Hands-On: Text Classification With Transformers 

Nicolás Jesús Peretti
July 16, 2020

Data Scientist @ Córdoba, Arg.

https://www.linkedin.com/in/nicolasjesusperetti/

https://github.com/nicoperetti/metadata-sadosky-santander

Topics

- Problem
  - Definition
  - Dataset
  - Metric
- Baseline.
- Preprocessing.
- Beto ✌️
- Data Augmentation

Problem

Definition

Clasificación de preguntas de clientes

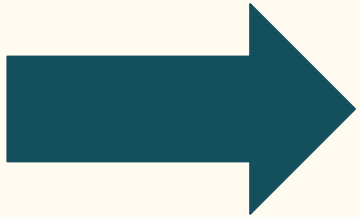
Santander nos propone un desafío basado en NLP, donde lo que se busca es entender las preguntas que hacen los clientes con el fin de ser más asertivos en las respuestas, esto es fundamental para brindar una mejor experiencia al usuario.

- Competition
  - hosted by: http://www.fundacionsadosky.org.ar/
  - link: https://metadata.fundacionsadosky.org.ar/competition/21/

Goal

Given a **question** we should return a **category**

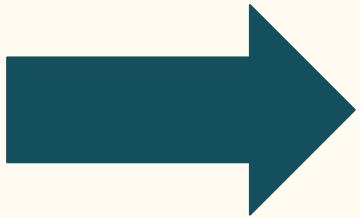
como hago para quitar un debito automatico?



cat_i

i, j $\in$ {1,..., N}

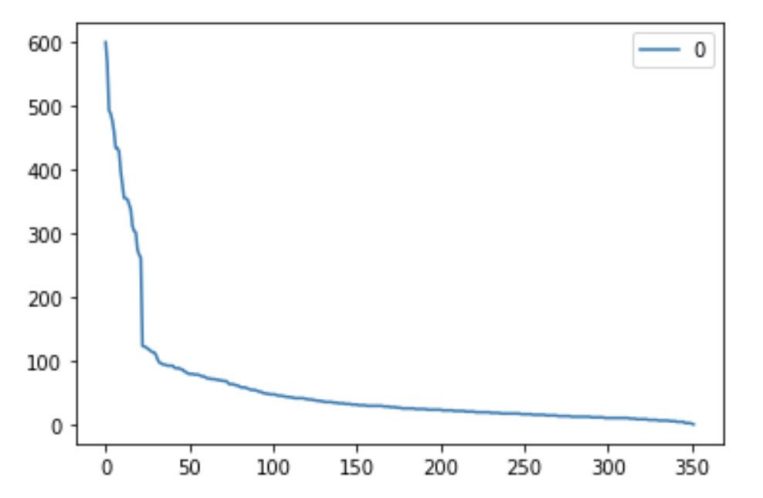
pedí una renovación de tarjeta hace un mes y todavía no me llegue nada.



cat_j

Dataset

- 350 Categories
- Distribution



- Size
  - Train: 20105 samples
  - Test: 6702 samples

Metric: Balanced Accuracy Score

- Binary Balanced Accuracy

$$\texttt{balanced-accuracy} = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right)$$

- From binary to multiclass metric:
  - Compute Recall for each class with the positive part $\frac{TP}{TP+FN}$
  - Average of each class.

[1] https://scikit-learn.org/stable/modules/model_evaluation.html#balanced-accuracy-score

Baseline (0 to 0.68)

Split + Text Vectorizer + TF-IDF

- Split Train Dataset into train and validation.

```
X = df.Pregunta
y = df.Intencion

X_train, X_test, y_train, y_test = train_test_split(df.Pregunta, df.Intencion, random_state = 13571113)
```

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((15078,), (5026,), (15078,), (5026,))
```

- Text Vectorization.

