amazon

MMA865 Final Project

Team Kipling

01
Problem
&
Objective

The Business Problem and Objective

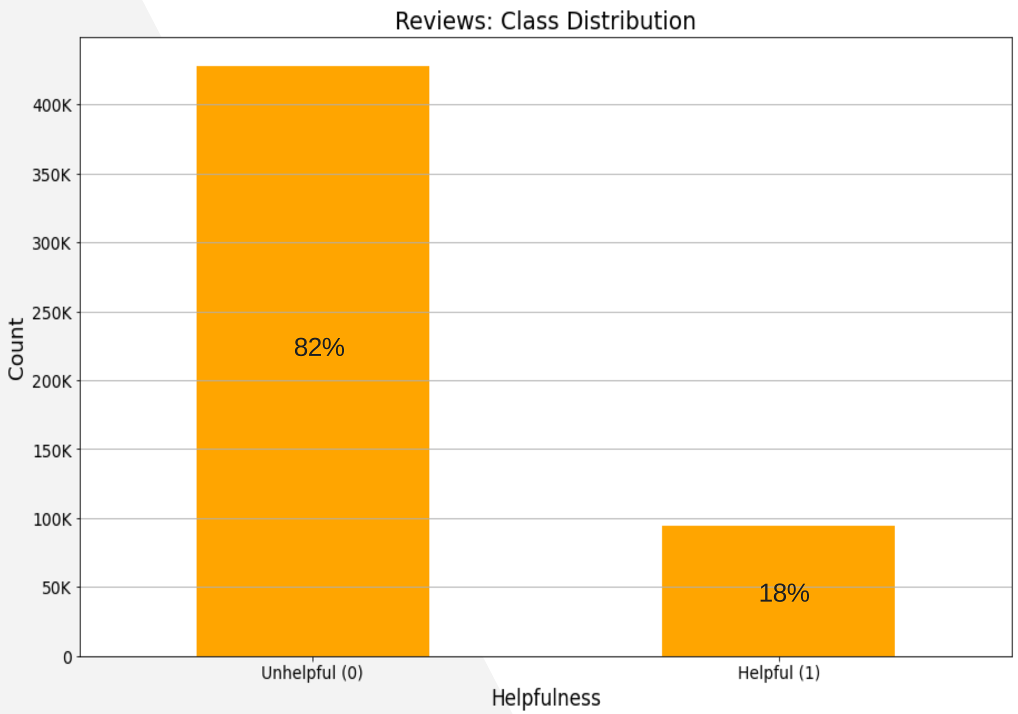
- **Business Problem**: According to research[1], 91 percent of people regularly or occasionally read online reviews, and 84 percent trust online reviews as much as personal recommendations. Amazon's customers rely on them to ensure that they’re getting what they’re paying for. However, not all reviews are legitimate or add value to their decision-making process.

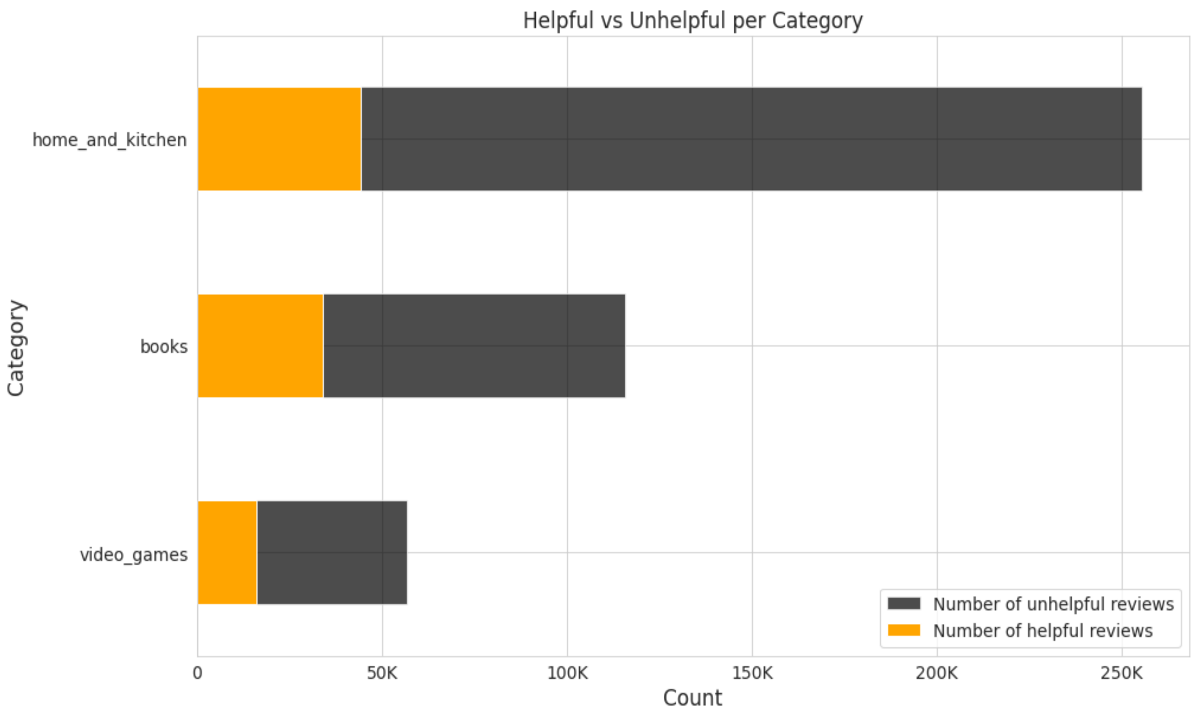
- **Business Objective:** How can Amazon accurately identify and leverage helpful reviews to improve customer experience and maintain customer loyalty?

