Production Line Performance

Project work in Languages and Algorithms for Artificial Intelligence class (Module 2)

Alessio Falai

alessio.falai@studio.unibo.it

September 13, 2020

Alma Mater Studiorum - University of Bologna

Table of contents

Datasets

- **Arrest**: Contains statistics, in arrests per 100.000 residents, for assault, murder, and rape in each of the 50 US states in 1973. It also gives the percent of the population living in urban areas.
- Adult: Aims at separating people whose income is greater than 50 thousands dollars per year from the rest.

- **Arrest**: Contains statistics, in arrests per 100.000 residents, for assault, murder, and rape in each of the 50 US states in 1973. It also gives the percent of the population living in urban areas.
- **Adult**: Aims at separating people whose income is greater than 50 thousands dollars per year from the rest.

- **Bosch**: Aims at predicting internal failures using thousands of measurements and tests made for each component along different assembly lines.

PREPROCESSING

- The Bosch dataset includes 3 subsets: numerical, categorical and time data.
- As stated in [3], categorical data is extremely sparse, and thus not exploited in subsequent stages.
- Our first analysis will be focused on just **numerical data**:
  - 968 anonymized features
  - 1 183 747 labeled examples
  - 0.58% of failed products
  - 78.5% of missing values

- **Stage I**: This step clusters data with similar processes together into process groups.

- Stage II: This step uses supervised learning to predict the failed products. Each cluster is treated as an independent dataset and has its own classifier.

- **Stage I**: This step clusters data with similar processes together into process groups.
- **Stage II**: This step uses supervised learning to predict the failed products. Each cluster is treated as an independent dataset and has its own classifier.

- **Common**: Columns containing null values and constant values in each row are dropped.

- **Clustering**: Values are binarized (0 meaning null value and 1 meaning not null value) and *PCA* is applied to binarized features.

- **Classification**: A *feature imputation* method (mean value over the column), followed by *feature standardization* (zero mean, unit variance) and *PCA*, is applied over non-binary values in each cluster.

- **Prediction**: A new example follows the same preprocessing scheme as the whole dataset.

- **Common**: Columns containing null values and constant values in each row are dropped.

- **Clustering**: Values are binarized (0 meaning null value and 1 meaning not null value) and *PCA* is applied to binarized features.

- **Classification**: A *feature imputation* method (mean value over the column), followed by *feature standardization* (zero mean, unit variance) and *PCA*, is applied over non-binary values in each cluster.

- **Prediction**: A new example follows the same preprocessing scheme as the whole dataset.

- **Common**: Columns containing null values and constant values in each row are dropped.
- **Clustering**: Values are binarized (0 meaning null value and 1 meaning not null value) and *PCA* is applied to binarized features.
- **Classification**: A *feature imputation* method (mean value over the column), followed by *feature standardization* (zero mean, unit variance) and *PCA*, is applied over non-binary values in each cluster.
- **Prediction**: A new example follows the same preprocessing scheme as the whole dataset.

- **Common**: Columns containing null values and constant values in each row are dropped.
- **Clustering**: Values are binarized (0 meaning null value and 1 meaning not null value) and *PCA* is applied to binarized features.
- **Classification**: A *feature imputation* method (mean value over the column), followed by *feature standardization* (zero mean, unit variance) and *PCA*, is applied over non-binary values in each cluster.
- **Prediction**: A new example follows the same preprocessing scheme as the whole dataset.

Custom implementation of the *PCA* workflow, taking into account:

- *Features assembly* and *features standardization.*

- Selection of the minimum number of principal components *explaining* the given percentage of *variance* in the data (defaults to 95%).

- *Transformation matrix* storage, so that new examples can be easily converted into principal components.

Custom implementation of the *PCA* workflow, taking into account:

- *Features assembly* and *features standardization.*
- Selection of the minimum number of principal components *explaining* the given percentage of *variance* in the data (defaults to 95%).
- *Transformation matrix* storage, so that new examples can be easily converted into principal components.

Custom implementation of the *PCA* workflow, taking into account:

- *Features assembly* and *features standardization.*
- Selection of the minimum number of principal components *explaining* the given percentage of *variance* in the data (defaults to 95%).
- *Transformation matrix* storage, so that new examples can be easily converted into principal components.

Clustering

The chosen clustering algorithm is k-means, since density-based methods (like DBSCAN) are still not available in Spark. So, the following issues needed to be addressed:

- **Problem**: Automatically select the right amount of clusters:

  - *Silhouette* and *elbow* methods are not ideal, since they require human analysis.
  - Spark 3.0.0 dropped support for *inertia* computation, maintaining only evaluation by silhouette scores.

- **Solution**: Ad-hoc implementation of the *Gap statistic* method, described in [2].

The chosen clustering algorithm is `k-means`, since density-based methods (like `DBSCAN`) are still not available in Spark. So, the following issues needed to be addressed:

- **Problem**: Automatically select the right amount of clusters:
    - *Silhouette* and *elbow* methods are not ideal, since they require human analysis.
    - `Spark 3.0.0` dropped support for *inertia* computation, maintaining only evaluation by silhouette scores.
- **Solution**: Ad-hoc implementation of the *Gap statistic* method, described in [2].

The chosen clustering algorithm is `k-means`, since density-based methods (like `DBSCAN`) are still not available in Spark. So, the following issues needed to be addressed:

- **Problem**: Automatically select the right amount of clusters:
    - *Silhouette* and *elbow* methods are not ideal, since they require human analysis.
    - `Spark 3.0.0` dropped support for *inertia* computation, maintaining only evaluation by silhouette scores.
- **Solution**: Ad-hoc implementation of the *Gap statistic* method, described in [2].

- For $k = 1, .., K$, where K is the maximum number of clusters, perform k-means and compute the resulting inertia I_k.

- Generate B reference datasets by sampling from a uniform distribution over each feature, where the support is directly identified by features ranges. Then, for $k = 1, .., K$ and $b = 1, .., B$, perform k-means and compute the resulting inertia I_{kb}. Finally, estimate $E^*[log(I_{kb})]$ as $\frac{1}{B} \sum_b log(I_{kb})$.

- Compute the Gap score as $Gap(k) = E^*[log(I_{kb})] - log(I_k)$.

- Compute the standard deviation sd_k of $log(I_{kb})$ and define $s_k = sd_k * \sqrt{1 + \frac{1}{B}}$.

- Select the minimum k s.t. $Gap(k) - Gap(k + 1) + s_{k+1} \geq 0$.

- For $k = 1, .., K$, where K is the maximum number of clusters, perform k-means and compute the resulting inertia I_k.
- Generate B reference datasets by sampling from a uniform distribution over each feature, where the support is directly identified by features ranges. Then, for $k = 1, .., K$ and $b = 1, .., B$, perform k-means and compute the resulting inertia I_{kb}. Finally, estimate $E^*[log(I_{kb})]$ as $\frac{1}{B} \sum_b log(I_{kb})$.
- Compute the Gap score as $Gap(k) = E^*[log(I_{kb})] - log(I_k)$.
- Compute the standard deviation sd_k of $log(I_{kb})$ and define $s_k = sd_k * \sqrt{1 + \frac{1}{B}}$.
- Select the minimum k s.t. $Gap(k) - Gap(k + 1) + s_{k+1} \geq 0$.

- For $k = 1, .., K$, where K is the maximum number of clusters, perform k-means and compute the resulting inertia I_k.
- Generate B reference datasets by sampling from a uniform distribution over each feature, where the support is directly identified by features ranges. Then, for $k = 1, .., K$ and $b = 1, .., B$, perform k-means and compute the resulting inertia I_{kb}. Finally, estimate $E^*[log(I_{kb})]$ as $\frac{1}{B} \sum_b log(I_{kb})$.
- Compute the Gap score as $Gap(k) = E^*[log(I_{kb})] - log(I_k)$.
- Compute the standard deviation sd_k of $log(I_{kb})$ and define $s_k = sd_k * \sqrt{1 + \frac{1}{B}}$.
- Select the minimum k s.t. $Gap(k) - Gap(k + 1) + s_{k+1} \geq 0$.