```
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(X_train)
```

- TF-IDF

```
tfidf_transformer = TfidfTransformer(sublinear_tf=True)
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
```

[1] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

[2] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html

Classifier

- Grid Search

```
parameters = {
    'kernel':['linear', 'rbf'],
    'C':[1, 10, 100, 1000],
    "class_weight": ["balanced", None]
}
svc = SVC()
clf = GridSearchCV(svc, parameters, n_jobs=-1, verbose=2)
```

```
clf.fit(X_train_tfidf, y_train)
```

```
clf.best_estimator_
```

```
SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='linear', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

Classifier: Fit & Predict

- Fit

```
clf = SVC(kernel="linear", C=10)
clf.fit(X_train_tfidf, y_train)
```

```
SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='linear', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

- Predict

```
X_test_counts = count_vect.transform(X_test)
X_test_tfidf = tfidf_transformer.transform(X_test_counts)
```

```
preds = clf.predict(X_test_tfidf)
```

- Balanced Accuracy:
  - valid: **0.669**
  - lb: **0.686**

Preprocessing (0.68 to 0.703)

Preprocessing


```
def preprocess_text(text, fix_unicode=True, lowercase=True,
                    no_urls=True, no_emails=True,
                    no_phone_numbers=True,
                    no_numbers=True, no_currency_symbols=True,
                    no_punct=True, no_accents=True):
    """Preprocess text."""
    clean_text = textacy_preprocess(text, fix_unicode=fix_unicode,
                                    lowercase=lowercase,
                                    no_urls=no_urls, no_emails=no_emails,
                                    no_phone_numbers=no_phone_numbers,
                                    no_numbers=no_numbers,
                                    no_currency_symbols=no_currency_symbols,
                                    no_punct=no_punct,
                                    no_accents=no_accents)
    return clean_text
```

- Balanced Accuracy:
  - valid: **0.674**
  - lb: **0.7031**

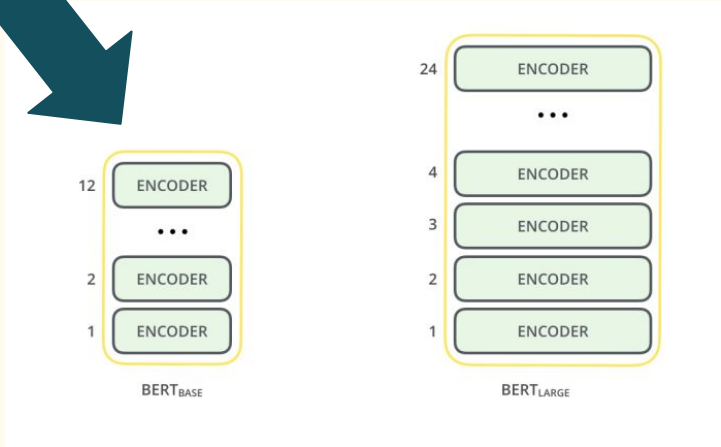
- Problem
  - Dataset
  - Metrics
- Baseline
- Preprocessing

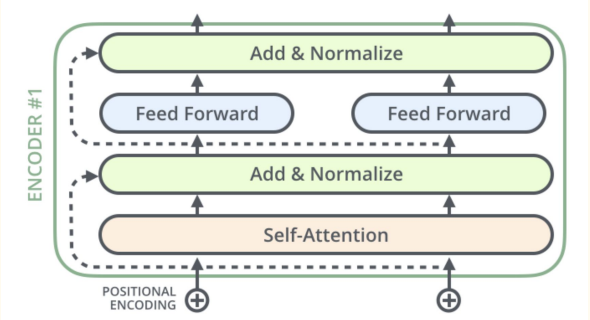
CHECK POINT

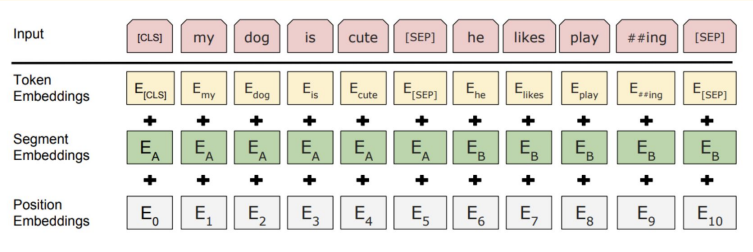
Beto ✌(0.70 to 0.81)

Beto ✌️

- BETO is a BERT copy trained on Spanish Corpus.[1][2]
- BERT: **B**idirectional **E**ncoder **R**epresentations from **T**ransformers.







Papers:

- BERT
- Attention

[1] https://github.com/josecannete/spanish-corpora
[2] https://github.com/dccuchile/beto

From Paper to Hugging Face

-  Data Preparation

