Task 3 Report

Julio Sánchez de las Heras Martín Consuegra[1,1*], Javier Santana Delgado[1,1*] and Alejandro Riquelme Castaño[1,1*]

*Corresponding author(s). E-mail(s): julio.sanchez6@alu.uclm.es; javier.santana1@alu.uclm.es; alejandro.riquelme1@alu.uclm.es;

Abstract

This report consist of supervised learning techniques applied to an insurance claims dataset.

1 Introduction

First of all, we have the data that we are going to work with in a csv file. The first step is **preprocessing process and** once the preprocessing is done, we will continue with **KNN, Decision Trees, Random Forests and Boosting algorithms** to create a model that predicts the target value, in this case, **UltimateIncurredClaimCost** .

2 Preprocessing

In this first step, we preprocess the data due to it has null and categorical features such as Gender or MaritalStatus so we have to **remove null values and convert categorical features into numerical features.** The code lines used to do this:

- df.dropna(axis=0, how='any', thresh=None, subset=None, inplace=True)

- df_OneHot = pd.get_dummies(df[['Gender','MaritalStatus', 'PartTimeFullTime']])

3 Baseline

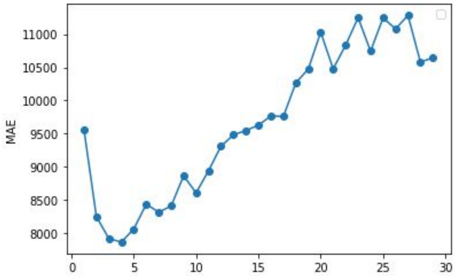
3.1 Feature Selection

To establish the features, we have to **exclude text-based, date-based features and categorical features** due to we have **converted** them **into numerical features** in the preprocessing step.

3.2 Decision Trees

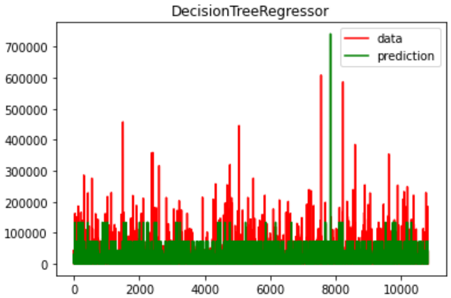
Once feature selection process is already finished, we divide data into two parts, **30% of test** and **70% of train** data. Once we have divided the data, the algorithm is calculated.

We try the algorithm from **one to thirty of maximum depth** and a **cross validation** with **ten splits**.



As we can see in the graphic, the **minimum MAE value is obtained with a maximum depth of four**. Now, we are going to create the definitive decision tree model with that value.

Once we create the model, we use it to predict the target feature. These are the results:



Now, we can obtain the **relevancies of the features** from the decision tree regressor. **The most relevant feature is InitialCurrentCalimsCost followed by DependentChildren, Age and WeeklyWages**.

The table of the **most relevant feautures** looks like this:

Table 1 Feature Relevancies

Attributes	Decision Tree
Age	**0.007101**
DaysWorkedPerWeek	0.000000
DependentChildren	**0.025436**
DependentsOther	0.000000
Gender$_F$	0.000000
Gender$_M$	0.000000
Gender$_U$	0.000000
HoursWorkedPerWeek	0.000000
InitialIncurredCalimsCost	**0.962034**
MaritialStatus$_M$	0.000000
MaritialStatus$_F$	0.000000
MaritialStatus$_U$	0.000000
PartialTimeFullTime$_F$	0.000000
PartialTimeFullTime$_P$	0.000000
WeeklyWages	**0.005429**

Furthermore, we are going to **repeat this process removing InitialCurrentCalimsCost** from the regressor **to see the relevance of the other features** as it is the largest of all and hoards almost the whole relevance. **Now the most relevant features are: WeeklyWages, DependentChildren and Age.** It's important to remark that some features that had no previous data have it now, like **DependentsOther, HoursWorkedPerWeek**.

