Documentation of process

Name: Darsh Kachroo

Mailing Address: kachroo.darsh@gmail.com

Problem Statement: Predicting Employee Attrition

Dataset: IBM HR Analytics Employee Attrition & Performance

Source: Kaggle

Introduction

This report delves into the IBM HR Analytics Employee Attrition & Performance dataset which provides information on employee demographics, work experience, compensation, job satisfaction, and, most importantly, their decision to stay with or leave the company (attrition).

I have used various methods and analysis to find the contributing factors for attrition which largely include:

- Cleaning Dataset
- Data Analysis
- Data Processing and Feature Selection
- Feature Extraction
- Model Training
- Model optimization
- Results

Cleaning Dataset

This step is the most important step for all tasks going forward as any inconsistencies not found will heavily impact the features and model performance.

Data Analysis

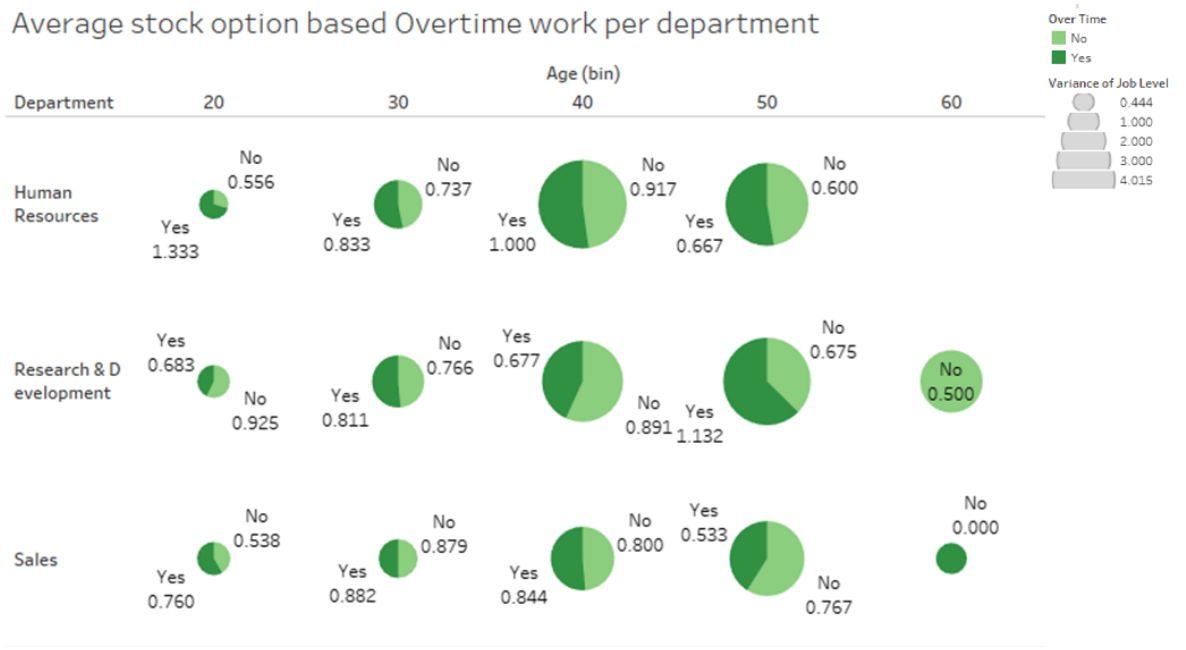
This step is as curtail as cleaning the dataset as the inferences are made in this step.

I start with loading the dataset into Tableau Desktop, which is an industry standard data visualization tool along with PowerBI and plotted comparative charts between significant columns.

