Machine Learning Final Project Report

Shreya Ravi Patki
Net ID: sxp210305

Problem: Binary Classification (Comparison Study of different ML models for classification)

Dataset: HR Analytics Job Change of Data Scientists
About the dataset: This dataset is from a company who is hiring data scientists and has a record of people who have passed tests that the company conducts. The company wants to know which of these people will continue working for this company after they complete their training. It consists of 19158 rows and 14 columns. The entire dataset in divided in to two csv files out of which one is the training set and the other is the testing set. The dataset has been taken from Kaggle.

Aim: To classify the candidates into two classes: Yes and No.

Features: Enrolled id, City, City development index, Gender, Relevant Experience, Enrolled University, Education Level, Major Discipline, Experience, Company Size, Company Type, Last New Job, Training Hours, Target. The 'Target' variable holds two values: 0.0 and 1.0 for 'no' and 'yes'. This dataset consists of more 0.0 values compared to 1.0 values.

Exploratory Data Analysis

1. Getting the dataset ready for pre-processing:

- The Kaggle dataset consisted of two csv files (aug_train.csv and aug_test.csv). Firstly, both these csv's were merged into a single pandas data frame and this data frame was then converted to a final csv. All the data was merged into a single file for pre-processing the data together. The final csv was then divided into train test and validation sets using train_test_split.
- The final dataset consisted of 298018 rows and 14 columns.
- Initial correlation heatmap was plotted which just showed a 4*4 matrix of features as the rest of the features were categorical. The dataset had a lot of nan values that had to be dealt with.

2. Irrelevant Features:

- Enrolled_id: This is the unique id for each candidate in the dataset. This feature had no importance as it did not contribute to the final prediction in any way. This feature was numerical and hence we had 298018 different values randomly allotted to the candidates. This feature was dropped from the data frame.
  Getting rid of the feature was important, because it would otherwise produce an unwanted ordinality in the enrolled ids (E.g. candidate with id 4503 could be given more importance than candidate with id 1902).
- City: This feature consists of city codes (city_01, city_34, city_180 etc). The column had around 200 distinct values. However, this feature was not giving any meaningful information about the data as it just consisted of the city id, and hence it was dropped. Additionally, the city development index is an attribute that has good correlation with the data and was giving useful additional information about the city.

3. Dealing with NaN values:

- The nan values from the following columns were directly dropped:
  Education_level, company_size, last_new_job, enrolled_university, experience.
  These were the columns where I did not feel that replacing the nan values with mean/media/mode or some similar values would be a good option, because they would skew the prediction results.
- For the column 'major_discipline' I could not drop the nan values, because the education level of primary school and high school do not have a major, hence dropping the nan values would result in loss of data and I would just be left with three distinct values in 'education_level'.
  To overcome this problem, I created a separate 'No Major' category and relaces all the nan values with this value.
- I faced a similar problem for the enrolled university column. Dropping the nan values was causing loss of data, hence I first encoded this column using one hot encoder where I added an additional category 'nan' and then dropped it after the encoding was complete.

4. Encoding the categorical data:

- Ordinal Encoding: The following columns were encoded using the ordinal encoder from sklearn:

- Relevant Experience: This column has only two values ‘has relevant experience’ and ‘no relevant experience’. Label encoder was used to encode these values into 1 and 0 respectively.
- Education level: This column has five values ‘primary school, high school, graduate, masters, Phd’. These values were encoded in the same order from 0 to 4 in order.
- Major Discipline: ‘No major, Other, Arts, Humanities, Business Degree, STEM’ were encoded from 0 5 in the same chronological order. Since this dataset is for the job of data scientists, I gave the highest preference to STEM degree and then the rest of them.

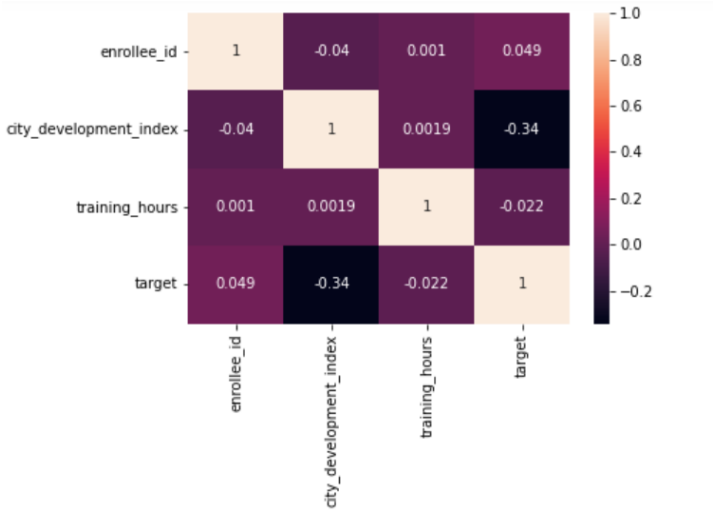
OrdinalEncoder is a pre-processing step used for encoding categorical features. It maps each unique value of a feature to a different integer. This encoding allows algorithms to correctly interpret the data and learn patterns from it. I have given the order of the mapping in the form of an array to the encoder.

