking(ranks, names, order=1):

minmax = MinMaxScaler()

ranks = minmax.fit\_transform(order\*np.array([ranks]).T).T[0]

ranks = map(**lambda** x: round(x,2), ranks)

**return** dict(zip(names, ranks))

colnames = df.iloc[:,:201].columns

**from** sklearn.feature\_selection **import** RFE

rf = RandomForestClassifier(n\_estimators=100,max\_depth=None, criterion='gini')

rf.fit(X, y)

# Extract feature importance coefficients as calculated by the trained model

ranks = {}

ranks["RF"] = ranking(rf.feature\_importances\_, colnames)

rf\_df = pd.DataFrame(list(ranks['RF'].items()), columns= ['Feature','alg\_importance'])

rfe = RFE(rf, n\_features\_to\_select=1, verbose =3 )

rfe.fit(X, y)

ranks["RFE\_RF"] = ranking(list(map(float, rfe.ranking\_)), colnames, order=-1)

thred = rf\_df["alg\_importance"].quantile(0.9)

Feature = rf\_df[rf\_df["alg\_importance"]>thred]['Feature'].values

## Chia tập test set

**from** skmultilearn.model\_selection **import** iterative\_train\_test\_split

X\_train, y\_train, X\_test, y\_test = iterative\_train\_test\_split(X, y, test\_size = 0.2)

X\_trains, y\_trains, X\_val, y\_val = iterative\_train\_test\_split(X\_train, y\_train, test\_size = 0.2)

# Fine-Tune và đánh giá

## SVM

### Training model base

* + - 1. **training base**

base\_lr = LinearSVC()

svm = ClassifierChain(base\_lr, order="random", random\_state=42)

svm.fit(X\_train\_norm, y\_train)

* + - 1. **Kết quả**

### Fine-tune hyperparameters lần 2

* + - 1. **Cài đặt Hyperparameters**
      2. **Kết quả**

### Fine-tune hyperparameters lần 3

* + - 1. **Cài đặt Hyperparameters**
      2. **Kết quả**

### Fine-tune hyperparameters lần 4

* + - 1. **Cài đặt Hyperparameters**
      2. **Kết quả**

## XGBoost

### Training model base

* + - 1. **training base**

xgb = XGBClassifier()

xgb.fit(X\_train, y\_train, eval\_set=[(X\_test, y\_test)])

* + - 1. **Kết quả**

### Fine-tune hyperparameters lần 2

* + - 1. **Cài đặt Hyperparameters**
      2. **Kết quả**

### Fine-tune hyperparameters lần 3

* + - 1. **Cài đặt Hyperparameters**
      2. **Kết quả**

### Fine-tune hyperparameters lần 4

* + - 1. **Cài đặt Hyperparameters**
      2. **Kết quả**

## RandomForest

### Training model base

* + - 1. **training base**

rf = RandomForestClassifier(n\_estimators=1000,max\_depth=None, criterion='gini')

rf.fit(X\_train,y\_train)

* + - 1. **Kết quả**
    1. **Fine-tune hyperparameters lần 2**
       1. **Cài đặt Hyperparameters**
       2. **Kết quả**

### Fine-tune hyperparameters lần 3

* + - 1. **Cài đặt Hyperparameters**
      2. **Kết quả**

### Fine-tune hyperparameters lần 4

* + - 1. **Cài đặt Hyperparameters**
      2. **Kết quả**

## MLP

**class** MLP(nn.Module):

**def** \_\_init\_\_(self, input\_dim, output\_dim):

super().\_\_init\_\_()

self.input\_fc = nn.Sequential( nn.Linear(input\_dim,256),

nn.ReLU())

self.hidden\_fc = nn.Sequential( nn.Linear(256, 128),

nn.ReLU(),

nn.Linear(128, 64),

nn.ReLU(),

nn.Linear(64, 32),

nn.ReLU())

self.output\_fc = nn.Sequential( nn.Linear(32, output\_dim),

nn.Sigmoid())

**def** forward(self, x):

h\_1 = self.input\_fc(x)

h\_2 = self.hidden\_fc(h\_1)

y\_pred = self.output\_fc(h\_2)

**return** y\_pred

**class** dataset(Dataset):

**def** \_\_init\_\_(self,input\_data,label):

self.input = input\_data

self.label = label

**def** \_\_len\_\_(self):

**return** len(self.label)

**def** \_\_getitem\_\_(self, idx):

input\_data = torch.from\_numpy(self.input[idx]).float()

label\_data = torch.from\_numpy(self.label[idx]).float()

**return** input\_data,label\_data

### Training model base

* + - 1. **training base**

train\_dataset = dataset(X\_train,y\_train)

train\_loader = torch.utils.data.DataLoader(dataset=train\_dataset,

batch\_size=48,

shuffle=True)

**from** tqdm **import** tqdm

**import** torchmetrics

device = 'cpu'

model = MLP(201,14).to(device)

criterion = nn.BCEWithLogitsLoss()

optimizer = torch.optim.Adam(model.parameters(), lr=0.002)

metric = torchmetrics.Accuracy(task= "multiclass",num\_classes=14)

total\_steps = len(train\_loader)

numbers\_epochs=100

pbar = tqdm (range(numbers\_epochs))

**for** epoch **in** pbar:

model.train()

**for** i, (images, label) **in** enumerate(train\_loader):

images = images.to(device)

label = label.to(device)

output = model(images)

loss = criterion(output, label)

optimizer.zero\_grad()

loss.backward()

optimizer.step()

pbar.set\_description("Epoch [{}/{}] ".format(epoch + 1, numbers\_epochs))

* + - 1. **Kết quả**

### Fine-tune hyperparameters lần 2

* + - 1. **Cài đặt Hyperparameters**
      2. **Kết quả**

### Fine-tune hyperparameters lần 3

* + - 1. **Cài đặt Hyperparameters**
      2. **Kết quả**

### Fine-tune hyperparameters lần 4

* + - 1. **Cài đặt Hyperparameters**
      2. **Kết quả**