Sentiment Analysis using Streaming Spark

BD_078_460_474_565

Anushka Hebbar
Shree Thavarakere
Vibha Masti
Shruvi D

Dataset

- Machine learning and spark streaming dataset
- It consists of 2 features:
  - Feature 0 - Sentiment (an integer - 0 or 4, which represents the label of the data
  - Feature 1 - Tweet (the actual twitter message)
- The dataset is divided into train and test sets

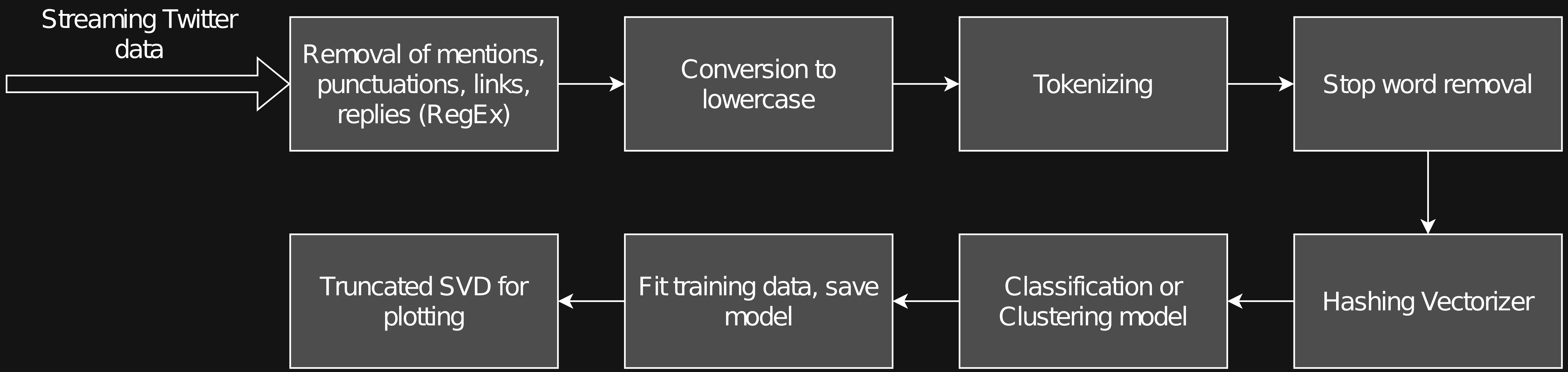
Structure of Repository

- train.py - acts as main file for training functionality
- preprocessing
  - preprocess.py
- classification_models
  - logistic_regression.py
  - multinomial_nb.py
  - passive_aggressive.py
- clustering_models
  - kmeans_clustering.py
  - birch_clustering.py
- test.py - acts as main file for testing functionality

Preprocessing the Stream

- Data is read in as a dstream
- For each RDD in the stream a function called 'process' is called
- In process, first a dictionary is created from the result of rdd.collect()
- This dictionary is then converted to a spark dataframe, which is used for further steps in preprocessing

NLP Pipeline



Preprocessing the Data

BASIC PREPROCESSING

- Removal of mentions, punctuations, links and replies are removed with Regular Expressions
- Conversion to lowercase

TOKENIZER

- Each record converted to an array of individual words

STOP WORDS REMOVER

- Stop words like 'a', 'the', 'on' etc removed

HASHING VECTORIZER

- Conversion of collection of text documents to Sparse Matrix (L2 norm)
- Size of matrix remains constant and vocabulary not stored in memory
- 2-gram

TRUNCATED SVD

- This is used for dimensionality reduction.

Libraries Used

Apart from the built in libraries of python, the following libraries were used:

- pyspark
    - For all data storage, manipulation and processing
    - For streaming support
- sklearn
    - For all the classification and clustering models
    - For some preprocessing and vectorizing techniques.
    - For decomposition - dimensionality reduction techniques.
    - For metrics.
- numpy
    - For processing of arrays, vectors and sparse arrays (sparse arrays were passed to the models)
- matplotlib
    - For plotting clusters
    - For visualizing results of classification models

Classification Models

Stochastic Gradient Descent Classifiers:
This estimator implements regularized linear models with stochastic gradient descent (SGD) learning. SGD allows incremental learning via the partial_fit method.
The model it fits can be controlled with the loss parameter

In our implementation, we used the following models:

- Logistic Regression model - 'log'
- Support Vector model - 'hinge'
- Perceptron algorithm model - 'perceptron'

Classification Models

Multinomial Naive Bayes:
MultinomialNB implements the naive Bayes algorithm for multinomially distributed data, and is one of the two classic naive Bayes variants used in text classification.
The hyperparameters here are:

- alpha: Additive (Laplace/Lidstone) smoothing parameter (0 for no smoothing). In our implementation we tried values of
  - 1.0
  - 0.5
  - 0.7
- fit_prior: Whether to learn class prior probabilities or not
  - We set this to True
- class_prior: Prior probabilities of the classes
  - We set this to None (we didn't have any prior probabilities)

Classification Models

Passive Aggressive Classifier:
The passive-aggressive algorithms are a family of algorithms for large-scale learning. It is an 'online-learning' algorithm, which makes it perfect for training streaming data.

- Passive: If the prediction is correct, keep the model and do not make any changes. i.e., the data in the example is not enough to cause any changes in the model.
- Aggressive: If the prediction is incorrect, make changes to the model. i.e., some change to the model may correct it.
- Hyperparameter- C : This is the regularization parameter, and denotes the penalization the model will make on an incorrect prediction.
- In our implementation, we used the following values of C:
    - 0.2
    - 0.5
    - 1.0

Clustering Models

KMeans:
The KMeans algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares.
Parameters used are:

- *n_clusters=2*
- *init='k-means++'* (selects initial cluster centers for k-mean clustering in a smart way to speed up convergence)
- *n_init=2* (Number of time the k-means algorithm will be run with different centroid seeds)
    - Varying this from 1-2 gave a very slight improvement in accuracy, but further increase had no impact
- *init_size=1000* (Number of samples to randomly sample for speeding up the initialization)
    - Varying this value from 0 -100-1000 did not have any impact on accuracy
- *verbose=False*
- *max_iter=1000*
    - Varying this from 1000-5000 had no impact on accuracy

Clustering Models

Birch Clustering - Tried, but not included in final project

The Birch builds a tree called the Clustering Feature Tree (CFT) for the given data. The data is essentially lossy compressed to a set of Clustering Feature nodes (CF Nodes). The CF Nodes have a number of subclusters called Clustering Feature subclusters (CF Subclusters) .

The CF Subclusters hold the necessary information for clustering.

Parameters:

- *n_clusters=2* (Number of clusters after the final clustering step, which treats the subclusters from the leaves as new samples)

However, the efficiency of this algorithm was very less, and it took a very long time. Also, it did not give any significant improvement over KMeans. Therefore, it was not used in the final training.

Results

Each model (3 variations of the 3 classification models and 1 clustering model) was tested on different batch sizes

The batch sizes trained on:

1. 2000
2. 2500
3. 3000
4. 4000
5. 5000

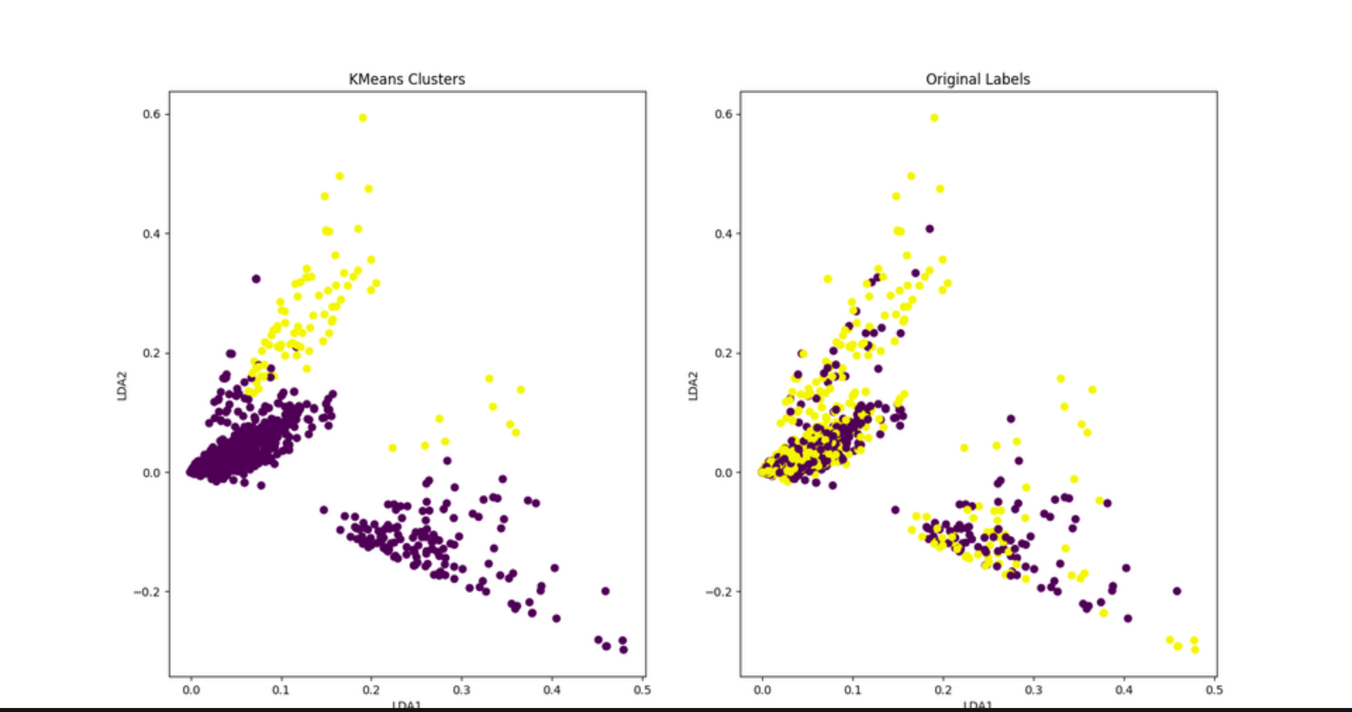
Accuracy:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

