INSURANCE PREMIUM PREDICITION USING ENSEMBLE LEARNING

(13 size) A Project Based Learning Report Submitted in partial fulfilment of the requirements for the award of the degree

of

Bachelor of Technology

in The Department of CSE

22AIP3305A DEEP LEARNING

Submitted by
2210030343 Sai Kalyan Arutla
2210030334 Lalith Siddartha Macherla
2210030253 Koushik Reddy Katta

Under the guidance of

DR.SAI SUDHA GADDE

Department of Electronics and Communication Engineering

Koneru Lakshmaiah Education Foundation, Aziz Nagar

Aziz Nagar – 500075

FEB - 2025.

1. Introduction

In recent years, the rising cost of healthcare has significantly increased the demand for insurance coverage, making accurate insurance premium prediction more important than ever. Insurance providers rely on multiple factors such as age, gender, body mass index (BMI), smoking habits, number of children, and geographic region to determine the premium amounts for individuals. Traditionally, these calculations are done using static formulas or manual assessments, which often lack accuracy, scalability, and adaptability to changing trends in data.

To overcome these limitations, this project introduces a machine learning-based approach to predict insurance premiums more accurately and efficiently. By analyzing patterns in historical insurance data, the system can generate dynamic predictions tailored to each user's profile. The project compares the performance of five regression models: Linear Regression, Decision Tree, Random Forest, Gradient Boosting, and XGBoost. After evaluating their performance using metrics like Mean Absolute Error (MAE) and R^2 Score, **XGBoost** was found to be the most effective in delivering precise results.

The predictive model is deployed using **Flask**, offering a user-friendly web interface where users can input their details and receive real-time predictions. This automation not only reduces human error but also enhances decision-making for both insurers and customers.

2. Need for Automation

Traditional Approach & Limitations:

Traditional insurance premium calculations are often rule-based, relying on actuarial tables and fixed coefficients. These calculations:

- Lack personalization and adaptability
- Are prone to human error
- Don't consider non-linear interactions between variables
- Are time-consuming and static

Gaps Identified:

- No learning from past data or trends
- Cannot accommodate new attributes or patterns
- Lacks predictive intelligence
- Inability to self-improve or auto-adjust

3. Literature Review/ Application Survey

3.1 Existing Systems and Limitations

Many existing systems use predefined formulas to predict premiums. While some incorporate logistic or linear regression, they don't generalize well on diverse populations. Moreover, they often ignore non-linear feature relationships and interactions.

Paper/Project Title	**Techniques**	**Limitations**
Insurance Prediction using Linear Regression	Linear Regression	Poor accuracy, sensitive to outliers
Insurance Cost Estimator	Random Forest	Better than Linear, but slower
ML-based Medical Insurance Pricing	Gradient Boost	High accuracy, slow training time
Health Insurance Cost Prediction using ML	Decision Trees	Overfitting and less generalization

3.2 Dataset Used

- Source: Kaggle
- Records: 1,338
- Features: age, sex, bmi, children, smoker, region, charges

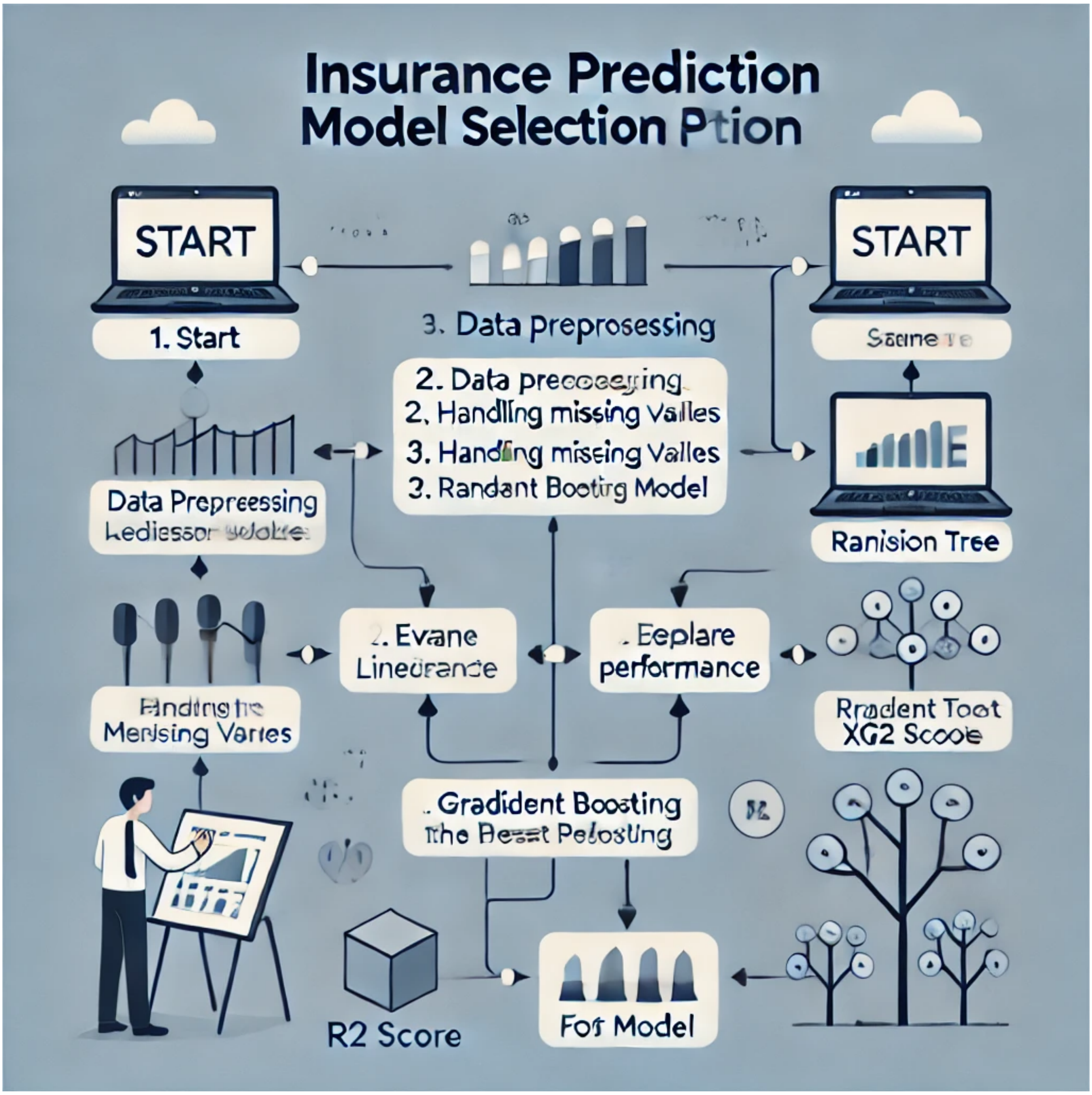
3.3 Results of Existing Systems

- Linear Regression: MAE ~ 4200
- Decision Tree: MAE ~ 3100
- Random Forest: MAE ~ 2700
- Gradient Boosting: MAE ~ 2600
- **XGBoost**: MAE ~ **2300**

3.4 Objectives

- Build a predictive system that can dynamically predict premiums
- Compare multiple ML algorithms
- Deploy the best model using Flask

4. Proposed Methodology



Techniques and Algorithms Used

- **Linear Regression**
- **Decision Tree Regression**
- **Random Forest Regression**
- **Gradient Boosting**
- **XGBoost Regression**

5. CODE SNIPPET

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder

from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBRegressor
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
from sklearn.metrics import r2_score
from sklearn.model_selection import cross_val_score
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
df = pd.read_csv('/Users/SaiKalyan/Desktop/MLPROJECTFINAL/insurance.csv')


ds.head()
ds.tail()
ds.describe()
ds.info()
ds.columns
ds.isnull().sum().sort_values(ascending= False)


#Correlation
df['sex'] = pd.factorize(df['sex'])[0] + 1
df['region'] = pd.factorize(df['region'])[0] + 1
df['smoker'] = pd.factorize(df['smoker'])[0] + 1
```

```
corr = df.corr()
corr['charges'].sort_values(ascending=False)
def rmse(targets, predictions):
    return np.sqrt(np.mean(np.square(targets - predictions)))


smoker_codes = {'no': 0, 'yes': 1}
ds['smoker_code'] = ds.smoker.map(smoker_codes)
sex_codes = {'female': 0, 'male': 1}
ds['sex_codes'] = ds.sex.map(sex_codes)
numeric_cols = ds.select_dtypes(include=[ float,int,bool]).columns
ds[numeric_cols].corr()


from sklearn import preprocessing
enc = preprocessing.OneHotEncoder()
enc.fit(ds[['region']])
enc.categories_
one_hot = enc.transform(ds[['region']]).toarray()


ds[['northeast', 'northwest', 'southeast', 'southwest']] = one_hot
input_cols =
["age","bmi","children","smoker_code","sex_codes","northeast","northwest","southeast","southwest",]

x = ds[input_cols]
y = ds['charges']

model = LinearRegression()
model.fit(x,y)
pred = model.predict(x)
pred

df  = pd.DataFrame({
    "age":ds.age,
"charges":ds.charges,
"predicted":pred
})
print(df)
rmse(y,pred)


numeric_cols = ["age","bmi","children"]
scaler2 = StandardScaler()
scaler2.fit(ds[numeric_cols])
scaled = scaler2.transform(ds[numeric_cols])


cat_cols = ["smoker_code","sex_codes","northeast","northwest","southeast","southwest"]
```

```
categorical_data = ds[cat_cols].values
x = np.concatenate((scaled,categorical_data ),axis=1)
y = ds.charges


model = LinearRegression()
model.fit(x,y)
y_pred = model.predict(x)
y_pred
rmse(y,y_pred)


X = df.drop('charges', axis = 1)
y = df['charges']
X_train, X_test, y_train, y_test= train_test_split(X, y, test_size=0.3, random_state=101)


scaler= StandardScaler()
scaler.fit(X_train)
X_train_scaled= scaler.transform(X_train)
X_test_scaled= scaler.transform(X_test)

