Classifying Labels for Dailysales

Task :

Implement a model for classifying around 144 labels of Daily Sales.

Approach:

1. Spacy textcat model using CNN architecture
2. Spacy transformer model
3. Using Fasttext embeddings
4. Xtreme Gradient boost algorithm(XGboost)

1. Spacy textcat model

Step1 : Prepare training data that supports spacy format

Tuples Format**:**

Each training example is a tuple containing the text data as the first element and the label(s) as the second element. If a text belongs to multiple categories, you can provide multiple labels.

Ex:- ("This is a positive review.", {"cats": {"POSITIVE": 1}})

Dictionary Format**:**

Each training example is a dictionary where the keys represent the text data and the values are dictionaries containing label(s) and their corresponding binary values (1 for present, 0 for absent).

Ex:- {"text": "This is a positive review.", "cats": {"POSITIVE": 1}}

In both formats:

- "text": Key containing the text data of the example.
- "cats": Key containing the dictionary of labels and their binary values.
- {"POSITIVE": 1}: The label "**POSITIVE**" with a binary value of 1 indicates the presence of the label**.**

Setting up the Environment:

1. **Install required libraries for spacy transformers:**

```
#Import all required libraries
!pip install -U spacy
!pip install spacy transformers
!pip install spacy-transformers

import spacy
import random
import time
import numpy as np
import pandas as pd
import re
import string


import sys
from spacy import displacy

from tqdm.auto import tqdm
from spacy.tokens import DocBin
```

2. **Pre-Processing the data:**

	text	label	label_id
0	early departure fee	early departure fees	0
1	rollaway bed rental	rollawayscribs	1
2	pet charge	pet fees	2
3	pet cleaning fee	pet fees	2
4	banquet breakfast	banquet food revenue breakfast	3

3. **Convert to Spacy format like above discussed.**

Ex:- {"text": "tw calif/local tourism f", "label": "rooms sales tax"}

4. **Split train and validation sets using sklearn library:**

```
from sklearn.model_selection import train_test_split

# Split the data into training and validation sets
train_data, valid_data = train_test_split('/content/drive/MyDrive/spacy_textcat/split_3.json', test_size=0.2, random_state=42)

# Example of how to access the training and validation data
# print("Training Data:")
# for text, label in train_data:
#     print(f"Text: {text}, Label: {label}")

# print("\nValidation Data:")
# for text, label in valid_data:
#     print(f"Text: {text}, Label: {label}")
```

5. **Convert train and valid sets to spacy binary format(.spacy)**

6. **Initialize configurations from the Spacy library**

```
✓ Auto-filled config with all values
✓ Saved config
/content/drive/MyDrive/spacy_textcat/textcat_config.cfg
You can now add your data and train your pipeline:
python -m spacy train textcat_config.cfg --paths.train ./train.spacy --paths.dev ./dev.spacy
```

Note: we can modify the hyperparameters in the configuration file depending upon the dataset and computational resources.

7. Debug the dataset and configurations using spacy debug tool

```
2024-01-10 05:57:37.250508: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register cuDNN factory: Attempting to register fac
2024-01-10 05:57:37.250565: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register cuFFT factory: Attempting to register factory f
2024-01-10 05:57:37.251942: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to register cuBLAS factory: Attempting to register factor
2024-01-10 05:57:38.287798: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT

============================ Data file validation ============================
config.json: 100% 481/481 [00:00<00:00, 2.83MB/s]
vocab.json: 100% 899k/899k [00:00<00:00, 3.54MB/s]
merges.txt: 100% 456k/456k [00:00<00:00, 2.51MB/s]
tokenizer.json: 100% 1.36M/1.36M [00:00<00:00, 4.64MB/s]
model.safetensors: 100% 499M/499M [00:05<00:00, 95.1MB/s]
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-base and are newly initialized: ['roberta.pooler.dense.weight', 'rober
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
✓ Pipeline can be initialized with data
✓ Corpus is loadable

============================== Training stats ==============================
Language: en
Training pipeline: transformer, ner
13734 training docs
3435 evaluation docs
⚠ 197 training examples also in evaluation data

============================== Vocab & Vectors ==============================
ℹ 6682424 total word(s) in the data (255270 unique)
ℹ No word vectors present in the package
```