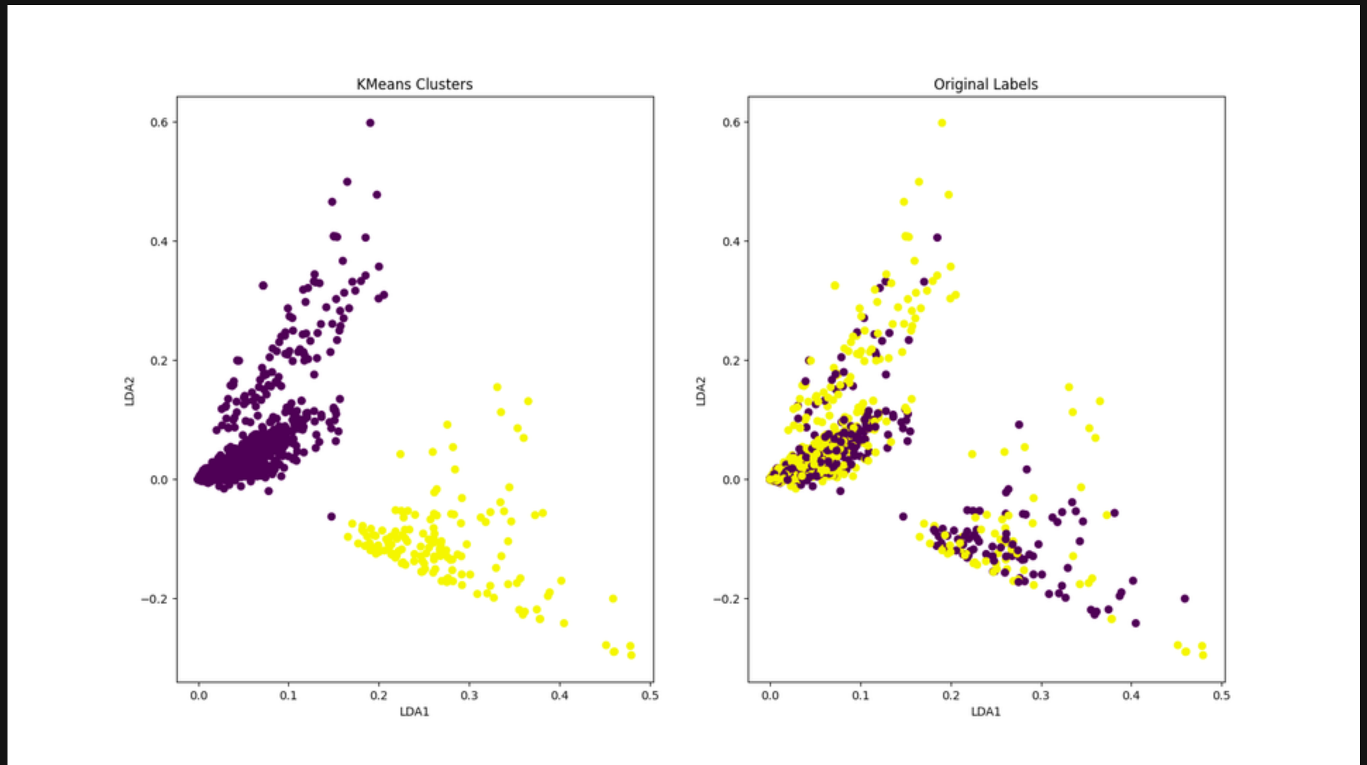
ROC curve: TPR vs FPR

Clustering Plots

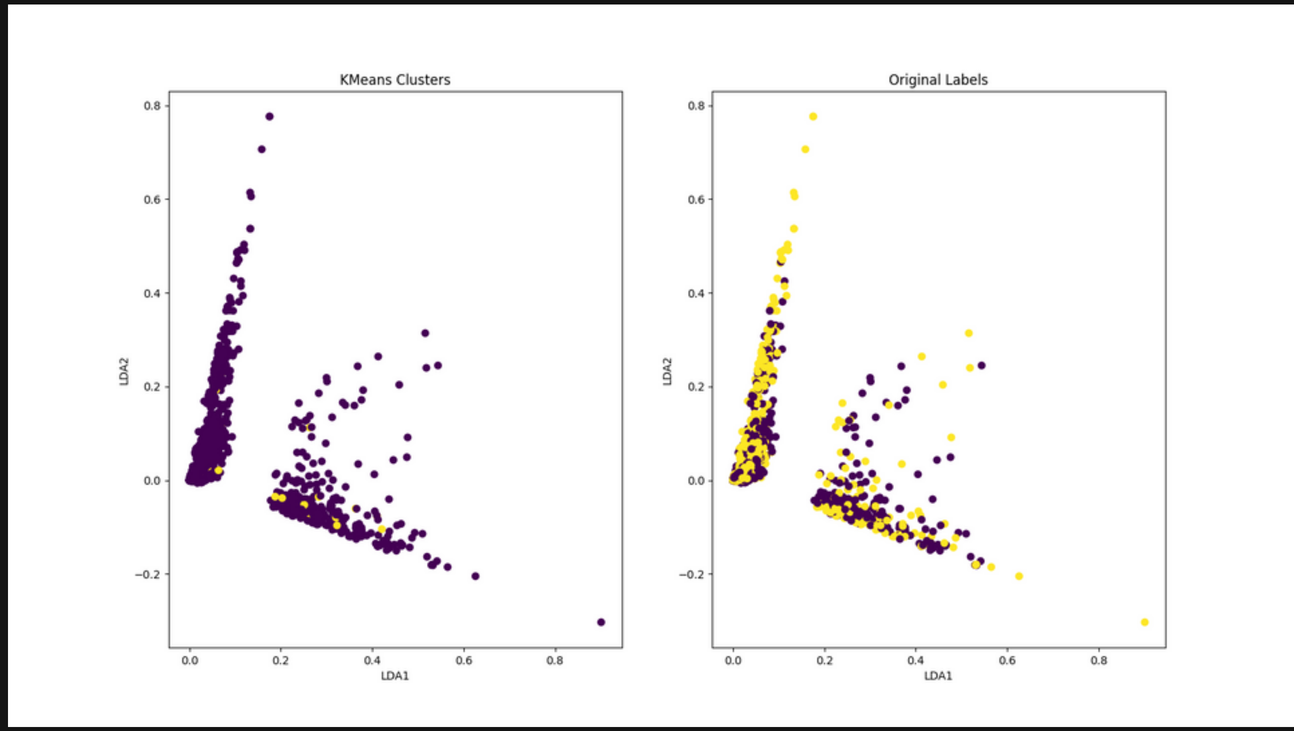
Batch size 2000



Batch size 2500