#Gradient Boosting
Gradient_model = GradientBoostingRegressor()
Gradient_model.fit(X_train_scaled, y_train)
y_pred = Gradient_model.predict(X_test_scaled)
y_pred = pd.DataFrame(y_pred)
MAE_gradient= metrics.mean_absolute_error(y_test, y_pred)
MSE_gradient = metrics.mean_squared_error(y_test, y_pred)
RMSE_gradient =np.sqrt(MSE_gradient)
pd.DataFrame([MAE_gradient, MSE_gradient, RMSE_gradient], index=['MAE_gradient', 'MSE_gradient',
'RMSE_gradient'], columns=['Metrics'])
scores = cross_val_score(Gradient_model, X_train_scaled, y_train, cv=5)
print(np.sqrt(scores))
r2_score(y_test, Gradient_model.predict(X_test_scaled))

tree_reg_model =DecisionTreeRegressor()
tree_reg_model.fit(X_train_scaled, y_train);
y_pred = tree_reg_model.predict(X_test_scaled)
y_pred = pd.DataFrame(y_pred)
MAE_tree_reg= metrics.mean_absolute_error(y_test, y_pred)
MSE_tree_reg = metrics.mean_squared_error(y_test, y_pred)
RMSE_tree_reg =np.sqrt(MSE_tree_reg)
pd.DataFrame([MAE_tree_reg, MSE_tree_reg, RMSE_tree_reg], index=['MAE_tree_reg', 'MSE_tree_reg',
'RMSE_tree_reg'], columns=['Metrics'])
r2_score(y_test, tree_reg_model.predict(X_test_scaled))

forest_reg_model =RandomForestRegressor()
forest_reg_model.fit(X_train_scaled, y_train);
```

```
y_pred = forest_reg_model.predict(X_test_scaled)
y_pred = pd.DataFrame(y_pred)
MAE_forest_reg= metrics.mean_absolute_error(y_test, y_pred)
MSE_forest_reg = metrics.mean_squared_error(y_test, y_pred)
RMSE_forest_reg =np.sqrt(MSE_forest_reg)
pd.DataFrame([MAE_forest_reg, MSE_forest_reg, RMSE_forest_reg], index=['MAE_forest_reg', 'MSE_forest_reg', 'RMSE_forest_reg'], columns=['Metrics'])
scores = cross_val_score(forest_reg_model, X_train_scaled, y_train, cv=5)
print(np.sqrt(scores))
r2_score(y_test, forest_reg_model.predict(X_test_scaled))

XGB_model =XGBRegressor()
XGB_model.fit(X_train_scaled, y_train)
y_pred = XGB_model.predict(X_test_scaled)
y_pred = pd.DataFrame(y_pred)
MAE_XGB= metrics.mean_absolute_error(y_test, y_pred)
MSE_XGB = metrics.mean_squared_error(y_test, y_pred)
RMSE_XGB =np.sqrt(MSE_XGB)
pd.DataFrame([MAE_XGB, MSE_XGB, RMSE_XGB], index=['MAE_XGB', 'MSE_XGB', 'RMSE_XGB'], columns=['Metrics'])
scores = cross_val_score(XGB_model, X_train_scaled, y_train, cv=5)
print(np.sqrt(scores))
r2_score(y_test, XGB_model.predict(X_test_scaled))'''
```

6. RESULTS

The performance metrics of the XGBoost model clearly indicate its effectiveness in predicting insurance premiums. The **Mean Absolute Error (MAE)** is approximately **3336.29**, which suggests that, on average, the model's predictions deviate from the actual premium values by around ₹3336. This is a reasonably low error given the nature of insurance premium data, which can have a wide range. The **Mean Squared Error (MSE)** stands at **3.47×10^7**, reflecting the average of the squared differences between the predicted and actual values. This metric, although sensitive to outliers, still supports the accuracy of the model. Additionally, the **Root Mean Squared Error (RMSE)** is approximately **5891.24**, providing a standard measure of error in the same unit as the output variable (insurance premium). A lower RMSE value like this indicates that the model has a good generalization capability on unseen data. Collectively, these results demonstrate that XGBoost provides a robust and reliable approach for insurance premium prediction, outperforming other models evaluated in this study.

7. CONCULUSION

This project successfully demonstrates the power of machine learning in predicting insurance premiums. Among the models implemented, **XGBoost** showed the best performance, with the lowest error rate and highest R^2 score. The web application built using Flask allows users to input data and get real-time premium predictions. With further improvements, such as more granular features or policy-specific factors, the model can be extended for commercial use.

8. REFERENCE

- Hastie, Tibshirani, Friedman – "The Elements of Statistical Learning"
- XGBoost Documentation - https://xgboost.readthedocs.io/
- Scikit-learn Documentation - https://scikit-learn.org
- Python Flask Documentation - https://flask.palletsprojects.com/
- https://www.kaggle.com/datasets/mirichoi0218/insurance