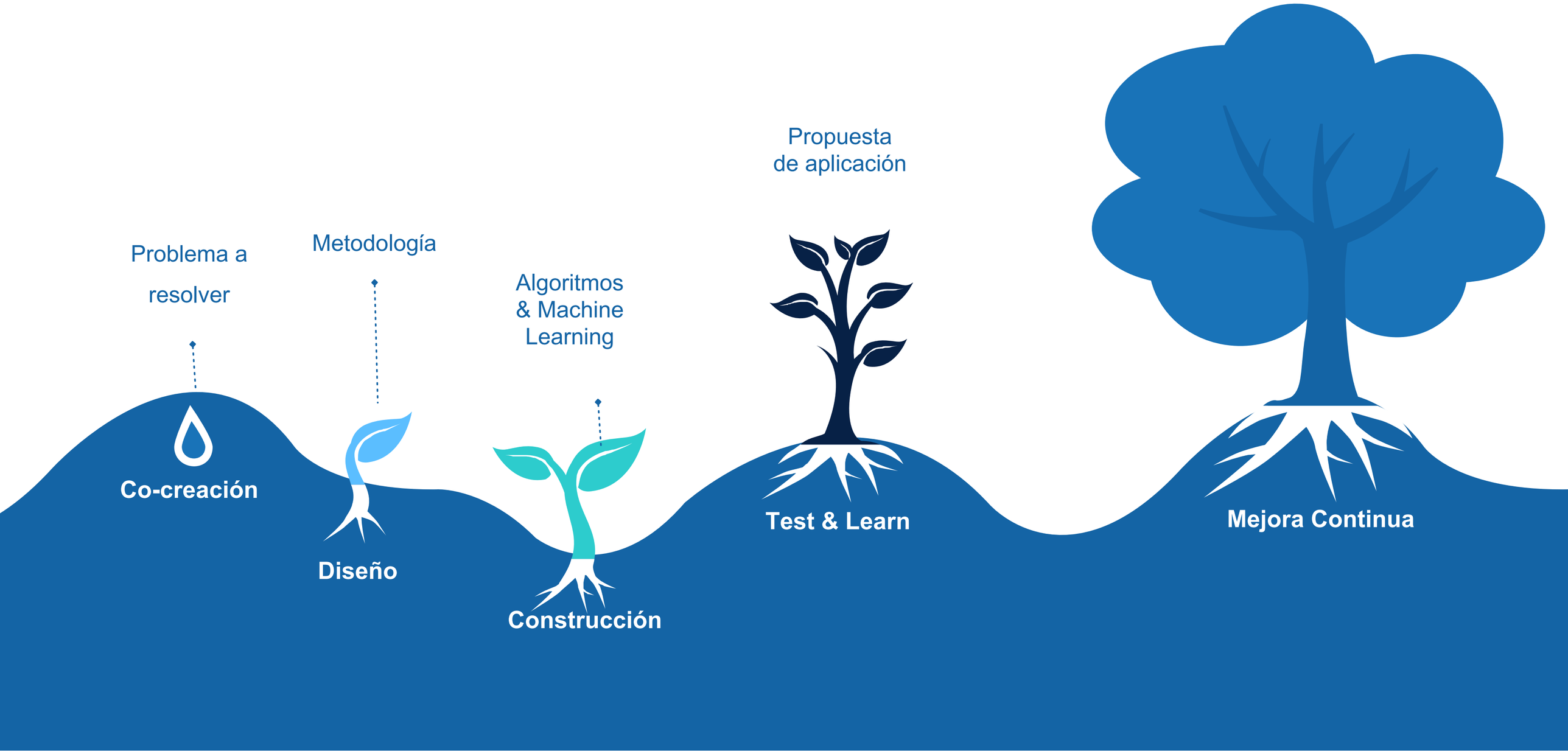
# KNN Algorithm Machine Learning Classification Parallel Implementation

LUIS BALAREZO
Software Engineering Student

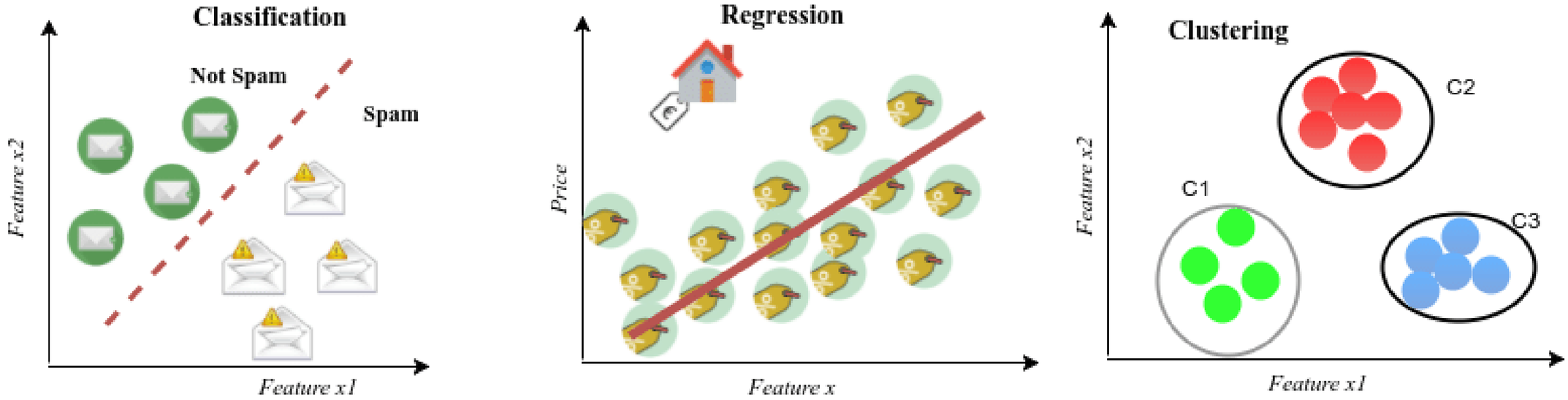
February 2024


# Contents


Problema a resolver
Co-creación
Metodología
Diseño
Algoritmos & Machine Learning
Construcción
Propuesta de aplicación
Test & Learn
Mejora Continua