1- Source: Brightlocal Local Consumer Review Survey 2020
https://www.brightlocal.com/research/local-consumer-review-survey/?SSAID=314743&SSCID=a1k5_3jwyh

02
Exploratory Data Analysis

EDA – Target Class Split





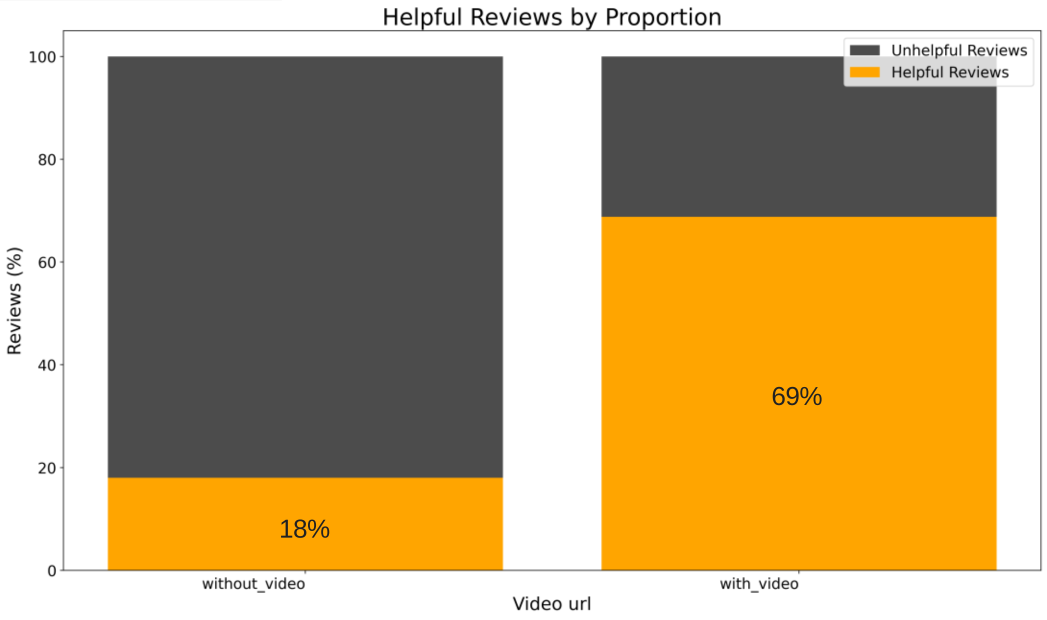
- Highly imbalanced - 18% of total reviews were helpful
- By category, 14%, 22.7%, and 22.0% from home and kitchen, books and videogames reviews were voted helpful by users respectively

