STEP 1: DATA COLLECTION AND TEXT PREPROCESSING

Data is collected from Steam's official website from May 14th to May 25th using the code in Figure 1(Figure 1 is in next page). The total number of reviews collected over this time period is 1719.The code for Step-1 is in spark_streaming_saving_example.ipynb. For step 2, code is in predictive_modelling.ipynb. For step 3, code is in livestream.ipynb.

Of the files collected during live streaming, the only relevant files containing game reviews were titled "part-0000". Other files with names and extensions such as "._SUCCESS.crc","_SUCCESS" and ".part-00000.crc" are not relevant for the text classification task. Hence, these files were removed by converting all the file names for files located under the root subdirectory to string format and then isolating for only those files starting with the letter "p" and not ending with the .crc extension. Figure 2 is a snippet of the code used to obtain the relevant files(Figure 2 in next page).

The obtained text reviews were then converted to Pandas dataframe and cleaned for natural language processing as well as machine learning. The data was then converted to csv and saved to local machine. The data was read in as csv and was converted to Spark dataframe for further data manipulation and predictive modelling. Before writing to csv, the is.ascii() method was used on the Pandas dataframe to ensure that reviews written in alphabets other than English were removed.After the data was loaded as csv and converted to Spark dataframe, some rows had the value "null" or "." either for the reviews column or the label column. These rows were dropped along with rows with NA values. Punctuation marks such as ",","!", etc. and digits were removed using regex pattern. The rest of the data manipulation was carried out within the context of predictive modelling(Step-2).

STEP 2: PREDICTIVE MODELLING

Further data preprocessing for predictive modellling included tokenization by Regex Tokenizer followed by removal of stop words by StopWordsRemover. The label column for the reviews was of string type and so it was cast into integer type for natural language processing.Feature extraction was carried out by three methods: by using CountVectorizer, Word2Vec and Tf-Idf. The models utilizing the input from these methods included logistic regression, naive bayes classifier, decision tree classifier, random forest classifier, linear support vector classifier, gradient boosting trees and One-vs-Rest classifier(with logistic regression as base classifier). A pipeline was constructed with the following stages for models using Count Vectorizer or Word2Vec:
tokenization− >stop words removal− >CountVectorizer− >model.
In the case of HashingTF, the pipeline was:
tokenization− >stop words removal− >hashingTF− >idf− >model

An example of this pipeline and its implementation is given in the context of logistic regression and Count Vectorizer in Figure-3.
The evaluation criteria used for these different models were: AUC, f1 score,recall, precision and confusion matrix. The results from fitting the different models are given in Figure-4.The confusion matrices for the three feature extraction methods for the different models are in Figures-5 to 7. Code used to obtain the evaluation results are in Figure-8.

With Count Vectorizer, logistic regression had the highest accuracy(0.9086), high precision for true positives(0.95) as well as reasonably good AUC in comparison to other models considered. High precision for true positives is incredibly useful for classifying Steam reviews as majority of the reviews collected in Steam have true label of being positive reviews(1277 of the initial 1719 reviews were positive). Other methods with good accuracy and precision are linear SVC, naive Bayes and Gradient Boosted Trees. However, decision trees,gradient boosted trees and linear SVC have much lower f1 scores for true negatives than logistic regression.In the case of Word2Vec model, logistic regression performs much better than other models overall in terms of accuracy,AUC,f1-score,precision and recall. The same pattern holds for TfIdf. Hence, logistic regression is chosen for classification of live streamed reviews.

```
import threading

# Helper thread to avoid the Spark StreamingContext from blocking Jupyter

class StreamingThread(threading.Thread):
    def __init__(self, ssc):
        super().__init__()
        self.ssc = ssc
    def run(self):
        self.ssc.start()
        self.ssc.awaitTermination()
    def stop(self):
        print('----- Stopping... this may take a few seconds -----')
        self.ssc.stop(stopSparkContext=False, stopGraceFully=True)
```

```
sc
```

SparkContext

Spark UI

Version
`v3.3.2`

Master
`local[*]`

AppName
`PySparkShell`

```
from pyspark.streaming import StreamingContext
```

```
ssc = StreamingContext(sc,1)
```

```
lines = ssc.socketTextStream("seppe.net", 7778)
```

```
# Change the path below to match yours
lines.saveAsTextFiles("file:///C:/Users/RimJhim/Desktop/spark_instructions/")
```

```
ssc_t = StreamingThread(ssc)
ssc_t.start()
```

```
# Don't run this cell unless you want to stop
ssc_t.stop()
```

```
----- Stopping... this may take a few seconds -----
```