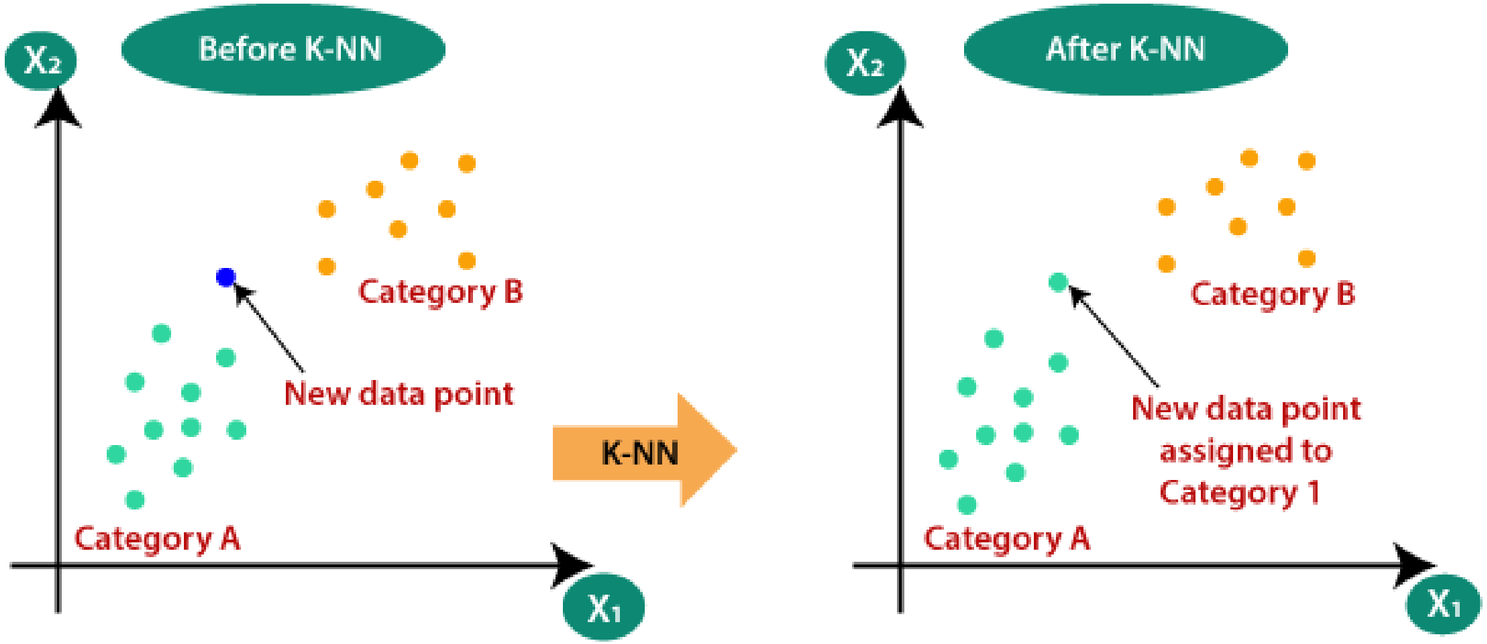
# Introduction

Classification is an important technique of Machine Learning in which a model will be trained on a data set with labels to predict a label for a new object. An algorithm, namely K nearest neighbors (KNN), is grouped into Classification problem with a purpose for producing the class label for a new instance by a major voting on KNN category



# Motivation

Although KNN is quite simple and can produce good results in some cases, it is still inefficient when working with large, multiple featured data. The reason is that KNN might take a huge cost of computation for the distances between the instance data point to all training data points. Therefore, parallel computation is essential to solve the above disadvantage. This is also the major motivation and goal of the project



# KNN Fundamentals

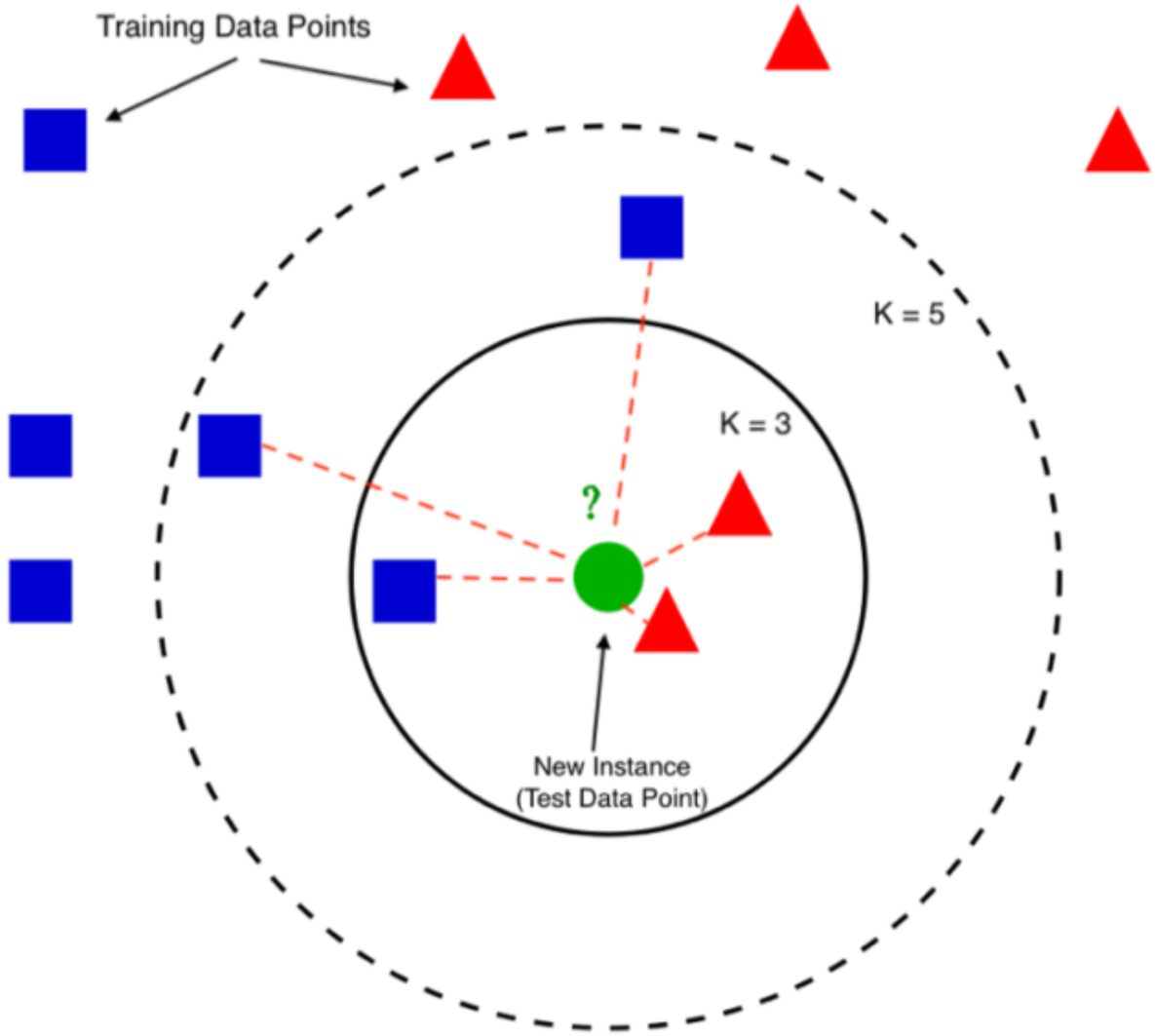
## How KNN works?

