Introduction - Libraries, GridSearch, SimpleImputer Pipeline

Pandas is used for gathering the dataset and do manipulation on it. Numpy is used for array manipulations. SciKit which is used in machine learning was used mainly used for this project. “GridSearchCV” is an estimator used with other estimators like SVM, decision tree etc. GridSearchCV takes an estimator, parameters of that estimator and number of cross-validation. Then by using cross validation it takes the best parameters, fit the data and then make prediction. This fitting and prediction process done according to that estimator (or machine learning algorithm being used) ("sklearn.model_selection.GridSearchCV — scikit-learn 0.24.0 documentation", 2021). SimpleImputer is a package used to deal with missing values within the dataset. For this project, missing values in the column will be replaced with mean of that column/attribute. Pipeline is used to pre-process the data before using the estimator. It able developers to do pre-processing before using the algorithm which is either classifier or regression ("sklearn.pipeline.Pipeline — scikit-learn 0.24.0 documentation", 2021). Example can be seen below shows that within the pipeline first SimpleImputer was used to deal with missing values ("sklearn.impute.SimpleImputer — scikit-learn 0.24.0 documentation", 2021). StandardScaler was used to normalise the values in the dataset ("sklearn.preprocessing.StandardScaler — scikit-learn 0.24.0 documentation", 2021) and then use classifier to make predictions.

```
#Used pipeline here, what will happen is that first fill the missing values, scaledown the values then
#pipeline will be used in later line
pipe = Pipeline(steps=[('imputer',imp),('scaler', sc), ('classifier', clf)])
```

Part 2 - In this section classification algorithms have been used to decide if the customer who have the insurance will file a claim or won’t file a claim which are decision tree classifier, support vector machine classifier and logistic regression classifier were used. For each different classification train_test_split method was executed therefore for each classifier, different training-set and testing-set generated from that dataset ("sklearn.model_selection.train_test_split — scikit-learn 0.24.0 documentation", 2021). Pipeline generated and inside SimpleImputer, StandardScaler and then classifier were placed as parameters. For each classifier, different parameters were which will be discussed in more detailed. Then pipeline and parameters were used in GridSearchCV with cv parameter set to 10 which means 10-fold cross validation will be set ("sklearn.model_selection.GridSearchCV

— scikit-learn 0.24.0 documentation", 2021). After parameters were set, model was fitted, model did prediction then showed the assessment on a classification report.

First algorithm was Decision Tree was used and got 78% accuracy. For decision tree only parameter was chosen was classification criterion. Other parameters were left as default values. Classification criterion parameter select either if gini or entropy should be considered to split the decision tree ("sklearn.tree.DecisionTreeClassifier — scikit-learn 0.24.0 documentation", 2021). The detailed assessment can be seen below:

	precision	recall	f1-score	support
False	0.80	0.80	0.80	244
True	0.76	0.77	0.76	206
accuracy			0.78	450
macro avg	0.78	0.78	0.78	450
weighted avg	0.78	0.78	0.78	450

After using GridSearchCV to make predictions, in next line best possible parameters were shown below. These parameters were stored in a variable called param_grid. The assessment of the parameters was according to values within the param_grid. Below show that tree will be divided according to information gain within each node within the tree ("sklearn.tree.DecisionTreeClassifier — scikit-learn 0.24.0 documentation", 2021).

```
{'classifier__criterion': 'entropy'}
```

Second algorithm was Support Vector Machine. However, with default parameters accuracy was better than decision tree with 81%. In prediction values which are false improved while predicting data having “true” as class wasn’t improve much. There is a 1% increase in f1 score.

	precision	recall	f1-score	support
False	0.82	0.83	0.82	244
True	0.79	0.78	0.79	206
accuracy			0.81	450
macro avg	0.81	0.80	0.81	450
weighted avg	0.81	0.81	0.81	450