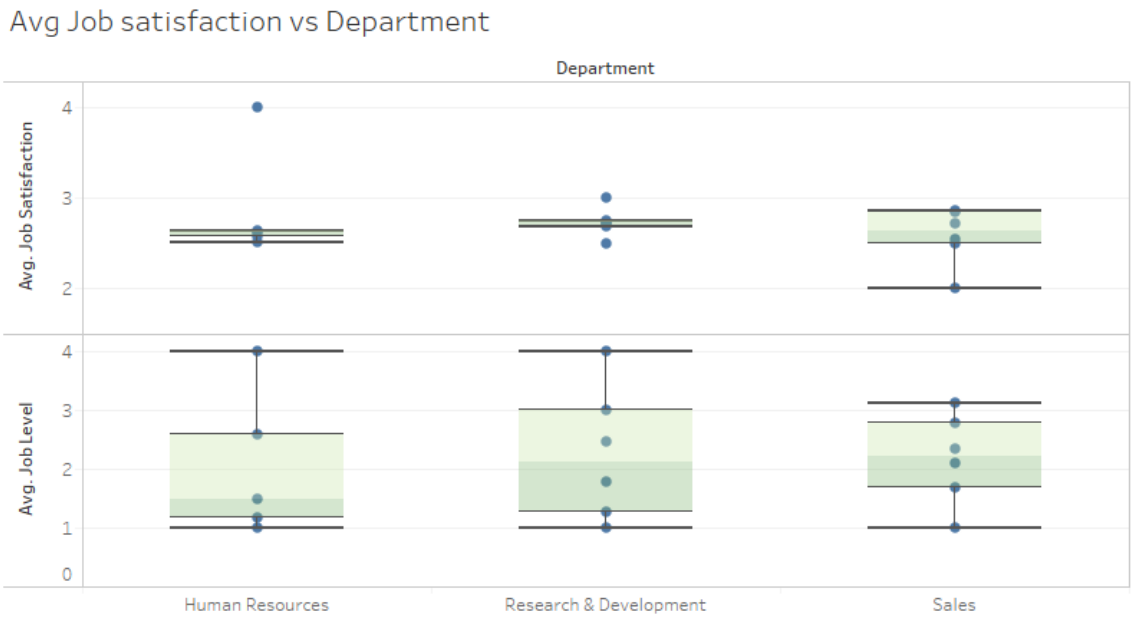
Tableau Dashboard created for the dataset

The created dashboard consists of 3 parts each analysing a different part of the dataset in a unique way. The first part - in the top left hand corner - represents the average stock options based on if the employee has done overtime work in their department, age, and the job level, the second part - in the top right hand corner - represents the average job satisfaction of the employee in their department, and finally - under both the sections - the Hourly rate vs the Training time of the employee based on their gender, and job satisfaction is visualized.

1. **Average Stock Option based on Over Time work:** Here each department is measure against each other by comparing the average stock option level the employee received based on if they had done overtime work or not. Additionally, the size of the pie chart represents the job level of the employee. Since it is taken per department, the size of the pie chart is determined by the variance of the job level in the department. This is done for all age group in increments of 10years.
   This showcases some clear trends:
   a. As age increases, the variance increases for all departments. This is attributed to freshers being placed at almost the same job level but as time increases, they start to get promoted “even out” the job levels.
   b. In the Human Resource department, the ratio of stock option level given to those who work overtime to those who don’t evens out to 50% after being highly biased in the start to employees who work overtime. This is attributed to companies wanting new hires to stay with the company longer and be more productive. Hence, better incentives are provided for them.
   c. In the Research and Development department, there is no clear pattern between the average stock option level to the overtime work but there is a human reason to the pattern based on the age. Since research and development requires skill and experience which freshers lack, they are given higher incentives to promote staying at the company (as backed by the significant difference around age 20-30.
   However, the experienced employees are provided with higher level stock options for not doing overtime as the company wants them to stay. This mean the incentives for the older experienced employees are more biased towards less work. Quality over Quantity.