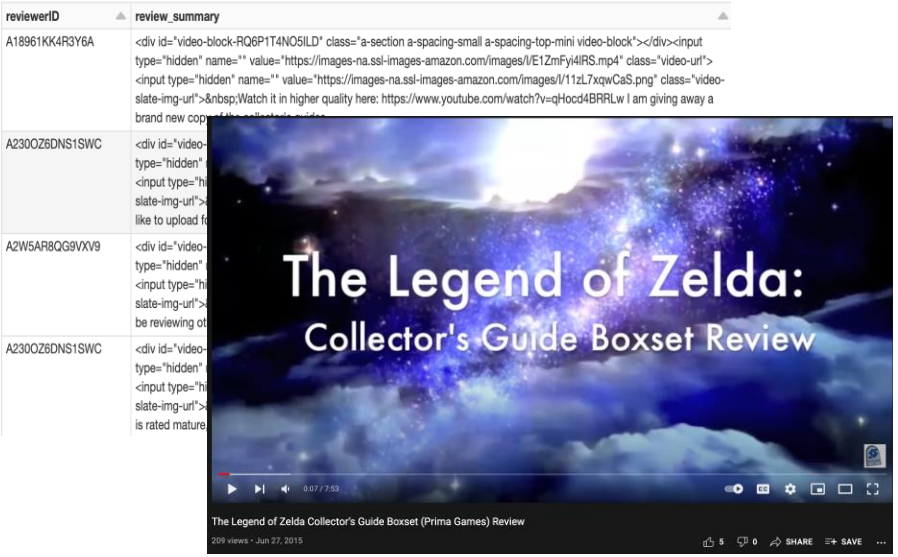
EDA –Video Reviews

Video Url Flag	Count	%
1	1,876	0.05%
0	3,485,455	99.95%

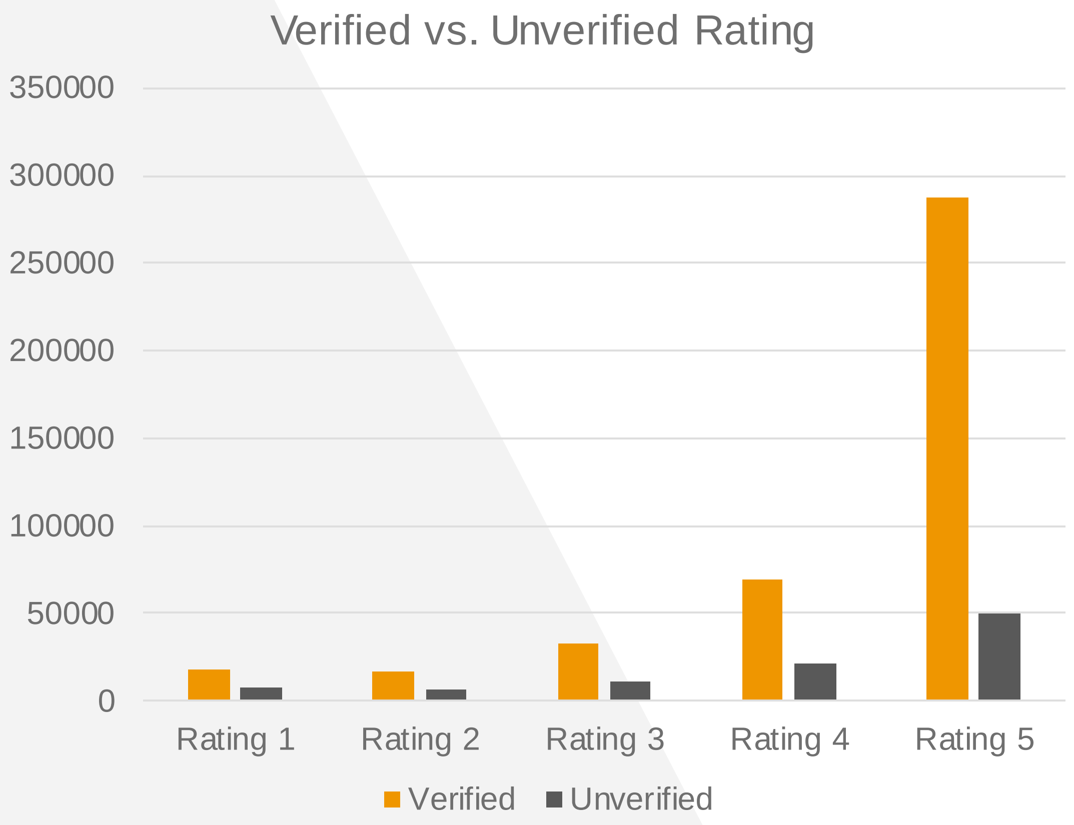
Insights:

- About 0.05% of reviews contained video links
- About 70% of the reviews with videos were helpful
- Videos and actual images of products from unbiased users can help other users to make purchase decisions