8. Start Training after debug using the command:

!python -m spacy train textcat_config.cfg --verbose --output ./textcat_output --paths.train textcat_data/textcat_train.spacy –gpu -id 0

Note: Observe the trans_loss, textcat_loss, and evaluation metrics during training and change the hyperparameters accordingly for the better convergence of the model.

Inference:

Test the data with the test samples and iterate the process until we get the desired results of the model**.**

Results:

output


```
{
    "label_0": 0.6285714285714286,
    "label_1": 0.9999999999999999,
    "label_10": null,
    "label_11": 0.13782051282051283,
    "label_12": 0.9285714285714286,
    "label_13": null,
    "label_14": 1.0,
    "label_15": null,
    "label_16": 1.0,
    "label_17": null,
    "label_18": 0.28301886792452835,
    "label_19": null,
    "label_2": 1.0,
    "label_20": null,
    "label_21": 0.05660377358490565,
    "label_22": 0.7742718446601942,
    "label_23": 1.0,
    "label_24": 0.047619047619047616,
    "label_25": 1.0,
    "label_26": 0.8927835051546392,
    "label_27": 1.0,
    "label_28": 1.0,
    "label_29": null,
    "label_3": 0.9764705882352942,
    "label_30": 0.9805825242718447,
    "label_31": null,
    "label_32": null,
    "label_33": 0.9745098039215686,
    "label_34": 1.0,
    "label_35": 1.0,
    "label_36": null,
    "label_37": 0.9904761904761905,
    "label_38": 0.7216981132075472,
    "label_39": null,
    "label_4": null,
    "label_40": 1.0,
    "label_41": null,
    "label_42": 1.0,
    "label_43": 0.9999999999999999,
    "label_44": null,
    "label_45": null,
    "label_46": null,
    "label_47": null,
    "label_48": null,
```

"label_49": 0.9999999999999999,
"label_5": 0.2095238095238095,
"label_50": 1.0,
"label_51": 0.30188679245283023,
"label_52": null,
"label_53": null,
"label_54": null,
"label_55": null,
"label_56": null,
"label_57": null,
"label_58": 0.3584905660377359,
"label_59": null,
"label_6": null,
"label_60": null,
"label_61": null,
"label_62": null,
"label_63": null,
"label_64": null,
"label_65": null,
"label_66": 1.0,
"label_67": null,
"label_68": null,
"label_69": 0.5283018867924528,
"label_7": null,
"label_70": null,
"label_71": null,
"label_8": 0.5188679245283019,
"label_9": 0.9887820512820513

Conclusion:

I experimented with 530 sample datasets. So, we get null for some labels. It is possible to have null labels for some data points that are not present in every validation example.

2. Using fasttext embeddings

Task :

Implement a model for classifying around 144 labels of Daily Sales.

2nd approach: Using FastText embeddings

Step1 : Prepare training data that supports FastText format

FastText expects training data to be in a specific format where each line contains a label prefixed by __label__ followed by the text associated with that label.

Example:
__label__class1 This is a sample text for class 1
__label__class2 This is another sample text for class 2
…

	text	label	label_text
0	early departure fee	__label__"early departure fees"	__label__"early departure fees" early departur...
1	rollaway bed rental	__label__"rollawayscribs"	__label__"rollawayscribs" rollaway bed rental
2	pet charge	__label__"pet fees"	__label__"pet fees" pet charge
3	pet cleaning fee	__label__"pet fees"	__label__"pet fees" pet cleaning fee
4	banquet breakfast	__label__"banquet food revenue breakfast"	__label__"banquet food revenue breakfast" banq...

