





# Team Kali P1

Dhaval Bhanderi, Andrew Durden,  
Rutu Gandhi, and Priyank Malviya



# New Naive Bayes Implementation

- Goal: To use spark functions to custom implement Naive Bayes
- Developed a testing suite using a local spark session to test our individual unit functions during development
- Discontinued Work as time passed and external library use restrictions were lifted

```
class PySparkTest(unittest.TestCase):

    @classmethod
    def suppress_py4j_logging(cls):
        logger = logging.getLogger('py4j')
        logger.setLevel(logging.WARN)

    @classmethod
    def create_testing_pyspark_session(cls):
        return (SparkSession.builder
                .master('local[2]')
                .appName('local-testing-pyspark-context')
                .enableHiveSupport()
                .getOrCreate())

    @classmethod
    def setUpClass(cls):
        cls.suppress_py4j_logging()
        cls.spark = cls.create_testing_pyspark_session()

    @classmethod
    def tearDownClass(cls):
        cls.spark.stop()
```

# Using Spark's Naive Bayes

- With Spark's Naive Bayes we built out a pipeline beginning with simple word counts and eventually adding bigrams.
- Using the pipeline structure from Spark's library allowed us to link a handful of steps together using the lazy property of spark for more time efficient predictions.
- With our final pipeline using Naive Bayes achieved an 83.75% accuracy

```
def create_pipeline():  
    """  
    creates model pipeline  
    Currently uses RegexTokenizer to get bytewords as tokens, hashingTF to  
    featurize the tokens as word counts, and NaiveBayes to fit and classify  
  
    This is where most of the work will be done in improving the model  
    """  
  
    tokenizer = RegexTokenizer(inputCol="text", outputCol="words",  
                              pattern="(?!<=\\s)..", gaps=False)  
    ngram = NGram(n=2, inputCol="words", outputCol="grams")  
    hashingTF = HashingTF(numFeatures=65792, inputCol=ngram.getOutputCol(),  
                          outputCol="features")  
    nb = NaiveBayes(smoothing=1)  
    pipeline = Pipeline(stages=[tokenizer, ngram, hashingTF, nb])  
  
    return(pipeline)
```

# Logistic Regression

- Leveraged the Naive Bayes pipeline
- Used single word term frequencies as features
- Fit spark's logistic regression model
- Used a low number of max iterations for speed
- Used cross validation on the small dataset with ParamGridBuilder which trained on 18 different models
- Achieved 83.8% accuracy

```
label_stringIdx = StringIndexer(inputCol="category", outputCol="label")
tokenizer = RegexTokenizer(inputCol="text", outputCol="words",
                           pattern="(?!<=\\s)\\.\\.", gaps=False)
hashingTF = HashingTF(numFeatures=256, inputCol=tokenizer.getOutputCol(),
                      outputCol="features")
lr = LogisticRegression(maxIter=1, featuresCol="features", labelCol="label",
                        family="multinomial")

pipeline = Pipeline(stages=[tokenizer, hashingTF, label_stringIdx, lr])
```

# Random Forest Approach

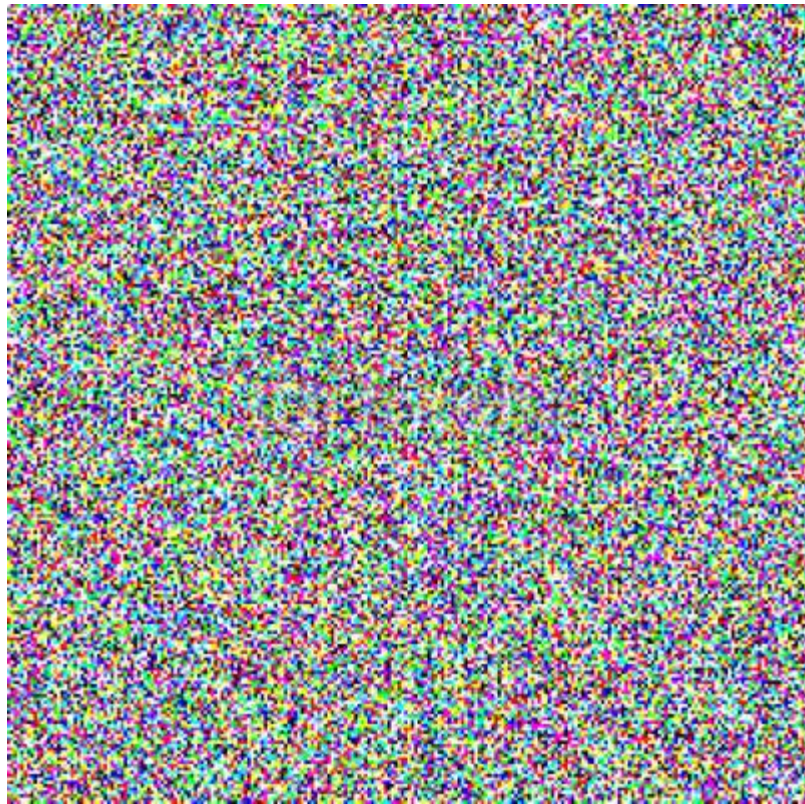
- We used a random forest model to try to achieve a better accuracy
- We used minimal preprocessing, only word counts
- Our random forest had a maxDepth of 8 and 20 trees
- At each point in our model we saved the data in order to avoid having to recompute previous steps on job failure
- Achieved 97% accuracy on the full dataset

```
rf = RandomForestClassifier(labelCol="label",  
                           featuresCol="indexedFeatures", numTrees=20,  
                           maxDepth=8, maxBins=32)
```

```
train_data.write.parquet('gs://project1_dsp_priyank/data/traindata_cv.parquet')  
cv.save('gs://project1_dsp_priyank/data/countvectorizer')
```

# Sparkdl library

- Converted the byte files to images
- Used the DeepImageFeaturizer() that sparkdl offers
- Got an accuracy of approximately 44%
- Improving the preprocessing steps could increase this



# Difficulties and Next Steps

- Difficulties
  - Computation Time
  - GCP out of memory issues and job failure
- Next Steps
  - Adding bigrams to our other models
  - Determining a larger set of stopwords
  - Multi-tier Classification