Figure 1: Streaming data from Steam and saving to local machine

STEP-3-LIVE STREAMING

In order to conduct live streaming, the chosen models needed to be saved and loaded in the context of the function designed to live stream Steam reviews. However, the code for saving the chosen models returned multiple py4j errors during the predictive modelling stage. Therefore, the full data cleaning and modelling codes used in the predictive modelling stage were used in the function that would ideally only have loaded the saved model and applied it to the streamed data. Code for streaming,cleaning and applying model for logistic regression and Word2Vec is given in Figure-9. TfIdf was also used and prediction using TfIdf can be found in the relevant Jupyter notebook. The accompanying predictions on livestreamed reviews can be found in Figure-10.

```
import pandas as pd
os.chdir("D:/spark")
rootPath="D:/spark"
dirpath="D:/spark"
import json
d=None

for root,dirs,files in os.walk(rootPath):
    for file in os.scandir(root):
        if (str(file).find('part')>1) and (str(file).find('crc')==-1):
            df2=pd.DataFrame(pd.read_json(file,lines=True))
            d=pd.concat([d,df2])
```

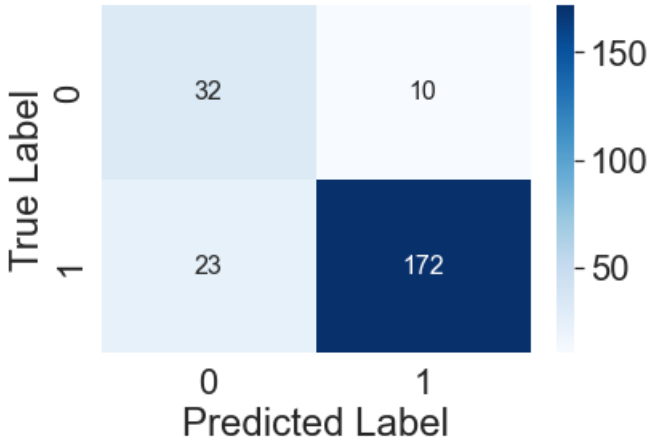
Figure 2: Extraction of relevant files for text classification

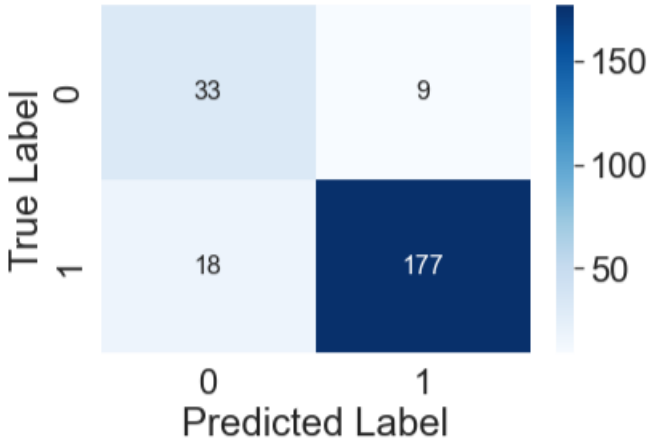
```
###PIPELINE-countvectorizer

##set up pipeline
from pyspark.ml import Pipeline
token=RegexTokenizer(inputCol="text", outputCol="text2", pattern="\\W")
remove= StopWordsRemover(inputCol=token.getOutputCol(), outputCol="text3")
count= CountVectorizer(inputCol="text3", outputCol="features")

model=LogisticRegression(maxIter=10, regParam=0.001)
##putting above:tokenizing,stop words removal, count vectorizing and model in a pipeline
pipeline = Pipeline(stages=[token,remove,count,model])
```

