



EECS E6893 Big Data Analytics

HW1: Clustering and Classification on Spark MLlib

Manav Munjal mm6840@Columbia.edu

Agenda

- HW1
 - Iterative K-means clustering
 - Binary Classification with Spark Mlib
 - Mini-Project

HW1

- Document clustering with K-means
 - Implement iterative K-means clustering in Spark
 - L1, L2 distance functions
 - Different initialization strategies
 - Plot the cluster assignment result with T-SNE dimensionality reduction
- Binary Classification
 - Load data into Spark Dataframe
 - Modeling and Evaluation

HW1 Part 1 Iterative K-means clustering on Spark

Apache Spark is an open-source engine for large-scale data processing, especially useful when working with big datasets across many machines.

- It supports in-memory computing, making it faster than Hadoop MapReduce.
- It provides high-level APIs in Python (PySpark), Java, and Scala.
- It includes MLlib, a library for scalable machine learning.

Iterative K-means

- In each iteration, k centroids are initialized, each point in the space is assigned to the nearest centroid, and the centroids are re-computed
- Pseudo code:

Algorithm 1 Iterative k -Means Algorithm

```
1: procedure ITERATIVE  $k$ -MEANS
2:   Select  $k$  points as initial centroids of the  $k$  clusters.
3:   for iterations := 1 to MAX_ITER do
4:     for each point  $p$  in the dataset do
5:       Assign point  $p$  to the cluster with the closest centroid
6:     end for
7:     for each cluster  $c$  do
8:       Recompute the centroid of  $c$  as the mean of all the data points assigned to  $c$ 
9:     end for
10:  end for
11: end procedure
```

Iterative K-means in Spark

Algorithm 1 Iterative k -Means Algorithm

```
1: procedure ITERATIVE  $k$ -MEANS
2:   Select  $k$  points as initial centroids of the  $k$  clusters.
3:   for iterations := 1 to MAX_ITER do
4:     for each point  $p$  in the dataset do
5:       Assign point  $p$  to the cluster with the closest centroid
6:     end for
7:     for each cluster  $c$  do
8:       Recompute the centroid of  $c$  as the mean of all the data points assigned to  $c$ 
9:     end for
10:   end for
11: end procedure
```

```
# iterative k-means
```

```
for _ in range(MAX_ITER):
```

```
    # Transform each point to a combo of point, closest centroid, count
```

```
    # point -> (closest_centroid, (point, 1))
```

```
    # Re-compute cluster center
```

```
    # For each cluster center (key), aggregate its value by summing up points and count
```

```
    # Average the points for each centroid: divide sum of points by count
```

Spark operations you might need:
map, reduceByKey, collect, keys

Document clustering

Goal: Group similar text documents together using the K-means clustering algorithm based on word importance features.

Dataset:

- data.txt contains 4601 documents, each represented as a 58-dimensional vector.
- Each feature represents the importance of a word (like TF-IDF) in the document.
- Idea: Similar documents → similar word distributions → same cluster.

Task:

- Run iterative K-means using:
 - Two initialization strategies (c1.txt and c2.txt)
 - Two distance metrics (L1 and L2)
- Evaluate clustering quality using within-cluster cost
- Visualize results using t-SNE (reduces 58-D to 2-D for plotting)

Plot the result with t-SNE

t-SNE (t-distributed Stochastic Neighbor Embedding) is a dimensionality reduction technique that helps you visualize high-dimensional data in 2D or 3D space.

```
from sklearn.manifold import TSNE
```

```
# RDD -> np array  
data_np = np.array(data.collect())
```

```
data_np.shape
```

```
(4601, 58)
```

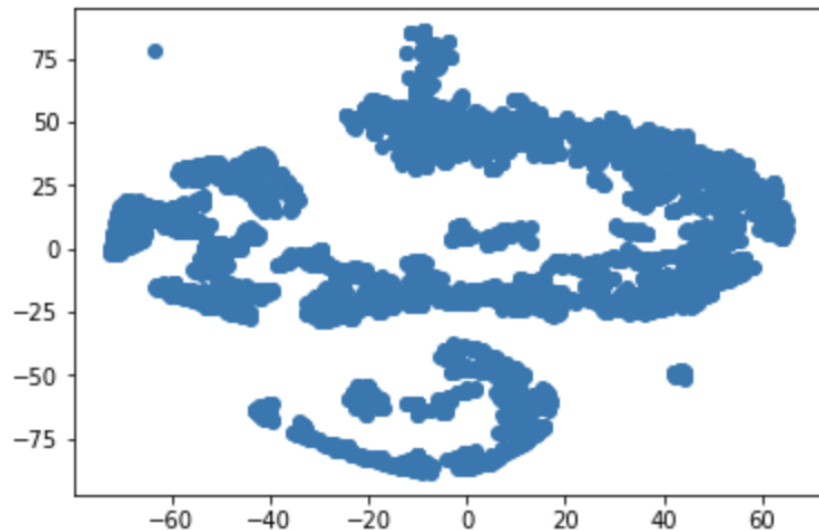
```
data_embedded = TSNE(n_components=2).fit_transform(data_np)
```

```
data_embedded.shape
```

