Case Study 3

Tobias Chin, Ryan Dalton, Jacob Ruppert, Rohan Prasad, and Bishoy Soliman Hanna

Introduction:

From the GitHub Scikit-Learn repo, we used Cornell's v2.0 polarity dataset on movie reviews, which is separated into positive and negative reviews.

The reviews were stored a text documents, below is an example:

```
plot : two teen couples go to a church party , drink and then
drive .
they get into an accident .
one of the guys dies , but his girlfriend continues to see him
in her life , and has nightmares .
what's the deal ?
watch the movie and " sorta " find out . . .
critique : a mind-fuck movie for the teen generation that
touches on a very cool idea , but presents it in a very bad
package .
which is what makes this review an even harder one to write ,
since i generally applaud films which attempt to break the
mold , mess with your head and such ( lost highway & memento ) ,
but there are good and bad ways of making all types of films ,
and these folks just didn't snag this one correctly .
they seem to have taken this pretty neat concept , but executed
it terribly .
so what are the problems with the movie ?
well , its main problem is that it's simply too jumbled .
it starts off " normal " but then downshifts into this " fantasy
" world in which you , as an audience member , have no idea
what's going on .
```

Data Analysis:

Part 1 - Grid Search:

In this case study we worked with the movie reviews in Cornell's v2.0 polarity dataset, using this data we initially pulled 2000 reviews and set 1500 as training data and 500 as testing data (see output below).

```
n_samples: 2000
Number of documents in training data: 1500
Number of documents in test data: 500
Number of labels in training data: 1500
Number of labels in test data: 500
```

Before producing any insights using this data, it required filtering to remove excessively rare or excessively frequent tokens using a tokenizer.

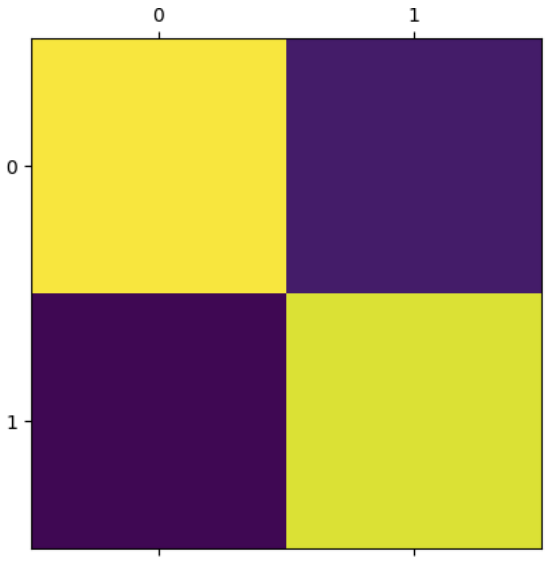
```
0 params - {'vect__ngram_range': (1, 1)}; mean - 0.83; std - 0.01
1 params - {'vect__ngram_range': (1, 2)}; mean - 0.85; std - 0.01
```

To predict whether the testing data were positive or negative reviews, we created a grid search. Grid search is a method of machine learning that works using cross-validation of various parameters based on training data to predict testing data.

	precision	recall	f1-score	support
neg	0.88	0.84	0.86	261
pos	0.83	0.87	0.85	239
accuracy			0.86	500
macro avg	0.86	0.86	0.86	500
weighted avg	0.86	0.86	0.86	500

The grid search method predicted 261 negative reviews and 239 positive reviews.

We sorted these predictions using a Confusion Method, a 2x2 table in which all values are sorted by if they're predicted to be positive or negative and if that prediction was accurate or not.



True Positive	False Positive
False Negative	True Negative

We correctly predicted 219 reviews to be positive and 209 reviews to be negative, but we incorrectly predicted 42 reviews as positive and 30 reviews as negative.

219	**42**
30	**209**

Part 2 - Vectorizers:

TfidfVectorizer is a class in the scikit-learn library that pulls all the words used in a document, weighs them based on importance, find's individual term frequency, and weighs down certain terms like "of" or "the".