The table of the most relevant feautures now looks like this:

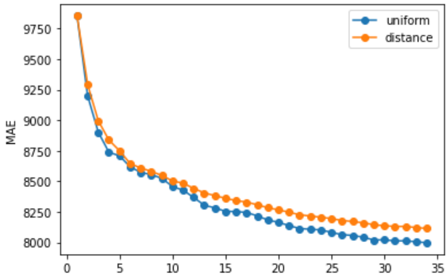
Table 2 Feature Relevancies without InitialIncurredCalimsCost

Attributes	Decision Tree
Age	**0.059532**
DaysWorkedPerWeek	0.000000
DependentChildren	**0.115948**
DependentsOther	**0.042435**
Gender$_F$	0.000000
Gender$_M$	0.000000
Gender$_U$	0.000000
HoursWorkedPerWeek	**0.024373**
MaritialStatus$_M$	0.000000
MaritialStatus$_F$	0.000000
MaritialStatus$_U$	0.000000
PartialTimeFullTime$_F$	0.000000
PartialTimeFullTime$_P$	0.000000
WeeklyWages	0**.757711**

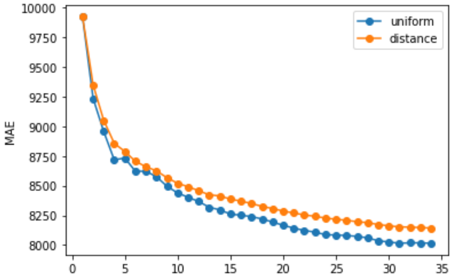
3.3 KNN

In this case, we are going to try different options in order to create the model. We will **test different distance metrics** such as **Manhattan** and **Euclidean** and we will **test different** combinations of **number of neighbors, from 1 to 35**. In addition, the **weights** we will test are **uniform** and **distance**.

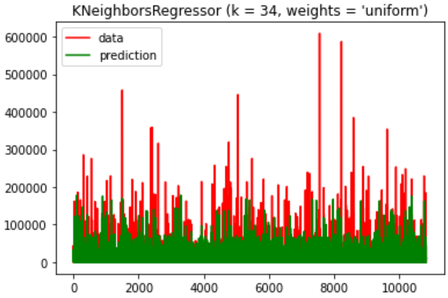
This is the plot for kNN algorithm and **Manhattan** distance metric.



This is the plot for kNN algorithm and **Euclidean** distance metric.

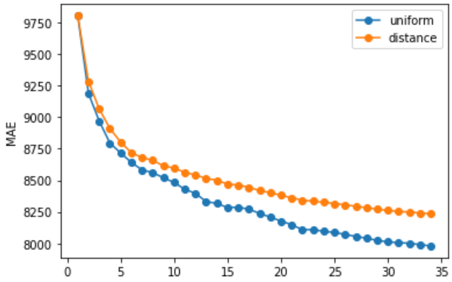


As we can see, the **lowest Mean Absolute Error** is obtained **with 34 neighbours using uniform** weights **with Manhattan** distance metric **(7996.817)**, so we are going to **test the model using these parameters**. The plot of the results obtained is shown below.

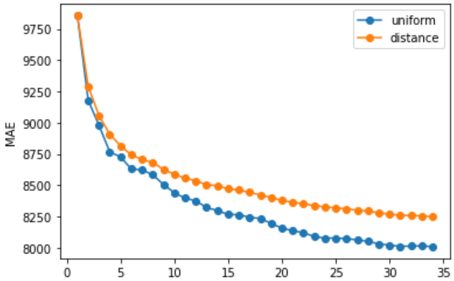


Later, taking into account the most relevant features obtained from the Decision Tree model, we are going to **repeat the previous step with** those **relevant features**, that is to say, **InitialIncurredCalimsCost, Age, DependentChildren and WeeklyWages.**

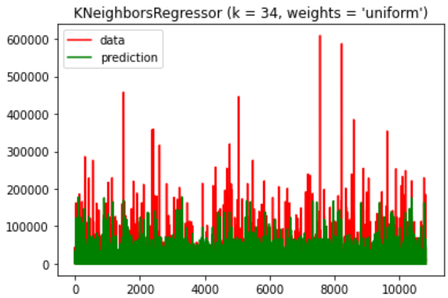
This is the plot for kNN algorithm and **Manhattan** distance metric with the **most relevant features**.



This is the plot for kNN algorithm and **Euclidean** distance metric with the **most relevant features.**



The **best result is obtained with 34 neighbours**, **the 4 most relevant features** and **uniform** weights **with Manhattan** distance metric **(7979.661)**.