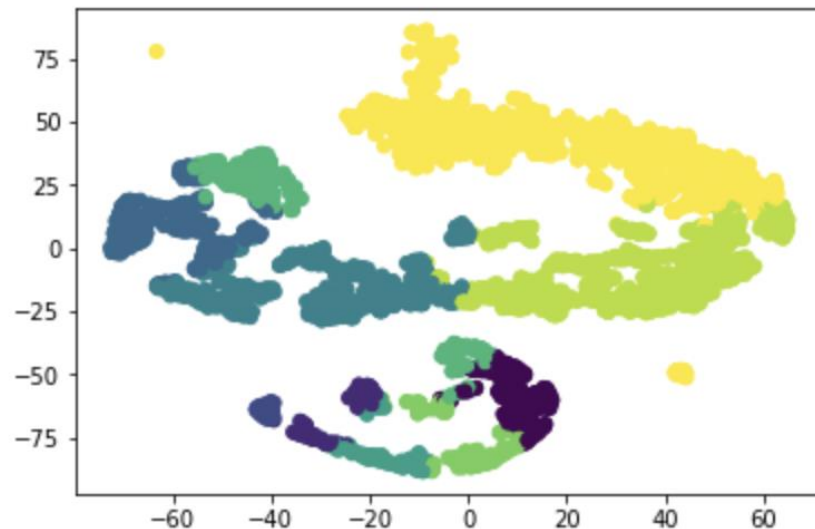
```
(4601, 2)
```

```
vis_x = data_embedded[:, 0]  
vis_y = data_embedded[:, 1]  
plt.scatter(vis_x, vis_y, cmap=plt.cm.get_cmap("jet", 10))  
plt.show()
```

Plot the result with t-SNE



Before clustering

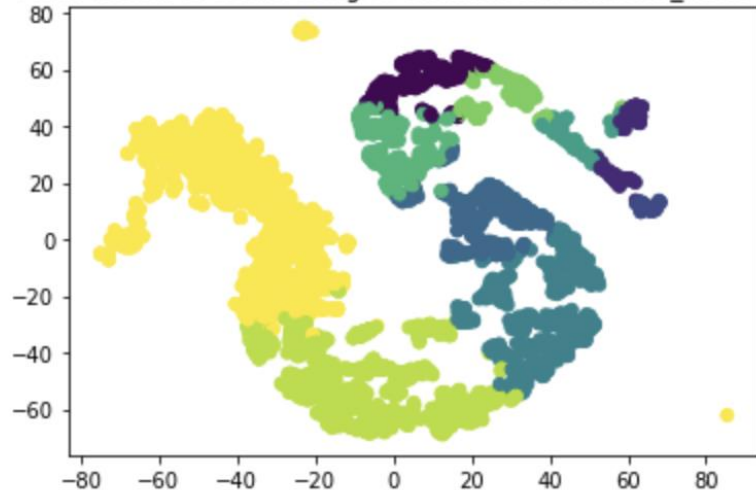


After clustering

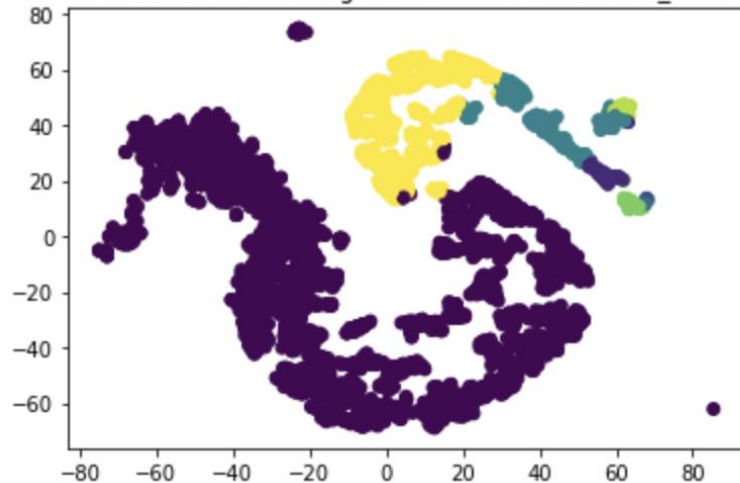
Each point is a document. Colors represent clusters assigned by K-means after dimensionality reduction via t-SNE.

Plot the result with t-SNE (set random state)

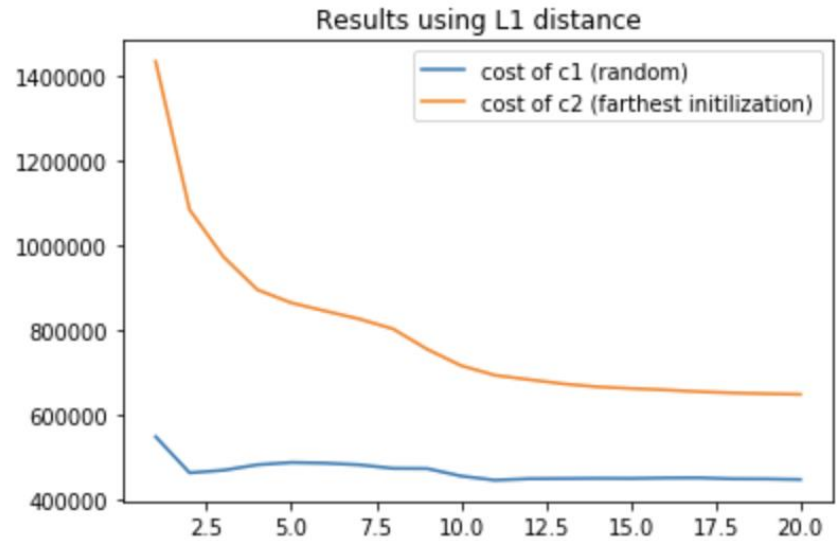
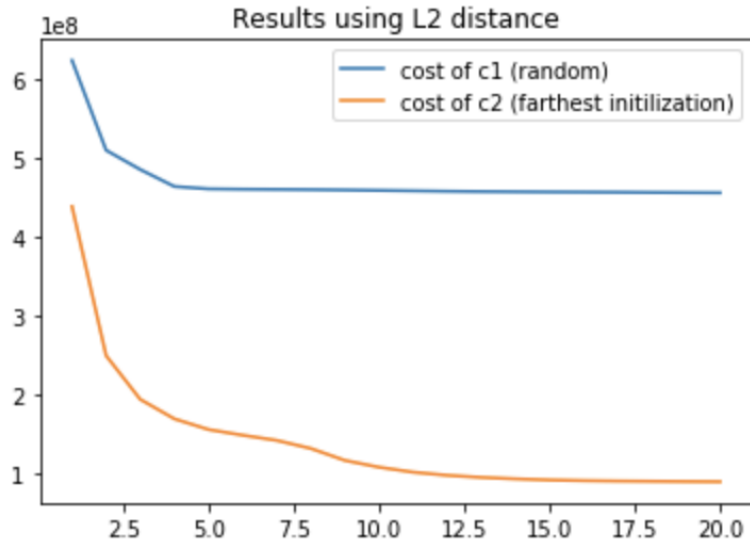
t-SNE results of c1.txt using L2 distance and random_state=100



