AIML427 Assignment 3 **Deveremma**
Individual Project Report **300602434**

---

1. Introduction

The machine learning task I've selected is the prediction of student success, specifically can we reliably predict a student's GPA based on their academic, behavioral, and social background? GPA is widely used as a measure of academic success and a predictor for future educational or employment opportunities. Accurately forecasting GPA can help identify at-risk students, personalize interventions, and allocate support efficiently.

Personal Motivations

- General curiosity on what truly contributes to academic success: Do study hours play a larger role than school attendance? Do extracurriculars help performance or hurt it? As such, this project goes beyond mere GPA prediction via regression coefficients and feature importance, more on this later in the report.

- Academic relevance as a master's student in AI: This project has two benefits; learning to work with the Hadoop Distributed File System and Spark ML for big data processing, while potentially learning and applying insights gained toward achieving improved student performance. This may likewise apply to the reader.

Project Scope

This project tackles the task of GPA prediction using **regression**, applied to a very large educational dataset. An end-to-end machine learning pipeline is developed, highlighting informed preprocessing, feature engineering, multiple regressor evaluation, and a final fine tuned model through cross validation. This program is trained and tested on the Victoria University Hadoop cluster using Apache Spark to ensure scalable handling.

Data Origin and Summary

Source: https://www.kaggle.com/datasets/neuralsorcerer/student-performance/data

The dataset is composed of records obtained from High School students (Ages 14 - 18). Provided files include train.csv (1.23 GB), validation.csv (158MB), and test.csv (158MB) Important: Due to the sufficient size of validation.csv and the general stigma of keeping test.csv unseen, validation.csv alone will be used to derive the training and testing sets.

Example Features and values:

- AttendanceRate: *Fraction of school days attended (0.70-1.00)*
- ParentalEductation: *Highest education (HS, SomeCollege, Bachelors+)*
- TestScore_Math: *Math achievement score (0–100).*
- Extracurricular: *Participation in clubs/sports (1 = yes, 0 = no)*
- StudyHours: *Average self-reported homework/study hours per day (0–4).*
- Romantic: *Currently in a romantic relationship (1 = yes, 0 = no)*
- FreeTime*:* *Free time after school on a scale from 1 (low) to 5 (high).*
- Locale: *School location (Suburban, City, Rural, Town)*
- **GPA*:*** *Cumulative Grade Point Average on a 0.0–4.0 scale.*

2. Exploratory Data Analysis

Validation.csv at 158MB contains:
999,229 instances of data

The Victoria University Submission System only supports a 100MB max attachment size

Therefore Validation.csv will be sampled to 80MB to create the dataset student.csv

In code, student.csv will be sampled again to 40MB to reduce computational overhead

The final dataset used in the program will contain 254,604 instances of data.

The dataset is clean, boasting no null values with a mixture of int, string, and double types

Column	DataType	NullCount
Age	int	0
Grade	int	0
Gender	string	0
Race	string	0
SES_Quartile	int	0
ParentalEducation	string	0
SchoolType	string	0
Locale	string	0
TestScore_Math	double	0
TestScore_Reading	double	0
TestScore_Science	double	0
GPA	double	0
AttendanceRate	double	0
StudyHours	double	0
InternetAccess	int	0
Extracurricular	int	0
PartTimeJob	int	0
ParentSupport	int	0
Romantic	int	0
FreeTime	int	0
GoOut	int	0

Data Preprocessing Tips

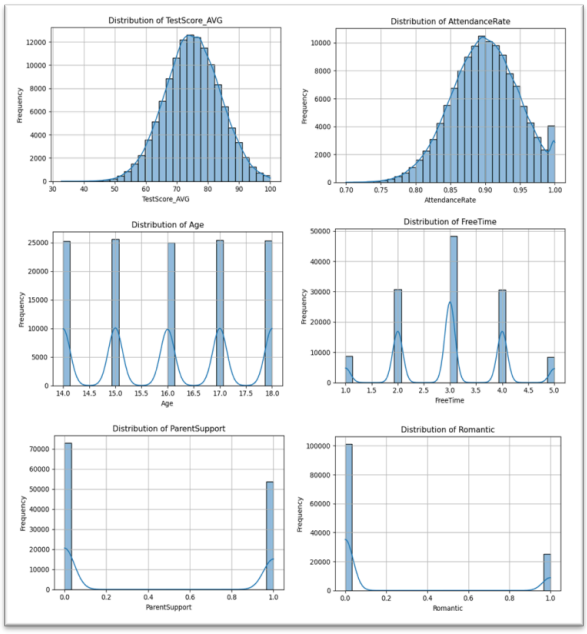
- **One-Hot Encoding:** For categorical features (`Gender` , `Race` , `SchoolType` , `Locale` , `ParentalEducation`).
- **Ordinal Encoding:** Map `SES_Quartile` , `FreeTime` , `GoOut` directly to integers.

