Supervised Machine Learning: Regression

IBM Skills Network

Project Report

By:

Fahd Seddik

Table of Contents

Main Objective

In a world of ever-growing population, the demand for resources is always very high. In recent years, the spread of disease has had a major impact on countless countries and on the global economy. The number of people that are in need of medical attention is at an all-time high. Due to both reasons mentioned, all medical supplies' prices sky-rocketed. Medical insurance companies are working as hard as they can to provide people with what they need. In order for a medical insurance company to sustain, such a company needs an estimate of the total budget a person would require. This would act as crucial information to estimate the cash flow needed for the whole company.

This project aims to try to estimate (prediction) the medical charges a person would require given some details about each person.

Brief Description

The data set presented consists of **6 features**.

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

The data set columns are **age**, **sex** (female/male), **bmi**, **children**, **smoker** (yes/no), **region**, and medical **charges**. This was collected for 1338 people (examples) and we were given the total charges spent on each of them. Furthermore, the column region is divided into 4 categories, **southwest**, **southeast**, **northwest**, and **northeast**. The first 6 columns in the data set represent our 6 features and the last column is our "target". In the next sections, we shall explore more information about the underlying patterns in this data set.

Data cleaning & Feature engineering

Upon reviewing the data.info() function we can see that there are no missing values in our dataset. Afterwards, as you can see below, a check for duplicates has been done and we have dropped the duplicate examples.

```
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1338 non-null   int64
 1   sex       1338 non-null   object
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64
 4   smoker    1338 non-null   object
 5   region    1338 non-null   object
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

```
data.charges.is_unique

False
```

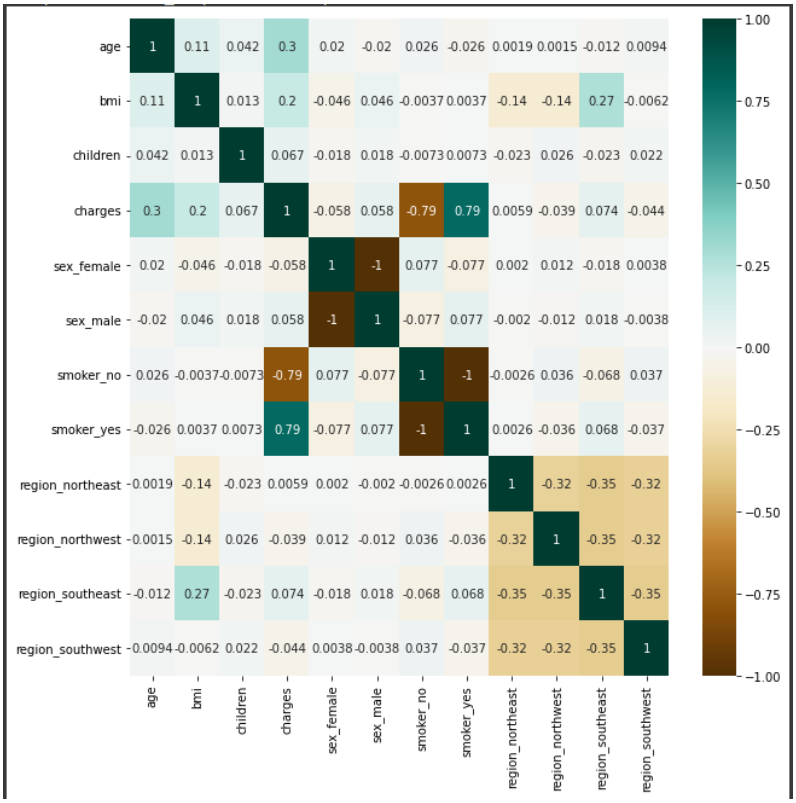
```
data.drop_duplicates(inplace=True)
```

```
data.charges.is_unique

True
```

When it comes to exploring the more correlated features in our data, we can start by plotting a heatmap to try and visualize which features would be of more significance when it comes to predicting our target.

In the figure, we can see that a person being a smoker or not heavily indicates whether they would need a high budget or not. This is indicated by the heatmap values and the legend provided. The correlation values are 0.79 and -0.79 for being a smoker and not being a smoker respectively which seems intuitive.

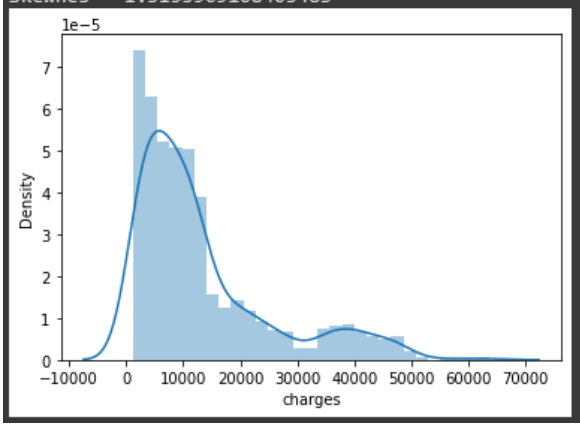


In order to better enhance the performance of our regression models, we need to make sure our target column is not heavily skewed. We will check the skewness of the target column 'charges' and correct the skewness if present as shown below.

BEFORE