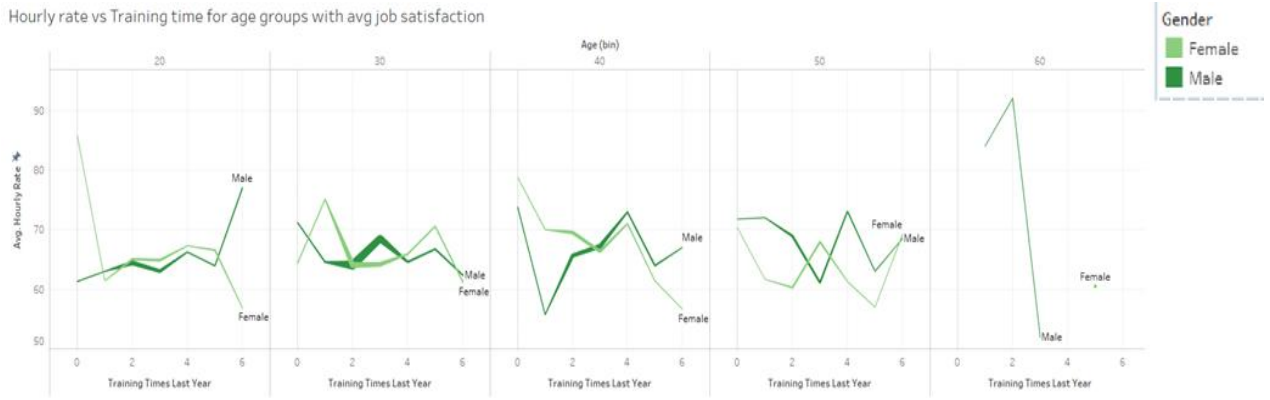


2. **Average job satisfaction and level vs Department:** This graph showcases the Interquartile range for the average job satisfaction in each department. Such data is useful as it allows us to see how the general employees feel about their work and find outliers. Negative outliers can be managed by the company via an intervention and discussions about their work.
   The average job level is also important as it provides insight on where the company should increase the hires, by what amount, and at what level.



3. **Hourly rate vs Training Time:** The hourly rate vs training time is one of the most important results in terms of team management and levels. It is also important since it is a topic which is highly personal to the employees and them feeling they are fairly compensated goes a long way for a company. Separating the data based on gender and making sure they as close is also important for the company to maintain.
   From the data, I observed that young woman who are freshers have the highest hourly pay whereas young men who are also freshers have the lowest hourly pay in their age group with

the difference of ~$20 per hour. It is further observed that as the young employees are trained, they are having similar pay and men showing more job satisfaction as compared to women. However, it should be noted that as women are trained for more time, their average pay declines with a maximum difference of ~$20 per hour for women in their 20's with 6 years of experience.
The cause for such extreme difference for both the sexes is concerning but many other factors may come into effect such as marriage and other opportunities.
Furthermore, it is observed that in general employees with medium amount of experience are the most satisfied with their job with little pay differences between the sexes.



Data Processing and Feature Selection

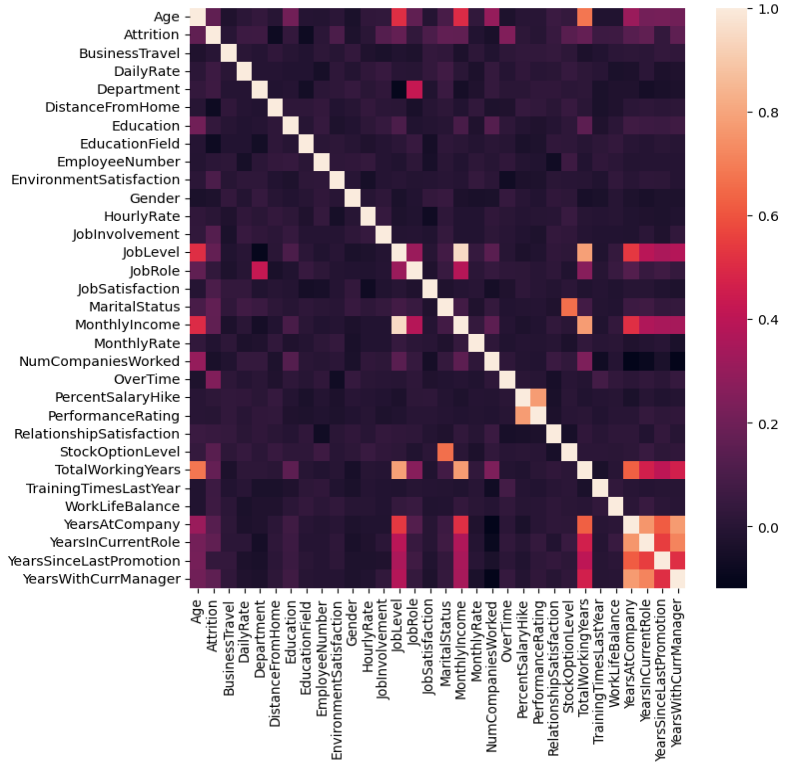
This step deals with the data from the perspective of a model. Processing the data such that we are able to make meaningful predictions from it is the core concept behind it.