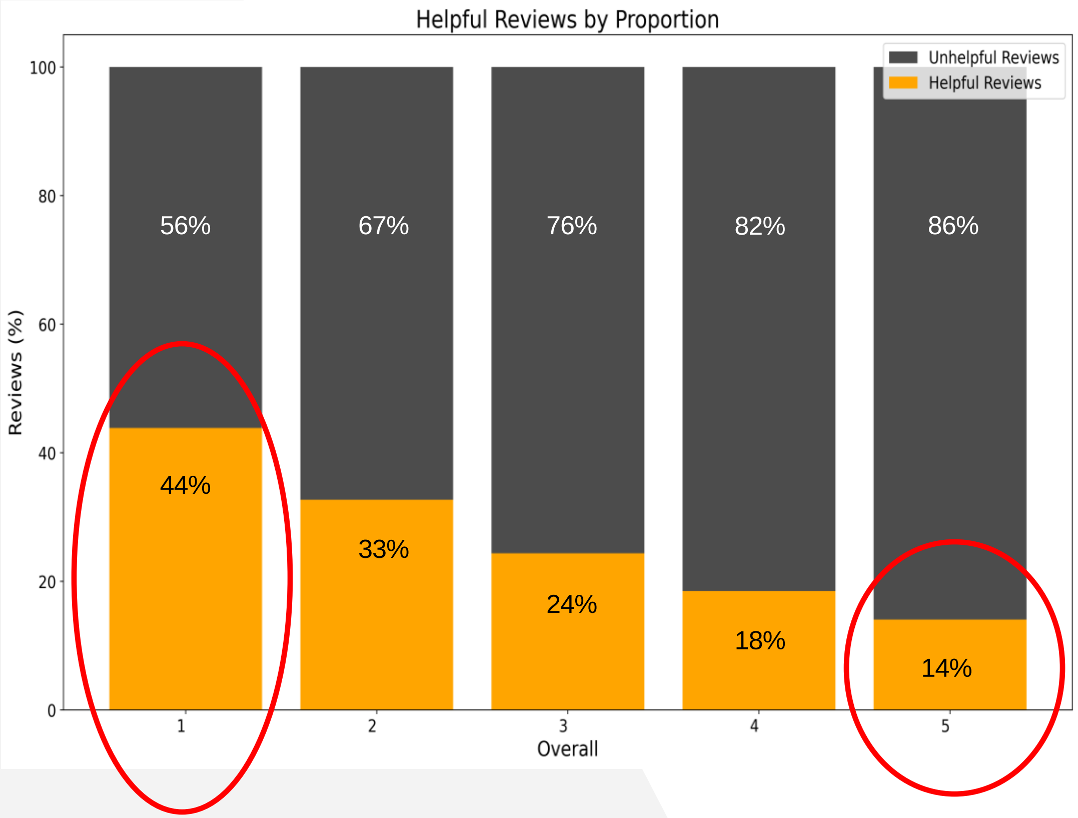
EDA – Unverified Reviews



Insights:

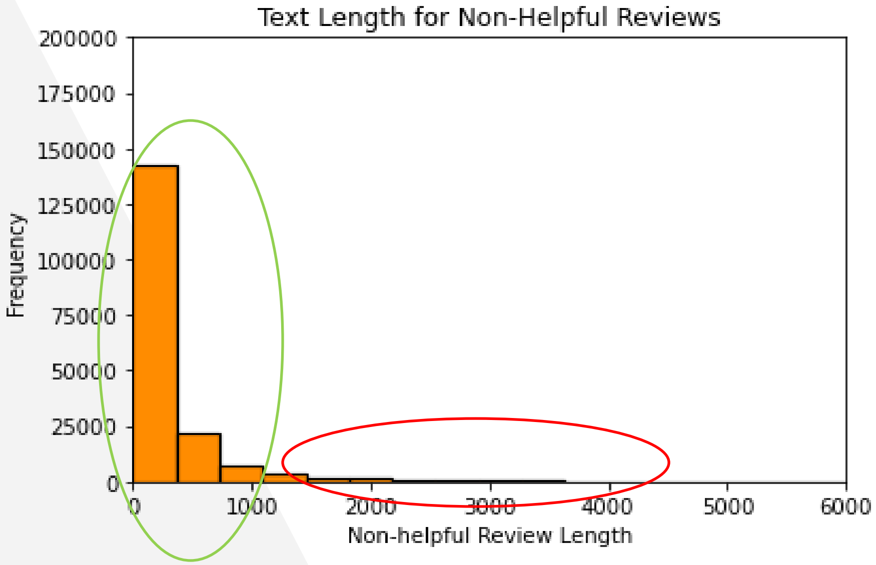
- Most accounts tend to give 5 stars to ratings
- Unverified accounts tend to review 5 stars the most: participative reviewers or bots?
- Sellers need to double think of their CRM strategy to deal with non-buyer reviewers; amazon needs to handle review bots at the same time

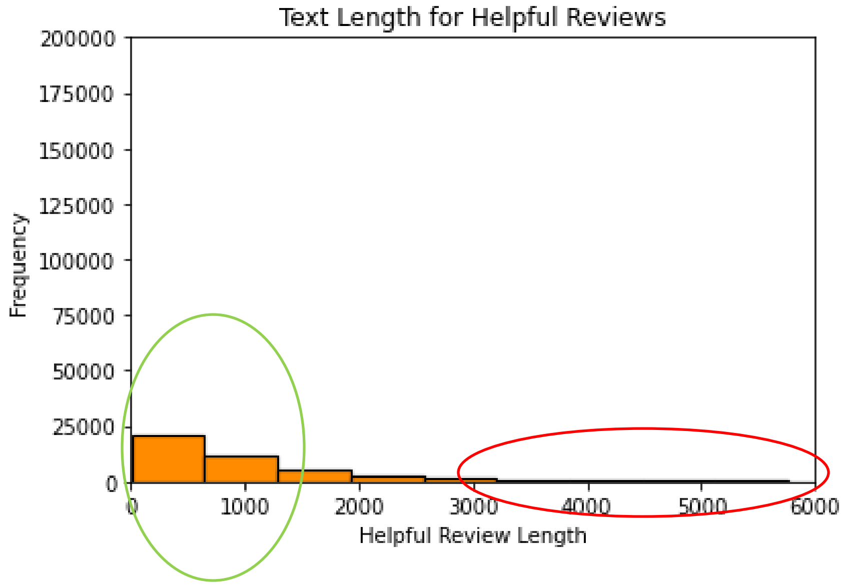
EDA – Helpful reviews per rating



- On the scale 1 to 5, the lower the rating, the higher the % of helpful reviews
- Potential skewness as users tend to write more about their negative experience when they are dissatisfied
- Higher ratings tend to have the least % of helpful reviews as users tend to spend less time writing reviews when their experience is positive