Running TfidfVectorizer on our 1500 training documents produced **35,199** unique terms across the training documents.

Once we added the max_df and min_df (metrics to check if a number appears too often to be considered or not often enough to be considered) we reduced this number of terms to **15,916**. We used a min_df of 3 terms per document and a max_df of 0.95 which checked if the term appeared in 95% of documents.

Next, we added an ngram range of (1,2) which considers both individual and pairs of words as separate terms which increased our number of terms to **63,076.**

Part 3 - Machine Learning:

Using the metrics determined in problem 2 we computed new training documents.

```
TfidfVectorizer
TfidfVectorizer(max_df=0.95, min_df=3, ngram_range=(1, 2))
```

The first ML method we used was LinearSVC. Linear Support Vector Classification is best suited for linearly separable datasets, it works by finding a plane to separate the various classes.

With an accuracy of 0.852, LinearSVC predicted 261 positive reviews and 239 negative reviews.

LinearSVC Accuracy: 0.852

	precision	recall	f1-score	support
0	0.88	0.84	0.85	261
1	0.83	0.87	0.85	239
accuracy			0.85	500
macro avg	0.85	0.85	0.85	500
weighted avg	0.85	0.85	0.85	500

The second ML method we used was the KNeighborsClassifier. This method is an instance-based algorithm that works by predicting individual data points based on a majority vote of its k nearest neighboring data points.

With an accuracy of 0.628, KNeighborsClassifier predicted 261 positive reviews and 239 negative reviews.

KNeighborsClassifier Accuracy: 0.628

	precision	recall	f1-score	support
0	0.69	0.52	0.59	261
1	0.59	0.74	0.66	239
accuracy			0.63	500
macro avg	0.64	0.63	0.63	500
weighted avg	0.64	0.63	0.62	500

The extent to which KNeighbors struggled was surprising and worth investigating.

```
Examples of misclassified documents by KNeighborsClassifier:
Document Index: 1
Document Text:
 b' " mandingo " has traditionally been seen as one of two things : either a much-needed revisionist look at slavery in the south ,
True label: Negative
Predicted label: Positive
--------------------------------------------------------------------------------
Document Index: 6
Document Text:
 b'the above is dialogue from this film , taken almost completely in context , and not jazzed up a bit to make it more inept than i
True label: Negative
Predicted label: Positive
--------------------------------------------------------------------------------
```

Some examples of misclassified articles can be seen above. The classifier likely inaccurately predicted these due to a lack of overtly negative/positive words such as ‘bad’ and ‘great’ and nostalgic tone appearing positive despite belonging to a negative review.

Part 4 - Pre-Trained Hugging Face Models:

The first model we investigated was the Adam Codd distilbert base uncased finetuned sentiment amazon, this model was trained using amazon reviews.

The validation performance of the model was:

Loss: 0.116

Accuracy: 0.961

F1_score: 0.960

The test performance on the v2.0 polarity dataset was:

Loss: 0.3342873454093933

Accuracy: 0.87

F1_score: 0.8771266540642723

Why was the test performance worse than the validation performance? The validation of the model was performed on amazon reviews not movie reviews; while they were similar enough that the model performed comparatively well on the movie dataset, it wasn't optimal.

The second model we investigated was the [JamesH Movie review sentiment analysis model](#).

Hugging Face contains thousands of pre-trained models, the James H. model was very applicable to the data we were attempting to predict.

The validation metrics reflect the performance of the model on previous datasets. We decided to fine-tune this model instead of the previous model as it was already built to perform sentiment analysis on movie reviews instead of amazon reviews.