t-SNE results of c2.txt using L2 distance and random_state=100



Plot the cost of each iteration



Spark MLlib

- Spark's scalable machine learning library
- Tools:
 - ML Algorithms: classification, regression, clustering, and collaborative filtering
 - Featurization: feature extraction, transformation, dimensionality reduction, and selection
 - Pipelines: tools for constructing, evaluating, and tuning ML Pipelines
 - Persistence: saving and load algorithms, models, and Pipelines
 - Utilities: linear algebra, statistics, data handling, etc.

Example: K-means clustering with Spark MLlib

```
from pyspark.mllib.clustering import KMeans
```

```
clusters = KMeans.train(data, 10, maxIterations=20, initializationMode="random")
```

```
# cluster centers  
len(clusters.centers)
```

10

HW1 Part 2 Binary classification with Spark MLlib

- Dataset: Adult dataset from UCI Machine Learning Repository
- Goal: Given information of a person, predict if the person could earn > 50k per year



Index of /ml/

- [Parent Directory](#)
- [Index](#)
- [adult.data](#)
- [adult.names](#)
- [adult.test](#)
- [old.adult.names](#)

Apache/2.4.6 (CentOS) Open

HW1 Part 2

Workflow

- **Data loading:** load data into Dataframe

Spark Dataframe Guide: <https://spark.apache.org/docs/latest/sql-getting-started.html>

```
#-----  
data.show(3)
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+  
|_c0|_c1|_c2|_c3|_c4|_c5|_c6|_c7|_c8|_c9|_c10|_c11|_c12|  
+_c13|_c14|  
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+  
| 39|State-gov| 77516.0| Bachelors|13.0| Never-married| Adm-clerical| Not-in-family| White| Male|2174.0| 0.0|40.0| United-  
States| <=50K|  
| 50|Self-emp-not-inc| 83311.0| Bachelors|13.0| Married-civ-spouse| Exec-managerial| Husband| White| Male| 0.0| 0.0|13.0| United-  
States| <=50K|  
| 38|Private|215646.0| HS-grad| 9.0| Divorced| Handlers-cleaners| Not-in-family| White| Male| 0.0| 0.0|40.0| United-  
States| <=50K|  
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+  
only showing top 3 rows
```

HW1 Part 2 Binary classification with Spark MLlib

Workflow

- **Data preprocessing:** Convert the categorical variables into numeric variables with ML Pipelines and Feature Transformers

```
dataset = df
```

```
root
|-- age: integer (nullable = true)
|-- workclass: string (nullable = true)
|-- fnlwt: double (nullable = true)
|-- education: string (nullable = true)
|-- education_num: double (nullable = true)
|-- marital_status: string (nullable = true)
|-- occupation: string (nullable = true)
|-- relationship: string (nullable = true)
|-- race: string (nullable = true)
|-- sex: string (nullable = true)
|-- capital_gain: double (nullable = true)
|-- capital_loss: double (nullable = true)
|-- hours_per_week: double (nullable = true)
|-- native_country: string (nullable = true)
|-- income: string (nullable = true)
```

age	workclass	fnlwt	education	education_num	marital_status	occupation	relationship	race	sex	capital_gain	capital_loss	hours_per_week	native_country	income
39	State-gov	77516.0	Bachelors	13.0	Never-married	Adm-clerical	Not-in-family	White	Male	2174.0	0.0	40.0	United-States	<=50K
50	Self-emp-not-inc	83311.0	Bachelors	13.0	Married-civ-spouse	Exec-managerial	Husband	White	Male	0.0	0.0	40.0	United-States	<=50K