- K Nearest Neighbors (KNN) is known as one of basic supervised Machine Learning algorithms. It is an instance-based algorithm to solve the Classification problem in Machine Learning

- The algorithm assigns a class to an instance based on the majority of the classes of its k nearest neighbors.

# KNN Fundamentals

- KNN does not train the model based on the training set. Instead, KNN predicts the class that a new instance (from the test set) belongs to by major voting on K nearest neighbors of that instance, found by calculating the distance between the instance and all data points in the training set



# Problem to solve

## Model training

Training the model does not involve any complications

## Making the model predictions

### High computational cost

- We have to look for the points closest to a certain given point.
- This apparently implies searching linearly through all the data.
- This is not good, it really is not good.


# Dataset

# Dataset

## Credit Card Fraud

- To perform the experiment for KNN, a real data with a relatively huge number of records and many features should be loaded. The following table demonstrates a brief overview of data samples:

| | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | V10 | V11 | V12 | V13 | V14 | V15 | V16 | V17 | V18 | V19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1.256228823... | -0.506943887... | 0.482417759... | 0.115657564... | -1.022264180... | -0.522546894... | -0.415670038... | -0.057158818... | -0.912657584... | 0.79662333... | 0.944966455... | 0.822816799... | -0.046876568... | 0.077825834... | -1.137199184... | -1.747086462... | 0.160523663... | 1.096806591... | -0.33 |
| 2 | 0.944330375... | -1.830558753... | -0.311509123... | -1.230705799... | -1.105964157... | 0.059691839... | -0.437649962... | -0.136275125... | -2.245879406... | 1.356976531... | 1.091572102... | 0.460466783... | 1.371581369... | -0.299259080... | -0.892198646... | -0.994788364... | 0.732843796... | -0.242082716... | 0.01 |
| 3 | 1.246314198... | 0.366946460... | 0.297958930... | 1.118263249... | -0.272566765... | -0.969191565... | 0.215064988... | -0.231450427... | -0.012874481... | 0.016361066... | -0.397618928... | 0.173088392... | -0.032009060... | 0.394807217... | 0.922598555... | 0.049843009... | -0.424111001... | -0.148721173... | -0.37 |
| 4 | -1.152825110... | 0.462866915... | 0.995502478... | 0.703442702... | 0.409894500... | -0.448309856... | -0.298842072... | 0.644459076... | -1.026941210... | -0.520883737... | 1.553125132... | 0.707760564... | -0.058300013... | 0.333154864... | 0.970176196... | 0.028399095... | 0.588620306... | 0.141831692... | 0.84 |
| 5 | 1.151825001... | 0.315841727... | 0.513422222... | 1.152739527... | -0.262600663... | -0.607619668... | 0.127760858... | -0.138175786... | -0.099943551... | -0.063738820... | 0.247758023... | 0.818838324... | 0.775078479... | 0.220189154... | 1.115554208... | -0.317490738... | -0.106470998... | -0.715426983... | -0.91 |
| 6 | -0.535774270... | -1.263040917... | 1.932969396... | -2.665949384... | -0.983061315... | -0.688778258... | -0.983746114... | 0.103225419... | -1.366597102... | 0.519185829... | -1.396361323... | -1.008109583... | 0.462428072... | -1.028318940... | -0.673949971... | 0.062635999... | 0.039522136... | 0.389353915... | -0.99 |
| 7 | -1.095775263... | 0.191872667... | 1.169597662... | -1.374673148... | -0.873063564... | -1.434041931... | 0.378825976... | 0.170079865... | -1.607298789... | -0.406239777... | -0.666918156... | 0.007572525... | 0.006155020... | 0.234771221... | -0.820900776... | -1.619158962... | 0.386971770... | 0.566545816... | -1.34 |
| 8 | 1.953224833... | -1.239730295... | -2.139097134... | -2.028096889... | -0.061366548... | -0.328376382... | -0.114860909... | -0.113020835... | 2.622596801... | -1.220184728... | -0.751415218... | 0.818119781... | -1.064386713... | 0.301483622... | -0.734358647... | -1.233741750... | -0.272199369... | 1.011000536... | 2.10 |
| 9 | -0.809799379... | -0.782676476... | 1.347029165... | -0.587108367... | -1.737703071... | -0.534112317... | 1.173839521... | -0.238622551... | -1.601029357... | 0.041356733... | -0.030290329... | -0.911154991... | 0.093608088... | -0.117711232... | 0.956579482... | 0.839588991... | 0.571523135... | -1.267180990... | 1.26 |
| 10 | 1.169427397... | 0.627229130... | 0.657882302... | 2.492570730... | -0.118150761... | -0.559891392... | 0.240835737... | -0.157698585... | -0.843037719... | 0.659861289... | -0.435088374... | 0.182176963... | 0.473687138... | 0.225795651... | 0.404178902... | 0.653762656... | -0.627747236... | -0.522143404... | -0.91 |
| 11 | 1.224215118... | -2.005040999... | 1.418589444... | -0.955406529... | -2.483135194... | 0.679499742... | -2.072197822... | 0.446719013... | -0.332826536... | 1.246474147... | -0.277494318... | -0.709374145... | -1.394854813... | -1.012488553... | -2.003253244... | -0.560399636... | 0.875682634... | 0.810264397... | 0.62 |
| 12 | 0.558096494... | -1.270713732... | -0.507024575... | -0.043515880... | -0.423402509... | -0.070655430... | 0.444575443... | -0.082018406... | 0.018798101... | -0.278998598... | 1.168004770... | 0.958614619... | -0.071665233... | 0.426229224... | -0.243280590... | -0.225009202... | 0.021573403... | -0.616442766... | 0.35 |
| 13 | 1.146379392... | -0.085976979... | 0.602791644... | 0.909603481... | -0.064318175... | 0.882654799... | -0.555275485... | 0.228359025... | 0.498324184... | -0.074955421... | -1.393444678... | 0.091751581... | 0.949551366... | -0.203289762... | 1.722089918... | 0.787043999... | -0.977036903... | 0.311439572... | -0.56 |
| 14 | 1.272833086... | 0.243313516... | 0.098921121... | 0.929030337... | -0.026214107... | -0.421218570... | 0.066568787... | -0.104569428... | 0.194345143... | 0.009632528... | -1.210944212... | -0.345992621... | -0.358743004... | 0.391691744... | 1.144328845... | 0.445408301... | -0.749228317... | 0.176623029... | -0.05 |
| 15 | 2.085799824... | -0.007503472... | -2.051017509... | 0.211331838... | 0.605291438... | -0.893311340... | 0.503455254... | -0.277635334... | 0.002208110... | 0.355756649... | 0.751126003... | 0.514154309... | -0.959033473... | 0.919623693... | -0.917714792... | -0.404184729... | -0.305976136... | -0.170261895... | 0.41 |
| 16 | -0.280825884... | 0.982833794... | -0.261327699... | -0.357156390... | 1.065531222... | -1.331703078... | 1.326842345... | -0.582912138... | 1.037273192... | -0.813015012... | -0.046429679... | -2.727209446... | 1.061295356... | 1.977857973... | -0.900450273... | -1.121908863... | 0.810147925... | -0.225098598... | 0.37 |