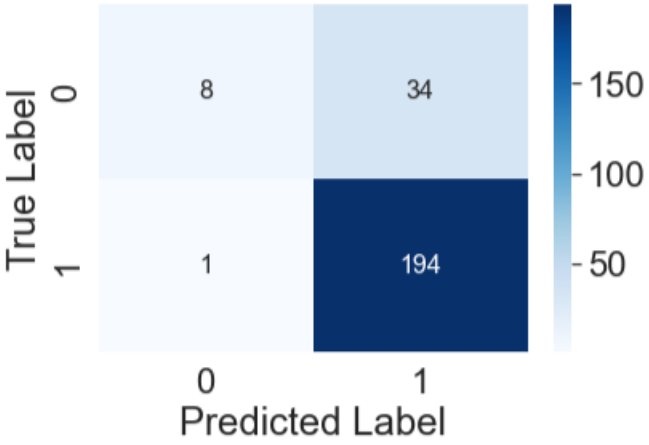
Figure 3: Pipeline for logistic regression-Count Vectorizer

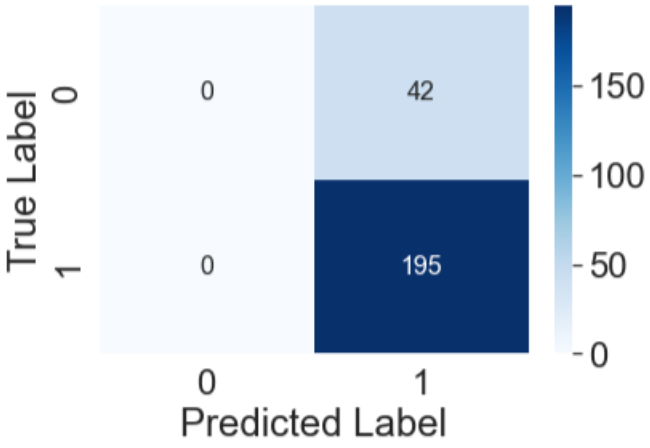
Feature Extraction/	Precision		Recall		f1-score		AUC	Accuracy
Evaluation	(0)	(1)	(0)	(1)	(0)	(1)		
Count Vectorizer								
Logistic Regression	0.58	0.95	0.76	0.88	0.66	0.91	0.868	0.9086
naive Bayes	0.65	0.95	0.79	0.91	0.71	0.93	0.9086	0.8902
Decision Trees	0.89	0.85	0.19	0.99	0.31	0.92	0.8103	0.6045
Random Forest	0	0.82	0	1	0	0.9	0.7711	0.7428
Gradient Boosted Trees	0.67	0.86	0.24	0.97	0.35	0.91	0.7978	0.812
Linear SVC	0.53	0.92	0.67	0.87	0.59	0.9	0.8983	0.8425
Word2Vec								
Logistic Regression	0.62	0.85	0.19	0.97	0.29	0.91	0.7797	0.7978
Decision Trees	0.32	0.83	0.14	0.93	0.2	0.88	0.5531	0.760
Random Forest	0	0.82	0	1	0	0.9	0.6748	0.743
Gradient Boosted Trees	0.15	0.82	0.05	0.94	0.07	0.88	0.6449	0.736
Linear SVC	0	0.82	0	1	0	0.9	0.63999	0.743
Tfidf								
Logistic Regression	0.47	0.96	0.86	0.79	0.61	0.87	0.8234	0.8017
naive Bayes	1	0.84	0.1	1	0.17	0.91	0.5476	0.8397
Decision Trees	0.89	0.85	0.19	0.99	0.31	0.92	0.5927	0.8523
Random Forest	0	0.82	0	1	0	0.9	0.5	0.8228
Gradient Boosted Trees	0.91	0.86	0.24	0.99	0.38	0.92	0.616	0.8608
Linear SVC	0.54	0.93	0.69	0.87	0.6	0.9	0.781	0.8397

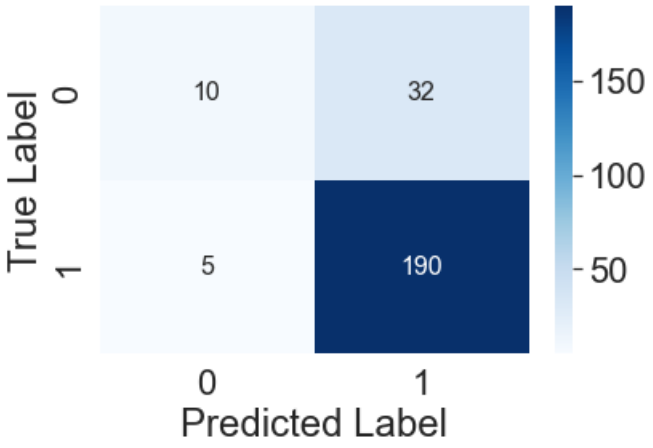
Figure 4: Evaluation criteria and results for different models and feature extraction methods