The dataset has 21 features initially, but through data preprocessing which is outlined by the dataset publisher (one-hot) the final total is exactly 35 preprocessed features.

Feature Distribution:

The distribution of numerical features (i.e Test_Score, AttendanceRate) is plotted to inform the scaling strategy used in later preprocessing. As all features generally follow a Gaussian distribution like those to the right, StandardScaler (z-score norm) is appropriate over MinMaxScaler

We can observe that TestScores have values e.g. 93.02 whereas attendance has values e.g. 0.93. Scaling therefore is crucial to prevent features with different scales from dominating during learning.



Addressing Data Leakage:

As a sideline discussion, EDA was performed prior to train/test splitting. In this instance, this is acceptable as the EDA tasks described above are data-agnostic operations that do not involve model training or preprocessing. Importantly, splitting is to be done prior to pre-processing to ensure statistics like the mean (necessary for scaling) are only calculated from the train set and applied to the test data (test data remains unseen).

Feature Engineering:

1. Stem_Ratio: ((TestScore_Math + TestScore_Science) / 2) / TestScore_Reading

New features were engineered due to personal investigation and curiosity. Stem_Ratio quantifies a student's strength in STEM subjects compared to their read/writing ability. Example 1.12 (Student scores higher in STEM subjects). Are STEM students smarter?

2. TestScore_AVG: (TestScore_Math + TestScore_Science + TestScore_Reading) / 3

This quantifies Charles Spearman's 1904 study "Students who did well in one subject, tended to do well in them all". Source: Veritasium, I took an IQ Test (YouTube) 04/23

Data Splitting: 0.8 Train to 0.2 Test split (Chosen as a reversal to the Pareto Principle)
Results in ~200,000 training instances to ~50,000 testing instances

3. Data Preprocessing

```
# One-Hot Encode Categorical Features as per recommended pre-processing
categorical_cols = ["Gender", "Race", "ParentalEducation", "SchoolType", "Locale"]

# Scale numeric double variables. Scaling method to be determined by feature distribution.
numerical_cols = ["Age", "Grade", "SES_Quartile", "TestScore_Math", "TestScore_Reading",
"TestScore_Science", "AttendanceRate", "StudyHours", "FreeTime", "GoOut", "Stem_Ratio",
"TestScore_AVG"]

# Acceptable binary integer variables.
binary_cols = ["InternetAccess", "Extracurricular", "PartTimeJob", "ParentSupport", "Romantic"]
```

Spark ML cannot handle string features, therefore **StringIndexer** and **OneHotEncoder** were applied to categorical features. StringIndexer converts categorical text values into indexed integers. OneHotEncoder as per the data publisher's guide converts indexed categorical features into sparse binary vectors. 5 base features were expanded to 18.

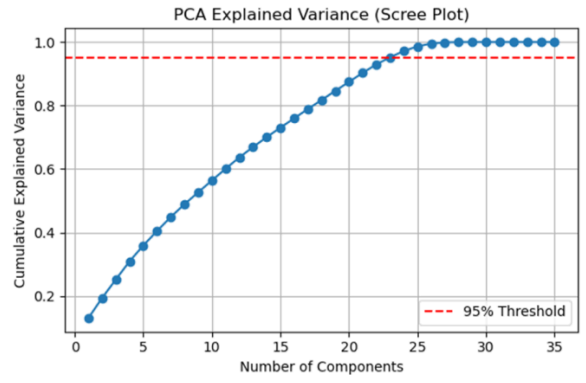
- All processed features are then merged using VectorAssembler. This converts all (now numerical features) into vector format which is required by PySpark ML.