EDA-Text Length of Helpful Reviews



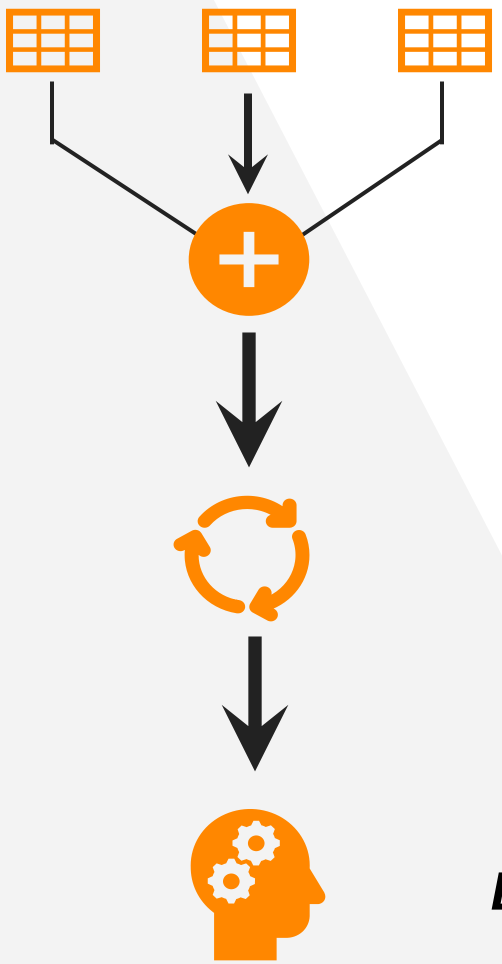


- On average, helpful reviews have higher text length both for review & summary
- Reviews that are not voted as 'helpful' are concentrated within the 1,000 characters in terms of text length

Label	Average Text Length
1	1,047
0	298

03
Modelling Journey

Data Cleaning and Feature Engineering



Data Integration

- *Sourced datafiles from the server*
- *Merged three files*

Data Cleaning

- *Removed rows with < 10 characters in "reviewText"*
- *Merged text from "reviewText" and "summary"*
- *Removed special characters, weblink and numbers*
- *Converted text to lowercase*
- *Removed non-English instances*

Feature Engineering

Introduced following new features:

- *# of characters in review text*
- *# of words in review text*
- *Flag for review with more than 1000 characters*
- *Flag for reviews with video and image URLs*

Machine Learning Pipeline

*Used **20%** data due to <u>Computational Challenges</u> !!!*

Review Text + Summary

Document Assembler

Universal Sentence Encoder (pretrained)

Feature Transformers Pre-Processing Pipeline

Embedding Finisher
Output as vector

One-Hot Encoding (*Verified* column)

Vector Assembler

Combine features from Encoder + Other variables

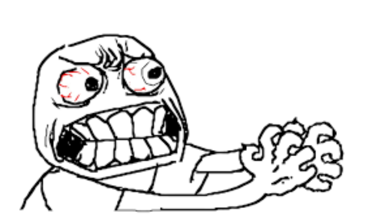
Weights to correct <u>class-imbalance</u>

Logistic Regression

Cross-Validation (n=5 folds)