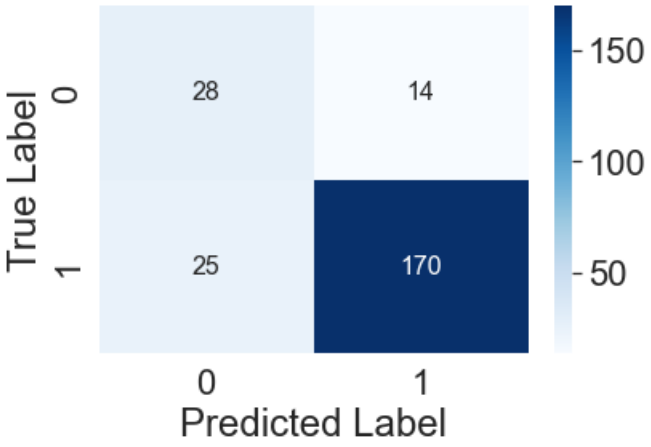

True Label
0
1
32
10
23
172
0
1
Predicted Label
150
100
50

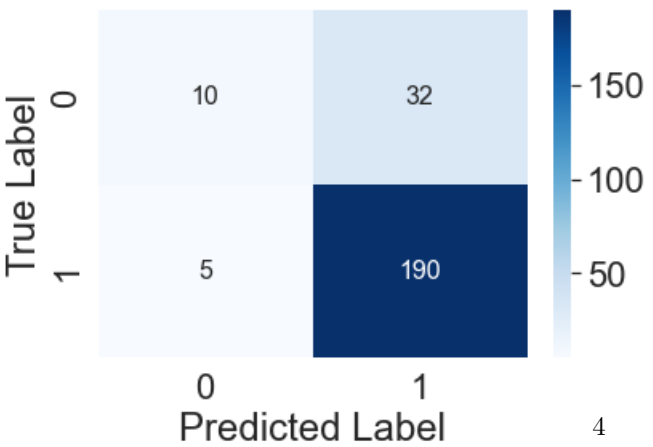

True Label
0
1
33
9
18
177
0
1
Predicted Label
150
100
50


True Label
0
1
8
34
1
194
0
1
Predicted Label
150
100
50


True Label
0
1
0
42
0
195
0
1
Predicted Label
150
100
50
0


True Label
0
1
10
32
5
190
0
1
