Insurance Premium Price Prediction

Web App using Flask
Zeru Zhou
2022.12.28
Data Glacier

Step 1: Loading Dataset and create several ML models with validation

Linear Regression:

Linear Regression

In [32]:
```
LR = sm.OLS(train_y, train_x).fit()
y_pred = LR.predict(test_x)
mape = mean_absolute_percentage_error(y_pred, test_y)
mae = mean_absolute_error(y_pred, test_y)
mape, mae
```

Out[32]: (0.24611767165221682, 4503.3562487906365)

SVM:

SVM

In [34]:
```
svr = SVR()
linearsvr = LinearSVR()

params = {'C': np.linspace(0.01, 3, 50)}
clf = GridSearchCV(svr, params).fit(train_x, train_y)
y_pred = clf.predict(test_x)
mape = mean_absolute_percentage_error(y_pred, test_y)
mae = mean_absolute_error(y_pred, test_y)
mape, mae
```

Out[34]: (0.22497192812045264, 5212.911422280605)

In [35]:
```
clf = GridSearchCV(linearsvr, params).fit(train_x, train_y)
y_pred = clf.predict(test_x)
mape = mean_absolute_percentage_error(y_pred, test_y)
mae = mean_absolute_error(y_pred, test_y)
mape, mae
```

Out[35]: (3.6243763603144177, 19623.383451753598)

Random Forest:

Tree Based Methods

In [36]:
```
rdf = RandomForestRegressor(criterion='absolute_error', max_features='sqrt')
params = {'ccp_alpha': np.logspace(-4, 4, 50)}

clf = GridSearchCV(rdf, params).fit(train_x, train_y)
y_pred = clf.predict(test_x)
mape = mean_absolute_percentage_error(y_pred, test_y)
mae = mean_absolute_error(y_pred, test_y)
mape, mae
```

Out[36]:
```
(0.07345996455865725, 1820.2702702702702)
```

Ridge and Lasso:

Ridge and Lasso

In [37]:
```
ridge = Ridge()
lasso = Lasso()
params = {'alpha': np.logspace(-4, 4, 50)}

clf = GridSearchCV(ridge, params).fit(train_x, train_y)
y_pred = clf.predict(test_x)
mape = mean_absolute_percentage_error(y_pred, test_y)
mae = mean_absolute_error(y_pred, test_y)
mape, mae
```

Out[37]:
```
(0.11740223809367199, 2815.0486289245514)
```

In [38]:
```
clf = GridSearchCV(lasso, params).fit(train_x, train_y)
y_pred = clf.predict(test_x)
mape = mean_absolute_percentage_error(y_pred, test_y)
mae = mean_absolute_error(y_pred, test_y)
mape, mae
```

Out[38]:
```
(0.11713108686280707, 2807.5822474897896)
```

As above, I'll choose Random Forest Regressor as my prediction method.

Step 2: Save the model using Pickle

In [39]:
```
x = MinMaxScaler().fit_transform(x)
```

In [40]:
```
rdf = RandomForestRegressor(criterion='absolute_error', max_features='sqrt')
params = {'ccp_alpha': np.logspace(-4, 4, 100)}
clf = GridSearchCV(rdf, params).fit(x, y)
```

In [42]:
```
pickle.dump(clf, open('model.pkl', 'wb'))
```

Step 3: Create a html template for app implementation

First, I used header and style, logo that defined by Data Glacier:

```
<!DOCTYPE html>
<html >
<head>
  <meta charset="UTF-8">
  <title>ML API Insurance Prediction</title>
  <link href='https://fonts.googleapis.com/css?family=Pacifico' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Arimo' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Hind:300' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300' rel='stylesheet' type='text/css'>
<link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">

</head>
```