For data processing, I have gone with label encoding and normalizations/scaling as the main processing steps.

I found that a few columns were strings and majority were integer values. To fix this inconsistency, I created a hash map which maps the string value to an integer value for every unique string in that column. Applying the map to each column, we successfully encoded the various strings into numeric data which can be processed, and the respective hash m aps were stored in case they are needed.

For feature selection, I have gone with chi squared test and correlation-based methods. Both are commonly used feature selection methods with the major difference being that chi squared method can give till k^{th} important feature whereas for the correlation method, it is dependent on how many features are above a threshold.

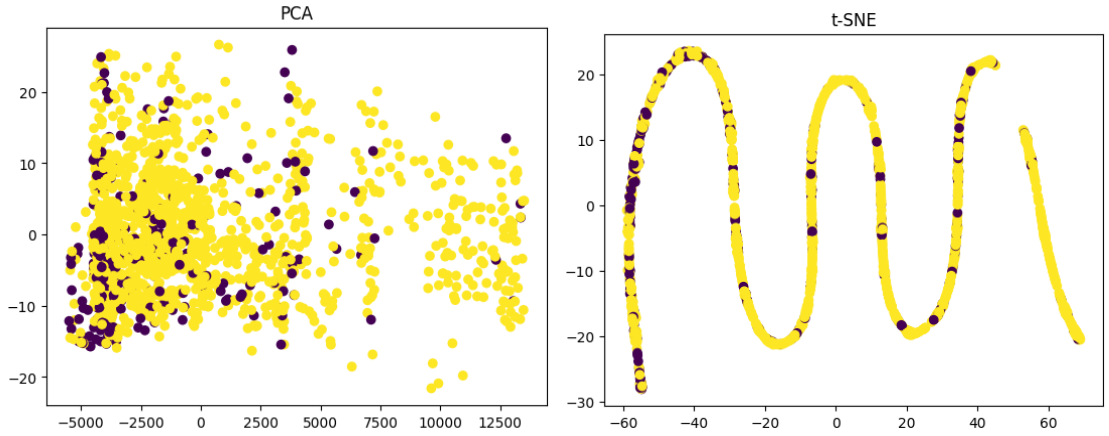
In the code, I have show with the example of feature selection with correlation method. There is an extra step which is done before getting the correlation matrix, it is to remove the columns where the standard deviation is zero. This would result in division by zero and hence would pollute the correlation matrix with null values, making them necessary to remove. These are significant as many columns like Over18 and EmployeeCount have 0 standard deviations.