Predicted Label
150
100
50


True Label
0
1
28
14
25
170
0
1
Predicted Label
150
100
50


True Label
0
1
10
32
5
190
0
1
Predicted Label
150
100
50

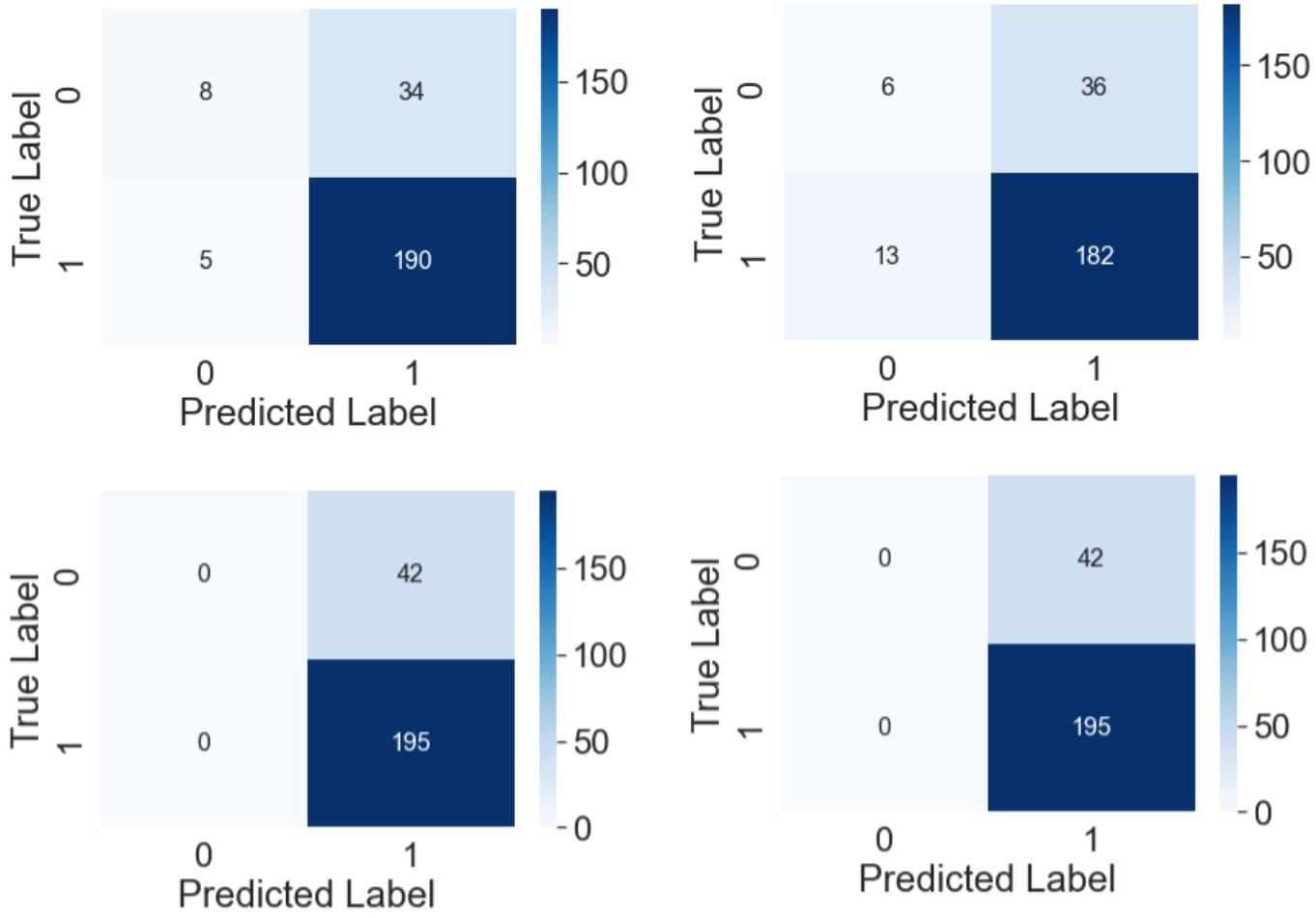


Figure 6: Confusion matrices for Word2Vec.**Left Panel-top to bottom:**Logistic Regression, Random Forest,**Right Panel-top to bottom:**Decision Trees,linear SVC