- One Hot Encoding: The following columns were encoded using one hot encoder:
  Enrolled university, gender, company type. The values in these columns did not have a hierarchy, and no column was supposed to be given more importance than the other. Hence, one hot encoding was used for encoding purpose.
  One hot encoding is another way to transform categorical features into numerical values. It creates a new binary column for each category of a categorical feature. For example, if a feature has three categories A, B, and C, it will create three new columns with binary values for each category. If a data point belongs to category A, the new column for A will have a value of 1 and the new columns for B and C will have a value of 0.
  Hence, the one hot encoding produced multiple columns having binary values. The original columns containing categorical values were dropped from the data frame and the newly encoded columns were appended to the data frame.

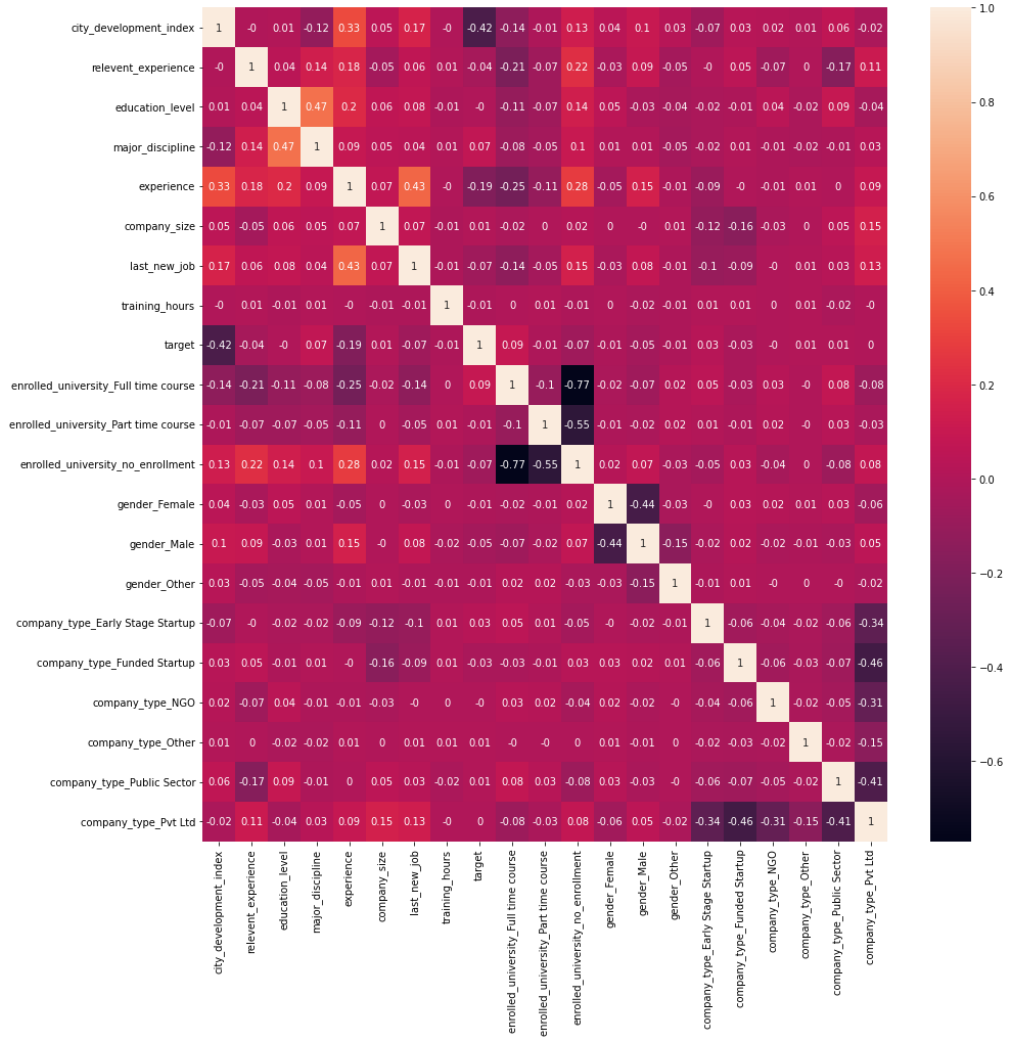
- Other ambiguous values:
  - Company size: This column had data in the form of range of values (50-99, 1000-4999 etc.) These values could not be directly used in the prediction, hence they had to replaced with mean of the range rounded off to the next whole number. (50-99 was replaced with 75, 1000-4999 was replaced with 3000 and so on). Two ambiguous values were ‘<10’ and ‘10000+’. There was no value below 10 and no value higher than 10000 which was explicitly stated in the dataset. Hence, these values were replaced with 5 and 15000 respectively.
  - Experience: This column again had a couple of ambiguous values that were ‘>20’ and ‘<1’. The >20 column was replaced with 21, as there was no other information about the candidates who had more than 20 years of experience. I had initially used a random number generator for digits between 20 to 40 to assign random values between 20-40 for >20 columns. However, it was not a very good idea as it would totally change the prediction every time because every time I train the model, the random values would be different. Hence replacing all values greater than 21 was the ideal option. The <1 values were replaced with 0.
  - Last new job: the column >4 was replaced with 5 for the same reasons as above. There was another value ‘never’ which was replaced with 0.