As informed by the above EDA, StandardScaler was selected over MinMaxScaler. This normalizes all features to a mean of 0 and std of 1 to ensure all features contribute equally to the model's learning process. However, as a side note, this step is optional.

Principal Component Analysis

PCA is aimed at dimensionality reduction via a small set of uncorrelated principal components. Steps to find this set size k:

- Transform the training set through the above pipeline to be fit for PCA
- Extract the cumulative explained variance for each PC (up to 35).
- Choose the smallest K such that a target explained variance is met




```
Total features/predictors in the fully preprocessed data: 35
============================================================
Optimal number of PCA components to retain 95.0% variance: 23
```

PCA was defined once with all 35 features. It is defined once again but with a K of 23.

Program Mode of Operation

```
mode = args.mode
# mode 1 = No Scaling / No PCA, mode 2 = Scaling / No PCA, mode 3 = Scaling / PCA
out = "features_vec" if mode == 1 else "scaled_features" if mode == 2 else "pca_features"
op = "non" if mode == 1 else "scaled" if mode == 2 else "pca"
if mode == 1:
    pipe = [encoder, assembler]
elif mode == 2:
    pipe = [encoder, assembler, scaler]
else:
    pipe = [encoder, assembler, scaler, pca]
```

To aid the methodology of obtaining a reliable student GPA predictor (and as per the brief), the model will undergo three different preprocessing pipelines defined under the variable mode. Mode can be set using the command line argument --mode n [1, 2, 3] Model performance derived from the three preprocessing pipelines will be compared.

Chosen Spark ML Regressors

Some regressors are better at capturing simple linear patterns (linear regression) while other regressors excel at learning non-linear relationships/interactions (tree-based regressors).

Testing a variety of models gives insight into underlying structure, whether it is simple or complex.

It is also conducted to identify the best regressor which will be taken for further fine tuning. Again, the goal of this project is to obtain a reliable predictor of student GPA.