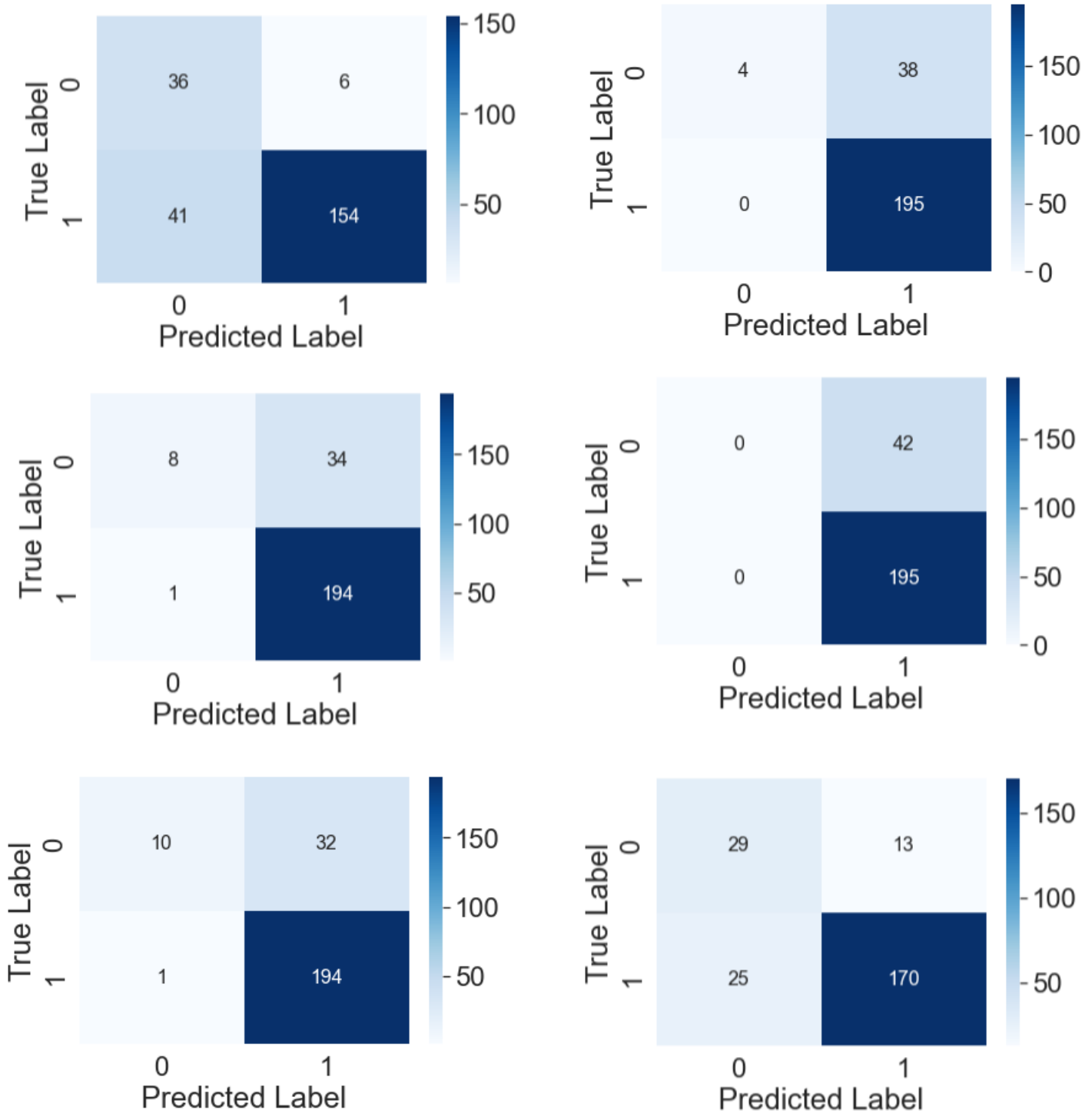


Figure 7: Confusion matrices for TfIdf: **Left Panel-top to bottom:**Logistic Regression, Decision Trees,Gradient Boosted Trees**Right Panel-top to bottom:**naive Bayes,Random Forest,linear SVC

```
##Evaluation of Logistic Regression

eval_lr = BinaryClassificationEvaluator()
auc_lr=eval_lr.evaluate(prediction)
print('Area Under ROC',auc_lr)
```

```
Area Under ROC 0.9086080586080587
```

```
##Logistic Regression Model Accuracy
pred=prediction.select("label","prediction")
evallr = MulticlassClassificationEvaluator()
evallr.setPredictionCol("prediction")
accuracy_lr=evallr.evaluate(pred)
print("Accuracy of Logistic Regression = %g"% (accuracy_lr))
print("Test Error of Logistic Regression = %g"% (1-accuracy_lr))
```

```
Accuracy of Logistic Regression = 0.867689
Test Error of Logistic Regression = 0.132311
```

```
###Logistic Model Confusion matrix, precision,recall
lab_true = pred.select(['label']).collect()
lab_pred = pred.select(['prediction']).collect()
print(classification_report(lab_true, lab_pred))
cm=confusion_matrix(lab_true, lab_pred)
```

```
              precision    recall  f1-score   support

           0       0.58      0.76      0.66        42
           1       0.95      0.88      0.91       195

    accuracy                           0.86       237
   macro avg       0.76      0.82      0.79       237
weighted avg       0.88      0.86      0.87       237

[[ 32  10]
 [ 23 172]]
```

```
##Confusion matrix
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
level= pd.DataFrame(cm, range(2), range(2))
# plt.figure(figsize=(10,7))
sns.set(font_scale=2)
d=sns.heatmap(cm, annot=True, annot_kws={"size": 16},cmap="Blues",fmt="g")
d.set_xlabel("Predicted Label")
d.set_ylabel("True Label")
plt.show()
```