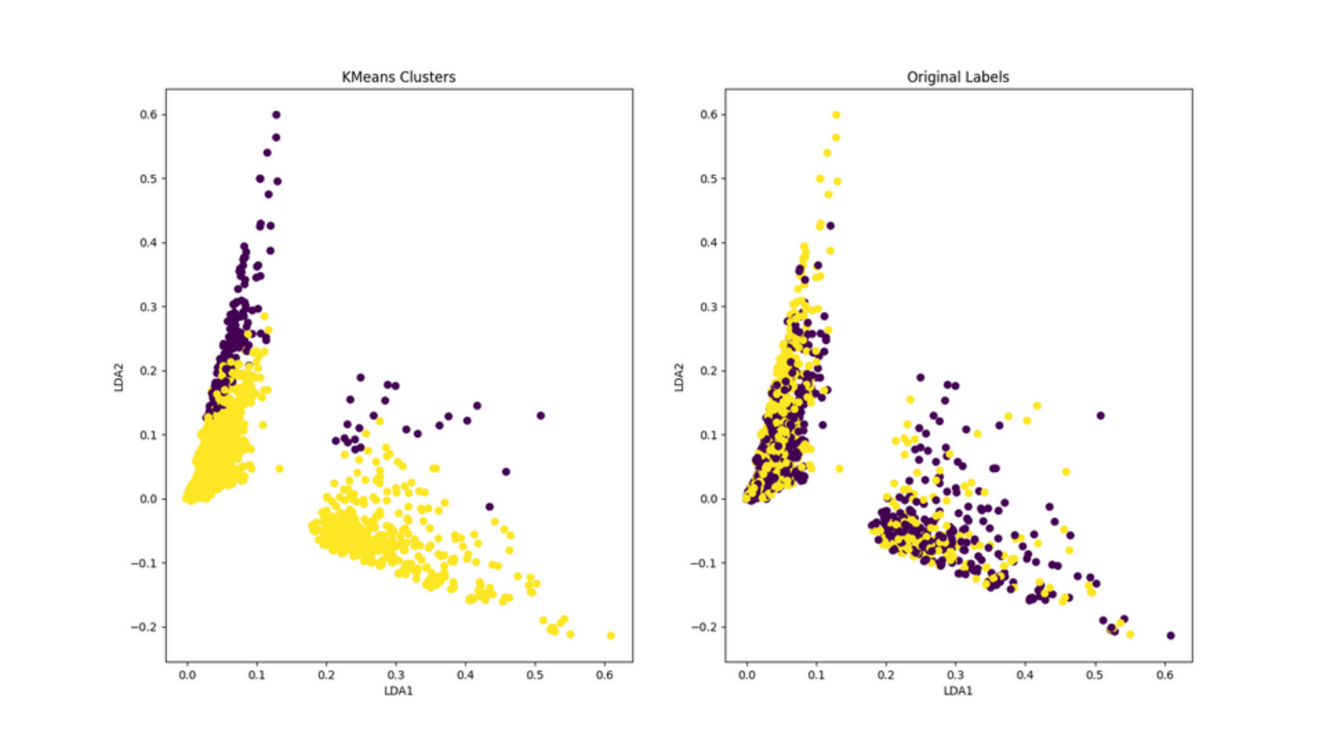


Batch size 3000

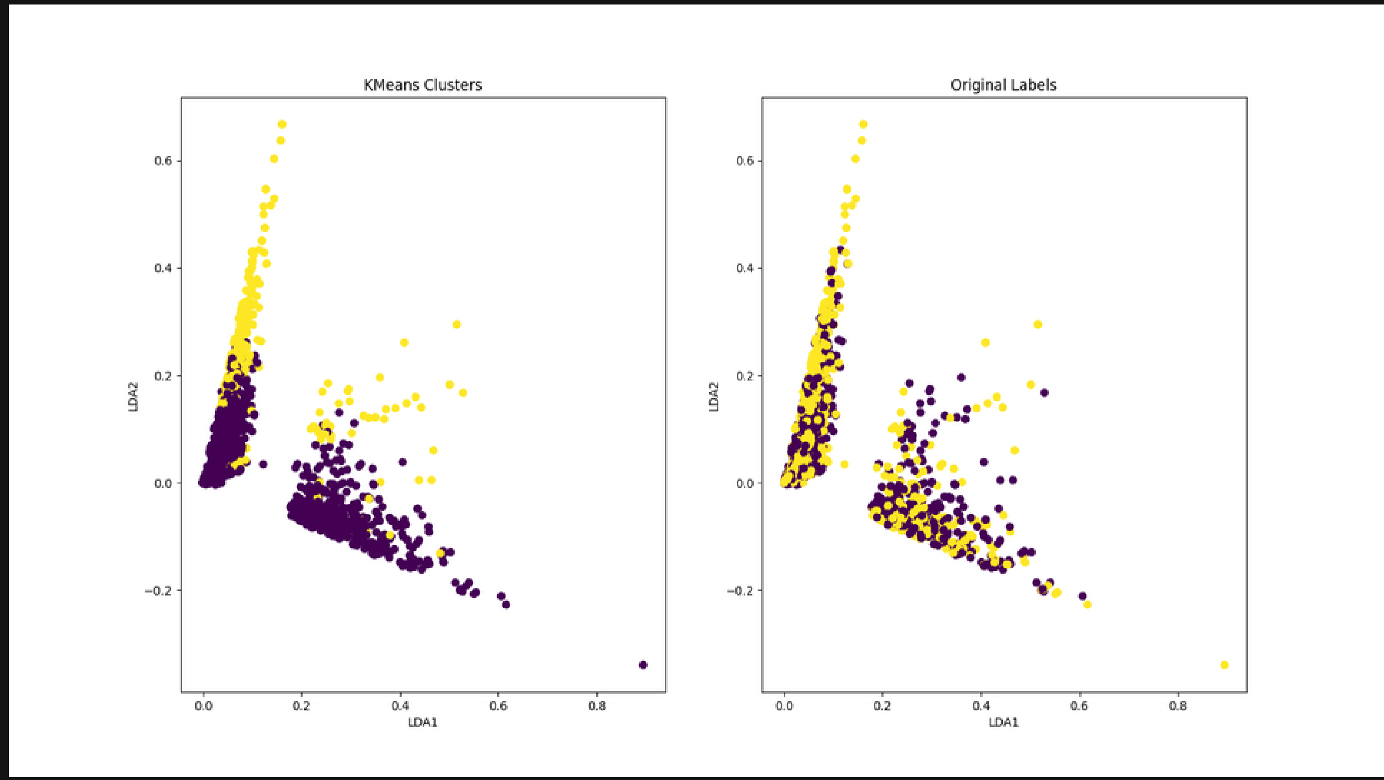


Clustering Plots

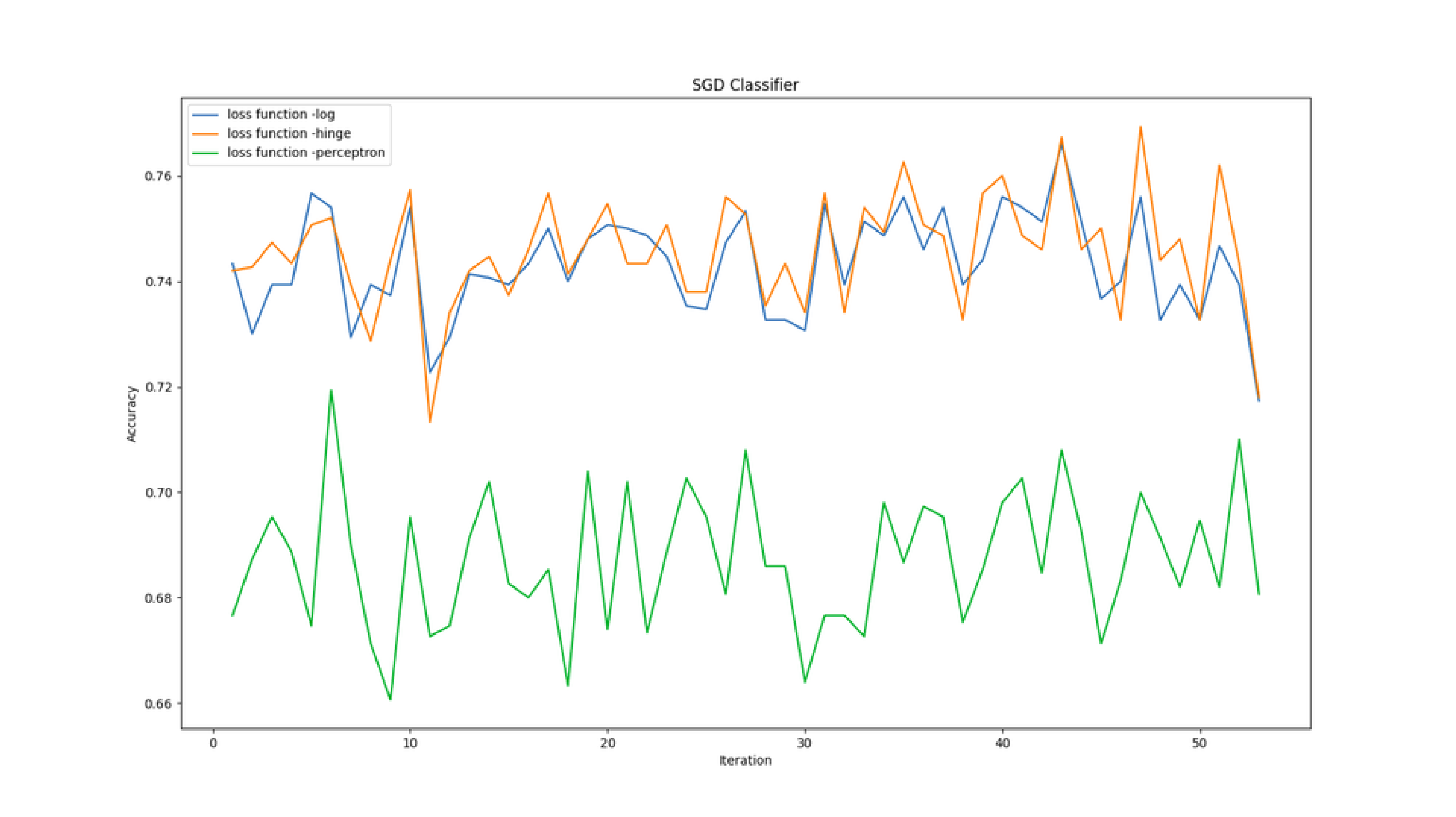