```
plot = sns.distplot(data['charges']);
print("Skewnes = {}".format(data['charges'].skew()))
```

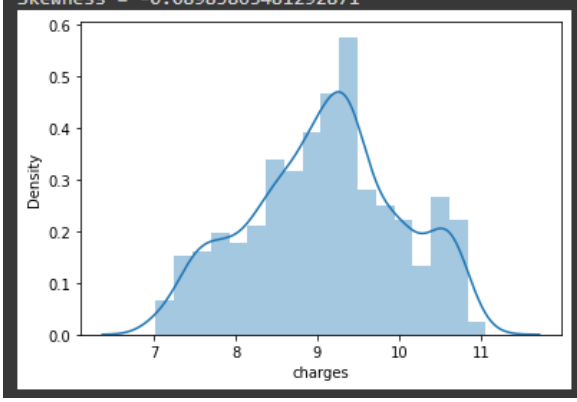
```
Skewnes = 1.5153909108403483
```



AFTER

```
log_transformed = np.log(data['charges'])
sns.distplot(log_transformed);
print(f"Skewness = {log_transformed.skew()}")
```

```
Skewness = -0.08983863481292871
```



We need to also encode our categorical data by using get_dummies() since we have only **3 categorical features** so it will not cause performance issues, thus not needing to use OneHotEncoder(). We will also drop first columns to insure we avoid the dummy variable trap. After that, we will divide our data set into our features 'X' and target 'y'. Calculating **polynomial features** will be done in the next section since we will be using a **pipeline**.

```
data = pd.get_dummies(data,drop_first=True)
```

```
data.head()
```

	age	bmi	children	charges	sex_male	smoker_yes	region_northwest	region_southeast	region_southwest
0	19	27.900	0	9.734176	0	1	0	0	1
1	18	33.770	1	7.453302	1	0	0	1	0
2	28	33.000	3	8.400538	1	0	0	1	0
3	33	22.705	0	9.998092	1	0	1	0	0
4	32	28.880	0	8.260197	1	0	1	0	0

```
d2 = data.copy()
y_col = 'charges'
X = d2.drop(y_col,axis=1)
y = d2[y_col].copy()
print('X shape:',X.shape)
print('Y shape:',y.shape)
```

```
X shape: (1337, 8)
Y shape: (1337,)
```

Linear Regression Models

We will use 3 regression models and determine which is best for the data set provided. Our approach would be constructing a cross-validation grid-search that would find the best hyperparameters for each of the models then we will compute the R^2 score for each of them as well as the RMSE and compare between each of the models at the end.

```
[33] estimator_lr = Pipeline([("polynomial_features",PolynomialFeatures()),
                              ("scaler",StandardScaler()),
                              ("linear_regression",LinearRegression())])
     estimator_las = Pipeline([("polynomial_features",PolynomialFeatures()),
                              ("scaler",StandardScaler()),
                              ("las_regression",Lasso())])
     estimator_r = Pipeline([("polynomial_features",PolynomialFeatures()),
                              ("scaler",StandardScaler()),
                              ("ridge_regression",Ridge())])

[35] from sklearn.model_selection import train_test_split

[36] from sklearn.model_selection import KFold

[38] kf = KFold(shuffle=True,random_state=42,n_splits=3)
     X_train, X_test, y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=40)

     params_r = {'polynomial_features__degree':[1,2,3],
                 'ridge_regression__alpha':np.geomspace(0.1,20,30)}
     params_las = {'polynomial_features__degree':[1,2,3],
                 'las_regression__alpha':np.geomspace(0.1,20,30)}
     params_lr = {'polynomial_features__degree':[1,2,3]}
```

The models we will be using are **LinearRegression(), Lasso(),** and **Ridge()**. We constructed our pipeline estimators for each of them and loaded with them a standard scaler and polynomial features. The parameters we intend to use for the grid-search will change the degree of the polynomial features and also change the alpha values for both the Lasso and Ridge models only.We also defined our cross-validation object and instantiated a K-fold cross-validation object with shuffle=True and K=3. This would try to decrease over-fitting our models to the data set.