Figure 8: Computation of AUC,accuracy,precision,recall,f1-score,support and construction of confusion matrix for logistic regression-count vectorizer combination

```
def pred(df):
    df = df.filter((df.label != 'null') | (df.review_text != 'null')|(df.review_text !="."))
    df=df.filter((df.label==1)|(df.label==0))
    df=df.dropna()
    df=df.withColumn("text",regexp_replace(col('review_text'), '\d+', ''))
    df = df.withColumn("text2", regexp_replace(col('text'), "[\"$#,<>+@=?!'/%-]",''))
    df = df.dropna()
    tokenize = RegexTokenizer(inputCol="text2", outputCol="text3")
    df = tokenize.transform(df)
    remove = StopWordsRemover(inputCol="text3", outputCol="text4")
    df = remove.transform(df)
    wv = Word2Vec(vectorSize=100, minCount=0, inputCol="text4", outputCol="wv")
    wv1 = wv.fit(df)
    df = wv1.transform(df)
    model_lr = LogisticRegression(featuresCol = 'wv', labelCol='label')
    model_lr2=model_lr.fit(df)
    res = model_lr2.transform(df)
    res = res.select("label","review_text","prediction")
    res.show()
    return(res)

predict_udf = udf(predict, StringType())

def process(time, rdd):
    if rdd.isEmpty():
        return

    print("========= %s =========" % str(time))

    # Convert to data frame
    df = spark.read.json(rdd)

    # Utilize our predict function
    predictions = pred(df)
    predictions.show()
```

```
ssc = StreamingContext(sc, 10)
```

```
lines = ssc.socketTextStream("seppe.net", 7778)
lines.foreachRDD(process)
```

```
ssc_t = StreamingThread(ssc)
ssc_t.start()
```

Figure 9: Predictive Modelling for livestreamed reviews

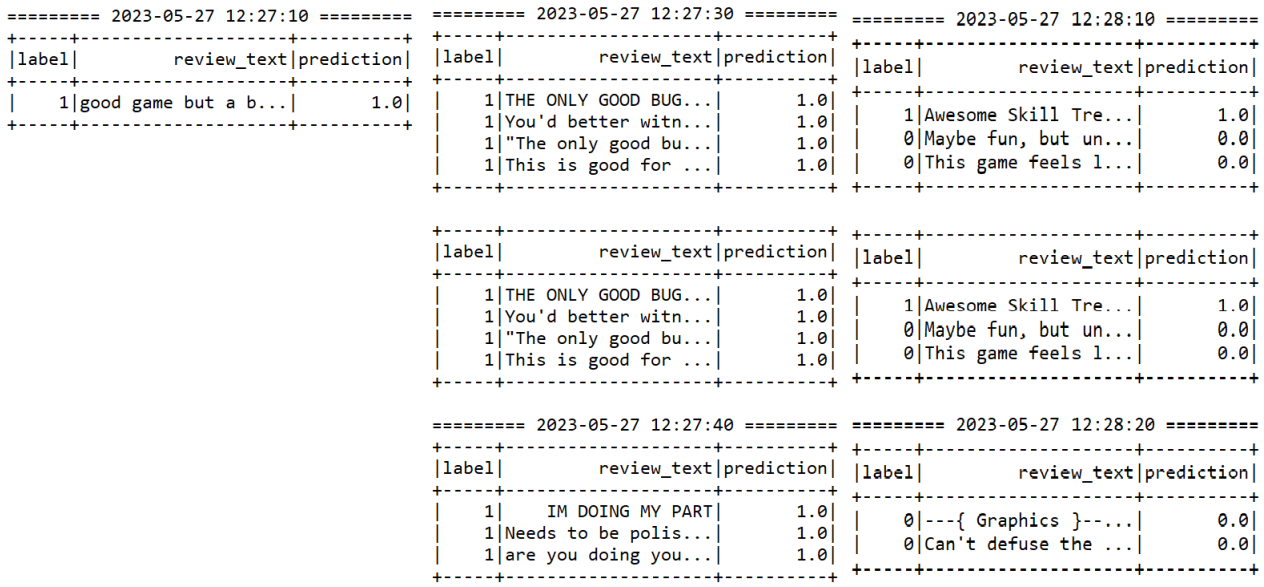


Figure 10: True Labels and Predictions for livestreamed reviews