only showing top 2 rows

```
In [43]: preppedDataDF.take(3)
```

```
Out[43]: [Row(age=39, workclass= 'State-gov', fnlwt=77516.0, education= 'Bachelors', education_num=13.0, marital_status= 'Never-married', occupation= 'Adm-clerical', relationship= 'Not-in-family', race= 'White', sex= 'Male', capital_gain=2174.0, capital_loss=0.0, hours_per_week=40.0, native_country= 'United-States', income= ' <=50K', workclassIndex=4.0, workclassclassVec=SparseVector(8, [4: 1.0]), educationIndex=2.0, educationclassVec=SparseVector(15, [2: 1.0]), marital_statusIndex=1.0, marital_statusclassVec=SparseVector(6, [1: 1.0]), occupationIndex=3.0, occupationclassVec=SparseVector(14, [3: 1.0]), relationshipIndex=1.0, relationshipclassVec=SparseVector(5, [1: 1.0]), raceIndex=0.0, raceclassVec=SparseVector(4, [0: 1.0]), sexIndex=0.0, sexclassVec=SparseVector(1, [0: 1.0]), native_countryIndex=0.0, native_countryclassVec=SparseVector(41, [0: 1.0]), label=0.0, features=SparseVector(100, [4: 1.0, 10: 1.0, 24: 1.0, 32: 1.0, 44: 1.0, 48: 1.0, 52: 1.0, 53: 1.0, 94: 39.0, 95: 77516.0, 96: 13.0, 97: 2174.0, 99: 40.0])),
Row(age=50, workclass= 'Self-emp-not-inc', fnlwt=83311.0, education= 'Bachelors', education_num=13.0, marital_status= 'Married-civ-spouse', occupation= 'Exec-managerial', relationship= 'Husband', race= 'White', sex= 'Male', capital_gain=0.0, capital_loss=0.0, hours_per_week=40.0, native_country= 'United-States', income= ' <=50K', workclassIndex=1.0, workclassclassVec=SparseVector(8, [1: 1.0]), educationIndex=2.0, educationclassVec=SparseVector(15, [2: 1.0]), marital_statusIndex=0.0, marital_statusclassVec=SparseVector(6, [0: 1.0]), occupationIndex=2.0, occupationclassVec=SparseVector(14, [2: 1.0]), relationshipIndex=0.0, relationshipclassVec=SparseVector(5, [0: 1.0]), raceIndex=0.0, raceclassVec=SparseVector(4, [0: 1.0]), sexIndex=0.0, sexclassVec=SparseVector(1, [0: 1.0]), native_countryIndex=0.0, native_countryclassVec=SparseVector(41, [0: 1.0]), label=0.0, features=SparseVector(100, [1: 1.0, 10: 1.0, 23: 1.0, 31: 1.0, 43: 1.0, 48: 1.0, 52: 1.0, 53: 1.0, 94: 50.0, 95: 83311.0, 96: 13.0, 99: 13.0])),
Row(age=38, workclass= 'Private', fnlwt=215646.0, education= 'HS-grad', education_num=9.0, marital_status= 'Divorced', occupation= 'Handle rs-cleaners', relationship= 'Not-in-family', race= 'White', sex= 'Male', capital_gain=0.0, capital_loss=0.0, hours_per_week=40.0, native_country= 'United-States', income= ' <=50K', workclassIndex=0.0, workclassclassVec=SparseVector(8, [0: 1.0]), educationIndex=0.0, educationclassVec=SparseVector(15, [0: 1.0]), marital_statusIndex=2.0, marital_statusclassVec=SparseVector(6, [2: 1.0]), occupationIndex=9.0, occupationclassVec=SparseVector(14, [9: 1.0]), relationshipIndex=1.0, relationshipclassVec=SparseVector(5, [1: 1.0]), raceIndex=0.0, raceclassVec=SparseVector(4, [0: 1.0]), sexIndex=0.0, sexclassVec=SparseVector(1, [0: 1.0]), native_countryIndex=0.0, native_countryclassVec=SparseVector(41, [0: 1.0]), label=0.0, features=SparseVector(100, [0: 1.0, 8: 1.0, 25: 1.0, 38: 1.0, 44: 1.0, 48: 1.0, 52: 1.0, 53: 1.0, 94: 38.0, 95: 215646.0, 96: 9.0, 99: 40.0]))]
```

HW1 Part 2 Binary classification with Spark MLlib

Workflow

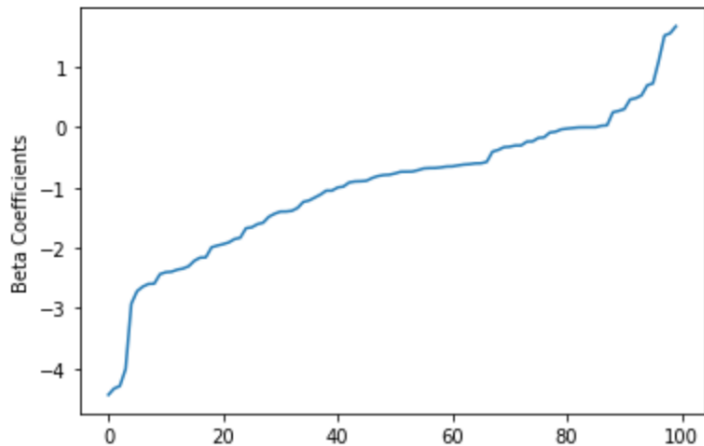
- **Modeling (9 models) :**
 - Logistic Regression
 - Random Forest
 - Naive Bayes
 - Decision Tree
 - Gradient Boosting Trees
 - Multi-layer perceptron
 - Linear Support Vector Machine
 - One-vs-Rest

HW1 Part 2 Binary classification with Spark MLlib

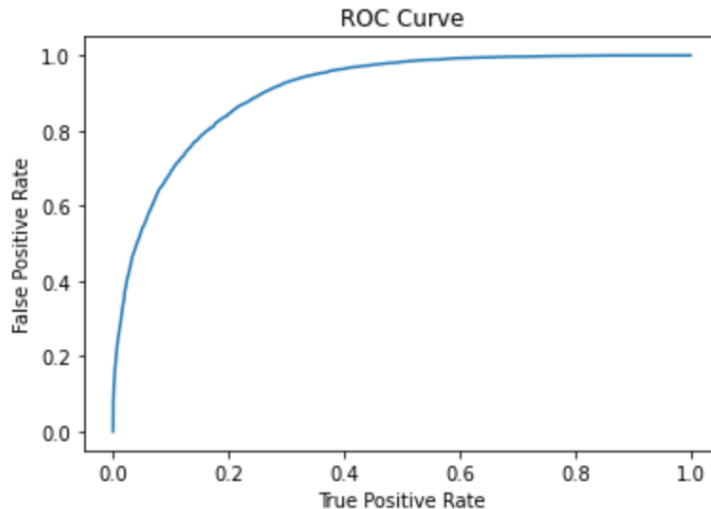
Workflow

- **Evaluation (Logistic Regression)**

```
plt.ylabel('Beta Coefficients')  
plt.show()
```



```
print('Training set areaUnderROC: ' + str(trainingSummary.areaUnderROC))
```

