1. Classification vs Regression

Your goal is to identify students who might need early intervention - which type of supervised machine learning problem is this, classification or regression? Why?

- ✓ This is a classification problem since we are predicting discrete classes rather than continues values.

2. Exploring the Data

Can you find out the following facts about the dataset?

- Total number of students
- Number of students who passed
- Number of students who failed
- Graduation rate of the class (%)
- Number of features (excluding the label/target column)

Use the code block provided in the template to compute these values.

- ✓ Total number of students: 395
- ✓ Number of students who passed: 265
- ✓ Number of students who failed: 130
- ✓ Number of features: 30
- ✓ Graduation rate of the class: 67.09%

3. Preparing the Data

Execute the following steps to prepare the data for modeling, training and testing:

- Identify feature and target columns
- Preprocess feature columns
- Split data into training and test sets

Starter code snippets for these steps have been provided in the template.

- ✓ Feature column(s):-['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu', 'Mjob', 'Fjob', 'reason', 'guardian', 'traveltime', 'studytime', 'failures', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery', 'higher', 'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc', 'Walc', 'health', 'absences']
- ✓ Target column: passed
- ✓ After pre-process we see 48 features instead of 30
- ✓ We keep 95 test and 300 training data

4. Training and Evaluating Models

Choose 3 supervised learning models that are available in scikit-learn, and appropriate for this problem. For each model:

- What are the general applications of this model? What are its strengths and weaknesses?
- Given what you know about the data so far, why did you choose this model to apply?
- Fit this model to the training data, try to predict labels (for both training and test sets), and measure the F1 score. Repeat this process with different training set sizes (100, 200, 300), keeping test set constant.
- Produce a table showing training time, prediction time, F1 score on training set and F1 score on test set, for each training set size.

Note: You need to produce 3 such tables - one for each model.

- ✓ I have run each of the following models 500 times and averaged their processing time and F1 results. At the **first level**, I have calculated their average statistics using out of the box model without any parameter optimization. At the **second level**, I have run the grid search for parameter optimization for all the models below and updated their parameters accordingly before calculating the statistics of processing time and F1 values. At the **third level**, I have run these models with PCA of smaller feature size. The main reason for PCA is to reduce the processing time while keeping the performance of the classifier at decent level.

- ✓ For each classifier, I have tree tables showing the results for each level. From the pdf file, the results of first level is shown in file: "*results_finetunemodel_False_pca_False_.pdf*", second level in file "*results_finetunemodel_True_pca_False_.pdf*", and the third level in file "*results_finetunemodel_True_pca_True_.pdf*"

- ✓ The parameters to control each level is as follows:
  - Fine_Tuned_Models = True
  - N_loop_stats_collect=500; # Number of Loops in Model Averaging
  - training_size = [300,200,100];
  - PCA_on = True;
  - n_components = 15 # Number of Features after PCA

- ✓ Keep in mind that PCA is not free, it requires decomposition and transform. I have collected average time of PCA Stats: Time for PCA Decomposition: [0.00290085]s , Time for PCA Transform: [0.00045978]s.

- ✓ **Decision Tree Classifier:**
  - **Strength:** Easy to understand the classification rules, easy to use / implement and build which requires low processing time, models non-linearity naturally, good fit for discrete data
  - **Weakness:** Sensible to noise in the data (small changes may cause instability), it is pruned to overfitting and not very good with continues data (increases complexity), not very good with linear combination of feature
  - **Why good?** This is a simple model and it provides results that is generally easy to understand with decision boundaries on different features. Furthermore, dimensionality of the dataset is **not** very large which still makes this model appealing since its complexity won't grow exponentially. Features are mainly discrete which makes it a good fit for decision tree based classifier.
  - **First Level:**
  - In this level, I used decision tree with no limit on depth of the tree. Thus for large training sets tree depth became high and model became complex. Thus in this case, I see that increasing training size did not help Test-F1 statistics because of overfitting.

Training Size	Training Time	Prediction Time of Training	Training F1	Prediction Time of Test	Test F1	Tree Depth
Tr=300	0.002349	0.000266	1.000000	0.000176	0.707578	13.808000
Tr=200	0.001547	0.000188	1.000000	0.000166	0.710980	11.944000
Tr=100	0.000855	0.000170	1.000000	0.000174	0.703222	8.998000

- **Second Level:**
- After running decision tree in gridsearch for optimized parameters such as "parameters_DT = {'max_depth':(list(range(3,11,1)))} " I found that tree-depth 4 did prune down the tree and reduced overfitting. In this case, I have seen almost 10% improvement on Test F1 values, while training F1 values decreased (this also shows the benefit of using smaller depth tree). Furthermore, reducing tree depth did not help processing time since, I assume, algorithm did grow the tree at its max size and pruned later. However, the processing time of decision tree is already at best compared to other models.