| V20 | V21 | V22 | V23 | V24 | V25 | V26 | V27 | V28 | Amount | Class |
|---|---|---|---|---|---|---|---|---|---|---|
| -0.492436622... | -0.444294985... | -0.703718469... | 0.020491339... | 0.550057163... | 0.403225511... | 0.337179761... | -0.012798239... | 0.008153787... | 28.0 | '0' |
| 0.257127128... | 0.024524687... | 0.025444184... | -0.360068196... | -0.243699799... | 0.554992306... | -0.064088600... | -0.015909852... | 0.041901701... | 285.27 | '0' |
| -0.148027873... | 0.026555774... | 0.125002197... | -0.097595299... | 0.397102217... | 0.686940178... | -0.328229693... | 0.013751854... | 0.020123148... | 3.3 | '0' |
| 0.163595302... | 0.010666348... | -0.359297428... | 0.072184007... | -0.014728305... | -0.703372946... | 0.173464643... | 0.024653440... | -0.034414492... | 0.89 | '0' |
| -0.109301482... | 0.136174347... | 0.497670612... | -0.061487402... | 0.428082411... | 0.587319726... | -0.290666714... | 0.040006232... | 0.025359899... | 14.9 | '0' |
| -0.225664477... | 0.002391110... | 0.274488164... | 0.149108265... | 0.005004020... | -0.638925799... | -0.390307160... | 0.179927122... | 0.192461363... | 40.55 | '0' |
| -0.462966485... | -0.239152946... | -0.488354252... | 0.020490313... | 0.946254981... | 0.116278912... | 0.895636551... | -0.154244997... | -0.015252710... | 64.99 | '0' |
| -0.047476882... | 0.311646886... | 1.169289555... | -0.475623415... | -1.295743292... | 0.792816558... | -0.326966375... | 0.016123023... | -0.071237306... | 110.66 | '0' |
| 1.001139929... | 0.253919672... | -0.053795796... | 0.914809572... | 0.657063766... | -0.420317155... | -0.484449724... | 0.017995760... | 0.175233238... | 358.78 | '0' |
| -0.102392261... | -0.181962430... | -0.587933020... | 0.073176116... | 0.365951952... | 0.363419227... | -0.190414687... | -0.011470737... | 0.028100258... | 15.14 | '0' |
| -0.276117672... | -0.157002749... | 0.054384565... | -0.122802784... | -0.004488174... | 0.293222006... | -0.045168301... | 0.079546037... | 0.030842562... | 96.0 | '0' |
| 0.645941078... | 0.172562550... | -0.178900722... | -0.408434258... | -0.164318061... | 0.272792578... | 1.464831240... | -0.178880361... | 0.032804755... | 364.11 | '0' |
| 0.020002871... | 0.088235280... | 0.264397066... | -0.200764707... | -1.309175371... | 0.458071351... | -0.237440647... | 0.073607522... | 0.032122571... | 50.0 | '0' |
| -0.127347120... | -0.109585097... | -0.332864205... | -0.125077258... | -0.476077444... | 0.632100741... | -0.386151209... | 0.015875365... | 0.017701025... | 13.99 | '0' |
| -0.256134173... | 0.137767674... | 0.500125276... | 0.007130828... | 0.826924367... | 0.336715484... | 0.660079497... | -0.126411140... | -0.089595096... | 0.76 | '0' |
| -0.236838920... | 0.041577291... | 0.438153673... | -0.406087759... | -0.080000876... | 0.483439690... | 0.285433028... | -0.326417473... | 0.022492698... | 10.0 | '0' |

# Dataset

## Credit Card Fraud

- The data are stored in csv-format files, and these data are about detection of credit fraud and is divided into training data and test data in prior. There are 227,845 training entries and 56,962 test entries

- For a specific entry, there are 29 float-datatype features/columns which will challenge the Euclidean distance computation and its last column shows up the class that the entry is credit fraud (‘1’) or not (‘0’).


# Code

# Main Concept

```python
class KNN:

    def __init__(self, k):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y
        valid = self.k <= len(X)
        if not valid:
            raise Exception('El valor de k debe ser menor que el tamaño de los datos de entrenamiento.')

    def predict(self, X):
        predicted_labels = [self._predict(x) for x in X]
        return predicted_labels
```

# Main Concept

```python
def _predict(self, x):
    # compute distance
    distances = [euclidean_distance(x, x_train) for x_train in self.X_train]
    # get k nearest samples, labels
    k_indices = np.argsort(distances)[0:self.k]
    k_nearest_labels = [self.y_train[i] for i in k_indices]
    # majority vote, most common label
    most_common_label = Counter(k_nearest_labels).most_common(1)[0][0]
    return most_common_label
```

# Main Concept

```python
def getLocalSortedDistances(self, X):
    k_local_sorted_distances = [self._getLocalSortedDistances(x) for x in X]
    return k_local_sorted_distances

def _getLocalSortedDistances(self, x):
    # array of tuples (distance, label)
    distances = []
    for i in range(0, len(self.X_train)):
        distance = (euclidean_distance(x, self.X_train[i]), self.y_train[i])
        distances.append(distance)
    # sort the array in ascending order by distance and pick k first elements
    k_local_sorted_distances = sorted(distances, key=lambda t: t[0])[0:self.k]
    return k_local_sorted_distances
```

```python
def euclidean_distance(p, q):
    return np.sqrt(np.sum((p-q)**2))
```