Classification Model

AUC (test): 0.78



Transformers

- BERT
- XL Net
- ELMo

Annotators

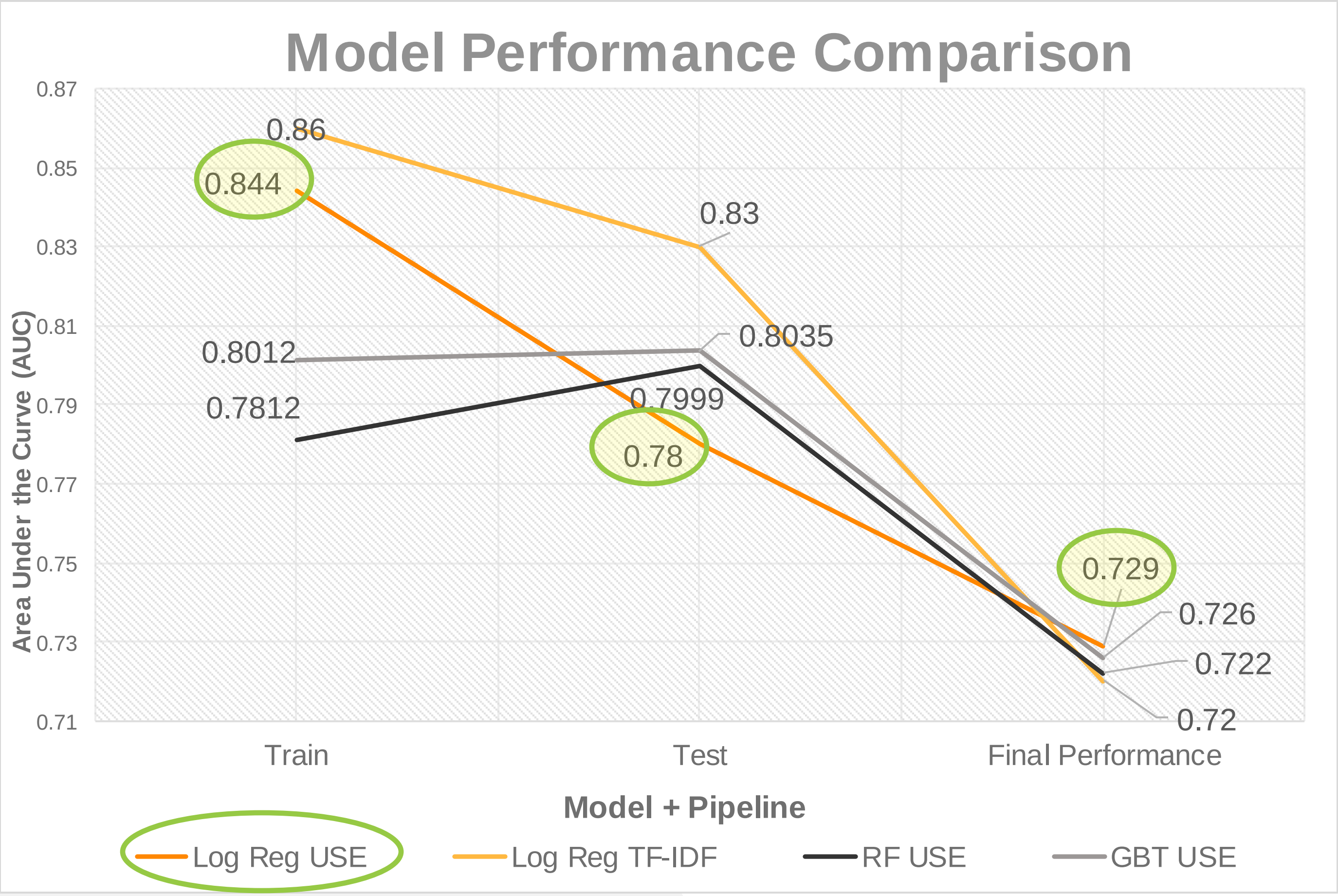
- ClassifierDL
- NorvigSweeting Spellchecker
- Sentiment Detector

Estimator

- RandomForest
- Gradient Boost
- Factorization Machines

Modelling Process – Model + Pipeline Results Comparisons

Model + Pipeline Comparison



Top Performing Model

- Logistic Regression with Universal Sentence Encoder Pipeline

Key Observations

- Significant overfitting of all models

Modelling Process – Best Model Performance

Classification Report

Class	Precision	Recall	F1-Score	Support
Non-Helpful (0)	0.87	0.96	0.91	57,152
Helpful (1)	0.64	0.34	0.45	12,579
---------------	--------------	-----------	-------------	-------------
Accuracy			0.85	69,731
Macro Avg	0.75	0.65	0.68	69,731
Weighted Avg (Micro)	0.83	0.85	0.83	69,731

Confusion Matrix

	Predicted Class	
Truth Class	**54,726**	2,426
	8,265	**4,314**

Current Kaggle Score/Position

- AUC: 0.7299
- Position: 13

04
Project Cost Estimate

Project Cost Estimate

Compute Costs

- Price/Hour = $1.173
- Hours used = 300
- Total Cost
- = **$351.90**

People Costs

- Data Scientist: $120k | 20hrs
- Data Engineer: $100k | 30hrs
- Analyst: $80k | 80hrs
- Total Wages = **$5,673**

Infrastructure Costs

- Azure Data Lake Storage
- 5GB Data
- $0.2304/GB
- Total Cost = **$1.15**

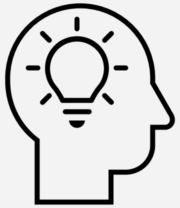
Grand Total Cost: $6,026.13

05
Conclusion & Next Steps

Conclusion & Next Steps

In Summary

- Helpful reviews often come from unhappy customers
- Length of reviews matter!
- Best Model: Logistic Regression with USE



What's Next?

- Main goal: Improve customer experience
- How? A/B test model to determine effectiveness

Appendix

Feature Engineering

```
#FE
import pyspark.sql.functions as f
#Adding in length of reviewText & Summary as features

df = df.withColumn('rt_length', f.length('reviewText'))
df = df.withColumn('sum_length', f.length('summary'))

df.show(10)
```

Appendix

Stratified Sampling

```
#Now, lets scale this down to 10% of our total data to train our algorithms
#Take a stratified sample

df = df.sampleBy('label', fractions={0:0.1, 1:0.1}, seed=47)

display(df.groupBy('label').count())
```

(2) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [reviewID: long, overall: double ... 13 more fields]

	label	count
1	1	62746
2	0	286301

Appendix

Feature Engineering

```
#Adding in rt_length indicator

df = df.withColumn('rt_length_ind', f.when(f.col('rt_length') >= 1000, 1).otherwise(0))

df.show(10)

#df.withColumn('D', f.when(f.col('B') > 0, "Yes").otherwise("No")).show()
```