Step2: Train the Classifier

Supervised Training:

FastText provides a simple and efficient way to train supervised text classifiers. Supervised training involves training a classifier to predict a predefined label or category for each input text based on its features.

- Use the fasttext.train_supervised function to train the supervised classifier.
- Provide the path to the training data file and any additional parameters such as the number of epochs, learning rate, etc.

Unsupervised Training:

Unsupervised training in FastText involves learning word embeddings or representations from unlabelled text data. The model learns to represent each word in a continuous vector space based on its context in the input text.

- Use the fasttext.train_unsupervised function to train the unsupervised model.
- Provide the path to the unlabelled text data file and any additional parameters such as the dimension of the word vectors, window size, etc.
- **Word Embeddings**: After training, you can access the learned word embeddings or representations using the get_word_vector method. This method allows you to obtain the vector representation of any word in the vocabulary.
- **Similarity:**You can also measure the similarity between words based on their vector representations using the get_nearest_neighbors method. This method returns the nearest neighbors or most similar words to a given word based on their vector representations.

```
[ ]  model.get_nearest_neighbors("banquet")
```

```
[(0.9997298121452332, 'market'),
 (0.9985095858573914, 'beverage"'),
 (0.9979128241539001, 'cmn'),
 (0.9979128241539001, 'campaig'),
 (0.9974956512451172, 'bs'),
 (0.9973974823951721, 'breakfast"'),
 (0.9973533153533936, 'sundries'),
 (0.996333658695221, 'r1'),
 (0.9959368109703064, 'breakfast'),
 (0.994982898235321, 'break"')]
```

3. Evaluate the Classifier

```
model.test('/content/dstools.test')
```

```
(102, 0.9509803921568627, 0.9509803921568627)
```

```
model.predict("early departure fee")
```

```
(('__label__"early',), array([0.85142648]))
```

4. Save the Model

```
from gensim.models import FastText

# Assuming you have trained and used your FastText model for predictions
# `model` is your FastText model object

# Save the model to disk
model.save_model("fasttext_model.bin")

# Now you can load the model from disk later using:
# loaded_model = FastText.load_model("fasttext_model.bin")
```

Conclusion:

We got a median accuracy of 56% from fasttext embeddings. The fastText uses **n gram technique** that helps to provide information about sequences of consecutive words in addition to individual words. When we train the model with Multi-word labels, it is only showing the first word. This has to be fixed.

3. DistilBERT - Fine tuning

Step1 : Package Installation and Module Import:

- Install the required packages, including the transformer package.
- Import relevant modules such as the DistilBERT tokenizer, TFDistilBertForSequenceClassification, and TextClassificationPipeline.

Step2: Label Encoding:

- Encode the category labels using the `cat.codes` method, which assigns unique numerical labels to each category.
- The encoded labels represent the five unique categories present in the dataset.

Step3: Split train and validation sets using sklearn library:

Step4: Import the Distil-BERT pre-trained model from Hugging face:

```
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')

train_encodings = tokenizer(train_texts, truncation = True, padding = True  )

val_encodings = tokenizer(val_texts, truncation = True, padding = True )
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:88: UserWarning:

The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in y
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
```

```
tokenizer_config.json: 100% 28.0/28.0 [00:00<00:00, 479B/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 3.64MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 2.34MB/s]
config.json: 100% 483/483 [00:00<00:00, 12.2kB/s]
```

Step5: Specify the model Configurations

```
from transformers import TFDistilBertForSequenceClassification, TFTrainer, TFTrainingArguments


training_args = TFTrainingArguments(
    output_dir='/content/sample_data/results',
    num_train_epochs=50,
    per_device_train_batch_size=32,
    per_device_eval_batch_size=128,
    warmup_steps=500,
    weight_decay=1e-5,
    logging_dir='/content/sample_data/logs',
    eval_steps=500
)

with training_args.strategy.scope():
    trainer_model = TFDistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased', num_labels = 72 )

trainer = TFTrainer(
    model=trainer_model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
)
```


Step 6: Training

Conclusion:

Give **39%** accuracy over 8k training data and 2k validation data.(without oversampling)

3. Spacy textcat using Roberta-base transformers model

1. Data preparation
2. Initialize configurations
3. Debug using Spacy debug tool
4. Start Training

Training Pipeline:

```
============================= Training pipeline =============================
ℹ Pipeline: ['transformer', 'textcat']
ℹ Initial learn rate: 0.0
E    #       LOSS TRANS...  LOSS TEXTCAT  CATS_SCORE  SCORE
---  ------  -------------  ------------  ----------  ------
  0       0           0.00          0.01        0.00    0.00
 42     200          14.55          3.81        5.30    0.05
 84     400          71.87          1.41       18.84    0.19
126     600          63.33          0.18       19.04    0.19
169     800          59.32          0.14       18.35    0.18
211    1000          60.50          0.15       18.09    0.18
254    1200          59.61          0.14       18.44    0.18
296    1400          53.93          0.13       18.52    0.19
339    1600          46.08          0.13       18.51    0.19
382    1800          48.05          0.14       18.17    0.18
425    2000          52.40          0.16       18.44    0.18
468    2200          49.42          0.14       18.87    0.19
✓ Saved pipeline to output directory
/content/drive/MyDrive/text_classification/dailysales1/model-last
```

Results:
The overall accuracy stood at 0.19 because there is a mismatch in training and validation labels.

Dataset size : 8k
Validation set : 2k

Training Pipeline

```
============================ Initializing pipeline ============================
✓ Initialized pipeline

============================== Training pipeline ==============================
ℹ Pipeline: ['transformer', 'textcat']
ℹ Initial learn rate: 0.0
E    #       LOSS TRANS...  LOSS TEXTCAT  CATS_SCORE  SCORE
---  ------  -------------  ------------  ----------  ------
  0       0           0.00          0.01        0.00    0.00
  4     200          37.60          3.88       16.56    0.17
  9     400         323.22          1.56       35.74    0.36
 13     600         548.59          0.64       39.58    0.40
 18     800         645.50          0.59       40.44    0.40
 23    1000         733.51          0.56       40.88    0.41
 27    1200         733.61          0.53       41.48    0.41
 32    1400         766.56          0.55       41.33    0.41
 37    1600         818.46          0.53       40.48    0.40
 41    1800         765.18          0.51       40.30    0.40
 46    2000         804.92          0.51       40.87    0.41
 50    2200         789.63          0.53       41.81    0.42
```

Results:

The overall accuracy rose up to 42 percent. Some labels didn't get the accuracy because of that labels are not present in the validation set.

3. Spacy textcat using distilBERT transformers model

1. Data Preparation by balancing the classes using data oversampling
2. Initialize base configuration of the distlBERT model
3. Debugging the configurations and data contains 74 labels
4. Training the 1k(8k by oversampling) data
5. Got accuracy **0.90**

Evaluation Report:

Got **53.40%** over Unseen data and **93%** over trained data from 10k data dstools file.

HyperParameter Tuning:

In the [components.textcat.model.linear_model] **section:**

- **Changed** ngram_size **from** 1 **to** 2**.**

In the [components.transformer.model] **section:**

- **Changed** name **from** "distilbert-base-uncased" **to** "bert-base-uncased"**.**

In the [training] **section:**

- **Changed** accumulate_gradient **from** 3 **to** 5**.**
- **Changed** dropout **from** 0.1 **to** 0.2**.**
- **Changed** patience **from** 1600 **to** 2000**.**
- **Changed** max_steps **from** 20000 **to** 25000**.**
- **Changed** eval_frequency **from** 200 **to** 500**.**
- **Changed** L2 **from** 0.01 **to** 0.001**.**
- **Changed** warmup_steps **from** 250 **to** 500**.**
- **Changed** total_steps **from** 20000 **to** 25000**.**
- **Changed** initial_rate **from** 0.00005 **to** 0.00003**.**

Explanation:

Text Classification Model (Linear Model):

- Changing ngram_size from 1 to 2: Increasing the ngram_size **to 2 allows the model to consider pairs of adjacent words (bigrams) during feature extraction. This can capture more complex patterns in the text data, potentially improving the model's performance.**

Transformer Model:

- **Changed** name from "distilbert-base-uncased" to "bert-base-uncased": BERT (Bidirectional Encoder Representations from Transformers) is a more powerful pre-trained transformer model compared to DistilBERT. Using BERT as the base model may lead to better performance due to its larger size and capacity for capturing more nuanced relationships in the data.