# Main Concept

```
def load_data():

    training_path = os.getcwd() + '/data/credit_train.csv'
    test_path = os.getcwd() + '/data/credit_test.csv'
    training_data = pd.read_csv(training_path)
    test_data = pd.read_csv(test_path)
    return training_data, test_data
```

```
def extract_data(data_records, training_data, test_data):
    # Extract a number of records according to testing cases
    X_train = training_data.iloc[:data_records,:5]
    y_train = training_data.iloc[:data_records,(training_data.shape[1]-1)]
    X_test = test_data.iloc[:500,:5]
    y_test = test_data.iloc[:500,(test_data.shape[1]-1)]
    # transform into numpy array for faster computation
    X_train = np.array(X_train)
    y_train = np.array(y_train)
    X_test = np.array(X_test)
    y_test = np.array(y_test)
    return X_train, y_train, X_test, y_test
```

# Sequential Version

**Fitting the model**

**Making predictions**

**Returning accuracy and predictions**

```
def run_sequential_KNN(k, X_train, y_train, X_test, y_test):
    knn = KNN(k)
    knn.fit(X_train, y_train)
    local_predictions = knn.predict(X_test)
    accuracy = accuracy_score(y_test, local_predictions)
    return accuracy, local_predictions
```

# Parallel Version

## Fitting the model

## Making predictions (parallel)

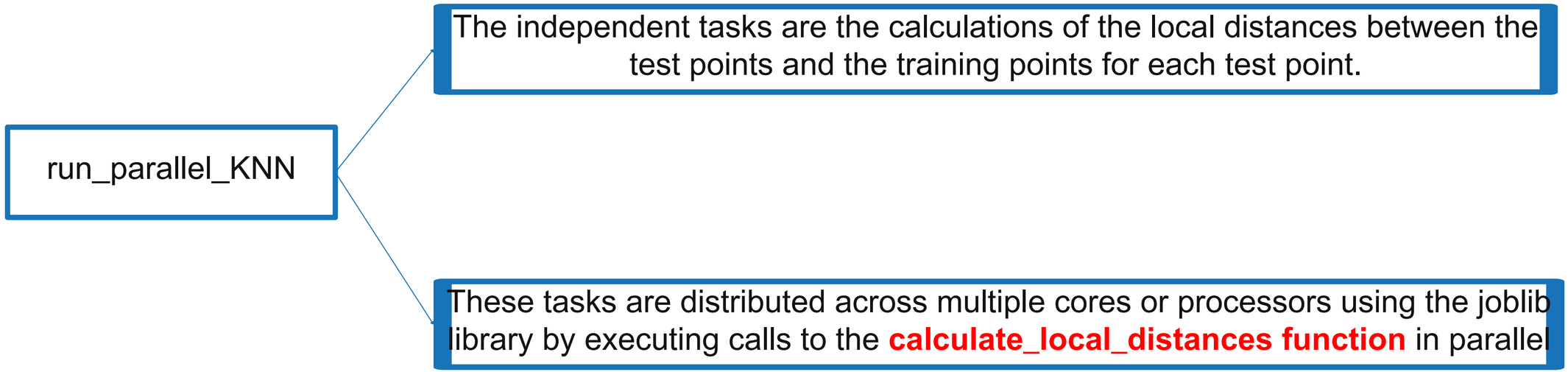
## Returning accuracy and predictions

```python
def run_parallel_KNN(n_jobs, k, X_train, y_train, X_test, y_test):
    knn = KNN(k)
    knn.fit(X_train, y_train)

    def calculate_local_distances(x):
        return knn.predict(np.array([x]))[0]  # Utilizando predict para predecir la etiqueta para x

    local_predictions = Parallel(n_jobs=n_jobs)(delayed(calculate_local_distances)(x) for x in X_test)

    accuracy = accuracy_score(y_test, local_predictions)

    return accuracy, local_predictions
```

# Parallelization strategies used

## Task Parallelism

- Task parallelism is a type of parallelism where different tasks or operations are executed simultaneously on separate processing units or threads.
- Each task can be executed concurrently without dependency on the completion of other tasks.



# Benchmarking

# Execution Time - Speedup - Efficiency Comparison

## n=2

| cant_filas | n_jobs | Tiempo de Ejecución Paralelo | Tiempo de Ejecución Secuencial | Speedup | Eficiencia |
|---|---|---|---|---|---|
| 200000 | 2 | 499,103 | 544,581 | 1,091 | 54,56 % |
| 50000 | 2 | 118,925 | 131,455 | 1,105 | 55,20 % |
| 20000 | 2 | 29,708 | 54,401 | 1,831 | 91,56 % |
| 2000 | 2 | 3,333 | 5,564 | 1,669 | 83,46 % |