Training Size	Training Time	Prediction Time of Training	Training F1	Prediction Time of Test	Test F1	Tree Depth
Tr=300	0.001281	0.000208	0.861910	0.000182	0.781806	4.000000
Tr=200	0.000945	0.000196	0.874062	0.000196	0.774612	4.000000
Tr=100	0.000649	0.000200	0.904669	0.000162	0.745971	4.000000

- **Third Level:**
- PCA did help to reduce processing time of Decision Tree but it also reduced the Test F1. The benefit on processing time for Decision tree is not high enough to make PCA appealing for this case. Because processing time is already very low.

Training Size	Training Time	Prediction Time of Training	Training F1	Prediction Time of Test	Test F1	Tree Depth
Tr=300	0.002264	0.000126	0.864120	0.000074	0.751590	4.000000
Tr=200	0.001439	0.000127	0.882524	0.000086	0.741036	4.000000
Tr=100	0.000789	0.000071	0.922279	0.000083	0.718612	4.000000

- ✓ **KNN:**
  - **Strength:** KNN is good at capturing non-linearity, and works well in complex models, cost of learning is zero (lazy learning). No assumption for underlying model.
  - **Weakness:** Selecting k is important since it may lead to under fitting with high k and overfitting with low k. Complexity of prediction grows rapidly with increasing data size. Curse of dimensionality may degrade the performance when the dimensionality increases.

- **Why good?:** This is a good algorithm to model non-linearity in the underlying model. It has zero learning time, and with small size data it is fast during prediction also. Furthermore, the number of features is low; so the processing should be fast.
- **First Level:**
- In this level, I used KNN with k=5 as this is its default value. As one can see, training time is very low since this is lazy learner (instance based) while prediction time is much higher compared to its learning time.

Training Size	Training Time	Prediction Time of Training	Training F1	Prediction Time of Test	Test F1
Tr=300	0.000898	0.006284	1.000000	0.002366	0.782292
Tr=200	0.000665	0.003092	1.000000	0.001716	0.773601
Tr=100	0.000531	0.001216	1.000000	0.001170	0.766102

- **Second Level:**
- After running KNN in gridsearch for optimized parameters such as "parameters_KNN = {'n_neighbors':(list(range(1,10,1))),'weights':('uniform','distance')}". Gird search in this case did not provide a significant change on the default settings of KNN. Thus, the results are similar to First level.

Training Size	Training Time	Prediction Time of Training	Training F1	Prediction Time of Test	Test F1
Tr=300	0.000969	0.006697	1.000000	0.002471	0.782123
Tr=200	0.000727	0.003228	1.000000	0.001816	0.773276
Tr=100	0.000586	0.001322	1.000000	0.001301	0.764432

- **Third Level:**
  PCA did help to reduce processing time of KNN since reducing the dimensionality helps this type of classifiers since they calculate the distance between data points. On the other hand, I saw that the Test F1 got reduced after PCA.

Training Size	Training Time	Prediction Time of Training	Training F1	Prediction Time of Test	Test F1
Tr=300	0.000713	0.004296	1.000000	0.001587	0.758129
Tr=200	0.000558	0.001906	1.000000	0.001101	0.761201
Tr=100	0.000408	0.000768	1.000000	0.000765	0.762576

- ✓ **SMV:**

- **Strength:** It generally provides high accuracy. With its kernel tricks, this algorithm provides great flexibility to produce very good prediction results. It typically has no local minima.
- **Weakness:** May not be easy to find the optimum kernel and its parameters. It may be complex for multi-class problem (even though this is not a multi-class problem), can be slow due to quadratic optimization of the model**.** Maybe sensitive to outliers
- **Why good?:** It generally provides high accuracy in prediction.
- **First Level:**
- In this level, I used SVM with its default parameters such as RBF kernel with certain C and gamma values. Its training and prediction times are comparable while it requires much higher time compared to simple decision tree. On the other hand, it provides the superior performance with its default parameters.

Training Size	Training Time	Prediction Time of Training	Training F1	Prediction Time of Test	Test F1
Tr=300	0.007615	0.005616	0.867047	0.001919	0.804625
Tr=200	0.003786	0.002681	0.870490	0.001414	0.804096
Tr=100	0.001364	0.000887	0.871254	0.000876	0.799529

- **Second Level:**
- One of the drawbacks of SVM is its flexibility which makes it hard to find the optimum setting for it. I have tried different parameters for kernel, gamma, C values as parameters_SVM = {'C':(0.1,1,10,100),'kernel':('linear','rbf'),'gamma':(1/(n_features-20),1/(n_features-10),1/n_features,1/(n_features+10),1/(n_features+10))}. After consirable duration, it gave parameters which were very close to its default setting as RBF with gamma=1. Its Test F1 value has improved slightly with these new settings.

Training Size	Training Time	Prediction Time of Training	Training F1	Prediction Time of Test	Test F1
Tr=300	0.009733	0.007066	0.977967	0.002432	0.816996
Tr=200	0.004694	0.003356	0.985659	0.001720	0.811647
Tr=100	0.001645	0.001174	0.994762	0.000979	0.804722

- **Third Level:**
- PCA has reduced the processing time of the SVM since its complexity is linearly proportional to dimensionality of the data. Even though PCA did hurt the Test F1 results, it reduced the processing time more than half (~(48-15) /48). Furthermore, PCA has the least effect on performance on SVM compared to other classifiers.
-

Training Size	Training Time	Prediction Time of Training	Training F1	Prediction Time of Test	Test F1
Tr=300	0.005033	0.003140	0.906843	0.001064	0.807746

Tr=200	0.002582	0.001527	0.918143	0.000800	0.803328
Tr=100	0.000938	0.000521	0.937224	0.000461	0.795271

- ✓ **Adaboost:**
  - **Strength:** It requires less number of parameters to tweak compared to complex models like SVM. It improves accuracy through boosting of dataset and do not incur overfitting as long as weak classifier has error rate less than 0.5
  - **Weakness:** Sensitive to noise and outlier in the dataset and maybe slow if the data is noisy.
  - **Why good?:** It is not prone to overfitting and very simple weak classifier can be used. It eliminates some of the downsides of decision tree based classifier (assuming week classifier is a very simple decision tree classifier).
  - **First Level:**
  - In this level, I used Adaboost with its default parameters such as depth=1 decision tree and learning curve=1. Its default parameters provided decent performance which were higher than simple Decision Tree. However training time of Adaboost was considerable higher than other classifiers. This might be related to noise in the data which typically makes Adaboost converge harder. Similarly, the prediction time of the Adaboost was also higher compared to other classifiers. His might be due to number of cascaded stages Adaboost construct and again it might be related to noise and outliers in the data.

Training Size	Training Time	Prediction Time of Training	Training F1	Prediction Time of Test	Test F1
Tr=300	0.066295	0.006080	0.857363	0.004416	0.771200
Tr=200	0.059891	0.005348	0.883479	0.004426	0.759039
Tr=100	0.052225	0.004381	0.971998	0.004339	0.735550

  - **Second Level:**
    After running Adaboost in gridsearch for optimized parameters such as parameters_Adaboost = {'base_estimator':(tree.DecisionTreeClassifier(max_depth=1),tree.DecisionTreeClassifier(max_depth=2)),'learning_rate':(0.3,0.5,0.75,1,2)}. Decision tree with smaller depth (default setting) outperformed the more complex weak learner with higher depth. I have seen improvement only on the learning rate which became 0.5 after gridsearch Test F1 values improved slightly after the parameter update.

Training Size	Training Time	Prediction Time of Training	Training F1	Prediction Time of Test	Test F1
Tr=300	0.068654	0.006335	0.849821	0.004461	0.793118
Tr=200	0.061315	0.005427	0.867338	0.004537	0.781788
Tr=100	0.053592	0.004539	0.932466	0.004547	0.754248

- **Third Level:**
  PCA did not help processing time of Adaboost, it even did worsen them. One reason might be the introduction of more uncertainty to the dataset with PCA did reduce the performance of Adaboost. Because, any processing on the data will reduce the information captured by its original value (based on Information Theory). Thus relative noise of the data might have increased with PCA.

Training Size	Training Time	Prediction Time of Training	Training F1	Prediction Time of Test	Test F1
Tr=300	0.091790	0.006642	0.894947	0.004752	0.769934
Tr=200	0.077455	0.005594	0.931839	0.004708	0.754109
Tr=100	0.061848	0.004550	0.996421	0.004552	0.734527

-
-

✓ **RandomForest:**

- **Strength:** Runs efficiently on large dataset. Easy to understand. Improves tree based learning by reducing the bias to data through randomization of features.
- **Weakness:** Maybe slow because of the number of decision trees. It may still have the overfitting problem.
- **Why good?:** It eliminates some of the drawbacks of decision tree and improves the accuracy but requires mode processing.
- **First Level:**
- In this level, I used Random Forest with its default parameters such as 10 randomized tree with sqrt features. Its performance is better than simple Decision tree.

Training Size	Training Time	Prediction Time of Training	Training F1	Prediction Time of Test	Test F1
Tr=300	0.011863	0.001448	0.992704	0.001111	0.745882
Tr=200	0.010711	0.001261	0.992494	0.001057	0.741060
Tr=100	0.009818	0.001069	0.989889	0.001015	0.731549

- **Second Level:**
- After running RandomForest in gridsearch for optimized parameters such as parameters_RF = {'n_estimators':(list(range(6,17,2))),'max_depth':(list(range(2,11,2)))}, I have seen that increasing the number of trees did help in this case. Test F1 values improved slightly after the parameter update, while I have seen that processing time also increased slightly due to larger number of random trees.

Training Size	Training Time	Prediction Time of Training	Training F1	Prediction Time of Test	Test F1
Tr=300	0.016095	0.001841	0.995774	0.001312	0.764829

Tr=200	0.014710	0.001581	0.996372	0.001301	0.757755
Tr=100	0.013473	0.001296	0.993671	0.001244	0.746216

- **Third Level:**
  Similar to Adaboost, I have seen that PCA did not reduce the processing time of RF while it degraded the Test F1 value. .

Training Size	Training Time	Prediction Time of Training	Training F1	Prediction Time of Test	Test F1
Tr=300	0.022213	0.001834	0.995318	0.001274	0.758466
Tr=200	0.018770	0.001530	0.996083	0.001262	0.753474
Tr=100	0.015914	0.001284	0.993761	0.001248	0.746181

-
-

5. Choosing the Best Model

Based on the experiments you performed earlier, in 2-3 paragraphs explain to the board of supervisors what single model you choose as the best model. Which model has the best test F1 score and time efficiency? Which model is generally the most appropriate based on the available data, limited resources, cost, and performance? Please directly compare and contrast the numerical values recored to make your case.

- ✓ In my opinion, there are two competing classifiers based on the processing resource and F1 performance. One of them is simple Decision Tree with max_depth=4 which has very low processing resource requirement, and other one is SVM-RBF which provides the best F1 score.
- ✓ SVM-RBF gave the best F1 test result which is 0.817 with reasonable time with processing time of training 0.01s and prediction 0.08s approximately. It provides a good combination of the processing time, performance and decent memory requirement. Since it is parametric learner, it requires less memory compared to instance based learners like KNN.
- ✓ On the other hand, simple Decision Tree achieved 0.781 Test F1 value after fine-tuning with the best processing time for training of 0.003s and test of 0.0003s approximately. These processing times are order of magnitude smaller than what SVM required which makes this model more appealing if the board is interested in something with minimum processing time with decent accuracy.
- ✓ Based on the two arguments above, I would like to propose simple Decision with max_depth=4 as the final classifier, because its F1 score is reasonable (even though not the best) and its processing resource requirement is almost one or two order lower than SVM. In addition it is more understandable and easier to explain its decision mechanism to board.

In 1-3 paragraphs explain to the board of supervisors in layman's terms how the final model chosen is supposed to work (for example if you chose a decision tree or support vector machine, how does it learn to make a prediction).

- ✓ **How the algorithm works:** Decision-tree builds tree like structure with several branches and nodes where each node is associated with a single feature of the student. It starts from the root node and use a single feature to divide the data into groups (labeled as passed and

failed) by looking at their single feature associated with the node. For each group of data in the leaf of the root node, it picks another feature and splits data into pass and fail groups accordingly. At each node, it picks a feature that splits the data with best split (that provides the maximum information gain).

- ✓ **How the algorithm learns during the training phase:** During training phase, algorithm looks at one values of a student at a time such as school, failure, age etc. and picks one of those that will give the best way to decide whether that value is important or not.
- ✓ For instance, it looks at failure value and tries to understand whether it helps to distinguish which student is going to fail by just looking at it only. For instance, when failure value is zero, it is likely that students will pass the class. Then it does the same for another value such as age of the student. It looks at all the values one by one and finds the value that helps most in the process of finding whether a student will pass or fail. It selects the best out of all values of the student. After that, it looks at all the values for the students except the one which is used in the previous step. This process goes on until algorithm hits one of the stopping criteria.
- ✓ In technical terms, algorithm will maximize information gain at every step by looking at one feature at a time. It only picks the best using a cost function such as entropy or gini index

- ✓ **How the algorithm makes a new prediction:** After building its decision mechanism, for any new student, it follows the tree starting from its starting point. At every step, it looks at one value of the student and determines whether student is more likely to be in one group or another.
- ✓ For instance, if the first value is failure and second value is absance determined in the training. It uses that value of the student to see whether student will be in passed or failed group. Then it looks at absence value to finalize that student belongs to passed or failed class.

Fine-tune the model. Use gridsearch with at least one important parameter tuned and with at least 3 settings. Use the entire training set for this.

- ✓ The procedure is described in Q4. At second level of each classifier, I have run grid search on each one of the classifiers to find best parameters.
- ✓ For Decision Tree, max_depth limited at 4 gave the best result
- ✓ I have used the scoring function of f1_score

What is the model's final F1 score?

- ✓ If it is DT it becomes 0.78