- For $k = 1, .., K$, where K is the maximum number of clusters, perform k-means and compute the resulting inertia I_k.
- Generate B reference datasets by sampling from a uniform distribution over each feature, where the support is directly identified by features ranges. Then, for $k = 1, .., K$ and $b = 1, .., B$, perform k-means and compute the resulting inertia I_{kb}. Finally, estimate $E^*[log(I_{kb})]$ as $\frac{1}{B} \sum_b log(I_{kb})$.
- Compute the Gap score as $Gap(k) = E^*[log(I_{kb})] - log(I_k)$.
- Compute the standard deviation sd_k of $log(I_{kb})$ and define $s_k = sd_k * \sqrt{1 + \frac{1}{B}}$.
- Select the minimum k s.t. $Gap(k) - Gap(k + 1) + s_{k+1} \geq 0$.

- For $k = 1, .., K$, where K is the maximum number of clusters, perform k-means and compute the resulting inertia I_k.
- Generate B reference datasets by sampling from a uniform distribution over each feature, where the support is directly identified by features ranges. Then, for $k = 1, .., K$ and $b = 1, .., B$, perform k-means and compute the resulting inertia I_{kb}. Finally, estimate $E^*[log(I_{kb})]$ as $\frac{1}{B} \sum_b log(I_{kb})$.
- Compute the Gap score as $Gap(k) = E^*[log(I_{kb})] - log(I_k)$.
- Compute the standard deviation sd_k of $log(I_{kb})$ and define $s_k = sd_k * \sqrt{1 + \frac{1}{B}}$.
- Select the minimum k s.t. $Gap(k) - Gap(k+1) + s_{k+1} \geq 0$.

Classification

- **Implemented models**: Decision Tree, **Random Forest** and Gradient Boosted Tree.
- **Training strategy**: Hyper-parameters selected by cross-validation on a parameter grid, mainly consisting of the following:
  - Maximum depth of each tree, ranging from 1 to 10 with step 5;
  - Minimum number of instances each child must have after split, ranging from 1 to 10 with step 2.

- **Implemented models**: Decision Tree, **Random Forest** and Gradient Boosted Tree.
- **Training strategy**: Hyper-parameters selected by cross-validation on a parameter grid, mainly consisting of the following:
  - Maximum *depth* of each tree, ranging from 1 to 30 with step 5.
  - Minimum *number of instances* each child must have after split, ranging from 1 to 10 with step 2.

Different evaluation strategies, based on a custom *confusion matrix* computation:

- **Accuracy**: $\frac{TP+TN}{TP+FP+FN+TN}$ (not well-suited to the Bosch dataset, given its high class imbalance).

- F_1-**score**: $2 \times \frac{p \times r}{p+r}$, where $p = \frac{TP}{TP+FP}$ and $r = \frac{TP}{TP+FN}$.

- **Area under ROC**: The two-dimensional area underneath the entire ROC curve, from $(0,0)$ to $(1,1)$.

- **Matthew's Correlation Coefficient (MCC)**:
  $$\frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP) \times (TP+FN) \times (TN+FP) \times (TN+FN)}}.$$

Different evaluation strategies, based on a custom *confusion matrix* computation:

- **Accuracy**: $\frac{TP+TN}{TP+FP+FN+TN}$ (not well-suited to the Bosch dataset, given its high class imbalance).

- F_1-**score**: $2 \times \frac{p \times r}{p+r}$, where $p = \frac{TP}{TP+FP}$ and $r = \frac{TP}{TP+FN}$.

- Area under ROC: The two-dimensional area underneath the entire ROC curve, from $(0, 0)$ to $(1, 1)$.

- Matthew's Correlation Coefficient (MCC):
  $\frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP) \times (TP+FN) \times (TN+FP) \times (TN+FN)}}$.

Different evaluation strategies, based on a custom *confusion matrix* computation:

- **Accuracy**: $\frac{TP+TN}{TP+FP+FN+TN}$ (not well-suited to the Bosch dataset, given its high class imbalance).

- F_1-**score**: $2 \times \frac{p \times r}{p+r}$, where $p = \frac{TP}{TP+FP}$ and $r = \frac{TP}{TP+FN}$.

- **Area under ROC**: The two-dimensional area underneath the entire ROC curve, from $(0, 0)$ to $(1, 1)$.

- Matthew's Correlation Coefficient (MCC):
  $\frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP) \times (TP+FN) \times (TN+FP) \times (TN+FN)}}$.

Different evaluation strategies, based on a custom *confusion matrix* computation:

- **Accuracy**: $\frac{TP+TN}{TP+FP+FN+TN}$ (not well-suited to the Bosch dataset, given its high class imbalance).

