Business Analytics : Churn Prediction

Maxime Manderlier

November 17, 2025

1 Introduction

Customer churn represents the proportion of users who discontinue a service. In subscription-based organisations, such as telecom operators, insurance companies, and online SaaS platforms, churn reduction is directly related to maintaining revenue stability, reducing customer acquisition costs, and improving long-term profitability.
Predicting churn allows businesses to:

- Detect customers at risk before they leave,
- Personalise retention strategies,
- Allocate marketing resources more efficiently,
- Increase customer lifetime value (CLV).

In this session, we will work with the **Telco Customer Churn dataset**. The final goal is to build a churn prediction model powered by a **Large Language Model (LLM)** using supervised fine-tuning.

2 Dataset Loading

The dataset is provided as `telco_churn.csv`.

```
from datasets import load_dataset, DatasetDict

raw_dataset = load_dataset("csv", data_files="./data/datasets/telco_churn
    .csv")

full_dataset = raw_dataset["train"]

train_val = full_dataset.train_test_split(test_size=0.2, seed=2025)

dataset = DatasetDict({
    "train": train_val["train"],
    "test": train_val["test"]
})
```

3 Class Distribution and Stratification

We must inspect the `Churn` label distribution in both train and test sets.

```
from collections import Counter

def proportions(counter):
    total = sum(counter.values())
    return {k: f"{v} ({v/total:.3%})" for k, v in counter.items()}

train_counts = Counter(dataset["train"]["Churn"])
test_counts = Counter(dataset["test"]["Churn"])

print("Train:", proportions(train_counts))
print("Test:", proportions(test_counts))
```

After the computation, answer:

- Is the dataset balanced?
- Does changing the random seed significantly alter class proportions?
- Which strategies could address imbalance if needed?

4 LLM as the Required Predictive Model

The company's technical manager imposes the use of a **Large Language Model** for churn prediction. No justification is provided. The model must:

- take customer profile text as input,
- output exactly one word: `CHURN` or `NO_CHURN`.

The required pretrained model is located at:

`./data/models/Llama-3.2-3B-Instruct`

5 Data Transformation for LLM Fine-Tuning

LLMs expect text, not tabular vectors. Each row must therefore be **converted into a textual customer description**.

You must decide:

- A consistent textual prompt structure,
- The training target label,
- How to integrate this into supervised fine-tuning.

Tokenization must produce three elements that will be used during training:

- `input_ids`: the sequence of numerical token identifiers corresponding to the tokenized textual prompt. They represent the actual content that is given to the model as input.
- `attention_mask`: a binary mask used by the transformer to differentiate real tokens from padding tokens. A value of `1` indicates that the token is meaningful and should be attended to, whereas a value of `0` indicates padding that must be ignored in the attention mechanism.
- `labels`: these contain the target tokens that the model must learn to predict. They are usually identical to `input_ids`, except that all positions that should not contribute to the loss are replaced with `-100`. The value `-100` is a special ignored index recognized by the loss function so that no gradient is computed for those tokens.

Because we are performing causal language modelling, the model learns to predict the next token at each time step; therefore, only the answer portion must contribute to the loss.

```
# TODO: format dataset into instruction-style training data
#       create columns: input_ids, attention_mask, labels
```

6 Tokenizer and Model Loading

Reuse knowledge from TP1 to load the tokenizer and model.

```
# TODO: load tokenizer and quantized model
```

Gradient Checkpointing

The following lines are required when training quantized models:

```
from peft import prepare_model_for_kbit_training
prepare_model_for_kbit_training(model)
```

7 LoRA Adaptation

LoRA (Low-Rank Adaptation) injects lightweight trainable matrices into attention layers, allowing:

- fine-tuning with limited GPU memory,
- faster experimentation,
- preventing full model overwriting.

```
from peft import LoraConfig, get_peft_model

lora_config = LoraConfig(
    r=8,
    lora_alpha=16,
    lora_dropout=0.05,
```

```
    bias="none",
    target_modules=["q_proj", "v_proj"],
    task_type="CAUSAL_LM",
)

model = get_peft_model(model, lora_config)

model.print_trainable_parameters()
```

8 Training Setup

The recommended modules are:

- `TrainingArguments`
- `Trainer`
- `DataCollatorForLanguageModeling`

```
from transformers import TrainingArguments, Trainer,
   DataCollatorForLanguageModeling
# TODO: Implement the training pipeline
```

9 Evaluation and Interpretation

You must compute:

- Global accuracy
- Per-class accuracy
- Confusion matrix (numeric and visual)
- Precision, Recall, F1-score
- Other metrics that might be relevant

```
# TODO: implement inference loop and compute metrics
```

Answer in written form:

- Did the model learn meaningful decision patterns?
- Does it favour the majority class?
- Is an LLM justified over traditional models?