Project Report - Spark for ML

Dataset Chosen

The dataset we have chosen is the Spam detection Dataset.

It consists of 3 features.

1. Subject of email
2. Body of email
3. Label: Spam or Ham

The dataset contains 33k samples, where 30k samples are used for training and 3k samples are used for testing.

A Quick peek into the dataset

```
+--------------------+-----+
|                text|Class|
+--------------------+-----+
|attached is the l...| spam|
|fyi , kim .
- - -...| spam|
|it did but tetco ...|  ham|
|the methanol plan...|  ham|
|i tried calling y...| spam|
|fyi , kim .
- - -...| spam|
|hi ,
i am forward...| spam|
|enron replaces fa...|  ham|
|attached is the l...| spam|
|start date : 2 / ...| spam|
|start date : 2 / ...| spam|
|start date : 12 /...|  ham|
|fyi , kim .
- - -...| spam|
|this is a complet...|  ham|
|enron tiger team ...|  ham|
|business highligh...|  ham|
|attached is the w...|  ham|
|attached is the l...| spam|
|attached is the l...| spam|
|start date : 2 / ...| spam|
+--------------------+-----+
only showing top 20 rows
```

Design details and Reason

Pre-processing

- The Text data is first Tokenized and then Stop Word removal is done. Al this is done using transforms present in the `pyspark mllib` library

```
tokenizer = Tokenizer(inputCol="text", outputCol="token_text")
stopremove = StopWordsRemover(inputCol='token_text',outputCol='stop_tokens')
```

- The labels - "Spam" and "Ham" were converted into numbers using `StringIndexer`

```
ham_spam_to_num = StringIndexer(inputCol='Class',outputCol='label')
```

- Initially the pre-procesising of the text was done via `CountVectoriation` and `TFID` transformations present in the `pyspark mllib` library.
  The issue: Although the encoding was done sucessfully, each batch was padded to the length of the max string in that batch, therefore incremental learning was not possible.

```
count_vec = CountVectorizer(inputCol='stop_tokens',outputCol='c_vec')
idf = IDF(inputCol="c_vec", outputCol="tf_idf")
```

- Enter Word2Vec
  Our solution was then to use Word2Vec and encode the text into an *n* dimension vector, wehre *n* was specified by us.
  We used several value sof *n*, [32,64,128], and we found no noteworthy difference between performances between these values

```
word2Vec = Word2Vec(vectorSize=128, seed=42, inputCol="stop_tokens",
outputCol="feature")
```

- PCA (For Clustering)
  - Plotting Cluster whose dimensions is *n* is difficult, hence we trained `kmean` incrementally on feature which had been reduced to 2 dimensions with `PCA`

    ```
    pca = PCA(k=2, inputCol="feature", outputCol="features")
    ```

- Pipelines: The Pipeline module from pyspark was used to streamlined the pre processing steps and for better code

```
data_prep_pipe = Pipeline(stages=
[ham_spam_to_num,tokenizer,stopremove,word2Vec])
```

Incremental Learning

- Online Learning or Incremental Learning was set up using the `partial_fit` function provided by some models in `sklearn`
- The data which was streamed in batches was pre-processed and the converted into a `numpy` array, and the dimension adjusted
- Finally, the numpy array was passed as an input to the `partial_fit` function of respective model we are training
- The model was saved every batch as a pickle file and also to be available for testing

Testing

- Testing was done on the 3k samples at once.

```
test_results['f1_score'] = sklearn.metrics.f1_score(y, y_pred)
test_results['accuracy'] = sklearn.metrics.accuracy_score(y, y_pred)
test_results['precision'] = sklearn.metrics.precision_score(y, y_pred)
test_results['recall'] = sklearn.metrics.recall_score(y, y_pred)
test_results['confusion_matrix'] =sklearn.metrics.confusion_matrix(y, y_pred)
```

- The result stored in a pcikle files, which is later accessed by a notebook to check the reulst and plt confusion matrix

Model Details

3 Classifier models which support `partial_fit` from the `sklearn` library were used. Since we are converting the text to fixed size vectors, this seemed a better choice than Naive Bayes, though, I'm skeptical given the results

1. `SGDClassifer`
2. `Perceptron`
3. `PassiveAggressiveClassifier`

Clustering

Clustering was done using `MiniBatchKMeans` module which support incremental learning, i.e `partial_fit`

- Data was streamed just like usual,and th e model trained and stored
- Data was again streamed, and this time the save model was used to predict, and store the prediction as a pickle file
- The pickle file was then read and plotted in the jupter notebook

Take away from Project

- We often find ourselves working with huge amounts of data in real life
- We have to learn to build models that scale and are efficient in these situations
- In this world of big data, the knowlegde to leverage these tools to build ML models that we build normally on notebooks is very important
- Main takeway was introdcution to building practical models that will be deployed or used, as most of the models I have built are on notebooks