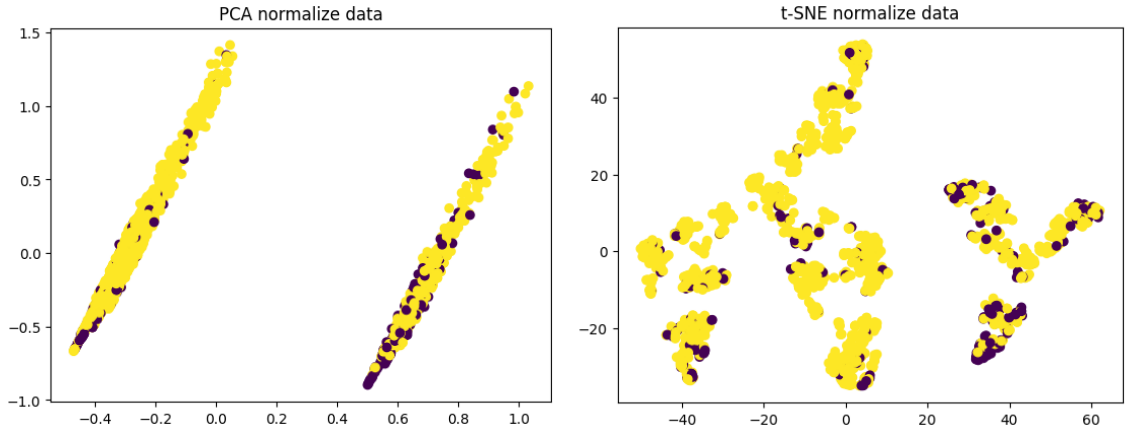


Feature extraction

After the feature columns are extracted, the data processing functions care called. The data is processed 6 times where PCA, t-SNE and no change is applied to both unnormalized and normalized data. This way we can accurately measure how well each of the dimension reduction methods worked in comparison to each other.

A constant which was included is that the dimensions are reduced to 2 in PCA and t-SNE solely for the purpose of being able to visualize the data clearly.





It is found that applying t-SNE to the unnormalize data gave rise to a wave like pattern. This gives us insight into which model might works the best for unscaled t-SNE data.

Model Training

Model training includes testing out various models for the binary classification task. I have taken 7 independent models to thoroughly test on the 6 data I created before:

1. Linear Regression
2. Logistic Regression
3. Stochastic Gradient Descent Classifier
4. Decision Tree
5. Random Forest
6. Multi-Layer Perceptron
7. Gaussian Naive Bayes

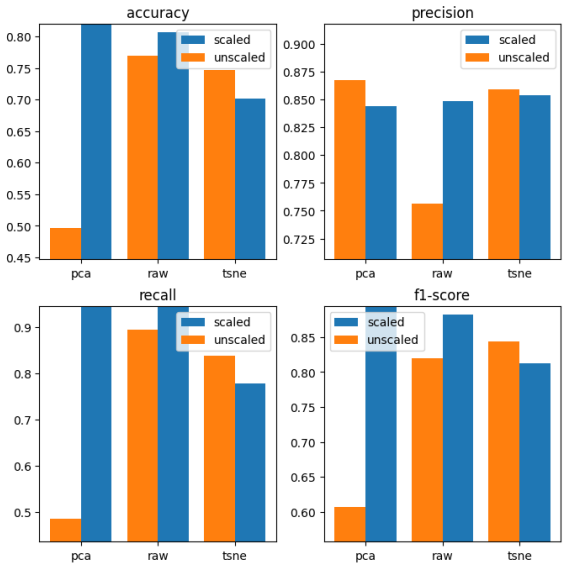
The models are trained, and their results are stored in a hash map as well as a table for easy visualization. The hash map allows us to obtain the model results by using the datatype used, normalization and the model's name. This gives us the total number of models trained to 42.

Each model is trained "N" number of times, and the average metrics are taken. This is done to make sure that "lucky" models aren't the one getting compared. It also provides us with a more general expectation from the model archetype.

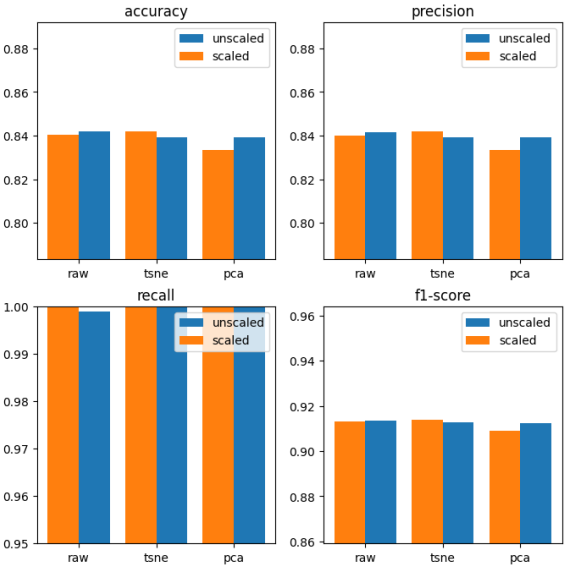
In the code N is set to 10 to speed up the training time.

The resulting table is sorted in descending order based on accuracy and the comparative model metrics are visualized.