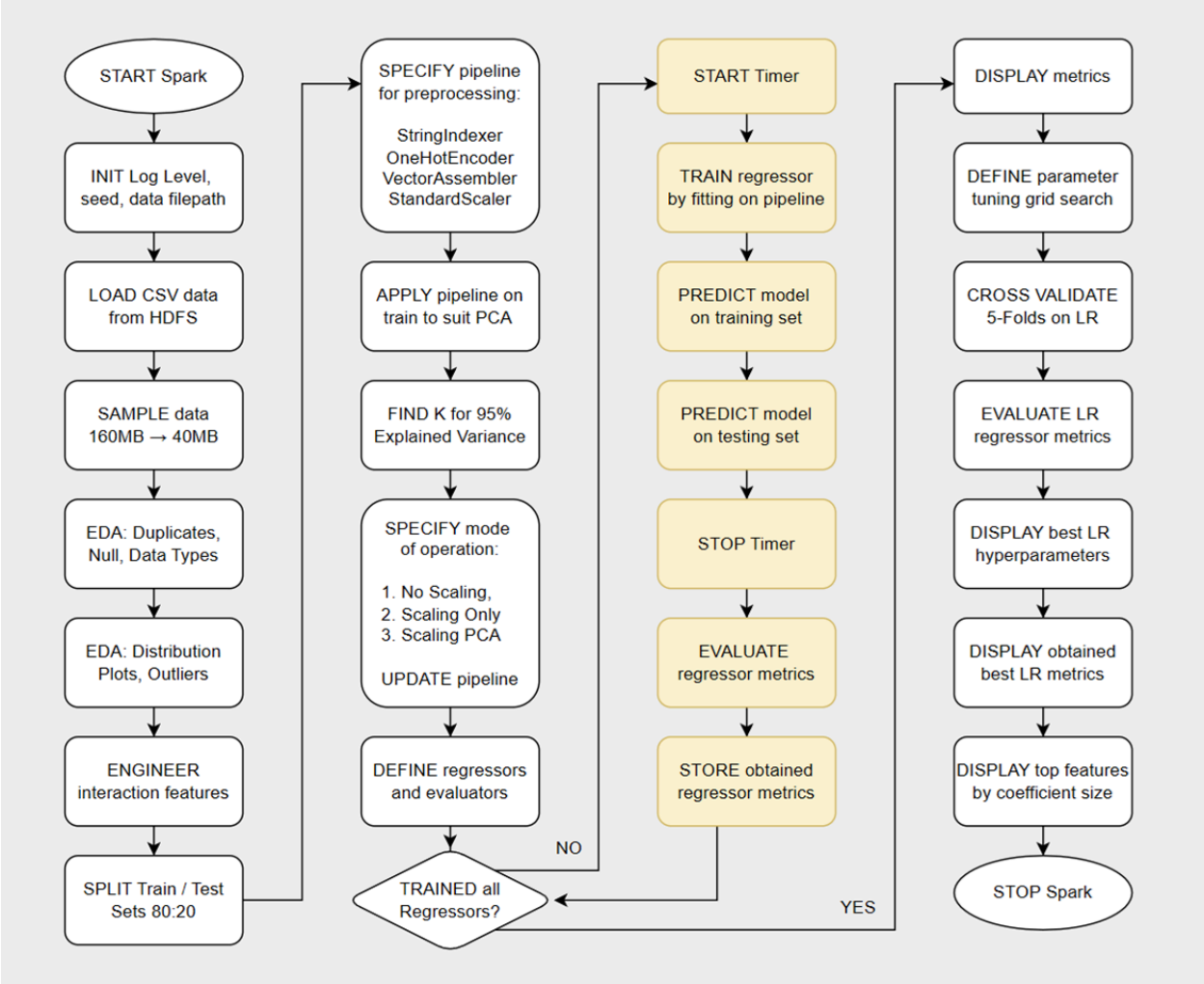
```
regressors = {
    "LinearRegression": LinearRegression(
        featuresCol=out,
        labelCol=label_col,
        regParam=0.1,              # Ridge regularization (L2)
        elasticNetParam=0.0,       # ElasticNet mix (0 = Ridge,
        maxIter=50
    ),
    "DecisionTree": DecisionTreeRegressor(
        featuresCol=out,
        labelCol=label_col,
        maxDepth=10,               # More depth for complexity
        minInstancesPerNode=5
    ),
    "RandomForest": RandomForestRegressor(
        featuresCol=out,
        labelCol=label_col,
        numTrees=20,               # More trees for stability
        maxDepth=5,
        subsamplingRate=0.8
    ),
    "Gradient-Boosted": GBTRegressor(
        featuresCol=out,
        labelCol=label_col,
        maxIter=50,                # More boosting rounds
        maxDepth=5,
        stepSize=0.1               # Conservative learning rate
    ),
    "GeneralizedLR": GeneralizedLinearRegression(
        featuresCol=out,
        labelCol=label_col,
        family="gaussian",
        link="identity",
        regParam=0.1,
        maxIter=100
    )
}
```

Chosen Regression Evaluators

```
# Evaluation metrics
evaluator_rmse = RegressionEvaluator(labelCol=label_col, predictionCol="prediction", metricName="rmse")
evaluator_r2 = RegressionEvaluator(labelCol=label_col, predictionCol="prediction", metricName="r2")
evaluator_mae = RegressionEvaluator(labelCol=label_col, predictionCol="prediction", metricName="mae")
```

Elapsed Time will also be measured (start time before fitting and end post predictions)

Program Described via Flowchart:



I omit discussion regarding above as it's self-explanatory and to prevent an overwhelmed reader

Program Described via Installation and Usage Steps:

- Download student.py, student.csv (80MB), SetupSparkClasspath.sh
- // Upload the above files into a directory within the Victoria University servers //
- // Access the Victoria University servers remotely (e.g. barretts) or physically //
- ssh co246a-8
- // Navigate to the directory containing above files //
- source SetupSparkClasspath.sh
- need java8
- hdfs dfs -mkdir -p /user/<USERNAME>/input
- hdfs dfs -put student.csv /user/<USERNAME>/input
- spark-submit student.py --dataset input/student.csv --seed 7 --mode 2
- // Dataset, seed, and mode are optional due to the defaults as seen above //