Then, I created the body part, to support the input that is required by the model:

```
<body>

<div class="login" style = "position:fixed; left:700px; top:200px;">

 <h1>Predict Insurance Premium Price</h1>


  <form action="{{ url_for('predict')}}"method="post">
  <input type="text" name="Age" placeholder="Age of the user, integer" required="required" />
      <input type="text" name="Diabetes" placeholder="If the user has diabete history, input 1 if has and 0 otherwise" required="required" />
  <input type="text" name="Blood pressure problems" placeholder="If the user has blood pressure problems history, input 1 if has and 0 otherwise" required="required" />
  <input type="text" name="Any transplants" placeholder="If the user has any transplants history, input 1 if has and 0 otherwise" required="required" />
  <input type="text" name="Any chronic diseases" placeholder="If the user has any chronic diseases history, input 1 if has and 0 otherwise" required="required" />
  <input type="text" name="Height" placeholder="Height of the user in cm" required="required" />
  <input type="text" name="Weight" placeholder="Weight of the user in kg" required="required" />
  <input type="text" name="Known allergies" placeholder="If the user has any known allergies history, input 1 if has and 0 otherwise" required="required" />
  <input type="text" name="History of cancer in family " placeholder="If the user has any history of cancer in family, input 1 if has and 0 otherwise" required="required" />
  <input type="text" name="Number of major surgeries " placeholder="Number of major surgeries the user had" required="required" />

      <button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>
  </form>

  <br>
  <br>
  {{ prediction_text }}


</div>
<img src="/static/images/Original.svg" style="width: 400px;position: absolute;bottom: 10px;left: 10px;" alt="Company Logo"/>

</body>
</html>
```

Once I run the app, after hitting the button, it will be directed to '/predict' page of my app, with the prediction result provided by the model.

Step 4: Create Flask app for both the home page and prediction page, load the ML model.

First, Load the model:

```
from flask import Flask, request, render_template
import pickle
from sklearn.preprocessing import MinMaxScaler
import numpy as np


model = pickle.load(open('model.pkl', 'rb'))
app = Flask(__name__)
```

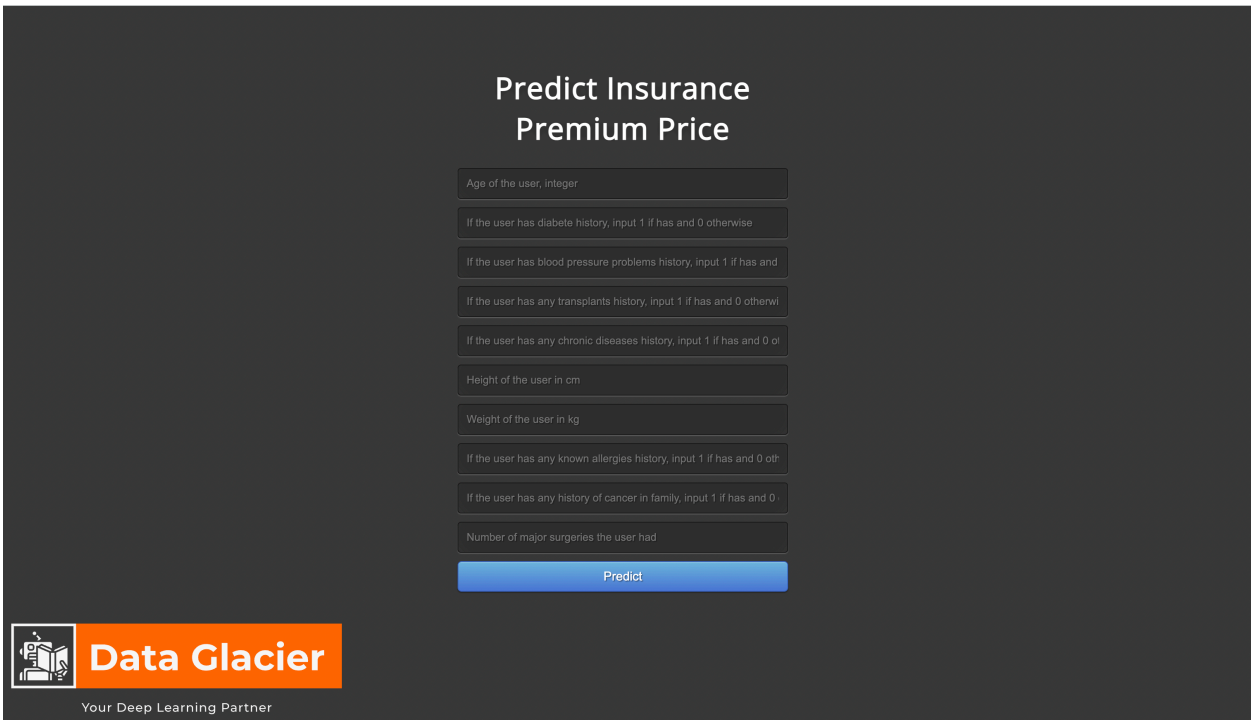
Then, design the routes and run the app, also taking the inputs as feature, scale the feature and predict using the pre-trained model:

```python
@app.route('/')
def home():
    return render_template('index.html')


@app.route('/predict', methods=['POST'])
def predict():
    feature = [int(x) for x in request.form.values()]
    feature = [feature]
    feature = MinMaxScaler().fit_transform(feature)
    result = model.predict(feature)
    return render_template('index.html', prediction_text=f'The predicted value of insurance price is ${int(result[0])}')


if __name__ == '__main__':
    app.run(port=5000, debug=True)
```