```
encode_dict = {}

def encode_cat(x):
    if x not in encode_dict.keys():
        encode_dict[x] = len(encode_dict)
    return encode_dict[x]

df['ENCODE_CAT'] = df['Intencion'].apply(lambda x: encode_cat(x))
NB_CLASS = len(encode_dict)
```

Tokenizer

- Prepare data in BERT format.
  - Input IDs
    - tokens to ids
  - Attention mask
    - tokens to perform attention
    - binary
  - Token Type IDs
    - Segment token
    - binary
  - targets: category

```python
import torch
from transformers import AutoModelWithLMHead, AutoTokenizer


def load_tokenizer():
    tokenizer = AutoTokenizer.from_pretrained("dccuchile/bert-base-spanish-wwm-cased")
    return tokenizer
```

```python
class Triage(Dataset):
    def __init__(self, dataframe, tokenizer, max_len, mode="train"):
        self.len = len(dataframe)
        self.data = dataframe
        self.tokenizer = tokenizer
        self.max_len = max_len
        self.mode = mode

    def __getitem__(self, index):
        pregunta = str(self.data.Pregunta[index])
        pregunta = " ".join(pregunta.split())
        inputs = self.tokenizer.encode_plus(
            pregunta,
            None,
            add_special_tokens=True,
            max_length=self.max_len,
            truncation=True,
            pad_to_max_length=True,
            return_token_type_ids=True
        )
        ids = inputs['input_ids']
        mask = inputs['attention_mask']
        token_type_ids = inputs["token_type_ids"]

        d = {'ids': torch.tensor(ids, dtype=torch.long),
             'mask': torch.tensor(mask, dtype=torch.long),
             'token_type_ids': torch.tensor(token_type_ids, dtype=torch.long)}
        col_target = "id" if self.mode == "submit" else "ENCODE_CAT"
        d['targets'] = torch.tensor(self.data[col_target][index], dtype=torch.long)
        return d

    def __len__(self):
        return self.len
```

Data Loaders

```python
def get_loaders(df, tokenizer):
    training_loader, testing_loader = None, None

    train_index, test_index = train_test_split(list(df.index), random_state=13571113)

    # Creating the dataset and dataloader for the neural network
    train_dataset = df.iloc[train_index].reset_index().drop(columns="index")
    test_dataset = df.iloc[test_index].reset_index().drop(columns="index")

    print("FULL Dataset: {}".format(df.shape))
    print("TRAIN Dataset: {}".format(train_dataset.shape))
    print("TEST Dataset: {}".format(test_dataset.shape))

    training_set = Triage(train_dataset, tokenizer, MAX_LEN)
    train_params = {'batch_size': TRAIN_BATCH_SIZE,
                    'shuffle': True,
                    'num_workers': 0}
    training_loader = DataLoader(training_set, **train_params)

    testing_set = Triage(test_dataset, tokenizer, MAX_LEN)
    test_params = {'batch_size': VALID_BATCH_SIZE,
                    'shuffle': True,
                    'num_workers': 0}
    testing_loader = DataLoader(testing_set, **test_params)

    return training_loader, testing_loader
```

```python
MAX_LEN = 512
TRAIN_BATCH_SIZE = 16
VALID_BATCH_SIZE = 16
```

Model

```
class BERTClass(torch.nn.Module):
    def __init__(self, nb_class):
        super(BERTClass, self).__init__()
        self.l1 = AutoModelWithLMHead.from_pretrained("dccuchile/bert-base-spanish-wwm-cased").base_model
        self.l2 = torch.nn.Dropout(0.3)
        self.l3 = torch.nn.Linear(768, nb_class)

    def forward(self, ids, mask, token_type_ids):
        _, output_1 = self.l1(ids, attention_mask=mask, token_type_ids=token_type_ids)
        output_2 = self.l2(output_1)
        output = self.l3(output_2)
        return output
```

- base_model: drop classification layer
- **l1** return a tuple (last_hidden_layer, **pooler_output**)

Loss Function & Optimizer