No Scaling / No PCA

BaseModel	RMSE_train	RMSE_test	MAE_train	MAE_test	R2_train	R2_test	TimeSec
Gradient-Boosted	0.2949	0.2961	0.2359	0.2375	0.5889	0.5854	22.9931
GeneralizedLR	0.2973	0.2966	0.2378	0.2379	0.5821	0.584	5.7921
LinearRegression	0.2973	0.2966	0.2378	0.2379	0.5821	0.584	7.2248
RandomForest	0.2977	0.2975	0.2384	0.2388	0.5809	0.5815	8.6383
DecisionTree	0.2932	0.2992	0.2344	0.24	0.5935	0.5766	9.2832

Scaling / No PCA

BaseModel	RMSE_train	RMSE_test	MAE_train	MAE_test	R2_train	R2_test	TimeSec
Gradient-Boosted	0.2949	0.2961	0.2359	0.2375	0.5889	0.5854	23.6546
LinearRegression	0.2973	0.2966	0.2378	0.2379	0.5821	0.584	8.3851
GeneralizedLR	0.2973	0.2966	0.2378	0.2379	0.5821	0.584	6.7378
RandomForest	0.2977	0.2975	0.2384	0.2388	0.5808	0.5815	9.7158
DecisionTree	0.2932	0.2992	0.2344	0.24	0.5935	0.5766	10.22

Results: No Scaling vs Scaling

As evidenced by the two essentially **identical** tables above, scaling had **no** impact on the performance on **all** my chosen regressors. While this certainly defies expectation, (for some) this outcome is a valid possibility and gives fair insight about our dataset.

Before a greater analysis on that, I had expected Tree-based regressors to be robust irrespective of scaling: Gradient Boosted Trees, Random Forest, and Decision Trees displayed identical performance pre- and post-scaling. This is expected behavior, as such models partition data based on feature thresholds (e.g., Attendance > 0.9), not distances, and are therefore invariant to monotonic transformations like standardization.

The one difference however, is time. Elapsed times consistently increased across all regressors when scaling was applied. This is due to the added computational overhead as a result of StandardScaler's fitting and transformation as well as the increased vector representation complexity, dense vectors instead of sparse when scaled. It's here that we can already conclude that 0 scaling would be the more optimal pathway for this data.

The main topic for analysis however, is Linear Regression which showed identical MAE RMSE, and R^2 across both setups. This is not expected as linear models are sensitive to feature magnitude. Scaling is conducted to prevent certain features from dominating due to inconsistent distance magnitudes. The only way to obtain identical results is if there exists a set of critical features that dominate equally in both scaled and unscaled forms, such that feature domination due to lack of scaling wasn't distorting training. We conclude that the relationship is simple: GPA correlates strongly with **certain features**.

Scaling / No PCA

BaseModel	RMSE_train	RMSE_test	MAE_train	MAE_test	R2_train	R2_test	TimeSec
Gradient-Boosted	0.2949	0.2961	0.2359	0.2375	0.5889	0.5854	23.6546
LinearRegression	0.2973	0.2966	0.2378	0.2379	0.5821	0.584	8.3851
GeneralizedLR	0.2973	0.2966	0.2378	0.2379	0.5821	0.584	6.7378
RandomForest	0.2977	0.2975	0.2384	0.2388	0.5808	0.5815	9.7158
DecisionTree	0.2932	0.2992	0.2344	0.24	0.5935	0.5766	10.22

Scaling / PCA

BaseModel	RMSE_train	RMSE_test	MAE_train	MAE_test	R2_train	R2_test	TimeSec
Gradient-Boosted	0.2985	0.3008	0.239	0.2414	0.5788	0.5722	40.6151
DecisionTree	0.2966	0.3049	0.2371	0.2444	0.584	0.5605	12.9021
GeneralizedLR	0.3065	0.3059	0.2456	0.2456	0.5559	0.5576	11.9314
LinearRegression	0.3065	0.3059	0.2456	0.2456	0.5559	0.5576	11.1293
RandomForest	0.3346	0.3347	0.2677	0.2679	0.4706	0.4704	12.1524

Results: PCA vs No PCA

Before diving into PCA, the main topic of analysis, an important discussion building upon the previous analysis and remains relevant here is the comparison between my chosen regressors. Observing the No PCA table, Gradient Boosted Trees outperforms Random Forest which outperforms Decision Tree. This is a direct reflection of their increasing algorithmic complexity and learning capacity, although their differences are marginal (0.2961 test RSME GB to 0.2992 test RSME DT). This is crucial because it tells us the underlying relationship is rather simple such that the patterns captured in the complex Gradient Boosted algorithm can also be captured in a simple decision tree.