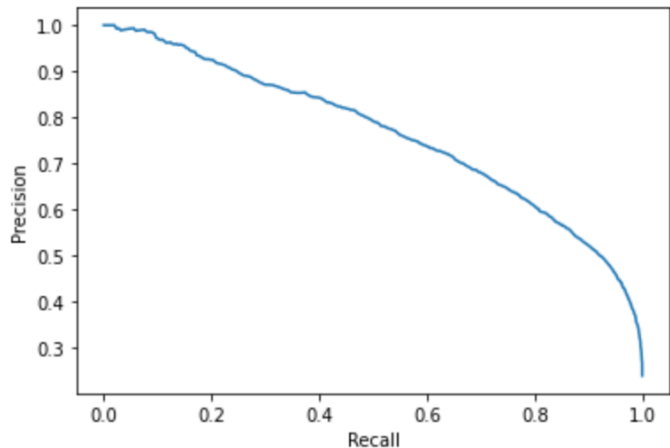


HW1 Part 2 Binary classification with Spark MLlib

- Workflow

- Evaluation (Logistic Regression)

```
plt.ylabel('Precision')  
plt.xlabel('Recall')  
plt.show()
```



```
print("Accuracy: %s\nFPR: %s\nTPR: %s\nF-measure: %s\nPrecision: %s\nRecall: %s"  
      % (accuracy, falsePositiveRate, truePositiveRate, fMeasure, precision, recall))
```

```
Accuracy: 0.8526629292221444  
FPR: 0.3172889625959339  
TPR: 0.8526629292221444  
F-measure: 0.8475083896808702  
Precision: 0.8464035949642436  
Recall: 0.8526629292221444
```

```
evaluator.evaluate(predictions)
```

```
Out[54]: 0.8993574699928725
```

```
In [55]: # accuracy  
correct = float(predictions.filter(pred  
total = float(predictions.count())  
print(correct, total, correct/total)
```

```
8218.0 9729.0 0.8446911296124987
```

References

- <https://spark.apache.org/docs/latest/sql-getting-started.html>
- <https://www.analyticsvidhya.com/blog/2016/10/spark-dataframe-and-operations/>
- <https://spark.apache.org/docs/latest/ml-guide.html>
- <https://towardsdatascience.com/machine-learning-with-pyspark-and-mllib-solving-a-binary-classification-problem-96396065d2aa>

E6893 Big Data Analytics

HOMEWORK 1 TUTORIAL



COLUMBIA ENGINEERING

The Fu Foundation School
of Engineering and Applied Science



QUESTION 1 - PROBLEM SETUP

Implement iterative K-means in Spark. We've provided you with starter code `kmeans.py` on Canvas, which takes care of data loading. **Complete the logic inside the for loop, and run the code with different initialization strategies and loss functions.** Feel free to change the code if needed, or paste the code into a Jupyter notebook. Take a screenshot of your code and results in your report

(1) Run clustering on `data.txt` with `c1.txt` and `c2.txt` as initial centroids and use **L1 distance as similarity measurement**. Compute and plot the within-cluster cost for each iteration. You'll need to submit **two graphs here**.

(2) Run clustering on `data.txt` with `c1.txt` and `c2.txt` as initial centroids and use **L2 distance as similarity measurement**. Compute and plot the within-cluster cost for each iteration. You'll need to submit two graphs here.

(3) t-SNE is a dimensionality reduction method particularly suitable for visualization of high-dimensional data. Visualize your clustering assignment result of (2) by reducing the dimension to a 2D space. You'll need to submit two graphs here.

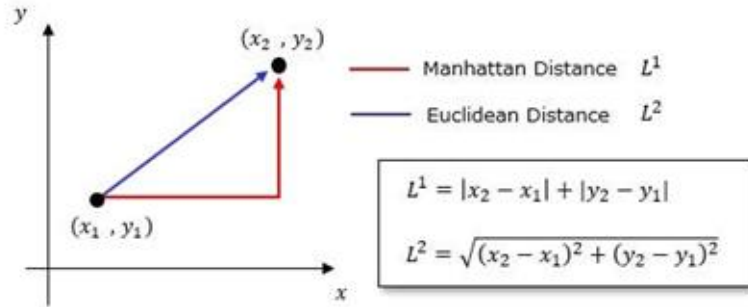
(4) For L2 and L1 distance, are random initialization of K-means using `c1.txt` better than initialization using `c2.txt` in terms of cost? Explain your reasoning.

(5) What is the time complexity of the iterative K-means?