```
tokenizer = load_tokenizer()

training_loader, testing_loader = get_loaders(df, tokenizer, validation)

device = 'cuda' if torch.cuda.is_available() else "cpu"
model = BERTClass(nb_class)
model.to(device)

loss_function = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params=model.parameters(), lr=LEARNING_RATE)
```

- CrossEntropyLoss: LogSoftmax + NLLLoss

$$\text{loss}(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right) = -x[class] + \log\left(\sum_j \exp(x[j])\right)$$

```
LEARNING_RATE = 1e-05
```


[1] https://pytorch.org/docs/master/generated/torch.nn.CrossEntropyLoss.html
[2] https://arxiv.org/pdf/1412.6980.pdf

Train

```python
for epoch in range(EPOCHS):
    model.train()
    for _, data in enumerate(training_loader, 0):
        ids = data['ids'].to(device, dtype=torch.long)
        mask = data['mask'].to(device, dtype=torch.long)
        token_type_ids = data['token_type_ids'].to(device, dtype=torch.long)
        targets = data['targets'].to(device, dtype=torch.long)

        outputs = model(ids, mask, token_type_ids)

        optimizer.zero_grad()
        loss = loss_function(outputs, targets)

        if _%1000 == 0:
            print(f'Epoch: {epoch}, Loss:  {loss.item()}')

        optimizer.zero_grad() # clear gradients
        loss.backward() # backpropagation
        optimizer.step() # updates parameters

    evaluate(model, device, training_loader)
    evaluate(model, device, testing_loader)
```

```
EPOCHS = 20
```

```
torch.save(model, output_model_file)
```

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 384.130                Driver Version: 384.130                   |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:00:1E.0 Off |                    0 |
| N/A   34C    P0    23W / 300W |      0MiB / 16152MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
```

Add Weight to loss function (0.81 to 0.825)

Loss Function

```
class_counter = Counter(df['ENCODE_CAT'])
weight_list = [1 / class_counter[i] for i in range(NB_CLASS)]
```

```
weights = torch.FloatTensor(weight_list).to(device, dtype=torch.float)
loss_function = torch.nn.CrossEntropyLoss(weight=weights)
```

- CrossEntropyLoss: LogSoftmax + NLLLoss

$$\text{loss}(x, class) = weight[class]\left(-x[class] + \log\left(\sum_j \exp(x[j])\right)\right)$$

Data Augmentation (0.825 to 0.846)

Back Translation

- Pivot
  - **EN**
  - Example:
- Googletrans

por que no puedo transferir usd por home banking?

¿Por qué no puedo transferir usd a través de la banca local?

[1] https://py-googletrans.readthedocs.io/en/latest/

Didn’t Work

Didn’t work

- Data Augmentation
  - Back Translation with: fr, pt, ar
  - Partial Back Translation: Populate minority classes.
  - Test Data Augmentation.

Ensemble (0.846 to 0.856)

Ensemble

- Calculate the mode between the output of several models

- Models:
  - CountVect + tf-idf + SVC → 0.68
  - Preprocess + CountVect + tf-idf + SVC → 0.71
  - Preprocess + Partial DA + weight loss + BETO → 0.829
  - Preprocess + DA + weight loss + BETO → 0.846

- Ensemble → **0.856**

Ensemble

- Why work?

```
Original signal:
1110110011

Encoded:
10,3 10101100111110110011111110110011

Decoding:
1010110011
1110110011
1110110011

Majority vote:
1110110011
```

- Correlated

```
1111111100 = 80% accuracy
1111111100 = 80% accuracy
1011111100 = 70% accuracy.
```

```
1111111100 = 80% accuracy
```

- Uncorrelated

```
1111111100 = 80% accuracy
0111011101 = 70% accuracy
1000101111 = 60% accuracy
```

```
1111111101 = 90% accuracy
```

[1] https://mlwave.com/kaggle-ensembling-guide/

- Problem
  - Dataset
  - Metrics
- Baseline
- Preprocessing
- BETO
- Weight Loss Function
- Data Augmentation
- Didn’t work
- Ensemble

CHECK POINT

Thanks!

Questions?