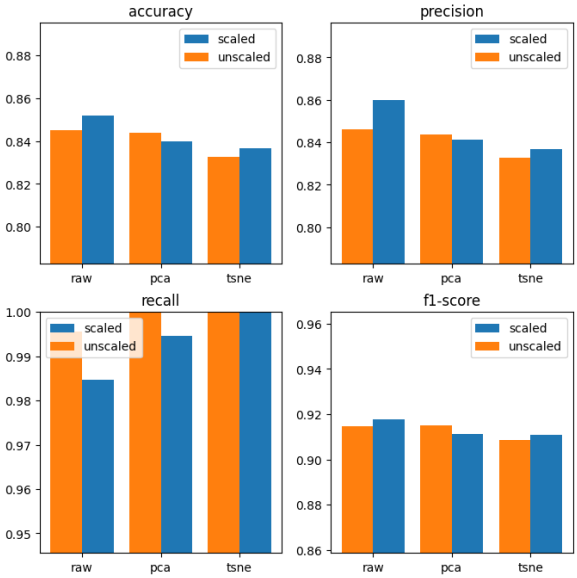
SGD Classifier results between data types



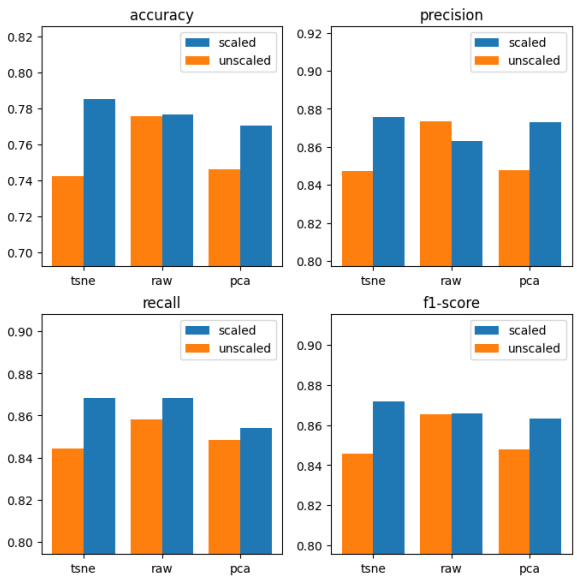
Linear Regression results between data types



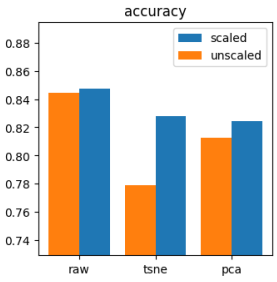
Logistic Regression results between data types

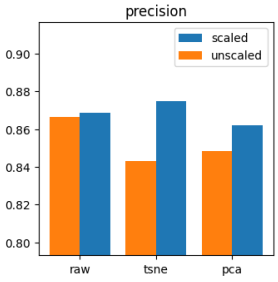


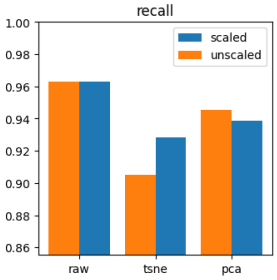
Decision Tree results between data types