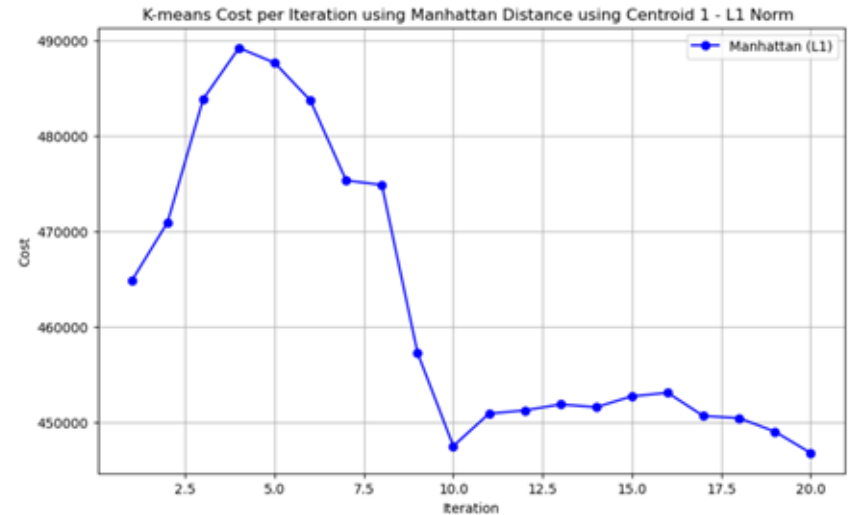
QUESTION 1 - PART 1

Restate the task

- **Input:**
 - `data.txt` → 4601 docs × 58 features
 - `c1.txt` and `c2.txt` → two sets of initial centroids
- **Distance function:** L1 (Manhattan)
- **Iterations:** 20, with $k = 10$
- **Output:** Plot within-cluster cost vs. iteration (two graphs: c1-init, c2-init)



Different from L2: adds absolute differences (linear growth) !

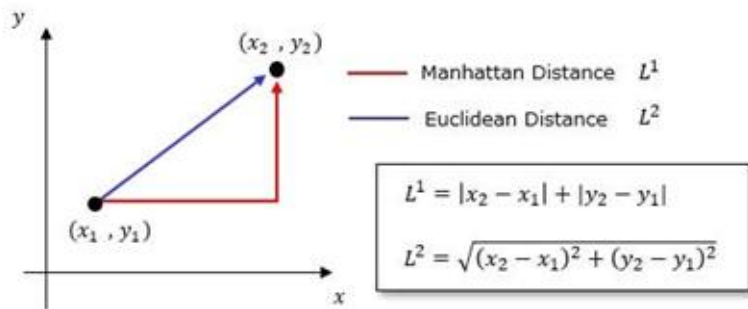


(please don't use this as your answer)

QUESTION 1 - PART 1

Restate the task

- **Input:**
 - `data.txt` → 4601 docs × 58 features
 - `c1.txt` and `c2.txt` → two sets of initial centroids
- **Distance function:** L1 (Manhattan)
- **Iterations:** 20, with $k = 10$
- **Output:** Plot within-cluster cost vs. iteration (two graphs: c1-init, c2-init)



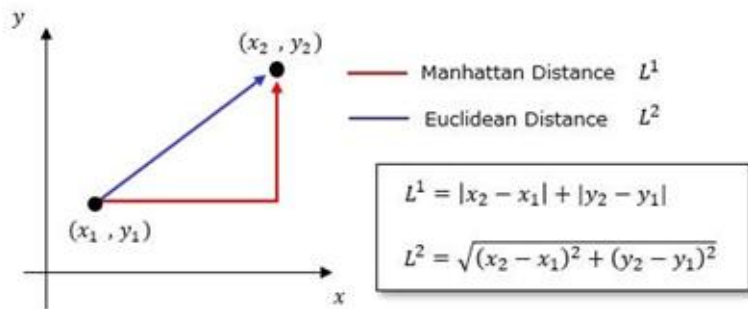
For **each iteration (1 to N)**:

1. Assign each document → nearest centroid using **L1**
(Spark: `map` each point → cluster ID)
2. Compute **cost**: sum of distances of each point to its assigned centroid.
(Spark: `map` distances → `reduce` to total)
3. Recompute centroids (mean of assigned points, unless you implement true median for L1).

QUESTION 1 - PART 2

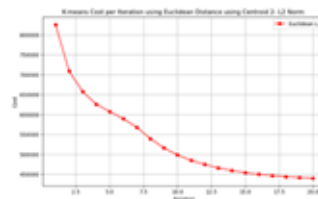
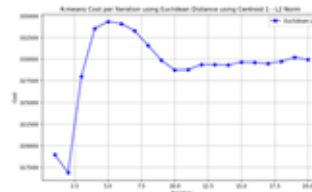
Restate the task / Similar to L1 Implementation

- **Input:**
 - `data.txt` → 4601 docs × 58 features
 - `c1.txt` and `c2.txt` → two sets of initial centroids
- **Distance function:** L2 (Euclidean)
- **Iterations:** 20, with $k = 10$
- **Output:** Plot within-cluster cost vs. iteration (two graphs: c1-init, c2-init)



For each iteration (1 to N):

1. Assign each document → nearest centroid using **L2**
2. (Spark: `map` each point → cluster ID)
3. Compute **cost**: sum of distances of each point to its assigned centroid.
(Spark: `map` distances → `reduce` to total)
4. Recompute centroids (mean of assigned points, unless you implement true median for L1).

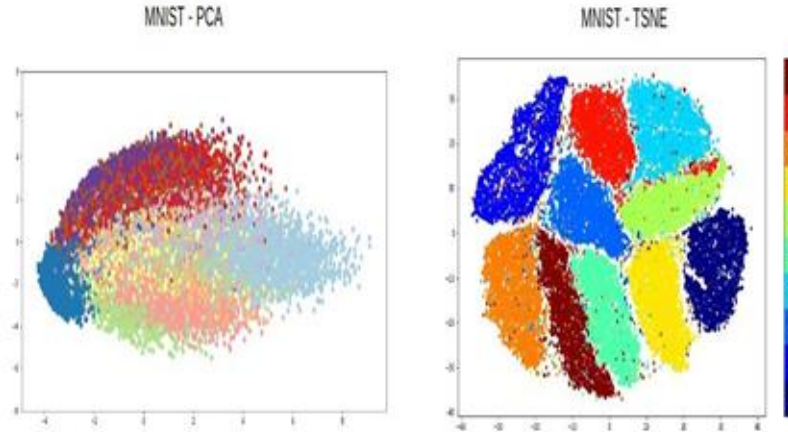


QUESTION 1 - PART 3

t-SNE : t-distributed Stochastic Neighbor Embedding

A **dimensionality reduction** algorithm (like PCA), but optimized for **visualization**.