Training Parameters:

- Changed accumulate_gradient from 3 to 5: Increasing the accumulate_gradient **parameter allows for larger effective batch sizes during training, which can lead to more stable training and potentially better generalization.**
- **Changed** dropout from 0.1 to 0.2: Increasing dropout regularization helps prevent overfitting by randomly dropping a larger proportion of units (in this case, 20%) during training.
- Changed patience from 1600 to 2000: Increasing the patience **parameter for early stopping allows the model to train for a longer period before halting training if there is no improvement in validation performance. This can help the model converge to a better solution.**
- **Changed** max_steps from 20000 to 25000: Increasing the maximum number of training steps allows the model to train for a longer duration, potentially allowing it to converge to a better solution.
- Changed eval_frequency from 200 to 500: Decreasing the evaluation frequency reduces the frequency of model evaluation during training,

which can speed up training while still providing regular feedback on performance.

- Changed L2 from 0.01 to 0.001: Decreasing the L2 regularization strength helps prevent overfitting by reducing the penalty on large weights in the model, allowing for smoother optimization.
- Changed warmup_steps from 250 to 500: Increasing the number of warm-up steps in the learning rate schedule allows the model's learning rate to increase more gradually at the beginning of training, potentially improving convergence.
- Changed total_steps from 20000 to 25000: Increasing the total number of training steps allows the model to train for a longer duration, potentially allowing it to converge to a better solution.
- Changed initial_rate from 0.00005 to 0.00003: Decreasing the initial learning rate can help stabilize training and prevent the model from diverging early on.

After HyperParameter tuning:

Model stood at **91% accuracy as** before.

4. Xgboost Classifier

1. Import libraries

```
##import libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

2. Load and preprocess the data

```
df = pd.read_csv('Desktop/data_999.csv')
```

```
df.columns
df
```

	text	label
0	AR Ledger Payments	Payments Received Towards City Ledger
1	Discount - D	transient discount
2	Industry Programs - V	transient retail
3	Food And Beverage Revenue	Food And Beverage Revenue
4	Other Revenue	miscellaneous income
...	...	...
8403	wine	banquet beverage revenue wine

3. Label Encoding

```
#Label Encoding
from sklearn.preprocessing import LabelEncoder
LE = LabelEncoder()
Y = LE.fit_transform(Y)
```

4. TF-IDF vectorization

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Convert NumPy array to list of strings
X_train_list = X_train.tolist()

# Ensure each element is a string
X_train_list = [str(x) for x in X_train_list]

# Initialize TF-IDF vectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=10000)  # Adjust max_features as needed

# Fit and transform the text data
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train_list)

# Convert to dense array if needed
X_train_tfidf = X_train_tfidf.toarray()
```

5. Train_Test_Split
6. Install xgboost Classifier
7. Fit model to train data

```
import xgboost as xgb
xgb_model = xgb.XGBClassifier(random_state = 0)
xgb_model.fit(X_train_tfidf, Y_train)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, objective='multi:softprob', ...)
```

8. Confusion matrix

```
from sklearn.metrics import confusion_matrix
confusion_matrix(Y_test, yp)
```

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

9. Results

```
from sklearn.metrics import accuracy_score
accuracy_score(Y_test, yp)*100
```

```
0.7728894173602854
```

Conclusion:

We got **77% accuracy** using the TF-IDF vectorizer. But, I hope it will increase when training with BERT embeddings.