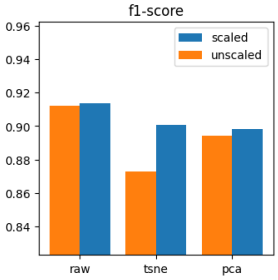


Random Forest results between data types

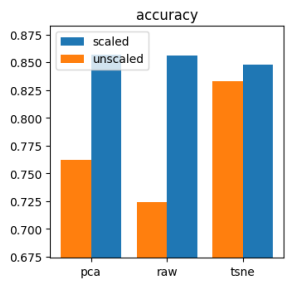


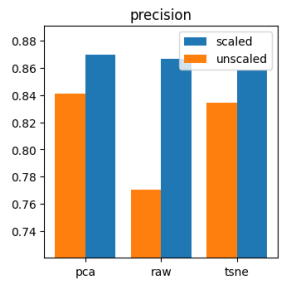


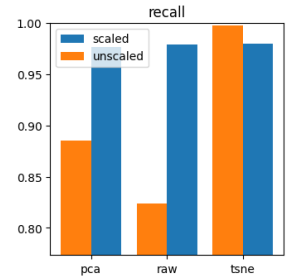


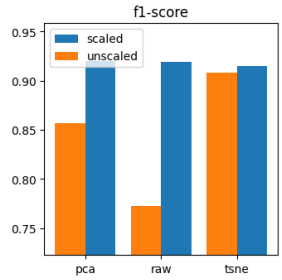


MLP results between data types

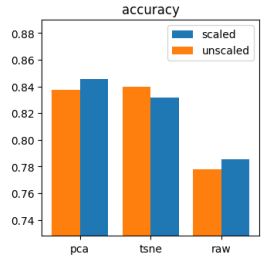


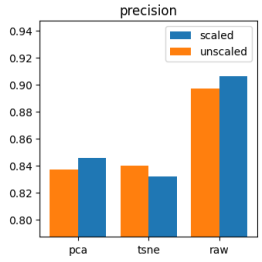


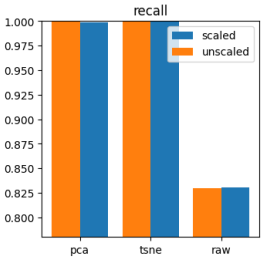


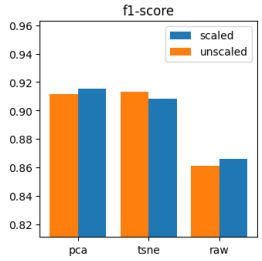


Gaussian Naive Bayes results between data types









Best Models based on all combination of models and data processing

	Model Name	model	data type	normalization	accuracy	precision	recall	f1-score
0	MLP	MLPClassifier()	pca	scaled	0.857202	0.869666	0.976852	0.919986
1	MLP	MLPClassifier()	raw	scaled	0.855967	0.866553	0.979424	0.919469
2	Logistic Regression	LogisticRegression()	raw	scaled	0.851852	0.859719	0.984684	0.917819
3	MLP	MLPClassifier()	tsne	scaled	0.847531	0.857908	0.980104	0.914865
4	Random Forest	(DecisionTreeClassifier(max_features='sqrt', r...	raw	scaled	0.847325	0.868697	0.963089	0.913388
5	Gaussian Naive Bayes	GaussianNB()	pca	scaled	0.845267	0.845478	0.998292	0.915492
6	Logistic Regression	LogisticRegression()	raw	unscaled	0.845062	0.846246	0.995558	0.914838
7	Random Forest	(DecisionTreeClassifier(max_features='sqrt', r...	raw	unscaled	0.844444	0.866581	0.962985	0.912176
8	SGD Classifier	SGDClassifier()	pca	scaled	0.843827	0.844176	0.999256	0.915159
9	Logistic Regression	LogisticRegression()	pca	unscaled	0.843827	0.843827	1.0	0.915195
10	Linear Regression	LinearRegression()	raw	unscaled	0.841975	0.841604	0.999031	0.913489
11	Linear Regression	LinearRegression()	tsne	scaled	0.84177	0.84177	1.0	0.913939
12	Linear Regression	LinearRegression()	raw	scaled	0.840535	0.839932	1.0	0.912964
13	Gaussian Naive Bayes	GaussianNB()	tsne	unscaled	0.839918	0.839918	1.0	0.912945
14	Logistic Regression	LogisticRegression()	pca	scaled	0.839712	0.84131	0.994584	0.911387
15	Linear Regression	LinearRegression()	tsne	unscaled	0.8393	0.8393	1.0	0.91262
16	Linear Regression	LinearRegression()	pca	unscaled	0.839095	0.839095	1.0	0.912479
17	Gaussian Naive Bayes	GaussianNB()	pca	unscaled	0.837243	0.837243	1.0	0.911347
18	Logistic Regression	LogisticRegression()	tsne	scaled	0.836626	0.836626	1.0	0.911011
19	Linear Regression	LinearRegression()	pca	scaled	0.833333	0.833333	1.0	0.909054

Model Optimization

In this we attempt to make the model better based on heuristics interpreted from data via analysis and visualization. Other factors also play a role such as model type, model parameters, data processing and so on and so forth.