Appendix

Feature Engineering

```
#Count total number of sentences
def countWordsInEachSentences(array):
    return [len(x.split()) for x in array]

countWordsSentences = f.udf(lambda x: countWordsInEachSentences(x.split('. ')))

df = df.withColumn("word_count", countWordsSentences(df['reviewText']))

df.show(10)
```

Appendix

Language Detection Pipeline

```
#Checking reviewText for non english before splitting
import sparknlp
from sparknlp.annotator import *
from sparknlp.common import *
from sparknlp.base import *
from sparknlp.pretrained import PretrainedPipeline

model_name = 'ld_wiki_tatoeba_cnn_375'

documentAssembler = DocumentAssembler()\
                    .setInputCol("reviewText")\
                    .setOutputCol("document")

sentence_detector = SentenceDetector() \
    .setInputCols(["document"]) \
    .setOutputCol("sentence")

languageDetector = LanguageDetectorDL.pretrained(model_name)\
      .setInputCols("sentence")\
      .setOutputCol("language")\
      .setThreshold(0.5)\
      .setCoalesceSentences(True)

nlpPipeline = Pipeline(stages=[ documentAssembler,
                               sentence_detector,
                                 languageDetector
                                 ])


langpipelineModel = nlpPipeline.fit(df)
```

Appendix

Universal Sentence Encoder Pipeline 1/2

```
#Downloading & Initializing the Unviversal Sentence Encoder + Importing relevant libraries
import sparknlp
from sparknlp.annotator import *
from sparknlp.common import *
from sparknlp.base import *
from pyspark.ml import Pipeline
from pyspark.ml.feature import CountVectorizer, HashingTF, IDF, OneHotEncoder, StringIndexer, VectorAssembler, SQLTransformer
from pyspark.sql.functions import udf, explode
import pandas as pd


useEmbeddings = UniversalSentenceEncoder.pretrained()\
      .setInputCols("document")\
      .setOutputCol("use_embeddings")
```

Appendix

Universal Sentence Encoder Pipeline 2/2

```
#Creating the pipeline
document_assembler = DocumentAssembler() \
      .setInputCol("reviewText") \
      .setOutputCol("document")

loaded_useEmbeddings = UniversalSentenceEncoder.load('/root/cache_pretrained/tfhub_use_en_2.4.0_2.4_1587136330099')\
      .setInputCols("document")\
      .setOutputCol("use_embeddings")

embeddings_finisher = EmbeddingsFinisher() \
      .setInputCols(["use_embeddings"]) \
      .setOutputCols(["finished_use_embeddings"]) \
      .setOutputAsVector(True)\
      .setCleanAnnotations(False)
```

Appendix

- Extracting Universal Sentence Encoder Embeddings

```
#Viewing embedding output
use_df_train.select('finished_use_embeddings').show(3)
```

(1) Spark Jobs

```
----------------------+
nished_use_embeddings|
----------------------+
 [[-0.056037165224...|
 [[-0.005085167475...|
 [[-0.045702368021...|
----------------------+
y showing top 3 rows

mand took 16.38 seconds -- by 20mg46@queensu.ca at 9/30/2021, 4:01:21 PM on MMA2022W-Kipling
```

```
#Adding in a features column based on embeddings
use_df_train= use_df_train.withColumn("fue", explode(use_df_train.finished_use_embeddings))

#use_df_train = use_df_train.withColumn('xln', explode(use_df_train.finished_xlnet_embeddings))
```

Appendix

Pipeline 2

```
#Now create new pipeline to encode verified and pass fue, overall and verified together into one column called "features"
#Now OHE the Verified feature and add in the overall feature as well

verified_indexer = StringIndexer(inputCol=('verified'), outputCol=('verified_indexer'))

ohe_verified_indexer = OneHotEncoder(inputCol=('verified_indexer'), outputCol=('verified_indexer_vec'))

#overall_indexer = StringIndexer(inputCol=('overall'), outputCol=('verified_overall'))

#ohe = OneHotEncoder()\
#   .setInputCols(['verified'])\
#   .setOutputCols(['verified_0'])


#assembler = VectorAssembler(inputCols=['fue', 'verified_indexer_vec', 'overall', 'rt_length', 'sum_length', 'rt_length_ind'], outputCol='features')
assembler = VectorAssembler(inputCols=['fue', 'verified_indexer_vec', 'overall', 'rt_length', 'rt_length_ind'], outputCol='features')

pre_proc_pipeline = Pipeline(stages=[verified_indexer, ohe_verified_indexer, assembler])

pre_proc_model = pre_proc_pipeline.fit(use_df_train)
```