Then different parameters were such as C value, gamma and kernel were considered. Parameter C and gamma tunes the bias vs variance trade-off (Citi, 2020). If C value become too high, then model can be overfit and have high variance. In other case model can be underfit with huge bias (Citi, 2020). The assessment below shows that there is huge improvement for predicting both classes.

	precision	recall	f1-score	support
False	0.90	0.89	0.90	244
True	0.88	0.89	0.88	206
accuracy			0.89	450
macro avg	0.89	0.89	0.89	450
weighted avg	0.89	0.89	0.89	450

After using gridsearchcv to make predictions, in next line best possible parameters were shown below. These parameters were stored in a variable called param_grid. The assessment of the parameters done according to values within the parameter. C value is 1000. However, higher C value might overfit the model and then performance can be suffered in later predictions.

```
{'classifier__C': 1000, 'classifier__gamma': 0.01, 'classifier__kernel': 'rbf'}
```

Lastly, Logistic Regression was used with its default values. It is a classifier which can be used to predict two different classes. Result was similar with SVC with default parameters and somewhat better then decision tree classifier in accuracy wise.

	precision	recall	f1-score	support
False	0.82	0.86	0.84	244
True	0.82	0.78	0.80	206
accuracy			0.82	450
macro avg	0.82	0.82	0.82	450
weighted avg	0.82	0.82	0.82	450

Part 3 - In this section, regression algorithms were used because in this section the task is to recognize if the customer will make a claim but also to predict the value of the claim. There were two different datasets which is training dataset and testing set. Both have non-numeric values which will be treated in later stage. Like in second section “GridSearchCV”, ”StandardScaler”, “Pipeline” and “train_test_split”. GridSearchCV is used to find best parameters that is given by developer as parameters, make cross-validation, fit the data and make prediction. However, GridSearchCV doesn’t do the fitting and prediction itself but instead done by classifier which is directly given as estimator or as a Pipeline which contain estimator. Therefore, fit and prediction done according to classifier. Standardscaler used to normalize the values within the dataset. “train_test_split” is used to split data. However, in this section, OneHotEncoder and “make_column_transformer” were used. OneHotEncoder used to transform non-numeric values to numeric values ("sklearn.preprocessing.OneHotEncoder — scikit-learn 0.24.0 documentation", 2021). “make_column_transformer” was used to change columns in the dataset which according to different method which is OneHotEncoder in this case ("sklearn.compose.make_column_transformer — scikit-learn 0.24.0 documentation", 2021).

In the “make_column_transformer”, takes OneHotEncoder, columns which are F1 and F10 and remainder value which is values which are either numeric or value that unspecified ("sklearn.compose.make_column_transformer — scikit-learn 0.24.0 documentation", 2021). “make_column_transformer” object will transform the columns F1 and F10 will be transformed according to “OneHotEncoder”. However, “make_column_transformer” will be used in “Pipeline”.

Three different classifiers were used which are; LinearRegresson, Support Vector Regressor or SVR and GradientBosstingRegresson and for each classifiers, “train_test_split” was used to split the data (which is training data in this case) to training and testing set randomly with random state parameter. After splitting data to training data and testing data, Pipline was created. In the “Pipeline”, “make_column_transformer”, Standardscaler then classifier was used. An example can be seen below.