1. *Hyperparameter tuning*

I have implemented a grid search by setting a state space for every model type.

```
models = [ # define all models
  LinearRegression(), LogisticRegression(), SGDClassifier(),
  DecisionTreeClassifier(), RandomForestClassifier(),
  MLPClassifier(), GaussianNB()
]
parameters = [ # define parametr space fo reach of the models
    {},
    {'C':[0.001,0.01,0.1,1,10,100], "solver":['lbfgs', "liblinear", 'newton-cg', 'newton-cholesky', 'sag', 'saga']},
    {'loss':['hinge', 'log_loss', 'modified_huber', 'squared_hinge', 'perceptron', 'squared_error', 'huber', 'epsilon_insensitive', 'squared_epsilon_insensitive']},
    {'criterion':['gini','entropy'],'max_depth':[4,5,6,7,8,9,10,11,12,15,20,30,40,50,70,90,120,150]},
    {'bootstrap': [True, False],'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None],'max_features': ['auto', 'sqrt'],'min_samples_leaf':[1, 2, 4],
    'min_samples_split': [2, 5, 10],'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]},
    {'hidden_layer_sizes': [(10),(20),(10,10),(10,20),(20,10),(20,20)],'activation': ['tanh', 'relu'],'solver': ['sgd', 'adam'],'alpha': [0.0001, 0.05],'learning_rate':
    {'var_smoothing': np.logspace(0,-9, num=100)}
]
```

This approach ensures that any model selected as the top model will be optimized. These models will be trained on every combination of the parameters to ensure the best model parameters are selected. This type of search is called an Exhaustive Search or Brute Force or Grid Search. It is useful in finding optimum parameters however the exhaustive nature of the method doesn't make it fast. Other search methods based on heuristics will be faster but may not give the best result. Since the state space is small for majority of the model, I have gone with this approach.

In the optimization, only the top "N" model is optimized as we do not want to waste time rerunning all 42 models. This makes only the models with the best accuracy to be optimized.

If the optimized model is performing better than existing model, then the existing model is replaced in the updated table.

There is a case where the trained model is not as good or better than the existing model, then we don't replace the model and the updated table column remains the same.

2. *Ensembling methods*

This method involves combining multiple models into a single model by taking the average of their predictions as a single prediction. For classification cases, majority voting is used instead of taking the average.

I have created the ensemble model by utilizing the top performing models via the updated table. However, each model has a different data requirement. These issues can be resolved by incorporating a list of preprocessing function along with the model. This way when a new prediction has to be made from a dataset which isn't previously made into the 6 datatypes, it can still predict it.

To get the proper processing functions I used the data processing class which was constructed before and having it create an instance of the processing functions. For future data, the same processing functions have to be used, otherwise the new data may be scaled inappropriately resulting in inconsistencies.

```python
class Ensamble_model():
  def __init__(self,models_arr,preprocessing_arr):
    self.models_ensemble = models_arr
    self.preprocessing_ensemble = preprocessing_arr
    self.setup = [False]*len(models_arr)

  def predict(self, x_test):
    results = []
    for i in range(len(self.models_ensemble)):
      if self.setup[i]:
        x = self.preprocessing_ensemble[i].transform(x_test)
      else:
        trained_processing_scalar,x = self.preprocessing_ensemble[i](x_test)
        self.preprocessing_ensemble[i] = trained_processing_scalar if trained_processing_scalar != -1 else self.preprocessing_ensemble[i]
        self.setup[i] = trained_processing_scalar != -1
      model = self.models_ensemble[i]
      results.append(model.predict(x))
    results = (np.mean(results,axis=0) > 0.5).astype(int)
    return results
```

It is assumed that for the first time when the model is used the preprocessing function will return the scaler function and the transformed weight. If the preprocessing function doesn't wish to return a scaler, then it will return -1 for which the preprocessing function should be a function of the data itself.

Results

Overall, it was found that the hyperparameter tuning was useful as it allowed for an increased accuracy as compared to the gird search. Moreover, ensembling also allowed the use of the top models such that they cover each of the pitfalls and many times increased the accuracy.

The average accuracy of each model can be clearly seen here