By extension, Linear Regression performs well, outpacing the more complex Random Forest and Decision Tree algorithms in all test metrics RMSE, MAE, and R^2. Though Decision Tree scores lower train metrics, this simply highlights overfitting as a result of training the one tree, hence negatively impacting test performance. Linear Regression's close test performance to GBT (+0.0005 RMSE, +0.0004 MAE, -0.0014 R^2) indeed suggests that the underlying relationship between features and GPA is primarily linear. Furthermore, Linear Regression's elapsed time being ~2.8x faster than GBT displays GBT's obsolete computation, ranking Linear Regression as the best overall regressor.

With the introduction of PCA, every regressor shows a drop in all performance metrics. This degradation is particularly evident in tree-based models as they rely heavily on feature splits which are rendered ineffective when input features are replaced with abstract principal components. RF is hit especially hard (0.33 RSME vs ~0.30 for other regressors) as the ensemble of trees multiplies this DT weakness imposed by PCA.

The elapsed time for all regressors has also consistently increased. This is due to the conversion of sparse vectors into dense format which is a requirement for PCA under the Spark ML library. Likewise is the overhead imposed by fitting and transforming PCA.

The purpose of PCA is to reduce dimensionality. It does so by selecting components that capture the most variance in the input features, not the variance that best predicts the target variable GPA. This is why Linear Regression is also negatively impacted by the PCA transformed dataset. PCA doesn't add any new modeling capability, simply a new coordinate system. This is sometimes helpful, but often harmful when there exists truly meaningful features –**which my previous observations appear to heavily imply**.

Beyond performance, PCA significantly reduces the interpretability of the regressors. Linear regression yields explicit coefficients tied to human-understandable features (e.g. AttendanceRate = +0.0353 GPA units). With PCA, principal components are abstract, composite combinations of original features, making interpretation nearly impossible. To conclude, PCA is unbeneficial for this dataset but more importantly, it is unbeneficial to my project objective which is to gain insight on **factors** that boost student performance.

Fine Tuning Process

```
# Define a parameter grid to search
paramGrid = ParamGridBuilder()\
    .addGrid(lr.regParam, [0.0, 0.01, 0.1])\
    .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0])\
    .addGrid(lr.maxIter, [50, 100])\
    .build()

# CrossValidator with 5 folds
crossval = CrossValidator(
    estimator=lr_pipeline,
    estimatorParamMaps=paramGrid,
    evaluator=evaluator_rmse,
    numFolds=5,
    parallelism=2,
    seed=seed
```

In an effort to obtain a reliable predictor for GPA as per my project objective, fine tuning is a necessary task. In my analysis above, linear regression was identified as the best overall regressor meeting high performance, low computational time, and adding to that is its interpretability through feature coefficients. As per assignment 2, lasso regression was concluded as superior. This fine tuning process aims to identify if this remains true.

```
Best Hyperparameters for Linear Regression:
  regParam:         0.0
  elasticNetParam:  0.5
  maxIter:          100
```

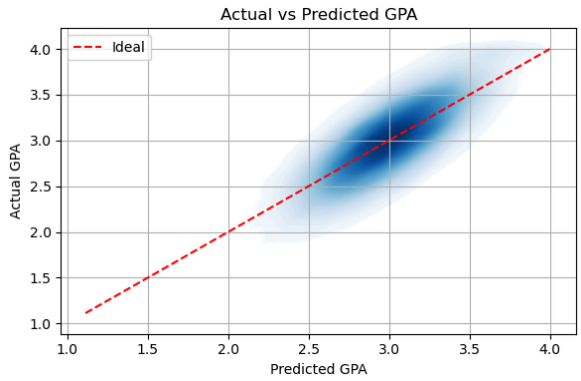
```
Cross-Validated Linear Regression Results:
  RMSE (Test): 0.2954
  MAE (Test) : 0.2367
  R2 (Test)  : 0.5874
```