- F_1-**score**: $2 \times \frac{p \times r}{p+r}$, where $p = \frac{TP}{TP+FP}$ and $r = \frac{TP}{TP+FN}$.

- **Area under ROC**: The two-dimensional area underneath the entire ROC curve, from $(0, 0)$ to $(1, 1)$.

- **Matthew's Correlation Coefficient (MCC)**:
  $\frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP) \times (TP+FN) \times (TN+FP) \times (TN+FN)}}$.

EXECUTION

- `Spark 3.0.0` is not yet available on the AWS EMR service (the last version that can be used is `2.4.4`).

- Spark 3.0.0 could be manually installed to EC2 clusters, but the Flintrock service already provides some shortcuts to create the desidered clusters and automatically install the selected Spark/Hadoop version.

- Unfortunately, Flintrock was not tested with the latest Spark version: this led to lots of trials and errors.

- Spark 3.0.0 is not yet available on the AWS EMR service (the last version that can be used is 2.4.4).
- Spark 3.0.0 could be manually installed to EC2 clusters, but the Flintrock service already provides some shortcuts to create the desidered clusters and automatically install the selected Spark/Hadoop version.
- Unfortunately, Flintrock was not tested with the latest Spark version: this led to lots of trials and errors.

- Spark 3.0.0 is not yet available on the AWS EMR service (the last version that can be used is 2.4.4).
- Spark 3.0.0 could be manually installed to EC2 clusters, but the Flintrock service already provides some shortcuts to create the desidered clusters and automatically install the selected Spark/Hadoop version.
- Unfortunately, Flintrock was not tested with the latest Spark version: this led to lots of trials and errors.

- Local:
    - **Type**: Macbook Pro 16-inch 2019
    - **CPU**: 2.3 Ghz 8-Core Intel Core i9
    - **RAM**: 16 GB 2667 MHz DDR4
- Cloud:
    - **Type**: `t2.2xlarge`
    - **Workers**: 3
    - **Total #cores**: 24
    - **Total RAM**: 91.2 GB

RESULTS

Data bosch-less-less

- Number of *examples* (random subset): 11 432
- Number of *features* (random subset): 50
- Percentage of *failures*: 0.0056%
- Identified number of *clusters*: 8
- **Random Forest** model with/without cross-validation:
    - **Accuracy**/F_1-**score**/**Area under ROC**: Almost 1.0 for each classifier
    - **MCC score**: Around 0.0 on every classifier (near random prediction)

DATA `bosch-less`

- Number of *examples*: 13 758

- Number of *features*: 968

- Percentage of *failures*: 50%

- Number of *clusters*: 6

- **Random Forest** model with cross-validation:
  - **Accuracy**: Mean of 0.74, maximum at 0.92 for cluster #1, minimum at 0.56 for cluster #5
  - F_1**-score**: Mean of 0.73, maximum at 0.92 for cluster #1, minimum at 0.55 for cluster #5
  - **Area under ROC**: Mean of 0.77, maximum at 0.96 for cluster #1, minimum at 0.58 for cluster #5
  - **MCC score**: Mean of 0.47, maximum at 0.83 for cluster #1, minimum at 0.13 for cluster #5

CONCLUSIONS

- A good Spark configuration is key to the success of the project: in [1], they provide a spreadsheet which gives the best configuration values for your setup.

- A wise management of Spark operations on RDDs/DataFrames is also of great importance.

- Mind your instances execution on AWS to avoid unpleasant surprises on your credit balance.

- Try to get good results on the whole **bosch** dataset.
- Implement a distributed feature augmentation method, like **SMOTE** (Synthetic Minority Oversampling TEchnique).
- Refactor the *Gap statistic* method to be more efficient.

THANK YOU FOR YOUR ATTENTION

📄 Anthony Shipman.
Apache spark: Config cheatsheet, 2016.

📄 Robert Tibshirani, Guenther Walther, and Trevor Hastie.
Estimating the number of clusters in a dataset via the gap statistic.
63:411–423, 2000.

📄 Darui Zhang, Bin Xu, and Jasmine Wood.
Predict failures in production lines: A two-stage approach with clustering and supervised learning.
pages 2070–2074, 12 2016.