Batch size 4000

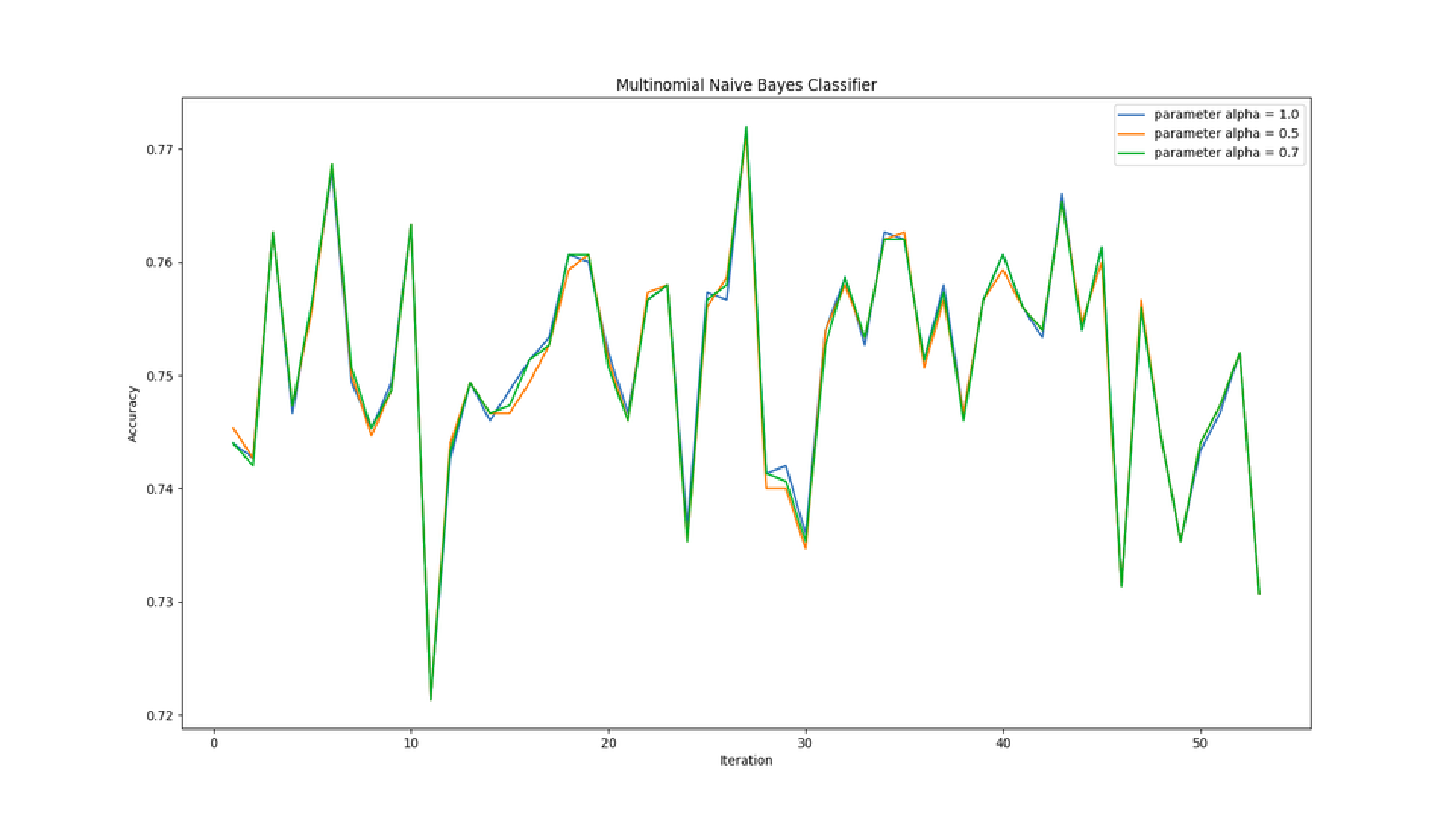


Batch size 5000

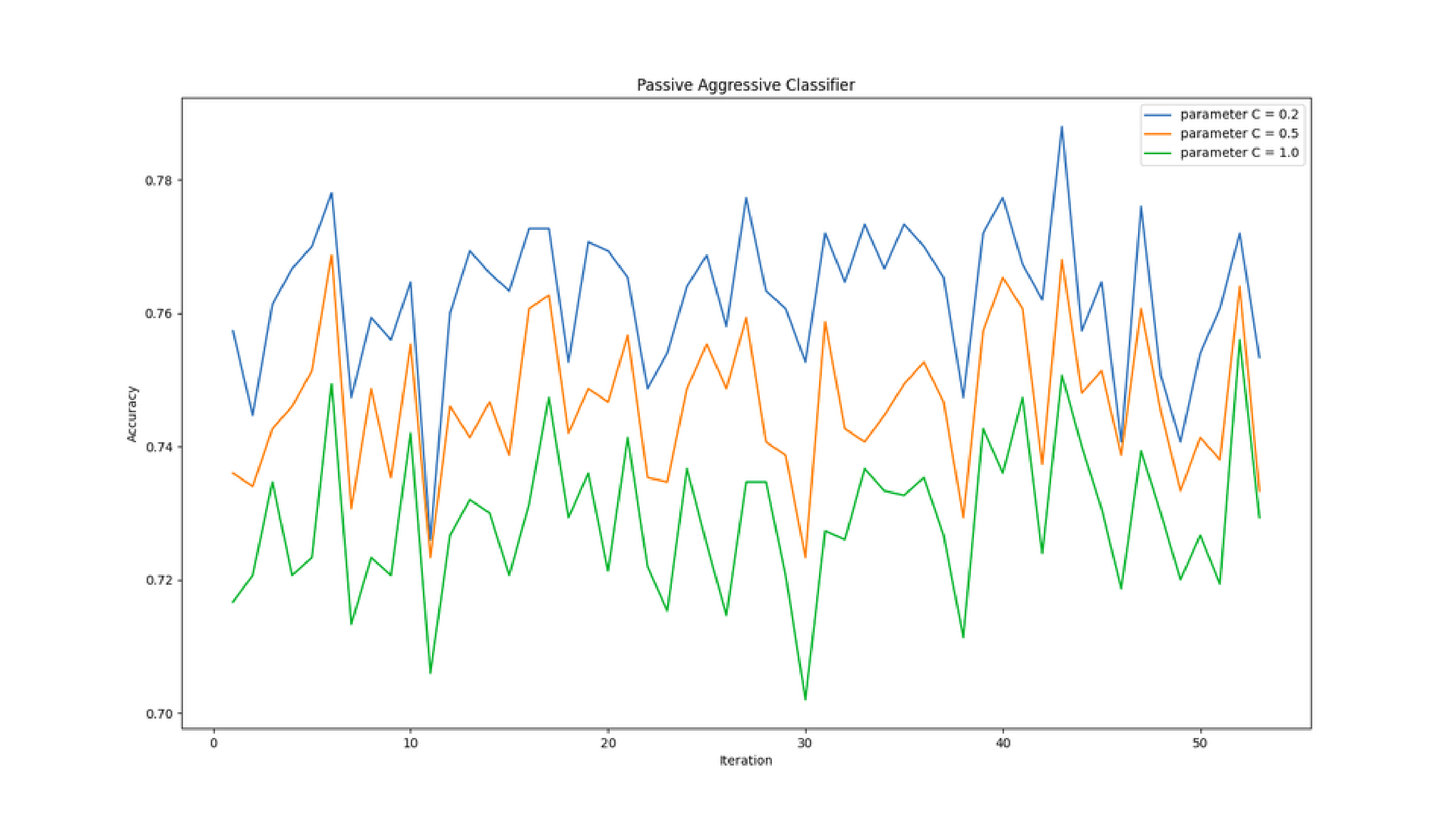


Results - Hyperparameters