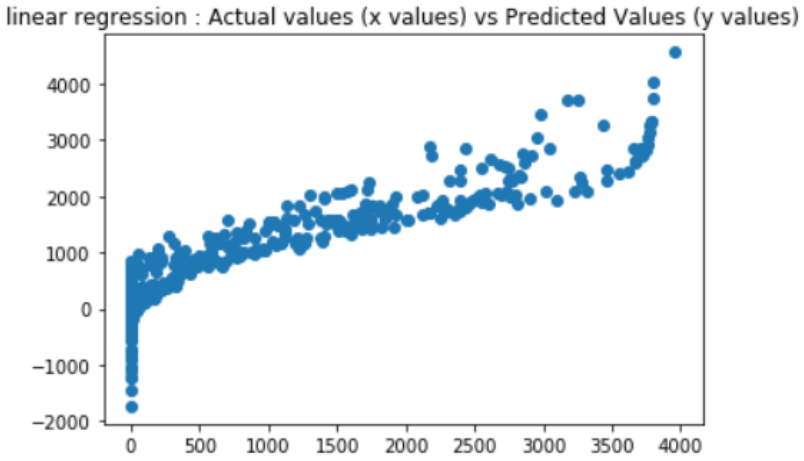
```
pipe = Pipeline(steps=[('trans',column_trans),('scaler',StandardScaler()), ('classifier', multiple_lr)])
```

Each of the parameters which are object will be executed sequentially within GridSearchCV. “column_trans” which is “make_column_transformer” will transform the non-numeric values to numeric values. Then, StandardScaler will work and multiple_lr which LinearRegression will work. Pipeline will be used in the GridSearchCV as parameter. Within the GridSearchCV, cv parameter which cross-validation set as 10 therefore 10 fold cross validation will be done. Then model will be fitted then predictions will be made and in the end assessment will be shown. This process done for each classifier. An example of this process can be seen below:

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state = 101)

from sklearn.linear_model import LinearRegression


multiple_lr = LinearRegression()
pipe = Pipeline(steps=[('trans',column_trans),('scaler',StandardScaler()), ('classifier', multiple_lr)])
param_grid = {'classifier__fit_intercept':[True,False]}
linear_gscv = GridSearchCV(pipe, param_grid, cv=10)
pipe.fit(X_train,y_train)
predictions= pipe.predict(X_test)
print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
plt.scatter(y_test,predictions)
```

First algorithm was Linear which results can be seen below. Parameter was named as “classifier__fit_intercept” where we decide either if intercept should be calculated or not. According to graphic, when estimator trying to predict data having “target” column 0 (so y value is 0) data in the graphic varies between -2000 to 1000. Additionally, after between 0 to 1000 at x, data start varies increasingly.

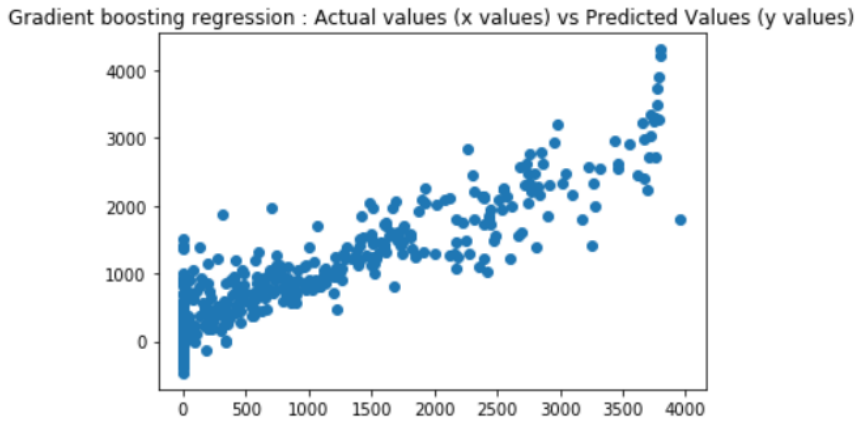
```
MAE for linear regression: 388.8554263859136
MSE for linear regression: 234093.30506316727
RMSE for linear regression: 483.83189752554273