4 Advanced Proposal

In this section the first task we are going to perform is a **Random Forest Regressor using a Randomized Search to get the best results**, with these results we will perform a **Grid Search**. Next step, we will apply a **Grid Search to Boosting Regressor** in particular **AdaBoosting** and **GradientBoosting**.

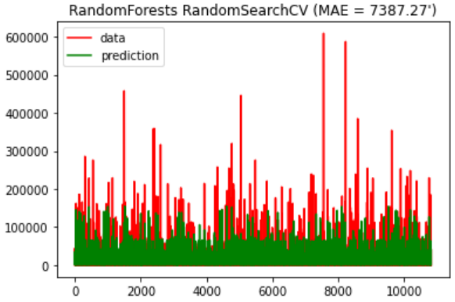
4.1 Random Forest

4.1.1 Randomized Search

To execute this type of search we will use the following **parameters**:

- **n_estimators**: From 8 to 1024 with an exponential growth
- **max_features**: auto and sqrt
- **max_depth**: These parameters are the maximum number of levels in tree (4,8,16,None)
- **min_samples_split**: A random number between 10 and 40 to split a node
- **min_samples_leaf**: Another random number betweeen 1 and 25 to each leaf node
- **bootstrap**: True, False
- **criterion**: mean square error and mean absolute error

The results of this search are the following:



With the function **rnd_regres.best_params_**, we get the **5 best parameters** that we will use in the next section.

```
Model with rank: 1
Mean validation score: 0.206 (std: 0.079)
Parameters: {'bootstrap': True, 'criterion': 'mse', 'max_depth': 4, 'max_features': 'auto', 'min_samples_leaf': 22, 'min_samples_split': 15, 'n_estimators': 512}

Model with rank: 2
Mean validation score: 0.195 (std: 0.084)
Parameters: {'bootstrap': True, 'criterion': 'mae', 'max_depth': 8, 'max_features': 'auto', 'min_samples_leaf': 4, 'min_samples_split': 37, 'n_estimators': 64}

Model with rank: 3
Mean validation score: 0.191 (std: 0.083)
Parameters: {'bootstrap': True, 'criterion': 'mae', 'max_depth': None, 'max_features': 'auto', 'min_samples_leaf': 19, 'min_samples_split': 13, 'n_estimators': 8}

Model with rank: 4
Mean validation score: 0.160 (std: 0.075)
Parameters: {'bootstrap': False, 'criterion': 'mse', 'max_depth': 4, 'max_features': 'sqrt', 'min_samples_leaf': 6, 'min_samples_split': 35, 'n_estimators': 256}

Model with rank: 5
Mean validation score: 0.146 (std: 0.020)
Parameters: {'bootstrap': True, 'criterion': 'mse', 'max_depth': 4, 'max_features': 'auto', 'min_samples_leaf': 2, 'min_samples_split': 16, 'n_estimators': 1024}
```

4.1.2 Grid Search

We decided to take only **3 of the top 5** to do this search and the parameters are the following:

- **n_estimators**: 8, 64 and 512
- **max_features**: auto
- **max_depth**: These parameters are the maximum number of levels in tree (8, 4, None)
- **min_samples_split**: 37
- **min_samples_leaf**: 19
- **bootstrap**: True
- **criterion**: mean square error

The final **5 best parameters** obtained from this search is:

```
Model with rank: 1
Mean validation score: 0.199 (std: 0.072)
Parameters: {'bootstrap': True, 'criterion': 'mse', 'max_depth': 4, 'max_features': 'auto', 'min_samples_leaf': 19, 'min_samples_split': 37, 'n_estimators': 512}

Model with rank: 2
Mean validation score: 0.199 (std: 0.073)
Parameters: {'bootstrap': True, 'criterion': 'mse', 'max_depth': 8, 'max_features': 'auto', 'min_samples_leaf': 19, 'min_samples_split': 37, 'n_estimators': 512}

Model with rank: 3
Mean validation score: 0.198 (std: 0.071)
Parameters: {'bootstrap': True, 'criterion': 'mse', 'max_depth': 4, 'max_features': 'auto', 'min_samples_leaf': 19, 'min_samples_split': 37, 'n_estimators': 64}

Model with rank: 4
Mean validation score: 0.197 (std: 0.071)
Parameters: {'bootstrap': True, 'criterion': 'mse', 'max_depth': 4, 'max_features': 'auto', 'min_samples_leaf': 19, 'min_samples_split': 37, 'n_estimators': 8}

Model with rank: 5
Mean validation score: 0.195 (std: 0.070)
Parameters: {'bootstrap': True, 'criterion': 'mse', 'max_depth': None, 'max_features': 'auto', 'min_samples_leaf': 19, 'min_samples_split': 37, 'n_estimators': 512}
```

4.2 Boosting

In this section, we will perfom a **GridSearch with AdaBoosting and** another with **GradientBoosting**.