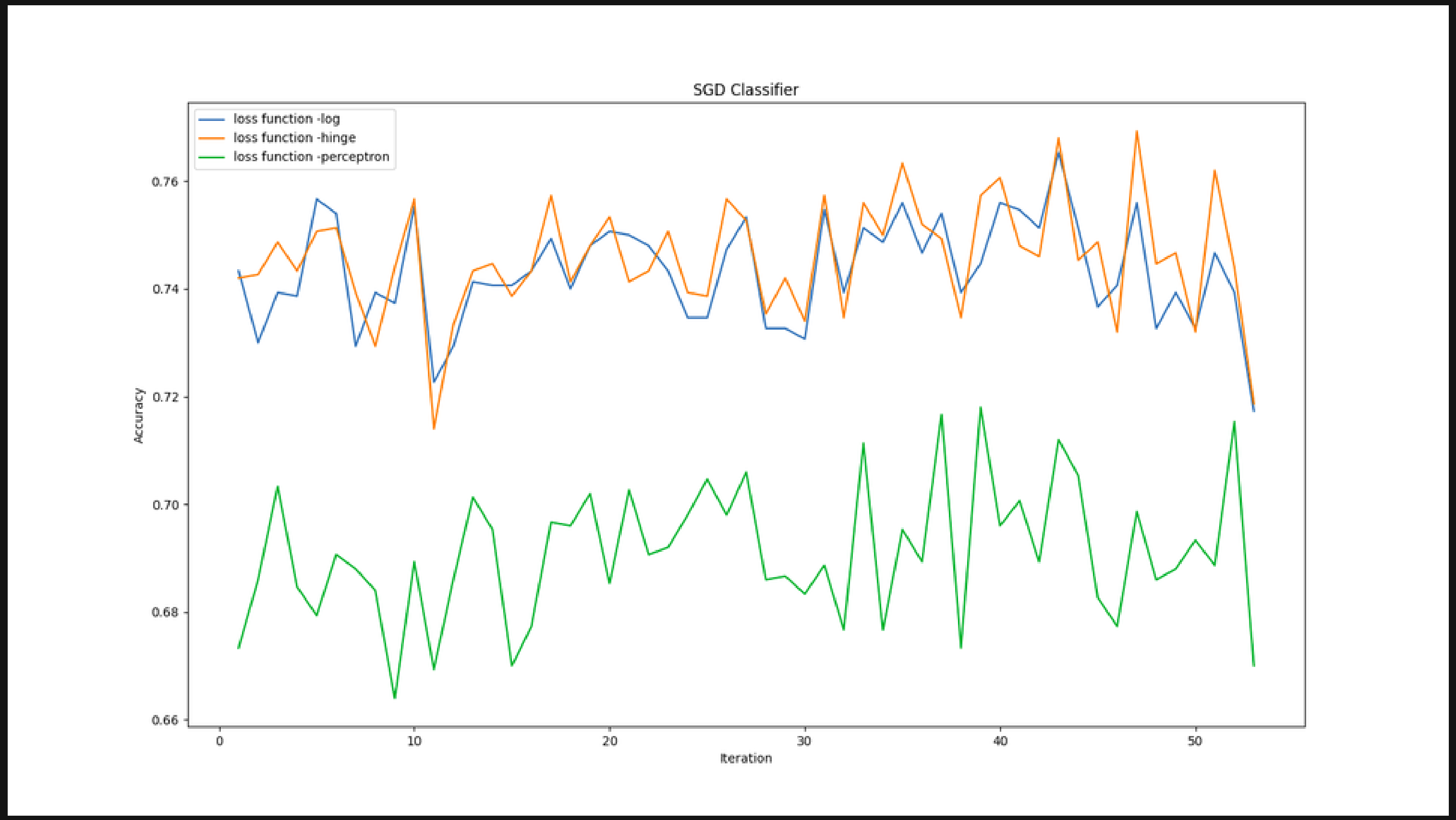


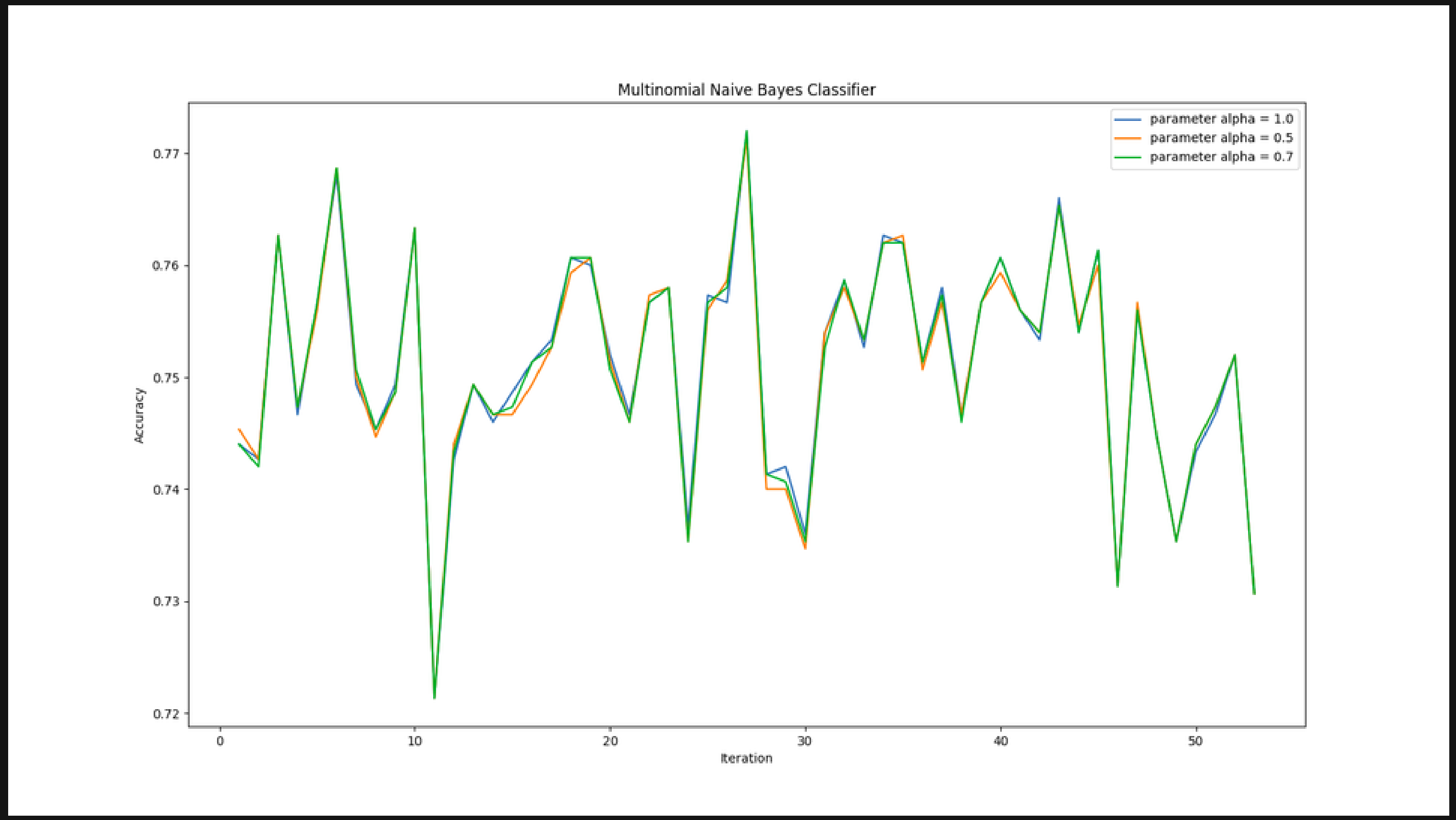


Batch Size 2000

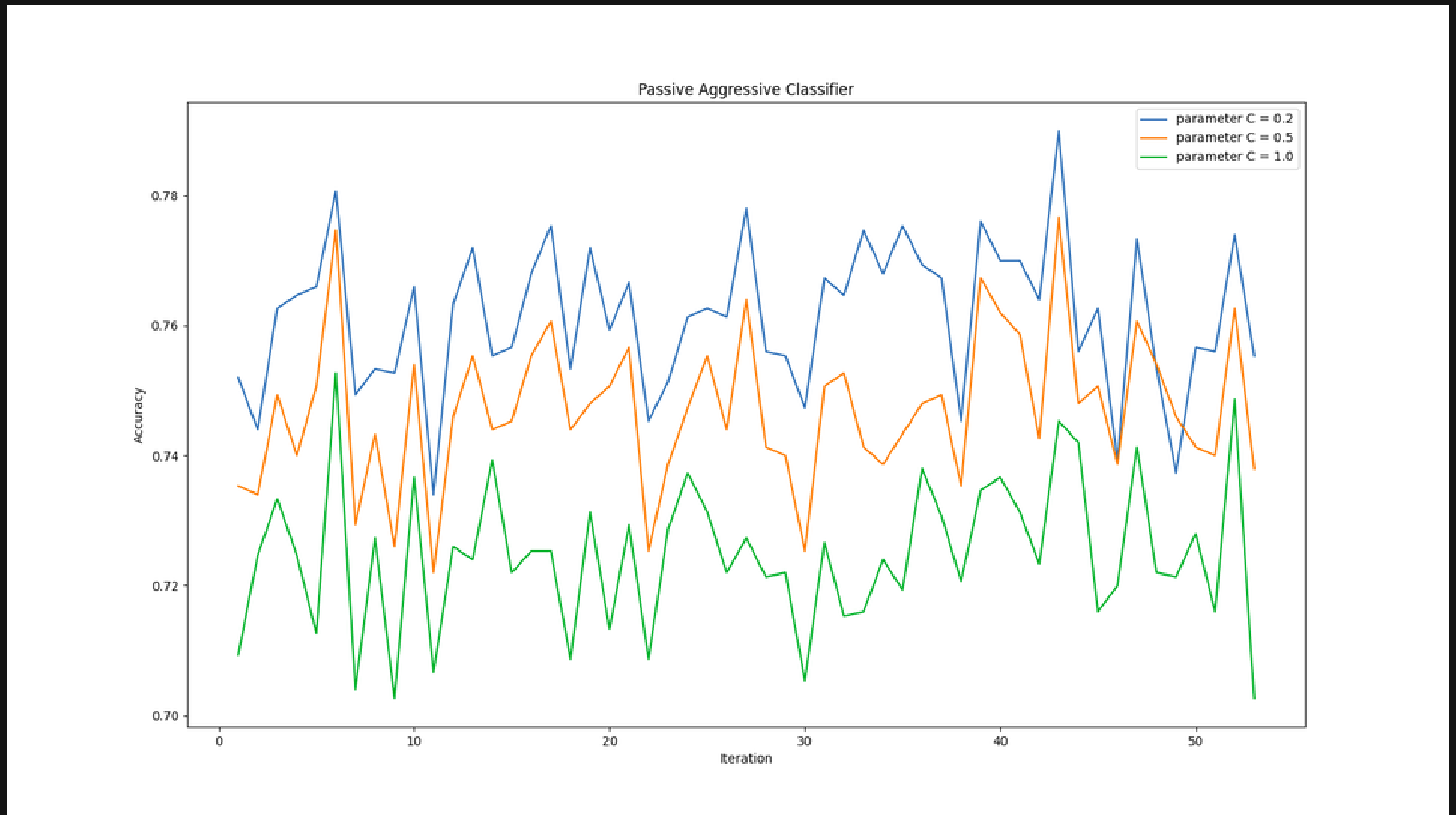