5. Feature Scaling:
   The two columns ‘training_hours’ and ‘company size’ were having values with very high ranges, hence they could possibly make the predictions biased. Therefor it was necessary to normalize these features. The standard scaler was used to scale these features down so as to not let them skew the prediction results.
6. Correlation matrix:
   After the entire dataset was pre-processed and all the values were now in place, I calculated the correlation matrix again. The correlation matrix stated that city development index and experience were the two most closely correlated features with the target variables. However, many features had a strong correlation with one another if not with the target variable. Hence, I decided not to drop any features that had a bad correlation with the target variable.

Correlation matrix before pre-processing:



Correlation matric after pre-processing



Therefore, it is evident that after converting all the categorical features into nominal and ordinal values, we can get a better idea of the correlation between different features.

Prediction using different ML models

A part of the dataset (10%) was set aside for validation purposes. The rest of the dataset was split into train and test sets. The validation set was used to run the GridSearch for finding the best set of hyper parameters and for hyper parameter tuning.

(Note: I played around a lot with the hyper parameters, but I have put the best few results for explanation.)

1. Support Vector Machine

Hyperparams	Accuracy	Precision	Recall	F1 Score	Cohen Kappa score	Exec Time
Kernel=linear, degree=1, gamma=0.001, C=0.05	82%	100%	0	0	0	2.06 sec
Kernel=polynomial, degree=1, gamma=0.1, C=0.05	82.3%	100%	0	0	0	1.04 sec
Kernel=sigmoid, degree=1, gamma=0.01, C=0.05	74%	21.1%	17.5%	19%	3%	1.97 sec
Kernel=rbf	82.3%	100%	0	0	0	5.314 sec

The first readings are the reading obtained by running grid search. The best hyper parameters obtained were kernel=linear, degree=1, gamma=0.001, C=0.05. When the model was run using these hyper parameter values, the accuracy and precision was very good. However, the rest of the metric values were 0. Hence this was a clear indication that the model was overfitting and was able to just predict one of the two classes correctly. The best performance among all these was the sigmoid kernel.

2. K Nearest Neighbors

Value of k	Accuracy	Precision	Recall	F1 Score	Cohen kappa score	Exec Time
5	81%	40%	13%	20%	12%	1.07 sec
8	82.2%	48%	5.8%	10%	6.8%	0.849 sec
14	82.2%	46.8%	4.4%	8%	5.2%	0.87 sec
20	82.6%	63%	4%	9%	6%	0.877 sec
40	83.4%	59.2%	2.3%	4.5%	3.2%	0.858 sec
100	82.2	37%	0.4%	0.8%	0.4%	0.99 sec

We can observe that as we increase the value of k, the recall and f1 scores start falling after k=8. This means as we increase the k neighbors, we start overfitting the data. At k=100, the recall, f1 and cohen kappa scores are almost close to 0. Hence the most ideal value for k is 8. K=20 can also be an ideal value but for k=8 the recall f1 and cohen kappa values are better. The grid search for best hyper parameters also gave a value of k=8.