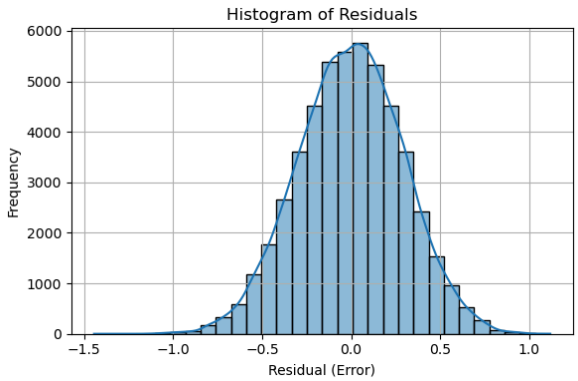
BaseModel	RMSE_train	RMSE_test	MAE_train	MAE_test	R2_train	R2_test	TimeSec
LinearRegression	0.2973	0.2966	0.2378	0.2379	0.5821	0.584	6.9937

Results: Fine Tuning

Contrary to assignment 2, Linear regression without regularization has outperformed all other regularized configurations. The base model linear regression used in the analysis prior was ridge regression, scoring 0.2966 test RSME in both unscaled and scaled data. Here, cross validated grid search deems traditional linear regression as superior which is evidenced by the marginally lower test RSME of 0.2954 (-0.0012), MAE of 0.2367 (-0.0012), and R^2 of 0.5874 (+0.0034). For clarification, we know that traditional linear regression is the best configuration as regParam is set to 0.0 despite 0.5 eNetParam.

What this tells us about our data is that there's little noise to suppress or complexity to penalize. The model benefits more from freely learning the true underlying weights without constraint. Moreover, if the relationship between the features and GPA is almost entirely linear, then a traditional linear regression is sufficient to capture this structure. Regularization may shrink useful coefficients unnecessarily, limiting higher performance.



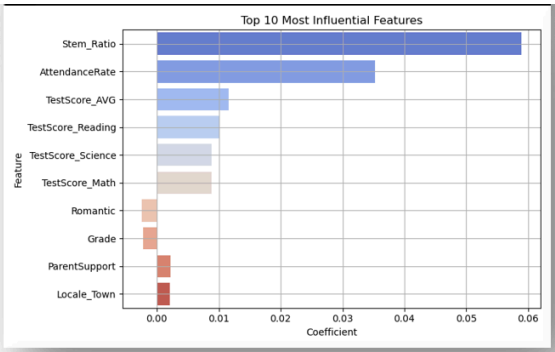


Plotting the fine tuned linear regression model's predictions on the test set, we observe that predictions closely track the GPA as contour density is relatively tight amid the ideal line. Symmetry in residuals plot to the right suggests no major bias in underpredicting or overpredicting. On average, predictions are off by 0.2367 GPA (MAE). This is ~5.9% error of the total GPA range (0.2367 / 4.0 = 0.059175). Larger errors inflate the RMSE calculation to 0.295. This is still ~7.4% of the GPA range. Overall, this performance is quite successful, again confirming a set of meaningful features that correlate with GPA.

Results: Coefficients

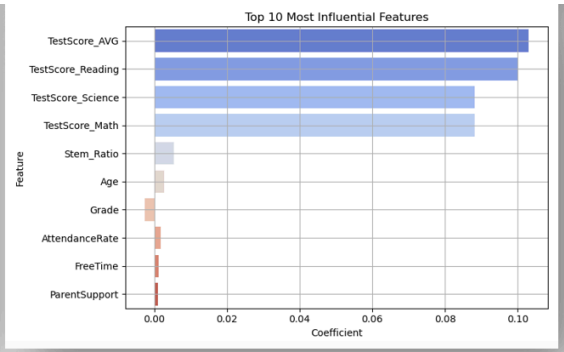
No Scaling / No PCA

```
==================================================
Top 10 Features from Final LR Model (Coefficient
==================================================

  Stem_Ratio              : +0.0589
  AttendanceRate          : +0.0353
  TestScore_AVG           : +0.0116
  TestScore_Reading       : +0.0100
  TestScore_Science       : +0.0089
  TestScore_Math          : +0.0088
  Romantic                : -0.0025
  Grade                   : -0.0022
  ParentSupport           : +0.0022
  Locale_Town             : +0.0021
```