Take **high-dimensional data** (58-D documents here) and show it in **2D** (or 3D) while **preserving neighborhood structure**.



Feature	PCA	t-SNE
Type	Linear	Non-linear
Goal	Capture max variance	Preserve local neighborhoods
Good for	Dimensionality reduction before ML	Visualization (2D/3D)
Global vs Local	Preserves global structure	Preserves local structure
Speed	Fast	Slower (iterative optimization)
Output	Any # of dims	Usually 2D or 3D only

QUESTION 2 - BINARY CLASSIFICATION ON SPARK

Frame the problem

- Dataset: **Adult (Census Income)** ~48k rows.
Goal: predict **income $\geq$ 50K?** → **binary classification**.
- Deliverables: **load** → **preprocess** → **train 9 models** → **evaluate & rank accuracy**.

Data loading

- Read CSV as **DataFrame** with `inferSchema` and header handling.
- **Rename columns** to:

```
["age", "workclass", "fnlwgt", "education", "education_num", "marital_status", "occupation",  
 "relationship", "race", "sex", "capital_gain", "capital_loss", "hours_per_week", "native_country", "income"]
```

- Check **schema & nulls/‘?’**.
- “Which columns look categorical vs numerical?”
- “What should we do with ‘?’ entries?”

QUESTION 2 - BINARY CLASSIFICATION ON SPARK

Preprocessing

Goal: turn mixed schema → single **features** vector + **label**.

1. **Clean string columns** (trim/lowercase; replace ? with “Unknown”).
2. **StringIndexer** on each categorical col (fit on **train only!**).
3. **OneHotEncoder** on indexed cats (dropLast=true).
Assemble numeric + OHE categorical into **features**.
4. **Label:** index **income** → {0, 1} (e.g., ≤50K→0, >50K→1).
5. **Split:** 70/30, **seed=100**.

Some Questions?

- “Why index *before* one-hot?”
- “Why assemble after encoding?”
- “Why split before fitting transformers?” (data leakage)

QUESTION 2 - BINARY CLASSIFICATION ON SPARK

Modeling (9 estimators)

The 9 models (Spark MLlib)

1. **Logistic Regression (LR)** – linear decision boundary; probs via sigmoid.
Hyperparams to mention: `regParam`, `elasticNetParam`, `maxIter`.
2. **Random Forest (RF)** – ensemble of trees; handles nonlinearity; robust.
`numTrees`, `maxDepth`, `featureSubsetStrategy`.
Naive Bayes (NB) – assumes feature independence; best with counts; try `multinomial`.
3. **Decision Tree (DT)** – interpretable; prone to overfit; `maxDepth`, `minInstancesPerNode`.
4. **Gradient-Boosted Trees (GBT)** – strong accuracy; `maxIter`, `maxDepth`. (Binary only.)
5. **Multilayer Perceptron (MLP)** – simple feedforward NN; specify **layers** like `[input, h1, ..., 2]`.
6. **Linear SVM (LSVC)** – margin-based linear classifier; fast; no probabilities by default.
One-vs-Rest (OvR) – meta-wrapper for multi-class; here it will just wrap a base linear model

Reuse the same preprocessed **features** for all models.

Wrap each model in a small **Pipeline**: `[indexers → encoder → assembler → estimator]`

QUESTION 2 - BINARY CLASSIFICATION ON SPARK

Modeling (9 estimators)

The 9 models (Spark MLlib)

1. **Logistic Regression (LR)** – linear decision boundary; probs via sigmoid.
Hyperparams to mention: `regParam`, `elasticNetParam`, `maxIter`.
2. **Random Forest (RF)** – ensemble of trees; handles nonlinearity; robust.
`numTrees`, `maxDepth`, `featureSubsetStrategy`.
Naive Bayes (NB) – assumes feature independence; best with counts; try `multinomial`.
3. **Decision Tree (DT)** – interpretable; prone to overfit; `maxDepth`, `minInstancesPerNode`.
4. **Gradient-Boosted Trees (GBT)** – strong accuracy; `maxIter`, `maxDepth`. (Binary only.)
5. **Multilayer Perceptron (MLP)** – simple feedforward NN; specify **layers** like `[input, h1, ..., 2]`.
6. **Linear SVM (LSVC)** – margin-based linear classifier; fast; no probabilities by default.
One-vs-Rest (OvR) – meta-wrapper for multi-class; here it will just wrap a base linear model

Reuse the same preprocessed **features** for all models.

Wrap each model in a small **Pipeline**: `[indexers → encoder → assembler → estimator]`



EECS E6893 Big Data Analytics

Spark Dataframe, Spark SQL

Agenda

- Spark Dataframe
- Spark SQL
- Hadoop metrics

Spark Dataframe

- An *abstraction*, an immutable distributed collection of data like RDD
- Data is organized into named columns, like a table in DB
- Create from RDD, Hive table, or other data sources
- Easy conversion with Pandas Dataframe

Spark Dataframe: read from csv file

```
# read data from csv into Dataframe
df = spark.read.format("csv").option("header", 'true').load("gs://big_data_ta/data/citibike_stations.csv")
```

```
type(df)
```

```
pyspark.sql.dataframe.DataFrame
```

```
df.show(1)
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+
|station_id|          name|short_name|  latitude|  longitude|region_id|rental_methods|capacity|eightd_has_key
_dispenser|num_bikes_available|num_bikes_disabled|num_docks_available|num_docks_disabled|is_installed|is_renting|is_r
eturning|eightd_has_available_keys|      last_reported|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+
|      382|University Pl & E...|  5905.11|40.73492695|-73.99200509|      71|KEY,CREDITCARD|      0|
false|          0|          0|          0|          0|      false|      false|fa
lse|      false|1970-01-01 00:00:00|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+
only showing top 1 row
```