Accuracy - Hyperparameters

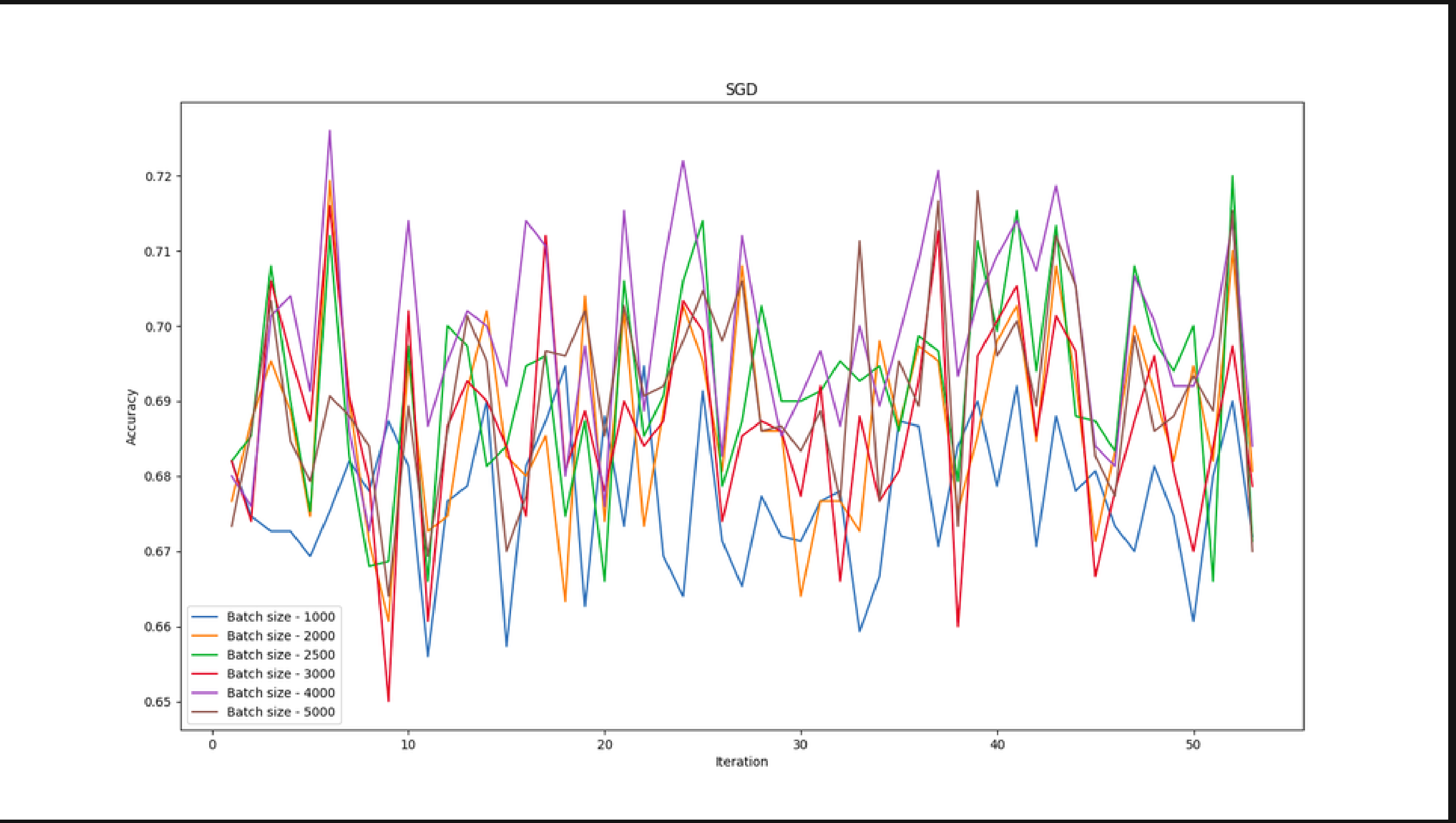


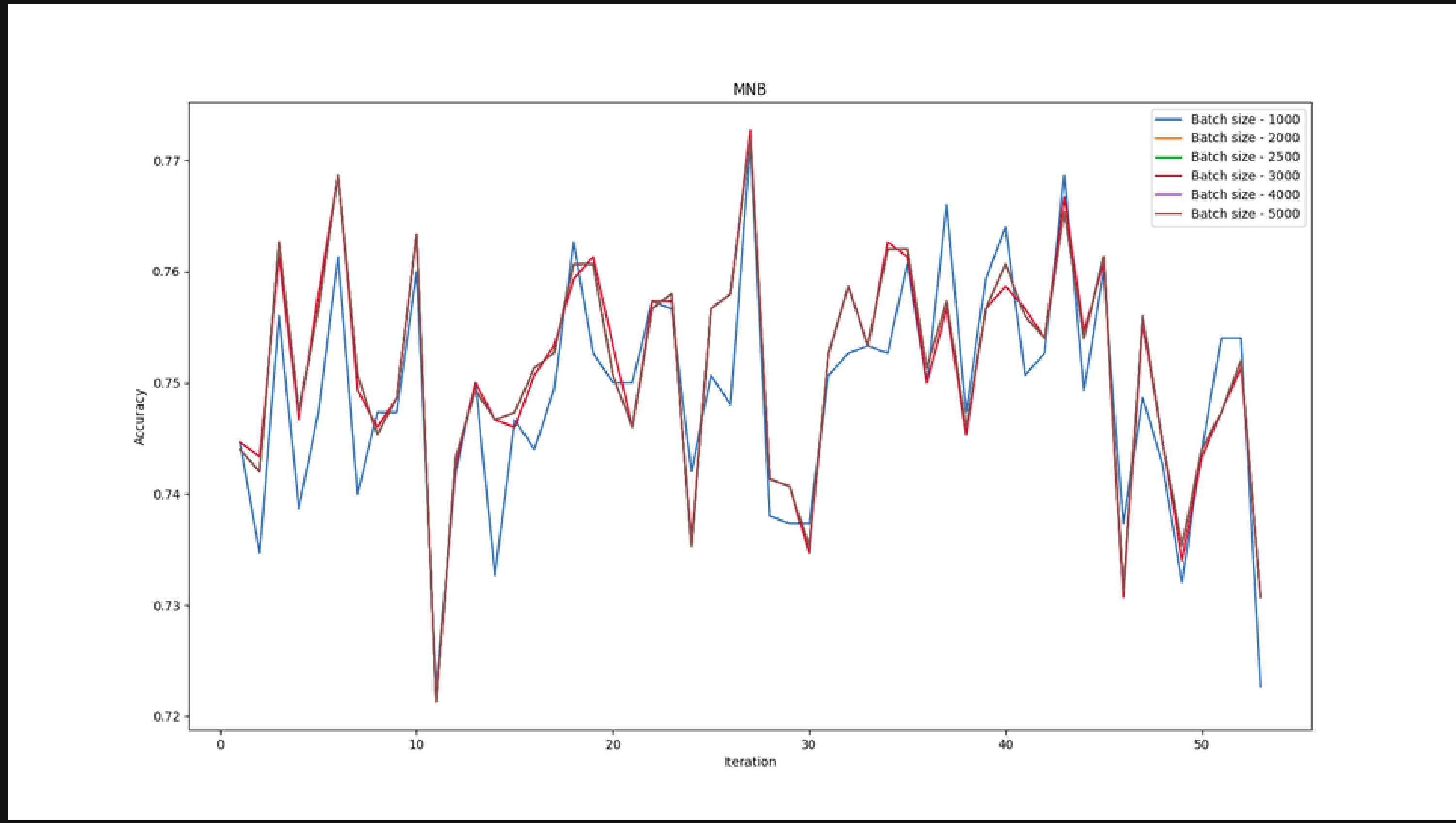


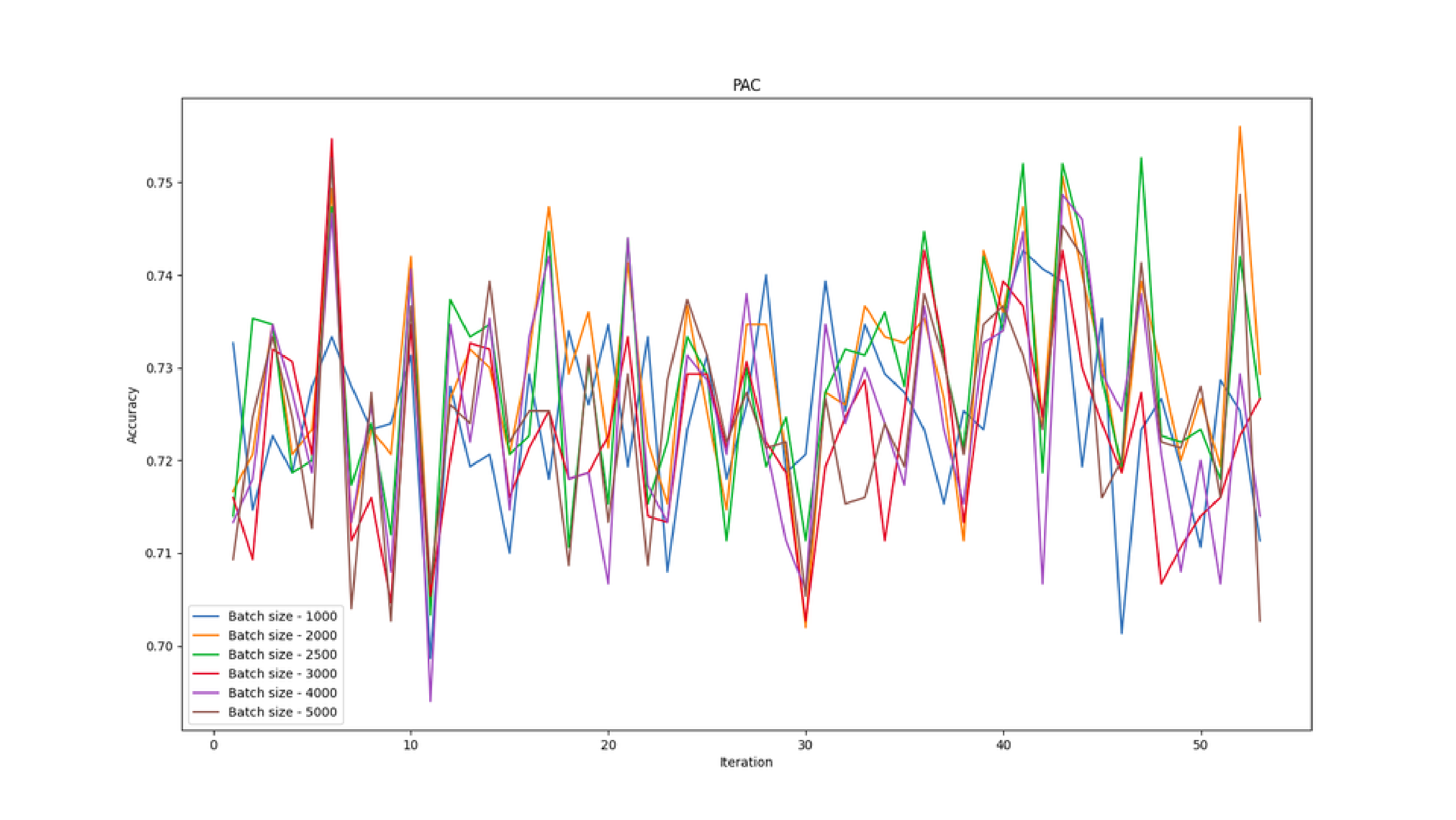
Batch Size 5000



