Project Support Ideas

Dear students,

If you are short in ideas for the BDA projects, let us *spark* it up.

In addition to the typical code that the BDA team has shown in class you can consider the following additions:

A. Implement some sort of feature importance of your models

Hereafter follows a piece of code to implement the typical FI for logistic regression:

```
# Train Logistic regression
lr = LogisticRegression(featuresCol="features", labelCol="label", maxIter=10)
lr_model = lr.fit(data)

# Show feature coefficients
coeffs = lr_model.coefficients.toArray()
for name, coef in zip(feature_cols, coeffs):
    print(f"{name}: {coef:.4f}")
```

There are many other methods that you can use and explore. Eventually some of them will require your own implementation.

Feature importance in clustering is not as straightforward as in supervised learning (like classification or regression. However, it is possible to assess **which features are important** for the clusters using several techniques. For example, with permutation importance using the silhouette score or Davies-Bouldin index.

B. Model Evaluation

As you know, evaluation isn't just about picking a single metric — it's about aligning model behavior with real-world impact, ensuring it generalizes, adapts to context, and can be trusted when deployed.

In real-world model scenario, we rarely care about just performance metrics (RMSE, R2, and f1, accuracy, etc.). We will give some examples of metrics that you could use and eventually implement.

Metric Type	Purpose	Example Tool
Discrimination	Can the model tell classes apart?	ROC curve
Calibration	Are predicted probabilities meaningful?	Calibration curves

Stability	How much does performance vary by fold/sample?	Cross-validation
Fairness	Does performance vary across subgroups?	Equalized odds, demographic parity
Interpretability	Can we explain individual or global behavior?	SHAP, LIME

Other ideas:

- Evaluate latency (we will talk about this in the next theoretical lecture), scalability, and resource usage.
- Monitor for concept drift: does the data distribution change over time?

You can play with the idea of cost-sensitive evaluation:

- Assign different weights to FP/FN depending on their cost and business value.

C. Time Series Forecasting

A sliding window transformation can be used to convert sequential time series data into a format suitable for supervised machine learning models. This means transforming the data so that **past values of a time series become input features**, and **future values become the prediction target**.

- In Spark, this transformation is implemented using **lag functions** within a **Window specification** that is ordered by time (e.g., a timestamp).

Beyond simple lag features, **rolling aggregations** are a powerful way to summarize trends or volatility over time.

- This is accomplished in Spark using Window functions with rowsBetween (e.g., from -3 to 0) to define a rolling frame over which aggregations like avg(), stddev(), min(), and max() are calculated.

Since Spark itself does not natively support time series models like ARIMA or Prophet, these can be integrated into Spark workflows using **pandas UDFs** and **grouped operations**. In this case you can use pandas.

- Prophet (developed by Meta) is a tool for time series forecasting that accounts for trend, seasonality, and holiday effects.
- ARIMA models (via libraries like pmdarima or statsmodels) offer classical statistical modeling for autoregressive processes, particularly when seasonality or trends are well understood.

- In Spark, you can use groupBy().apply() along with a pandas_udf to run these models in parallel across many time series groups (e.g., by location, store, or sensor ID).

D. Custom Algorithms in Spark

Creating **custom algorithms in Apache Spark** is a powerful way to implement specialized machine learning logic or business rules that are not covered by the built-in MLlib API. Spark is flexible enough to allow custom implementations at both the **low level (RDD/DataFrame)** and **high level (ML pipelines)**. Examples of custom algorithms are the following:

- Custom **clustering** algorithm (e.g., DBSCAN, hierarchical, dynamic time warping)
- Domain-specific **anomaly detection**
- Ensemble logic combining **MLlib models**

E. Advanced Feature Selection

There are many ways to do advanced feature selection. If you have a long list of features, might be worth exploring:

- Tree-based Feature Importance: Uses built-in importance scores from models like Random Forest or Gradient-Boosted Trees to rank and select the most relevant features.
- Chi-Square Selector: A statistical test that selects categorical features with the strongest dependence on the label. Best suited for classification tasks with discrete inputs.
- L1 Regularization (Lasso): Applies logistic regression with L1 penalty to shrink less useful feature coefficients to zero, effectively selecting only the most informative ones.
- Mutual Information: Evaluates how much knowing a feature reduces uncertainty about the label. Useful for ranking discrete features in classification problems.

It is important to draw the line between the ML and Big Data Analytics. You can implement a very sophisticated ML solution but have only a small subset of distributed code (pyspark). This will hardly lead to an excellent grade.

You will be evaluated by your ability to use the code from PySpark or to create code that is distributed. It is better to implement a simple distributed code (RDD DBSCAN) and to show full understanding of your code than to implement a big solution of typical ML. Remember that the course is Big Data Analytics.