Spark Dataframe: common operations

```
df.printSchema()
```

```
root
|-- station_id: string (nullable = true)
|-- name: string (nullable = true)
|-- short_name: string (nullable = true)
|-- latitude: string (nullable = true)
|-- longitude: string (nullable = true)
|-- region_id: string (nullable = true)
|-- rental_methods: string (nullable = true)
|-- capacity: string (nullable = true)
|-- eightd_has_key_dispenser: string (nullable = true)
|-- num_bikes_available: string (nullable = true)
|-- num_bikes_disabled: string (nullable = true)
|-- num_docks_available: string (nullable = true)
|-- num_docks_disabled: string (nullable = true)
|-- is_installed: string (nullable = true)
|-- is_renting: string (nullable = true)
|-- is_returning: string (nullable = true)
|-- eightd_has_available_keys: string (nullable = true)
|-- last_reported: string (nullable = true)
```

```
df.count()
```

Spark Dataframe: common operations

```
df.columns
```

```
['station_id',  
 'name',  
 'short_name',  
 'latitude',  
 'longitude',  
 'region_id',  
 'rental_methods',  
 'capacity',  
 'eightd_has_key_dispenser',  
 'num_bikes_available',  
 'num_bikes_disabled',  
 'num_docks_available',  
 'num_docks_disabled',  
 'is_installed',  
 'is_renting',  
 'is_returning',  
 'eightd_has_available_keys',  
 'last_reported']
```

Spark Dataframe: common operations

```
df.describe().show()
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
|summary|station_id|name|short_name|latitude|longitude|region_id|rental_methods|capacity|eightd_has_key_dispenser|num_bikes_available|num_bikes_disabled|num_docks_available|num_docks_disabled|is_installed|is_renting|is_returning|eightd_has_available_keys|last_reported|
+-----+-----+-----+-----+-----+-----+-----+-----+
|count|843|843|843|843|843|843|843|843|843|843|843|843|843|843|843|843|843|843| |
|mean|2434.425860023725|null|5806.786515151487|40.73212559944772|-73.9749901186049|70.93950177935943|null|31.419928825622776|null|14.565836298932384|0.5693950177935944|16.18979833926453|0.05219454329774614|null|null|null|null|null|
|stddev|1421.1204113008778|null|1175.6743390458948|0.0387451696148341|0.031207687758326202|0.23854913482063428|null|12.052012437572532|null|11.188256063195926|0.8613434732614029|13.158075664204775|0.6499620307701626|null|null|null|null|null|
|min|116|1 Ave & E 110 St|3460.01|40.655399774478312|-73.9077436|70|KEY,CREDITCARD|0|false|false|false|1970-01-01 00:00:00|0|0|false|0|false|false|false|
|max|83|York St|JC106|40.814394437915816|-74.0836394|71|KEY,CREDITCARD|79|true|9|6|9|9|true|true|true|true|2019-09-02 00:00:00|
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
```

Spark Dataframe: common operations

```
df.describe('capacity').show()
```

summary	capacity
count	843
mean	31.419928825622776
stddev	12.052012437572532
min	0
max	79

```
df.select('station_id').distinct().count()
```

```
843
```

Spark Dataframe: conversion with Pandas

```
# conversion with Pandas
```

```
import pandas as pd
```

```
pandaDf = df.toPandas()
```

```
pandaDf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 843 entries, 0 to 842
```

```
Data columns (total 18 columns):
```

```
station_id      843 non-null object
```

```
name            843 non-null object
```

```
short_name      843 non-null object
```

```
latitude        843 non-null object
```

```
longitude       843 non-null object
```

```
region_id       843 non-null object
```

```
rental_methods  843 non-null object
```

```
capacity        843 non-null object
```

```
eightd_has_key_dispenser 843 non-null object
```

```
num_bikes_available 843 non-null object
```

```
num_bikes_disabled 843 non-null object
```

```
num_docks_available 843 non-null object
```

```
num_docks_disabled 843 non-null object
```

```
is_installed     843 non-null object
```

```
is_renting       843 non-null object
```

```
is_returning     843 non-null object
```

```
eightd_has_available_keys 843 non-null object
```

```
last_reported    843 non-null object
```

```
dtypes: object(18)
```

```
memory usage: 118.6+ KB
```

Work with Spark SQL

```
# Play with Spark SQL  
# Register the DataFrame as a SQL temporary view  
df.createOrReplaceTempView("citibike")
```

```
sqlDF = spark.sql("""  
    SELECT COUNT (DISTINCT station_id)  
    FROM citibike  
    """)  
sqlDF.show()
```

```
+-----+  
|count(DISTINCT station_id)|  
+-----+  
|                        843|  
+-----+
```

```
# get data out of df  
sqlDF.select("count(DISTINCT station_id)").collect()[0]["count(DISTINCT station_id)"]
```

843

References

- <https://spark.apache.org/docs/latest/sql-getting-started.html>
- <https://www.analyticsvidhya.com/blog/2016/10/spark-dataframe-and-operations/>
- <https://spark.apache.org/docs/latest/ml-guide.html>
- <https://towardsdatascience.com/machine-learning-with-pyspark-and-mllib-solving-a-binary-classification-problem-96396065d2aa>
- <https://www.datadoghq.com/blog/collecting-hadoop-metrics/#namenode-and-datanode-metrics-via-jmx>
- <https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/Metrics.html>