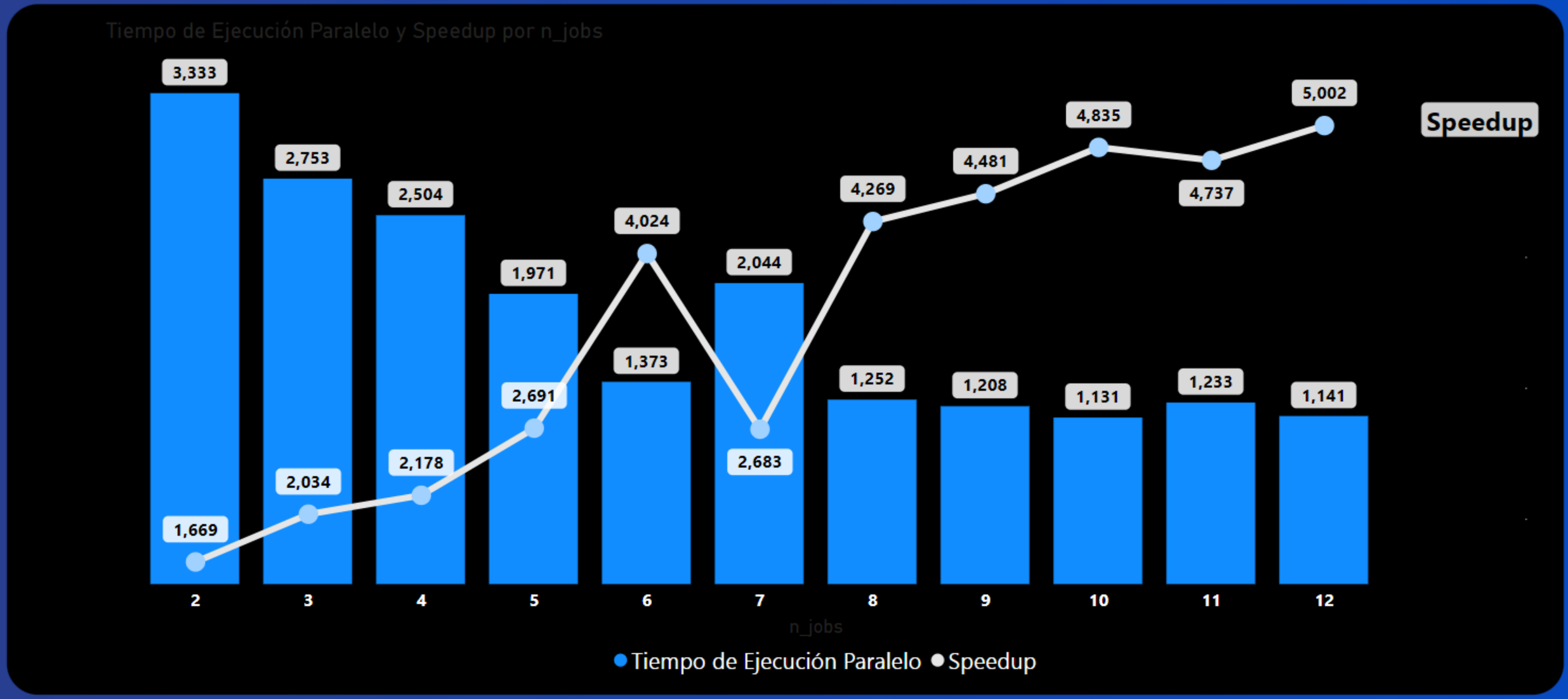
## n=6

| cant_filas | n_jobs | Tiempo de Ejecución Paralelo | Tiempo de Ejecución Secuencial | Speedup | Eficiencia |
|---|---|---|---|---|---|
| 200000 | 6 | 241,520 | 558,149 | 2,311 | 38,52 % |
| 50000 | 6 | 52,383 | 133,832 | 2,555 | 42,58 % |
| 20000 | 6 | 13,997 | 55,399 | 3,958 | 65,97 % |
| 2000 | 6 | 1,373 | 5,527 | 4,024 | 67,07 % |

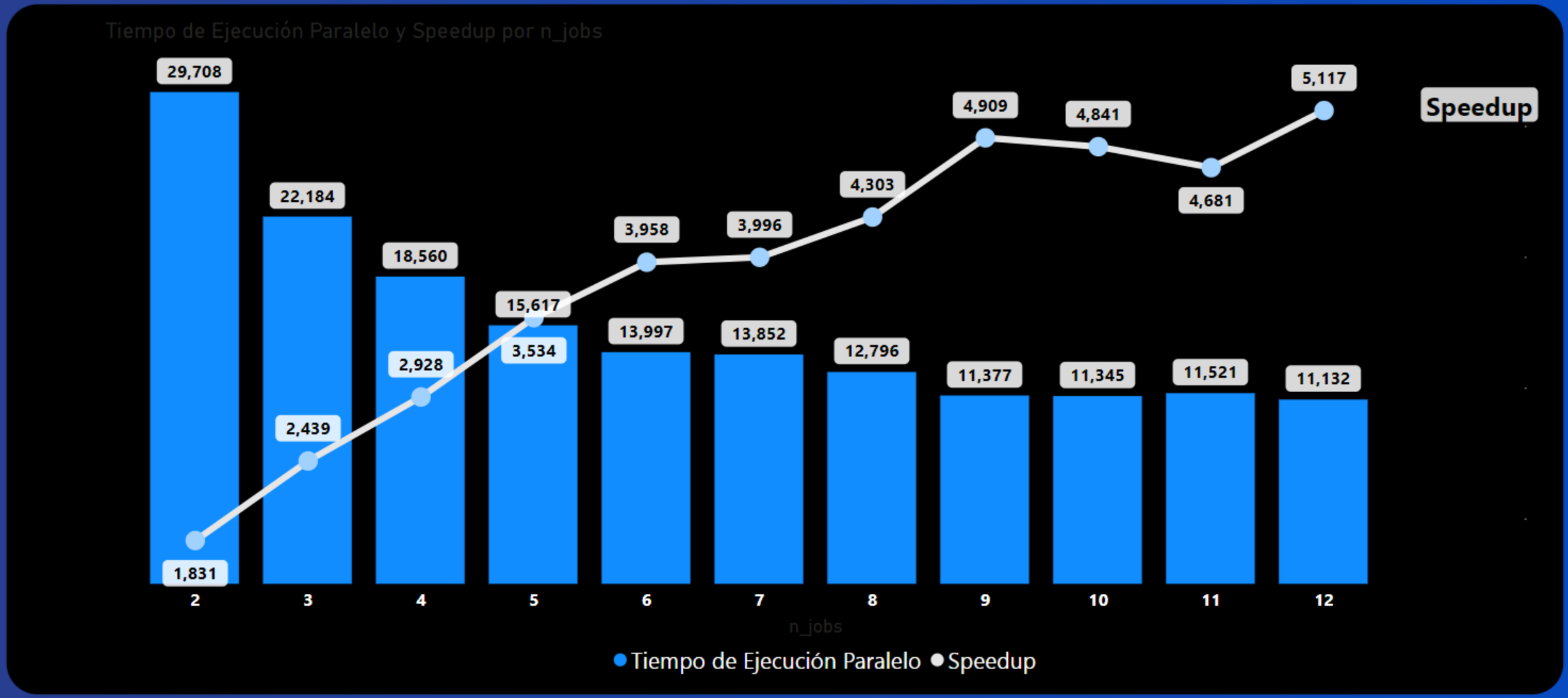
## n=12

| cant_filas | n_jobs | Tiempo de Ejecución Paralelo | Tiempo de Ejecución Secuencial | Speedup | Eficiencia |
|---|---|---|---|---|---|
| 200000 | 12 | 207,037 | 589,411 | 2,847 | 23,72 % |
| 50000 | 12 | 48,145 | 141,227 | 2,933 | 24,44 % |
| 20000 | 12 | 11,132 | 56,963 | 5,117 | 42,64 % |
| 2000 | 12 | 1,141 | 5,708 | 5,002 | 41,68 % |