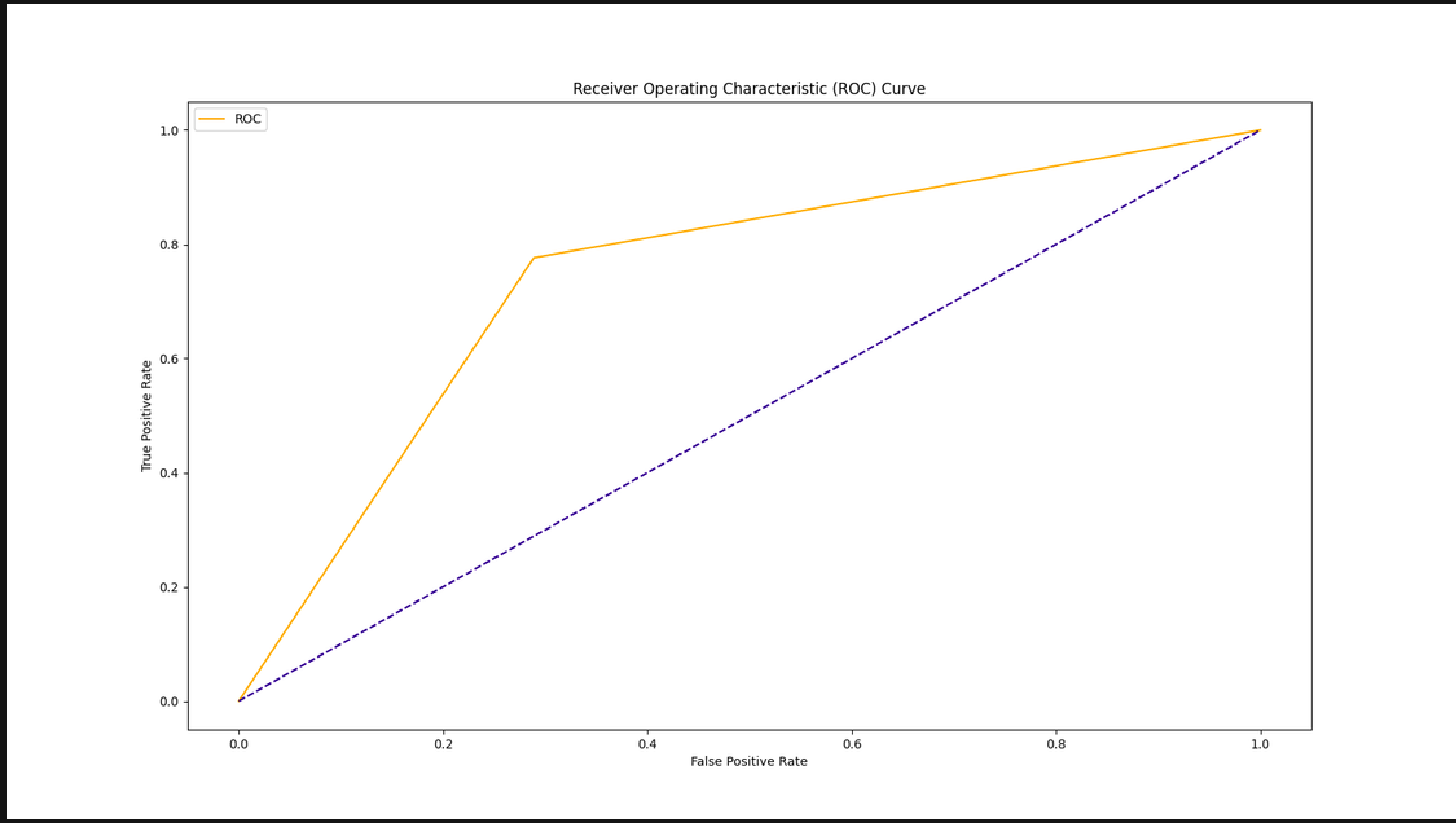
Accuracy (Batch Sizes) - 3 models



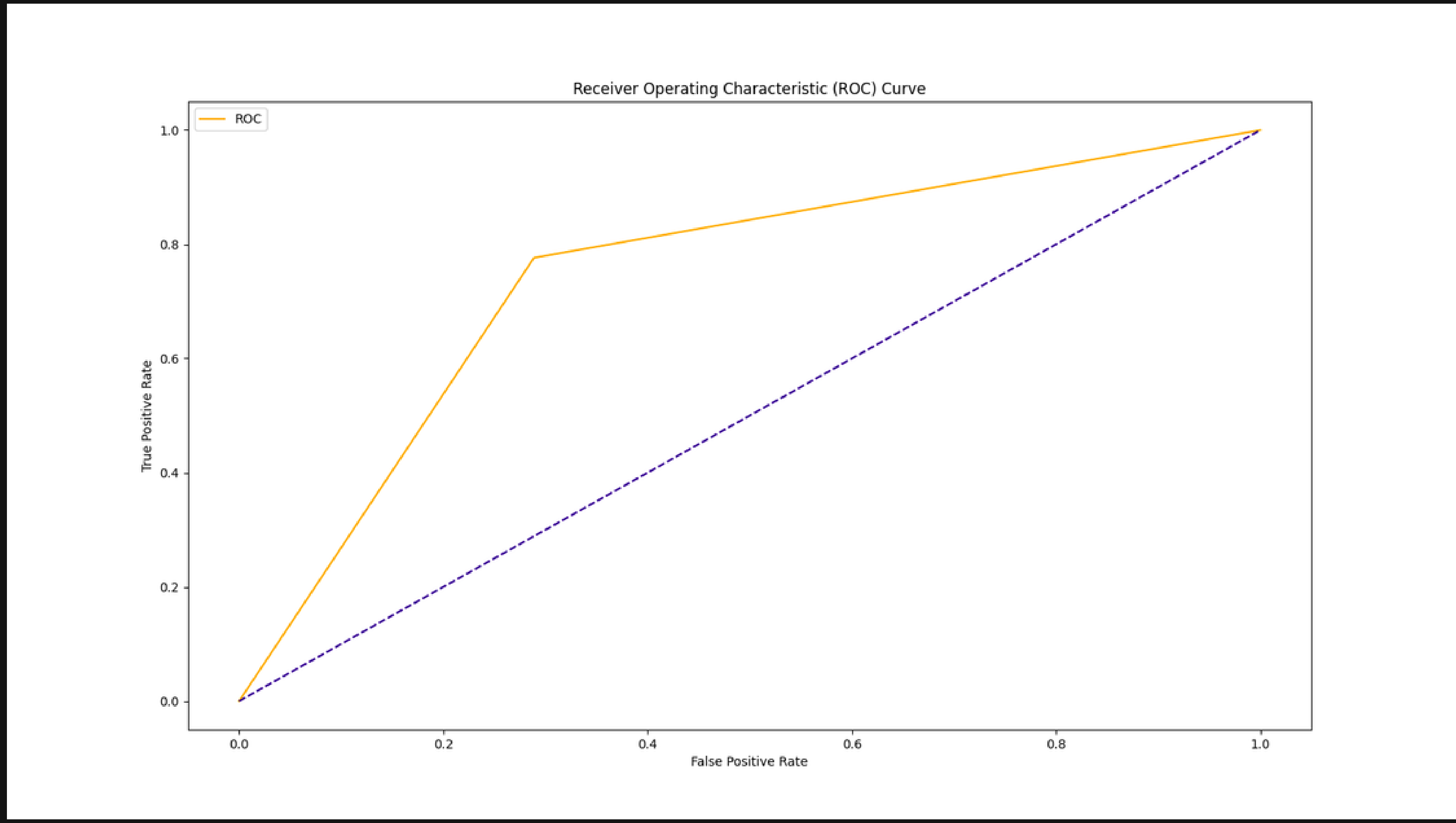




Results - ROC Curves (SGD)