4.2.1 AdaBoosting

The **parameters** of AdaBoostingRegressor are:

- **n_estimators**: 8, 32, 64 and 128
- **learning_rate**: 0.05, 0.1 and 0.25
- **loss**: linear, square and exponential

The **Mean Absolute Error** obtained is **6186.606**

4.2.2 GradientBoosting

The **parameters** of GradientBoostingRegressor are:

- **n_estimators**: 32, 64 and 128
- **learning_rate**: 0.01,0.05,0.1,0.25 and 0.5
- **max_features**: auto and sqrt
- **loss**: ls, lad, huber and quantile
- **criterion**: Mean square error and friedman_mse

The **Mean Absolute error** with this Boosting is: 10785.869

5 Improvements

In order **to improve the models obtained**, we are going to take into account the **"ClaimDescription" feature**, which **is a text-based feature**.

To join the numerical features with this new feature, a **preprocessing step is needed**.

5.1 Preprocessing

First of all, we **convert all capital letters** present in the text **into lower letters**. After that, we **remove possible repeated words** and **stopwords** which will not be useful for the model. Then, we **lemmatize all terms** in order to reduce the amount of text and o**btain the most meaningful part of each word**. Eventually, we **correct the wrong words**.

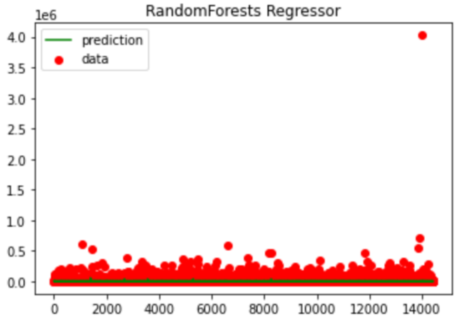
At this point, we have the data processed and we save it into a **pickle file**.

5.2 Vectoricer

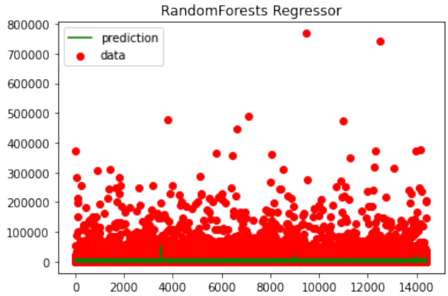
Once we have preprocessed the data, we are going to **vectorice it**. In order to do that, we are going to use the **TF-IDF vectoricer**.

With the vectoriced data, we are going to **divide it accompanied by the target feature, "UltimateIncurredClaimCost"**, **70% for the train** part and **30% for test**.

Later, **we supply the data to** a **Random Forest Regressor** with **4 estimators**, a **max depth of 2** and a **MAE criterion**. The results are shown below.



After we calculate the Random Forest Regressor, we are going to **perform a TFIDF vectorizer with n-grams for** the **ClaimDescription feature** but taking the **SelectKBest output**. The plot resulting:



If we compare the two graphs, we can see that the data is a **little more spread out in the last plot**.

6 Conclusion

As we observed in this task, the most relevant feature that affects the target **"UltimateIncurredClaimCost"** is the **"InitialCurrentCalimsCost"**, but also if we take into account some **text-based features "ClaimDescription"** it can also **give us a lot of information about the target**.

More technically related, we have noticed that the **KNN** and the **Decision Tree** can be useful for **regression** or for **predicting the target**. However, if the **Hyperparameter Optimisation** is used **with the Optimised model**, the results can be more **precise**. In addition, the **Support Vector Machine Model** applied to the **"ClaimDescription"**, gives better results, however, as we have demonstrated, a preprocessing step is needed because it is a text-based model.

To conclude, we are a group that belongs to another intensification but by working so many hours on this task, we have finally found it interesting and we have a very good current thinking about people who process large amounts of data.