Model Trained Using AutoTrain

- Problem type: Binary Classification
- Model ID: 1883864250
- CO2 Emissions (in grams): 6.9919

Validation Metrics

- Loss: 0.175
- Accuracy: 0.950
- Precision: 0.950
- Recall: 0.950
- AUC: 0.986
- F1: 0.950

The test performance on the v2.0 polarity dataset was:

Loss: 0.1482

Accuracy: 0.9500

F1 Score: 0.9482

Part 5 - Fine-Tuning:

Using the Hugging Face model as base we can train a new model specifically for our dataset. Similar to the vectorizers used previously we'll be determining parameters to improve the performance of the model.

```
training_args = TrainingArguments(output_dir="finetuned_movie_sentiment",
                                  use_cpu=False,
                                  evaluation_strategy="epoch",
                                  learning_rate=3e-5,
                                  per_device_train_batch_size=4,
                                  per_device_eval_batch_size=4,
                                  num_train_epochs=3,
                                  weight_decay=0.01,
                                  warmup_steps=500,
                                  max_grad_norm=1.0,
                                  logging_dir='./logs',
                                  logging_steps=50,
                                  save_strategy="epoch",
                                  lr_scheduler_type="linear",
                                  adam_epsilon=1e-8,
                                  seed=42,
                                  gradient_accumulation_steps=4,
                                  fp16=True
                                  )
```

Shown above is the parameters we used to fine-tune the Hugging Face model, some of the most important parameters are: a learning rate of $3e^{-5}$, a weight decay of 0.01, a max gradient norm of 1.0, adam epsilon of $1e^{-8}$, and 3 training epochs.

All these hyperparameters represent different methods adjusting the models performance:

output_dir: Specifies the directory where outputs such as model checkpoints will be saved. This is where your trained model's state and configuration are stored.

use_cpu: Determines whether to use the CPU for training. Setting this to False lets the model use a GPU if available, which is typically much faster.

evaluation_strategy: Defines when the evaluation of the model should be performed. "epoch" means the evaluation will occur after every training epoch.

learning_rate: The rate at which the model learns. A smaller learning rate means slower learning, but too large a learning rate can cause the model to converge too quickly to a suboptimal solution, or not converge at all.

per_device_train_batch_size: Number of training examples used in one iteration for each device (like a GPU). Smaller batch sizes often mean more updates and potentially smoother convergence but can take longer to train.

per_device_eval_batch_size: Number of examples used in one iteration during model evaluation. This does not impact the training process but can affect the speed and memory usage during model evaluation.

num_train_epochs: The number of times the training data will be passed through the model. More epochs can lead to a better fit but also increase the risk of overfitting.

weight_decay: This adds a regularization term to the loss function to prevent the model from fitting the training data too closely (overfitting).

warmup_steps: Number of steps to increase the learning rate from 0 to the initially set learning rate. This gradual increase can help improve model stability and performance.

max_grad_norm: Used for clipping gradients to prevent them from getting too large, which can cause training instability or a poor model.

logging_dir: Specifies the directory where training logs will be stored.

logging_steps: Determines how often to log training information. For example, 50 means log information every 50 steps.

save_strategy: Defines when to save the model checkpoints. "epoch" means the model is saved at the end of each epoch.

lr_scheduler_type: The type of learning rate scheduler to use. "linear" means the learning rate decreases linearly from the initial lr set.

adam_epsilon: A small constant for numerical stability in the Adam optimizer. This helps prevent division by zero in some cases.

seed: Sets a seed for the random number generator. This is used for reproducibility of results.

gradient_accumulation_steps: Allows you to effectively train with larger batch sizes. The gradients are accumulated for the specified number of steps before a backward pass is done.

fp16: Indicates whether to use 16-bit floating-point precision (as opposed to 32-bit) for training. This can reduce memory usage and speed up training but might lead to precision issues in some models.

For each of the Epochs the model produced different training losses, validation losses, and accuracies.

Epoch	Training Loss	Validation Loss	Accuracy
1	0.134500	0.181348	0.955000
2	0.098900	0.245246	0.955000
3	0.060800	0.240997	0.955000

The performance on the v2.0 polarity dataset was:

Loss: 0.2410

Accuracy: 0.9550

F1 Score: 0.9554

Accuracy isn't the only factor to consider when modeling the dataset, the runtime is also incredibly important.