<matplotlib.collections.PathCollection at 0x1773ee06d88>
```



Second algorithm was Gradient boosting regression. Gradient boosting regression is an algorithm uses decision tree regression. It takes a learning rate and uses residuals. Then tree look at the data that's being predicted and decide which residual it should be used in the decision tree. Then using value of the data, learning rate and residual new value gathered new prediction was made then comparing it to data within the dataset and get new residuals (Starmer,J, 2021).

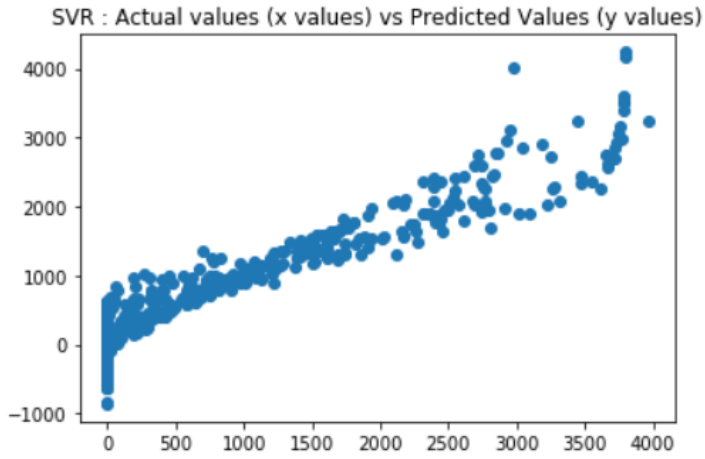
New tree created with those residuals and next time, the residual within the new tree will also be used with previous tree. Until no improvement in prediction, this continues. Mean absolute error (therefore Mean Square Error and Root mean square error) are lower than first algorithm therefore, it indicates improvement in performance (Starmer, 2021). Data in the graphic can be below, predictions vary below 0 to 1000 at x = 0 which indicate improvement from second and first algorithms. After x=1500, predictions vary more than second algorithm which might show how mean squared error is higher than second algorithm.

```
MAE for Gradient boosting regression: 338.26810947275965
MSE for Gradient boosting regression: 227763.35412412955
RMSE for Gradient boosting regression: 477.24559099496093

<matplotlib.collections.PathCollection at 0x2c2d17b1e08>
```



Third algorithm was support vector regression which is basically making regression by using support vector machine algorithm. Parameters were same as support vector classifier in second section of this report. In case of Mean squared error, it indicates improvement compare to first and second algorithm. This time, variation in predictions x=0 have more variance than second algorithm. However, after X=0 variance is less than second algorithm. Like second algorithm, predictions vary more after x=1500.

```
MAE for Support vector regression: 287.1743989822622
MSE for Support vector regression: 153452.56262418668
RMSE for Support vector regression: 391.7302166340844

<matplotlib.collections.PathCollection at 0x249d1b3af08>
```



1) Citi, L. (2021). *Support Vector Machines* [Ebook]. Retrieved from https://moodle.essex.ac.uk/pluginfile.php/1184436/mod_resource/content/2/CE802_Lec_LinearDiscrim_handouts.pdf

2) sklearn.tree.DecisionTreeClassifier — scikit-learn 0.24.0 documentation. (2021). Retrieved 18 January 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

3) sklearn.model_selection.GridSearchCV — scikit-learn 0.24.0 documentation. (2021). Retrieved 18 January 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

4) sklearn.pipeline.Pipeline — scikit-learn 0.24.0 documentation. (2021). Retrieved 18 January 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

5) sklearn.model_selection.train_test_split — scikit-learn 0.24.0 documentation. (2021). Retrieved 18 January 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

6) sklearn.preprocessing.StandardScaler — scikit-learn 0.24.0 documentation. (2021). Retrieved 18 January 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

7) sklearn.impute.SimpleImputer — scikit-learn 0.24.0 documentation. (2021). Retrieved 18 January 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html

8) sklearn.compose.make_column_transformer — scikit-learn 0.24.0 documentation. (2021). Retrieved 18 January 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.compose.make_column_transformer.html

9) sklearn.preprocessing.OneHotEncoder — scikit-learn 0.24.0 documentation. (2021). Retrieved 18 January 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html

10) Starmer, J. (2021). Retrieved 18 January 2021, from https://www.youtube.com/watch?v=3CC4N4z3GJc