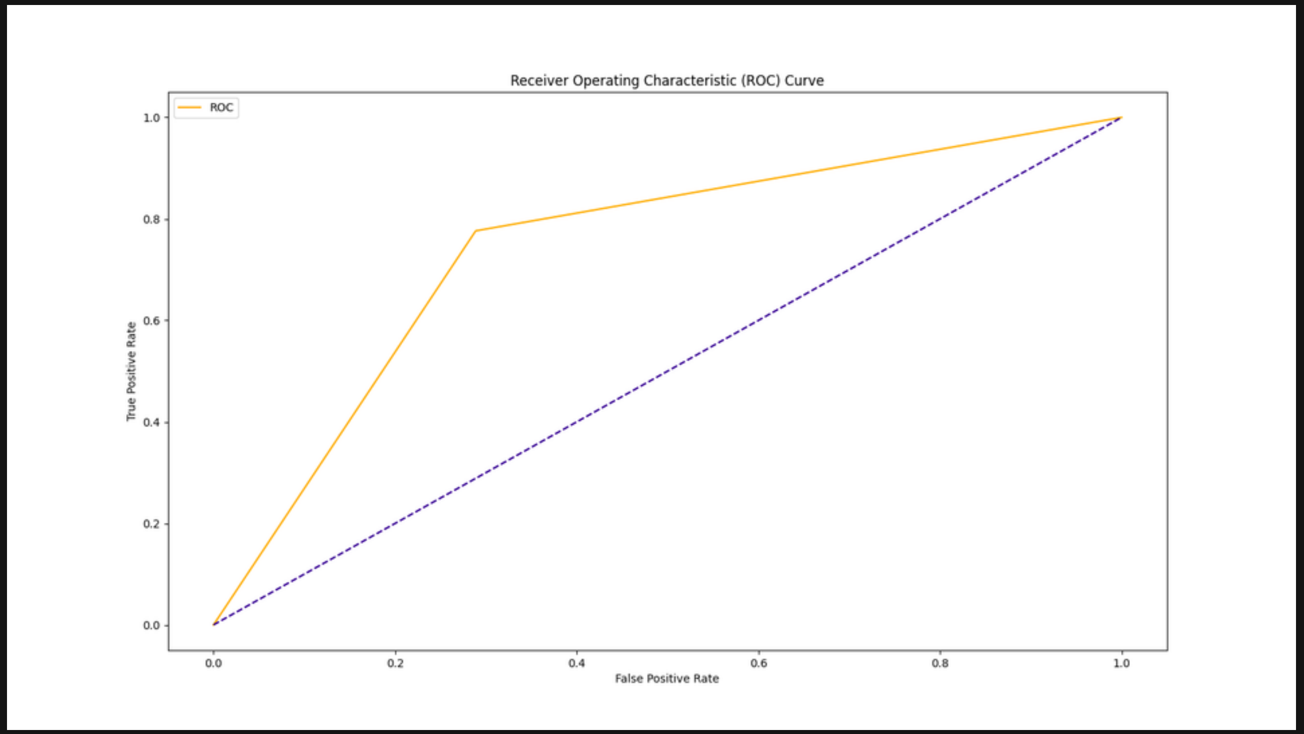


Batch Size 2000 - First SGD Model

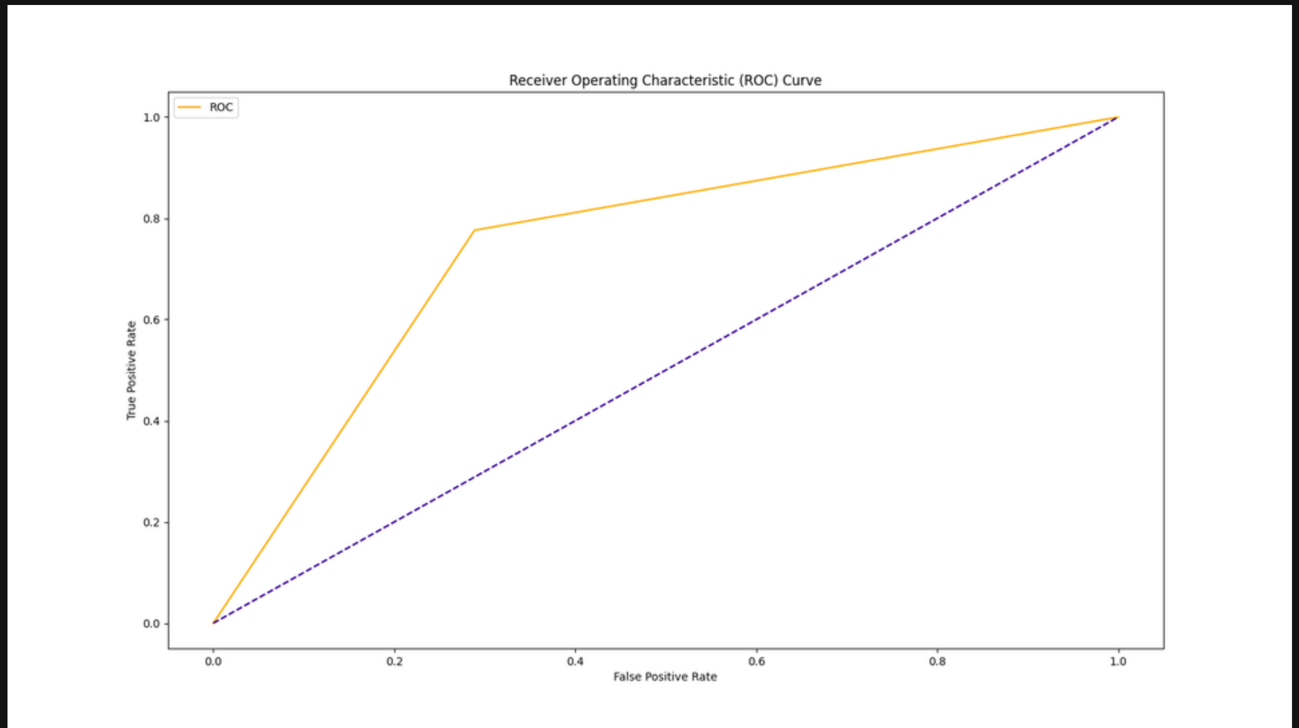


Batch Size 2500 - First SGD Model

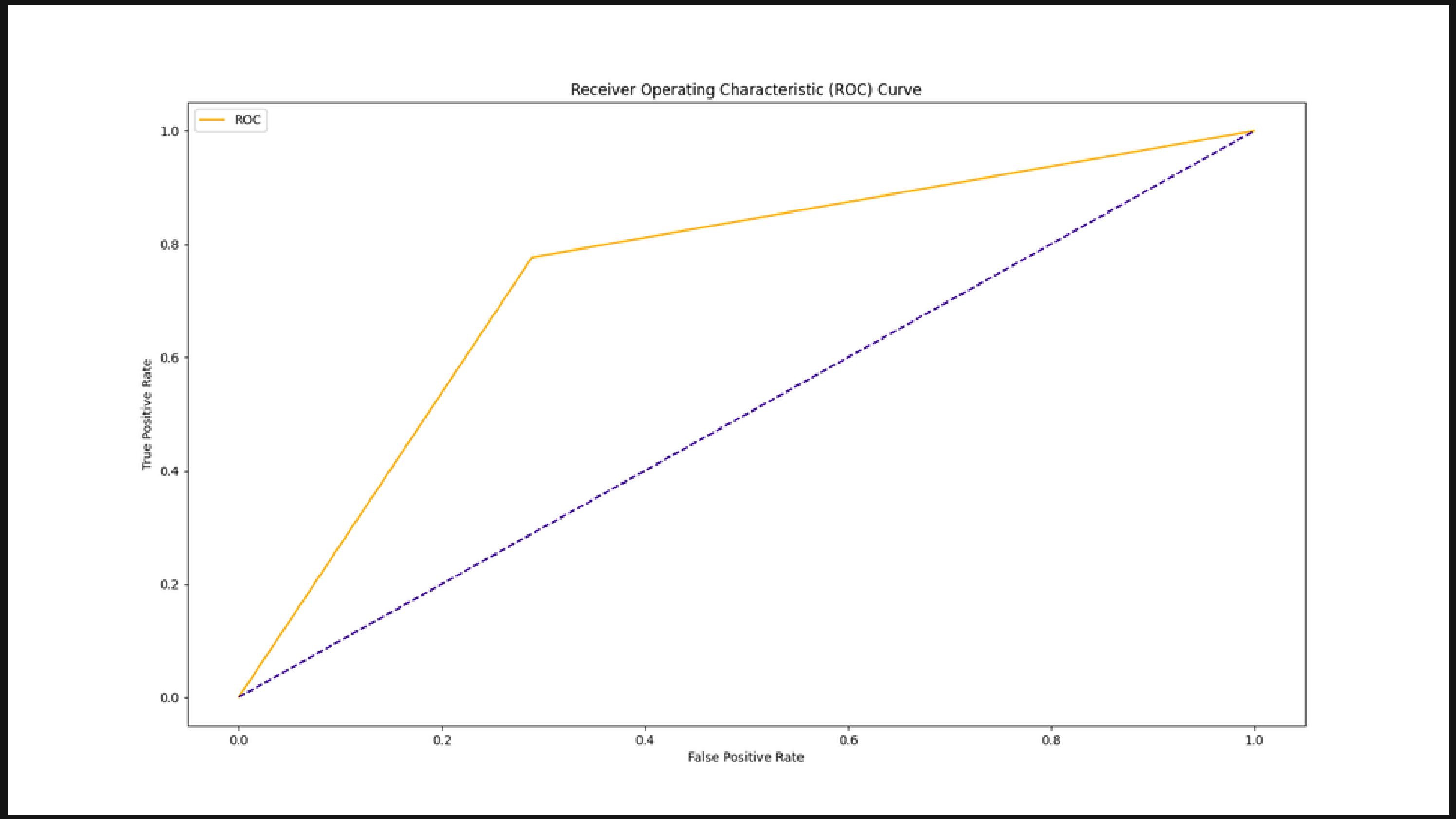
Results - ROC Curves (SGD)



Batch Size 3000 - First SGD Model



Batch Size 4000 - First SGD Model



Batch Size 5000 - First SGD Model

Summarised Results

Model	Batch Size	Hyperparameter	Average Test Accuracy
SGD	2000	Log loss	0.7432
		Hinge loss	**0.7455**
		Perceptron loss	0.6872
	3000	Log loss	0.7431
		Hinge loss	**0.7453**
		Perceptron loss	0.6867
	4000	Log loss	0.7432
		Hinge loss	**0.7456**
		Perceptron loss	0.6983
	5000	Log loss	0.7432
		Hinge loss	0.7456
		Perceptron loss	0.6909

Summarised Results

Model	Batch Size	Hyperparameter	Average Test Accuracy
MNB	2000	**1.0**	**0.7511**
		0.5	0.7509
		0.7	0.7501
	3000	**1.0**	**0.7511**
		0.5	0.7508
		0.7	0.7509
	4000	**1.0**	**0.7511**
		0.5	0.7509
		0.7	**0.7511**
	5000	**1.0**	**0.7511**
		0.5	0.7509
		0.7	**0.7511**

Summarised Results

Model	Batch Size	Hyperparameter	Average Test Accuracy
PAC	2000	**0.2**	**0.7624**
		0.5	0.7463
		1.0	0.7289
	3000	**0.2**	**0.7623**
		0.5	0.7461
		1.0	0.7231
	4000	**0.2**	**0.7624**
		0.5	0.7461
		1.0	0.7247
	5000	**0.2**	**0.7614**
		0.5	0.7470
		1.0	0.7246

Conclusion

- Accuracy did not change much when batch size varied from 2000-5000
  - Could not train smaller batch sizes on our systems
- LR and SVM performed similarly, better than perceptron
- Changing alpha in MNB had almost no impact
- Changing C in PAC had slight impact
  - Optimal C: 0.2
- All models - ~75% accuracy
- K-means clustering not effective

- Possible future improvements
  - DBScan clustering
  - Smaller batch sizes
  -

Thank you