Baseline (Sci-Kit Learn SVC) | 2.27 ms ± 339 µs

Pre-trained Hugging Face | 11.2 s ± 389 ms

Fine-tuned Hugging Face | 11.2 s ± 265 ms

SVC is significantly faster, this speed is expected for traditional machine learning models.

The Hugging Face models were much slower which is expected from a notably more complex model like a deep-learning model. Even using GPU's the Hugging Face models were still slower.

Part 6 - Takeaways:

The traditional machine learning models such as SVC were far faster and demanded less processing power than deep-learning models.

The best methods for sentiment prediction were generally the Hugging Face models as they are especially well-suited to recognizing contextual nuances of text datasets.

Between the pre-trained Hugging face model and the custom fine-tuned model: fine-tuned models are typically more accurate.

	Grid Search	LinearSVC	KNearest-Neighbor	HuggingFace Pre-Trained	HuggingFace Fine-Tuned
Accuracy	0.86	0.85	0.63	0.950	0.955

Of all the methods used, KNearestNeighbor struggled the most due to issues with recognizing unclear language in articles.

	HuggingFace Pre-Trained	HuggingFace Fine-Tuned
Loss	0.1482	0.2410
F1	0.9482	0.9554

As predicted the fine-tuned model's accuracy improved as the adjusted hyperparameters better suited the model to our dataset.

Business Model:

Suppose you had a machine learning algorithm that could detect the sentiment of articles that was highly accurate. What kind of business could you build around that?

Public Relations is a significant aspect of many companies but identifying and recognizing shifts and trends towards certain businesses can be challenging. In 2021 alone the PR industry encompassed ~6.6 billion dollars in spending. If we could create a business that specializes in recognizing public perception and sentiments toward a company we could act as an alternative to outsourcing to large PR agencies. With a highly accurate ML algorithm, we could identify public sentiments toward certain products and marketing campaigns. We can integrate News API to offer highly accurate sentiment analysis of news articles.

Who would be your competitors, and what are their sizes?

Our competitors would be large global communications firms and public relation agencies like Edelman or MSL, these are large global brands that generate hundreds of millions of dollars in gross revenue each year. A major advantage of these firms over in-house PR representatives is their greater access to resources but these agents are often too expensive for medium/small businesses. If we could contract our services to bolster the capabilities of in-house PR departments we could fill the gap between global PR brands and internal representatives.

What would be the size of the market for your product?

As previously mentioned the PR industry is a multi-billion dollar industry and PR specifically plays an important role in almost every major industry.

In addition, assume that your machine learning was slow to train, but fast in making predictions on new data. How would that affect your business plan?

Slow model training and fast data prediction would have affect our business plan in various ways:

- Training Speed: The slow training speed may require careful planning and resource management during the training phase. This could involve optimizing

algorithms, utilizing parallelization, and using cloud resources for more efficient training.
- Prediction Speed: The fast prediction on new data is advantageous, especially for real-time applications. This would enable the platform to provide up-to-the-minute insights on breaking news and emerging trends.

To minimize the issues associated with slow training we could initially reuse the model for different businesses and instead fine-tune it for individual companies.

How could you use the cloud to support your product?

We could use cloud computing to reduce the cost associated with utilizing the processing power to run the algorithm. We could also use remote data storage over the cloud to more efficiently store large data sets of articles. If we made our business remote to reduce the cost of our business to below that of our competitors we would also benefit from global access to our resources.

8. Citations:

"News API – Search News and Blog Articles on the Web." News API: Search News and Blog Articles on the Web, News API – Search News and Blog Articles on the Web. Accessed 1 Nov. 2023.

OpenAI. (2023). ChatGPT (November 6 version) [Large language model]. https://chat.openai.com

MongoDB. (2023). Home. https://www.mongodb.com/

"PR Statistics 2023." TrueList, 7 Jan. 2023, PR Statistics 2023 - TrueList.