```
grid_r = GridSearchCV(estimator_r,params_r,cv=kf)
grid_r.fit(X_train,y_train)
grid_lr = GridSearchCV(estimator_lr,params_lr,cv=kf)
grid_lr.fit(X_train,y_train)
grid_las = GridSearchCV(estimator_las,params_las,cv=kf)
grid_las.fit(X_train,y_train)
```

Recommended Model

To determine which of our models is best, we will try to calculate a few metrics about each model to see if they managed to successfully beat the other models. We will be using r2_score() and mean_square_error() from the sklearn.metrics library. We first construct a function that would help us compute the RMSE instead of the MSE shown below.

```
def rmse(y_pred,y_test):
  return np.sqrt(mean_squared_error(y_pred,y_test))
```

This would act as a simple function that would be used in the next steps. We calculate each of the models' y_predict and we will calculate both the R^2 score and the RMSE of three the models. After that, we would decide which of the models is the best based on those results.

As we can see, Lasso regression under-performed compared to the other models. On contrast, both linear and ridge regression have relatively high R^2 scores. Based on this results the best choice would be simple **Linear Regression.**

```
-=Linear Regression=-
R2 score:  0.8765291103882433
RMSE:  0.30480205657676673


-=Lasso Regression=-
R2 score:  0.47786362544602157
RMSE:  0.4683601941252894


-=Ridge Regression=-
R2 score:  0.8741727302651208
RMSE:  0.30600665322500986
```

Key Findings and Insights

In this section we would dive deeper into the construction of each of the models. We would look into the values and insights about the weights of each of the three models used. First let's look into the weights given by each model for each feature.

Simple Linear Regression

	0	1
4	smoker_yes	-7.564983e-02
6	region_southeast	-1.852936e-02
7	region_southwest	-7.499821e-03
0	age	-1.204636e-17
2	children	2.432955e-01
5	region_northwest	2.493527e-01
3	sex_male	4.220240e-01
1	bmi	7.445560e-01

Ridge Regression

	0	1
4	smoker_yes	-0.075090
6	region_southeast	-0.021738
7	region_southwest	-0.009662
0	age	0.000000
2	children	0.206422
5	region_northwest	0.250839
3	sex_male	0.399773
1	bmi	0.678911

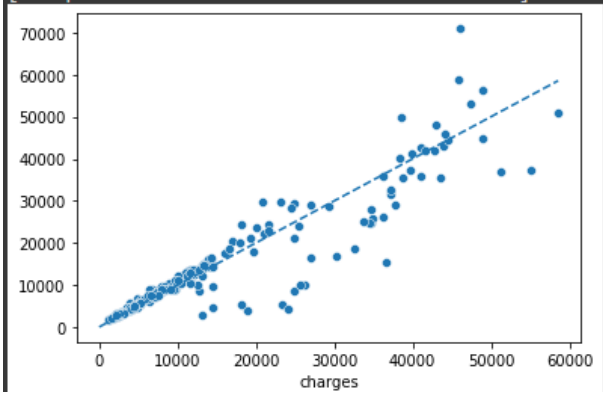
Lasso Regression

	0	1
0	age	0.000000
2	children	0.000000
4	smoker_yes	-0.000000
6	region_southeast	0.000000
7	region_southwest	-0.000000
5	region_northwest	0.024130
3	sex_male	0.025571
1	bmi	0.384213

As we can see, lasso has 5 zero weights which is expected from a lasso regression model. On contrast, ridge regression has only 1 zero weight as expected which is age. Surprisingly, as we can see our simple linear regression model did not over-fit and does not have large weight values this is most probably due to using cross-validation. The next figure would show our simple linear regression model estimates of the test data.

```
sns.scatterplot(np.exp(y_test),np.exp(grid_lr.predict(X_test)));
l = np.linspace(0,np.exp(y_test).max(),500)
plt.plot(l,l,ls='--')
```

```
[<matplotlib.lines.Line2D at 0x7f875600b890>]
```



Suggestions

The data set we used allows us to use a plethora of models to try and predict the best model that would not over-fit and a good bias-variance tradeoff. Some suggestions would include trying **ElasticNet**s as a model with cross-validation. We can also try to revisit our models we constructed to have a better interpretability of the data set. On top of that, we can try to use grid-search with more parameter ranges and values for all the models to try to see if it better suits our data.