Appendix

Logistic Regression Model

```
#Log Reg TV
import mlflow
import mlflow.spark
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, auc
import pandas as pd
from pyspark.ml.classification import LogisticRegression, GBTClassifier
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.tuning import CrossValidator, TrainValidationSplit, ParamGridBuilder

#Define Model
lr = LogisticRegression(labelCol='label', featuresCol='features')

#Define Evaluator
evaluator = BinaryClassificationEvaluator(labelCol='label', metricName='areaUnderROC')

lr_paramGrid = ParamGridBuilder() \
  .addGrid(lr.regParam, [0.0, 0.5, 0.7]) \
  .addGrid(lr.maxIter, [10,25,50,100]) \
  .build()

lr_tvs = TrainValidationSplit(estimator=lr, estimatorParamMaps=lr_paramGrid, evaluator=evaluator,
    parallelism=4, seed=42)
lr_grid_model = lr_tvs.fit(train_preprocessed)
```

Appendix

Logistic Regression Model 2

```
#Logistic Regression
import mlflow
import mlflow.spark
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, auc
import pandas as pd
from pyspark.ml.classification import LogisticRegression, GBTClassifier
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.tuning import CrossValidator, TrainValidationSplit, ParamGridBuilder

#Let's start off with a Logistic Regression model
lr = LogisticRegression(elasticNetParam=0.0)

#Define our evaluator
evaluator = BinaryClassificationEvaluator(labelCol='label', metricName='areaUnderROC')

#Define a hyperparameter search space
lr_paramGrid = ParamGridBuilder() \
  .addGrid(lr.regParam, [0.0, 0.5, 0.7]) \
  .addGrid(lr.maxIter, [10,25,50,100]) \
  .build()


lr_grid = CrossValidator(estimator=lr,
                         estimatorParamMaps=lr_paramGrid,
                         evaluator=evaluator,
                         numFolds=5)


# Run cross-validation, and choose the best set of parameters.
with mlflow.start_run():
  lr_grid_model = lr_grid.fit(train_preprocessed) #Running cv on the training dataset; will return the best model it found

  #Evaluate the best model's performance on the test set and log the results
  #test_metric = evaluator.evaluate(lr_grid_model.transform(use_df_test))
  #mlflow.log_metric('test_' + evaluator.getMetricName(), test_metric)

  #Log the best model
  mlflow.spark.log_model(spark_model=lr_grid_model.bestModel, artifact_path='best-model')
```

Appendix

Random Forest Model

```
#RF TV
import mlflow
import mlflow.spark
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, auc
import pandas as pd
from pyspark.ml.classification import LogisticRegression, GBTClassifier, RandomForestClassifier
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.tuning import CrossValidator, TrainValidationSplit, ParamGridBuilder

#Define Model
rf = RandomForestClassifier(labelCol='label', featuresCol='features')

#Define Evaluator
evaluator = BinaryClassificationEvaluator(labelCol='label', metricName='areaUnderROC')

rf_paramGrid = ParamGridBuilder() \
  .addGrid(rf.maxBins, [10,20,30]) \
  .addGrid(rf.maxDepth, [4,6,8]) \
  .addGrid(rf.impurity, ['gini', 'entropy'])\
  .build()


rf_tvs = TrainValidationSplit(estimator=rf, estimatorParamMaps=rf_paramGrid, evaluator=evaluator,
    parallelism=4, seed=42)
rf_grid_model = rf_tvs.fit(train_preprocessed)
```

Appendix

Gradient Boost Tree Classifier

```
#Gradient Boosted Tree Classifier
import mlflow
import mlflow.spark
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, auc
import pandas as pd
from pyspark.ml.classification import LogisticRegression, GBTClassifier
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.tuning import CrossValidator, TrainValidationSplit, ParamGridBuilder

#Define Model
gbt = GBTClassifier(labelCol='label', featuresCol='features')

#Define Evaluator
evaluator = BinaryClassificationEvaluator(labelCol='label', metricName='areaUnderROC')

gbt_paramGrid = ParamGridBuilder() \
  .addGrid(gbt.maxDepth, [3,5,10]) \
  .addGrid(gbt.maxBins, [2,12,24,32]) \
  .build()

gbt_tvs = TrainValidationSplit(estimator=gbt, estimatorParamMaps=gbt_paramGrid, evaluator=evaluator,
    parallelism=4, seed=42)
gbt_model = gbt_tvs.fit(train_preprocessed)
```

Appendix

Compute Cost Calculations

Azure VM								
RAM	vCPUs	DBU Count	VM Price/CPU	VM Price	DBU Price	Total Price/Hour	Total Hours Used	Total Cost
28GB	4	1	0.1173	$ 0.4690	$ 0.7040	$ 1.173	300	$ 351.90

Appendix

Employee Wage Cost Calculations

Employees				
Title	**Pay**	**Hours Used**	**Hourly Salary**	**Total Cost**
Data Scientist	$120,000	20	$ 57.69	$ 1,153.85
Data Engineer	$100,000	30	$ 48.08	$ 1,442.31
Analyst	$ 80,000	80	$ 38.46	$ 3,076.92
				$ 5,673.08

Appendix

- Infrastructure Cost Calculations

Infrastructure		
GBs Used	Price	Total
5	$ 0.2304	$ 1.15