# BIG DATA COMPUTING PROJECT

AIKATERINI BELIGIANNI - 2036595

# PROJECT IDEA

A parallel machine learning system developed with PySpark that reads from a dataset of Amazon customer reviews on products and performs a basic sentiment analysis.

# Tasks to be addressed

Binary classification on text based reviews with distributed computation.

INPUT: Historical customer reviews and their respective labeled sentiment as input

OUTPUT: A prediction function which predicts whether a review is positive or negative

# PYSPARK AND COLAB SET-UP

At first, we need to:

1. Install PySpark and related dependencies
2. Import useful PySpark packages
3. Create Spark context
4. Check that all dependencies were installed

# COMMENTS ON HARDWARE

Due to the limited resources provided by Google Colab, for demonstration purposes we have set a limit on the amount of training data, as we train multiple models which requires more computational cores and more memory in order for Spark to parallelise well and be efficient.

# DATA ACQUISITION

We will be using this dataset from Kaggle that has 2 csv files with 3 columns each corresponding to class index (1-negative or 2-positive), review title and review text.

- We download the dataset folder in our Google drive, using the Kaggle API and load the two datasets (train.csv, test.csv) as spark dataframes.

  In order to use the Kaggle API we need to use a JSON file provided by Kaggle , which need to be uploaded to Google Drive.

# DATA PREPROCESSING

Our data as with every NLP task needs to be preprocessed in order to be ready for feature preprocessing and model training. Specifically we:

- Change the column names (label, title, text)
- Drop the title column as it won't be needed later on
- Delete any null values
- Make labels of binary form (good reviews are 0 – bad reviews are 1)
- Clean the text through a pipeline which includes: Case normalization, Trimming, filtering out punctuation and white spaces, Tokenisation, Stopword removal and Stemming.

# FEATURE PREPROCESSING

In order to train our model, we need a technique to extract numerical features from words in a set of documents. In this case we will use TF-IDF (Term Frequency — Inverse Data Frequency), where:

- Term Frequency (TF): gives us the frequency of the word in each document in the corpus.
- Inverse Data Frequency (IDF): is used to calculate the weight of rare words across all documents in the corpus.

After we obtain out TF-IDF representation of our dataframes we remove zero-length vectors and we are ready for model training.

# MODEL TRAINING WITH DIFFERENT HYPERPARAMETER CONFIGURATIONS AND K-FOLD CROSS VALIDATION

In the following sections, we implement a pipeline making use of different hyperparameters and also of k-fold cross validation (in this case k=5) to get a better estimate of the generalization performance of our 3 models. For this purpose, we will use the CrossValidator, ParamGridBuilder and BinaryClassificationEvaluator objects provided by the PySpark API.

Our 3 classifiers which are:

1. Naive Bayes classifier
2. Logistic Regression classifier
3. Random Forest classifier

# NAIVE BAYES CLASSIFIER

Firstly, we will train a naive bayes classifier, using the NaiveBayes object provided by the PySpark API within the package pyspark.ml.classification.

- Our ParamGridBuild object will have  6 values for alpha (the smoothing parameter) which helps tackle the problem of zero probability.
- Our BinaryClassificationEvaluator will have the “areaUnderROC” metric.

# NAIVE BAYES CLASSIFIER: Model performance on training set

The best model according to k-cross validation has a smoothing value of: 0.000

- Area Under ROC Curve (ROC AUC): 0.469

# NAIVE BAYES CLASSIFIER: Predictions on the test set

We evaluate performance on our test set and with our BinaryClassificationEvaluator we obtain the results of:

- Area Under ROC Curve (ROC AUC): 0.469
- Area Under Precision-Recall Curve: 0.464

# NAIVE BAYES CLASSIFIER: Observations

Since we know that we have two classes to classify, a random predictor would have a theoretical maximum of 50% accuracy. For us to say that a model performed well enough, it should have a measurable advantage over the random predictor. This is not the case, as with the Naive Bayes Classifier, the model doesn't seem to generalise well.

# LOGISTIC REGRESSION CLASSIFIER

We then train a logistic regression model, using the LogisticRegression object provided by the PySpark API within the package pyspark.ml.classification.

More specifically, we will tune the two hyperparameters in our ParamGridBuilder object:

- lambda = regParam which is the regulation parameter ([0.0, 0.05, 0.1])
- alpha = elasticNetParam that is the tradeoff parameter for regularization penalties ([0.0, 0.5, 1.0])

# LOGISTIC REGRESSION CLASSIFIER - Model performance on training set

The best model according to k-cross validation has:
lambda=[0.100], alpha=[0.000]

- Area Under ROC Curve (ROC AUC): 0.924
- Accuracy : 0.8461657699461967
- Recall: [0.8674067557014206, 0.8244240050997177]
- Precision: [0.834896953319758 7, 0.858646854252292 1]

# LOGISTIC REGRESSION CLASSIFIER: Predictions on the test set

We evaluate performance on our test set and with our BinaryClassificationEvaluator we obtain the results of:

- Area Under ROC Curve (ROC AUC): 0.901
- Area Under Precision-Recall Curve: 0.898

# LOGISTIC REGRESSION CLASSIFIER: Observations

This time we can see that this model in about the same training time performs much better than the Naive Bayes Classifier without the space complexity that is required by NB. For a non deep learning method, the results achieved across both metrics are very good.

# RANDOM FOREST CLASSIFIER

At last, we will train a Random Forest Classifier using the RandomForestClassifier object provided by the PySpark API within the package pyspark.ml.classification.

More specifically, we will tune the two hyperparameters in our ParamGridBuilder object:

- MaxDepth ([3, 5])
- NumTrees ([10, 50])

# RANDOM FOREST CLASSIFIER - Model performance on training set

The best model according to k-cross validation has an numTrees value of: 50  and a maxDepth value of: 3.

We obtain the below results:

- Area Under ROC Curve (ROC AUC): 0.821
- Recall: [0.8650540238634229, 0.5818332675631647]
- Precision: [0.679225365777933, 0.808145826599019]
- Accuracy: 0.7250935056103366

# RANDOM FOREST CLASSIFIER - Predictions on test set

We evaluate performance on our test set and with our BinaryClassificationEvaluator we obtain the results of:

- Area Under ROC Curve (ROC AUC): 0.817
- Area Under Precision-Recall Curve: 0.814

# RANDOM FOREST CLASSIFIER - Observations

This time although the model performs better than the NB classifier, yielding measurable results, compared to the model which used Logistic Regression for Classification, it achieves worse metric results while it takes more time and more memory.

# FINAL REMARKS

As we observed, out of the 3 models, the model which used the Logistic Regression Classifier was able to outperform the other two. While not trading off many computational resources, it was able to achieve great results for a non DL method.