3. Decision Tree Classifier

Hyperparams	Accuracy	Precision	Recall	F1 Score	Cohen kappa Score	Exec Time
Max depth=1, criterion=entropy	85.4%	59%	54%	57%	48%	0.018 sec
Max depth=4, criterion=gini	85.5%	60%	53%	58%	48%	0.02 sec
Max depth=6, criterion=entropy	85.2%	59%	52%	55%	47%	0.028 sec
Max depth=12, criterion=gini	81.%%	47%	41%	44%	33%	0.03 sec

As observed from the above data, for max depth=12, the tree definitely overfits as all the metric values start reducing drastically. Overall, by looking at all the evaluation metrics, depth=4 and criterion gini gives the best results. Max depth=4 and criterion=gini was the result given by grid search.

4. Logistic regression

Hyperparams	Accuracy	Precision	Recall	F1 Score	Cohen Kappa score	Exec Time
Penalty=l2, C=0.6	82.9%	55.8%	17.9%	27%	20%	0.046 sec
Penalty=l1, C=1	83%	57%	24%	34%	23%	0.05 sec
Penalty=l2, C=5	83.6%	58%	25.1%	35%	27%	0.04 sec
Penalty=l1, C=20	83.5%	58%	25.26%	35%	27%	0.127 sec

The grid search for hyperparameters gave the values l1 and C=1 for best values of hyperparameters. However, playing around with the values gave me two more readings that were better than the grid search by a small fraction. The last two values look slightly better than the one given by grid search. Hence, I selected the last value for the final comparison study.

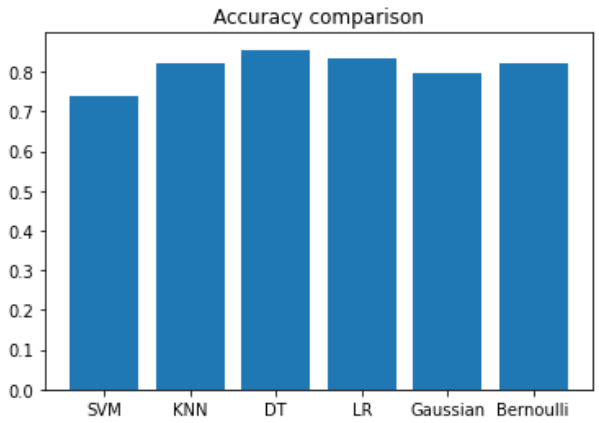
5. Naïve Bayes

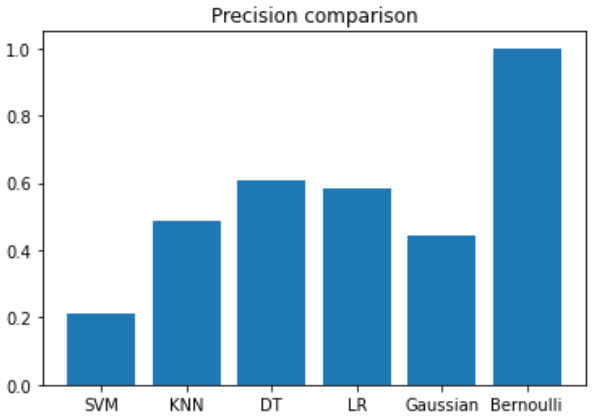
Type of NB	Accuracy	Precision	Recall	F1 Score	Cohen kappa Score	Exec Time
Bernoulli	82.3%	100%	0	0	0	0.015 sec
Gaussian	79.6%	44%	55.3%	49%	36%	0.014 sec

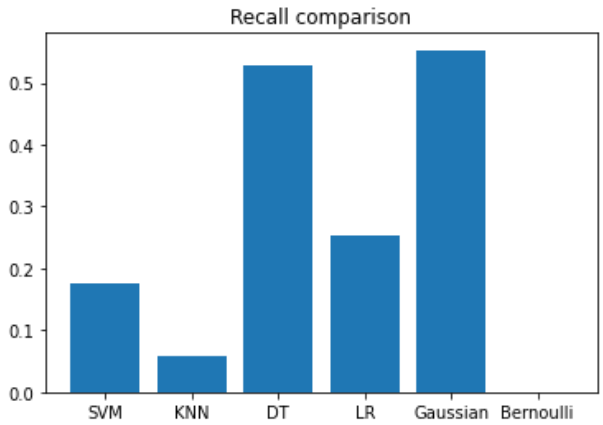
In naïve bayes, I used hyper parameter tuning, however the different hyper parameters were still giving me very similar results to what I have stated in the above table. Initially when I just had the accuracy measure, my Bernoulli Naïve Bayes seemed to perform better than Gaussian. When I added the rest of the parameters it was clear that the model was overfitting.

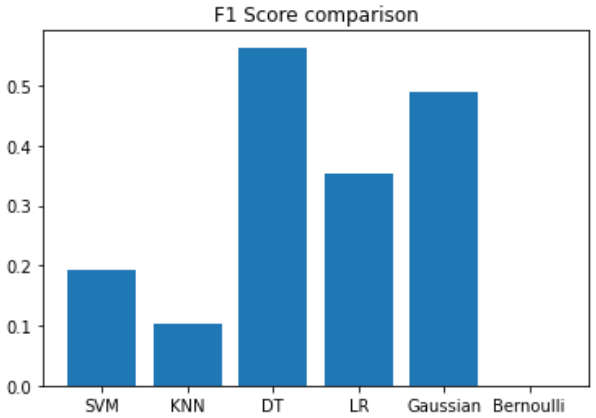
Comparison Study

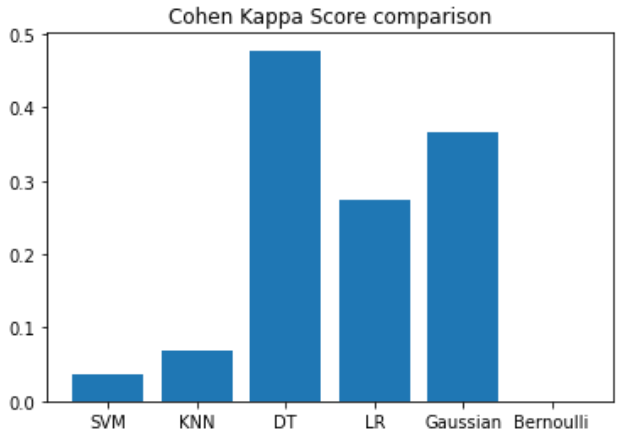
For each classifier I chose the best performing one for the comparison study. Following are the results in the form of tables for each evaluation metric:











- In terms of accuracy, all models are performing more or less the same. The decision tree has the highest accuracy of 85% and SVM has the lowest accuracy of 74%. Overall, just looking at the accuracies, all the models seem to be performing quite fine with the data.
- When we look at the precision score, Bernoulli Naïve Bayes has the best precision score of 1. This means that it is able to classify all the positive labels correctly. SVM has the lowest precision score which is around 0.2
- Recall and F1 scores give us a lot more clarity about the performance of our classifiers. Bernoulli NB has a recall and F1 score of 0. Hence, we can say that the classifier is not able to predict the negative class at all. KNN also has very low recall and f1 scores indicating its weakness in predicting negative class labels. Decision tree and Gaussian have a pretty good recall and f1 that states they are good with predicting both class labels.
- SVM and KNN have low Kohen Kappa scores that indicate that these models are not good while dealing with outliers. The decision tree has the highest Cohen Kappa score which shows that the classifier performs well even in presence of outliers.

Conclusion:

After an overall evaluation of the metrics, we can clearly see that the decision tree has the best scores and is performing well in all aspects compared to the rest of the classifiers. Gaussian Naïve bayes can be a close second.

There can be a few reasons as to why decision tree and naïve bayes perform better than the other classifiers:

1. Decision Tree is not sensitive to the choice of hyperparameters and Naïve Bayes is less sensitive to the choice of hyperparameters. SVM is sensitive to the choice of kernel, KNN is sensitive to the choice of k neighbors and logistic regression is sensitive to the choice of regularization parameter.
2. Decision trees use splits to separate the data into different regions. Hence, they are not greatly affected by outliers which is why we have the highest Cohen Kappa score for decision trees. Other models like SVM, KNN and logistic regression are affected by outliers very easily.
3. Both, decision trees and Gaussian Naïve Bayes can handle non-linear data well. SVM and KNN are weak at handling non-linear data while logistic regression handles non-linear data well if we perform feature engineering.

Decision Tree is the best performing algorithm on the given dataset. Ensemble methods like random forests, Adaboost and gradient boosting were not considered in this project however they might have a better performance than what the decision trees, as they combine the results of multiple models to produce a more accurate prediction.

References

https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn
https://machinelearningknowledge.ai/knn-classifier-in-sklearn-using-gridsearchcv-with-example/
https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search/
https://www.simplilearn.com/tutorials/scikit-learn-tutorial/sklearn-decision-trees
https://www.projectpro.io/recipes/optimize-hyper-parameters-of-decisiontree-model-using-grid-search-in-python
https://www.analyticsvidhya.com/blog/2021/01/gaussian-naive-bayes-with-hyperpameter-tuning/https://bait509-ubc.github.io/BAIT509/lectures/lecture6.html