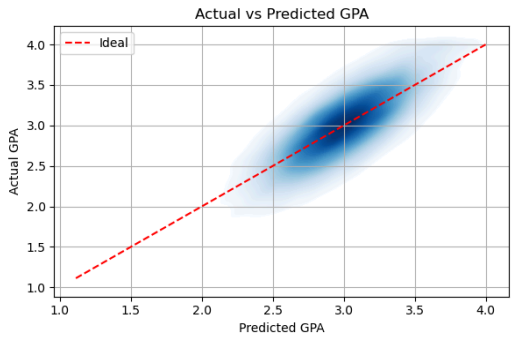
Scaling / No PCA

```
==================================================
Top 10 Features from Final LR Model (Coefficient
==================================================

  TestScore_AVG           : +0.1030
  TestScore_Reading       : +0.1001
  TestScore_Science       : +0.0882
  TestScore_Math          : +0.0880
  Stem_Ratio              : +0.0055
  Age                     : +0.0028
  Grade                   : -0.0026
  AttendanceRate          : +0.0017
  FreeTime                : +0.0012
  ParentSupport           : +0.0011
```



As hinted throughout the analysis, there is indeed a set of critical features which greatly correlate with student GPA and quite linearly too. These features are TestScore_Avg, TestScore_Math, TestScore_Science, and TestScore_Reading. These test scores are also the features with the highest numerical magnitudes 0-100 and therefore dominated the unscaled data. As these features remained prevalent in both unscaled and scaled data, their performance fared no difference. Additional observations include positive stem_ratio which means we are in the correct field! Positive AttendanceRate is sensical. Negative Romantic gives insight into our priorities as students. Positive ParentSupport shouldn't be overlooked. Lastly, as expected, PCA feature coefficients are nonsensical.

```
==================================================
Top 10 Features from Final LR Model (Coefficient
==================================================

  Gender_Female           : -0.1629
  Romantic                : +0.0023
  ParentalEducation_Bachelors+: +0.0014
  ParentSupport           : +0.0012
  InternetAccess          : +0.0012
  Extracurricular         : +0.0011
  SchoolType_Public       : +0.0007
  Race_Two-or-more        : +0.0005
  ParentalEducation_HS    : +0.0005
  Locale_Suburban         : +0.0004
```



Conclusion

This project concludes with the successful delivery of an end-to-end machine learning pipeline for predicting student GPA using Apache Spark ML. Using a dataset with a size of 158MB, this was applied at scale via the Victoria University Hadoop cluster. Through systematic model selection and cross-validated tuning, a low RMSE GPA regressor was obtained, indicating a strong fit between the model selection and the underlying data.

Beside a reliable predictor was greater insight on the factors that led to greater student performance. These were well interpretable through the coefficients provided by the linear regression model. Although test scores predominantly saturated these factors, other insight is gained such as positive attendance rate and ongoing parental support.

The project also provided an opportunity to explore the effects of data preprocessing techniques. It was shown that scaling had limited benefit due to the dominance of certain key features, while PCA, despite its value in contexts of big data, degraded performance in this task by removing interpretable and predictive feature structure.

Overall, this work served as a practical introduction to distributed machine learning workflows. It involved setting up and executing Spark jobs within a Hadoop ecosystem, and integrating multiple stages of feature transformation and model evaluation. This project demonstrates not only technical competency in big data tools and machine learning methods, but also the ability to critically evaluate modeling decisions and communicate findings in a clear, structured manner.

Future Directions and Improvements

Looking back, the main criticism is the bias toward big data and the objective of having 35 preprocessed features to work with. As such feature selection was unfairly omitted. Future work would certainly see more rigorous feature selection, such as addressing multicollinearity through correlation heatmaps and especially the removal of test scores. This would reframe the project objective to focus more purely on social factors, opening the door for a more complex and interesting look into student performance prediction.