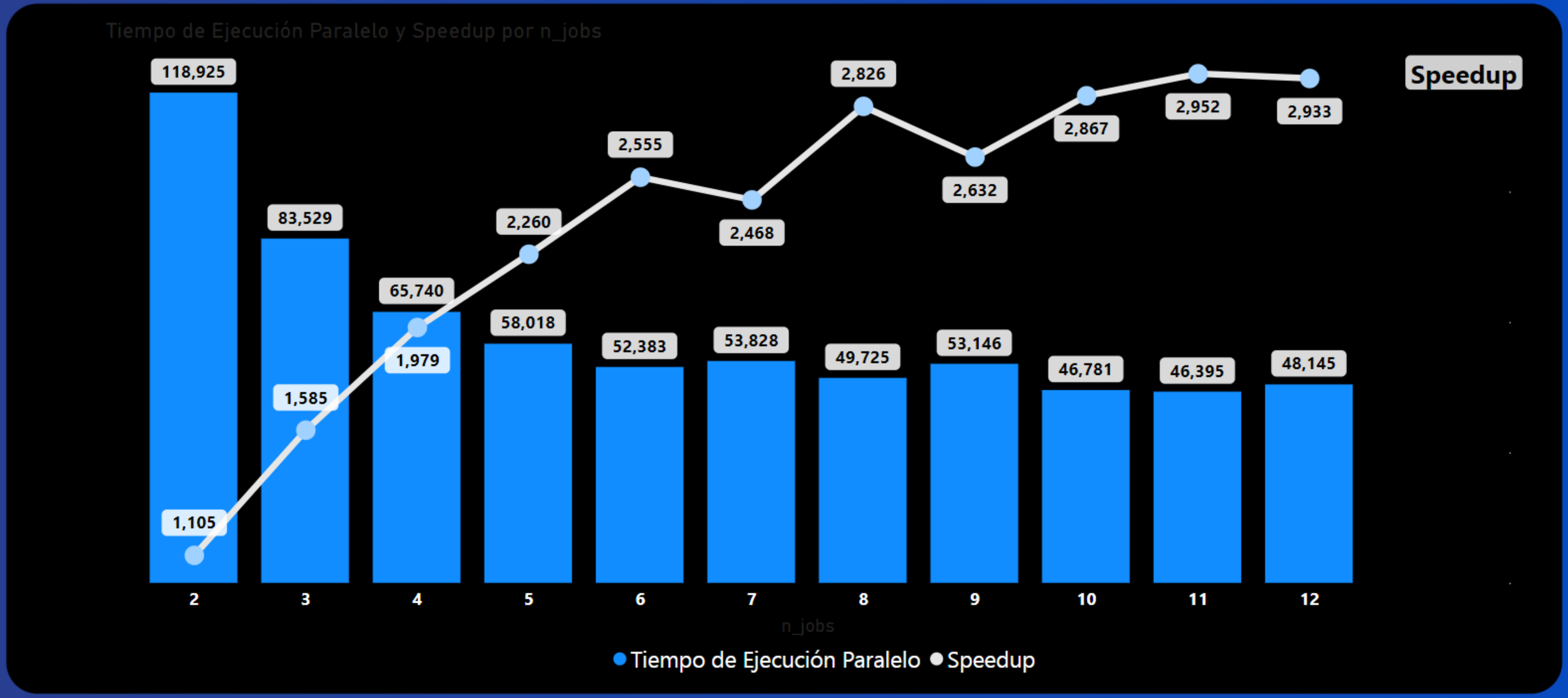
# Parallel Execution Time & Speedup 2k



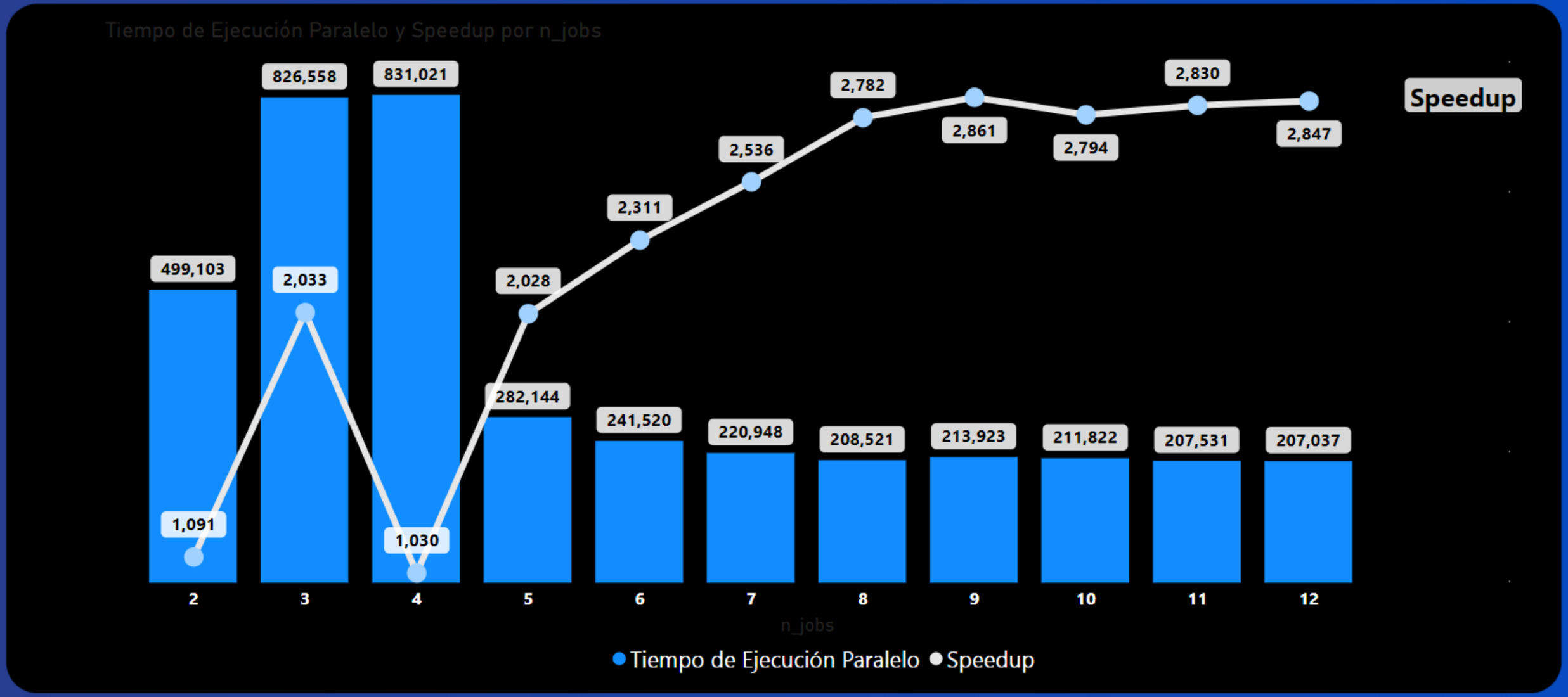
# Parallel Execution Time & Speedup 20k



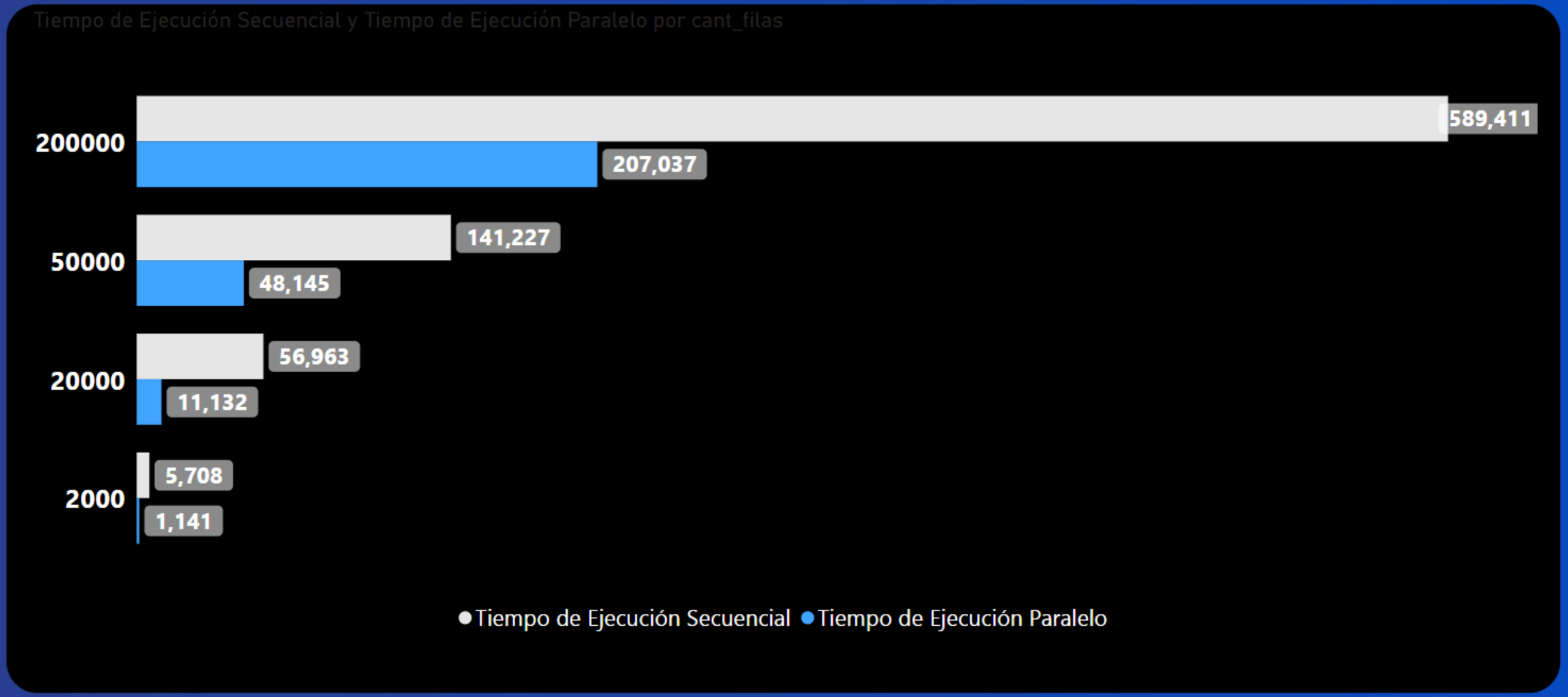
# Parallel Execution Time & Speedup - 50K



# Parallel Execution Time & Speedup 200k



# Parallel (n=12)  VS Sequential Comparison Time



# Conclusions

# Conclusions

- The aim of this project is to give a general view of parallel programming and implementation of KNN algorithm to reach the boost of performance

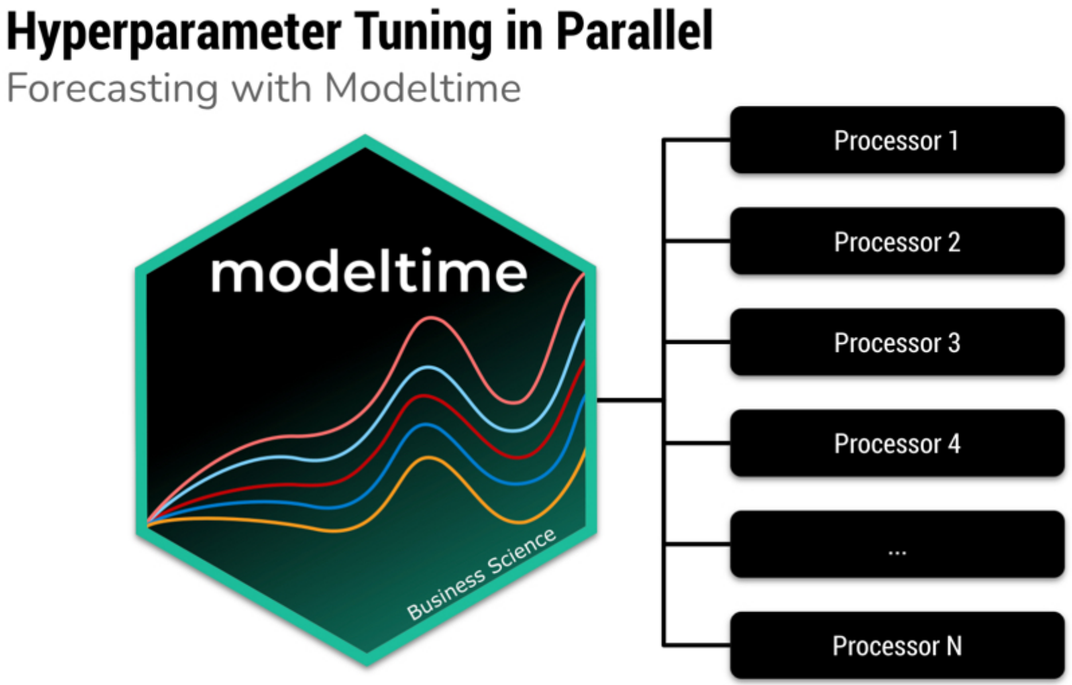
- Applying parallelism to algorithms instead of doing sequential implementation, which saves execution time, optimizes the use of available resource, and gives out many solutions to problems.

- Execution times decrease significantly as we increase the number of workers when running it in parallel, as we increase the number of records in our dataset

# Future Work

# Future Work

- Parallel implementation of hyperparameter tuning
- Parallel implementation for finding the most optimal model.



# Thank you for your attention.