Step 5: Run the Flask App on Web

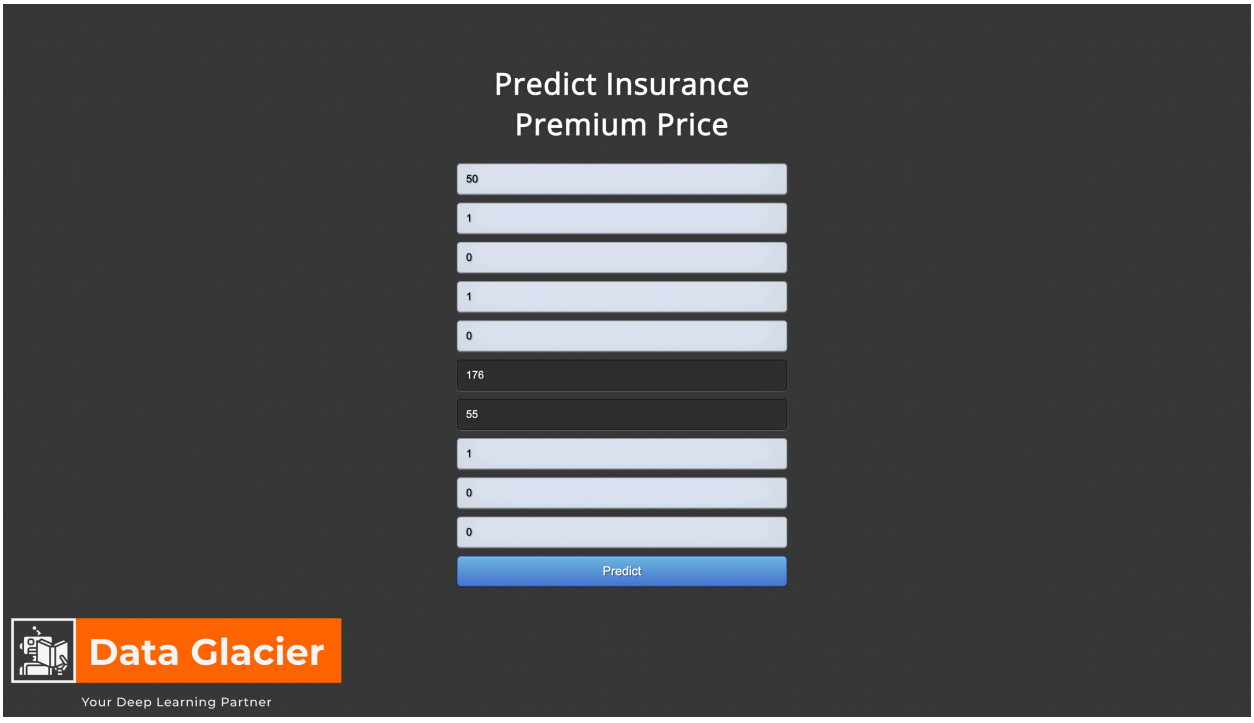
```
(base) zhouzeru@Zerus-MacBook-Pro Flask ML Model Deployment % python3 Flask\ Implementation.py
 * Serving Flask app "Flask Implementation" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with watchdog (fsevents)
 * Debugger is active!
 * Debugger PIN: 837-856-694
```

Click the link, we will be directed to the Web App:



Fill in the blanks. **Notice that there are some blanks that needed to fill in 0 or 1 since they are categorical.**

For Example:



Then, click the predict button, we'll get the result from our model: