Feature Engineering for Insurance Premium Prediction

This lecture details the feature engineering process for a machine learning model predicting insurance premiums. We'll transform raw data into suitable numerical representations for the model, addressing categorical and numerical features, handling missing values, and performing feature selection.

1. Handling Complex Categorical Features

The `medical_history` column contains a list of diseases for each individual. Since machine learning models require numerical input, we convert this list into a numerical "health risk score." Each disease is assigned a specific risk score based on domain expertise (e.g., diabetes = 6, heart disease = 8, no disease = 0). The total risk score for an individual is the sum of the risk scores of all their diseases.

```python
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

# Sample medical_history data
medical_history = ["diabetes, high blood pressure", "heart disease", "none", "diabetes"]

# Dictionary mapping diseases to risk scores
scores = {
    "diabetes": 6,
    "high blood pressure": 2,
    "heart disease": 8,
    "none": 0
}

# Calculating risk scores
risk_scores = []
for history in medical_history:
    diseases = [d.strip().lower() for d in history.split(',')]
    score = sum(scores.get(disease, 0) for disease in diseases)  # Handles unknown diseases
    risk_scores.append(score)

print(risk_scores)  # Output: [8, 8, 0, 6]
```

Explanation:

1. **Splitting Diseases:** The `medical_history` string is split into individual diseases using a comma delimiter and converted to lowercase to ensure consistency.
2. **Mapping Scores:** Each disease is mapped to its corresponding risk score using the `scores` dictionary. If a disease isn't found in the dictionary, a default score of 0 is assigned.
3. **Summing Scores:** The scores for all diseases an individual has are summed to obtain the total risk score.

Normalization:
After calculating the total risk score, we normalize it to a 0-1 range using Min-Max scaling. This ensures that features with different scales do not disproportionately influence the model.

```
# Example normalization
scaler = MinMaxScaler()
normalized_risk_scores = scaler.fit_transform(pd.DataFrame(risk_scores))

print(normalized_risk_scores)  # Outputs normalized scores between 0 and 1
```

2. Encoding Ordinal Categorical Features

Features like `insurance_plan` (bronze, silver, gold) represent ordinal categories. We use label encoding to assign numerical values while preserving the order (e.g., bronze = 1, silver = 2, gold = 3). This approach captures the inherent ranking within the categories.

```
# Sample insurance_plan data
insurance_plans = ["bronze", "silver", "gold", "bronze"]

# Mapping ordinal categories
plan_mapping = {"bronze": 1, "silver": 2, "gold": 3}
encoded_plans = [plan_mapping[plan] for plan in insurance_plans]

print(encoded_plans)  # Output: [1, 2, 3, 1]
```

3. Encoding Nominal Categorical Features

Nominal categorical features like `gender` and `region` have no inherent order. We use one-hot encoding to create dummy variables for each category. This avoids introducing artificial ordinal relationships between the categories. The `drop_first` parameter in `pd.get_dummies` is used to prevent multicollinearity by dropping one dummy variable from each category.

```
# Sample nominal data
data = {
    'gender': ['male', 'female', 'female', 'male'],
    'region': ['southwest', 'southeast', 'northwest', 'southeast']
}
df = pd.DataFrame(data)

# One-hot encoding
df_encoded = pd.get_dummies(df, columns=['gender', 'region'], drop_first=True)

print(df_encoded)
```

Output:

```
   gender_male  region_northwest  region_southeast  region_southwest
0            1                 0                 0                 1
1            0                 0                 1                 0
2            0                 1                 0                 0
3            1                 0                 1                 0
```

4. Dropping Redundant Features

After calculating the `normalized_risk_score`, the original `medical_history` column and the intermediate columns used for calculating the score become redundant and are dropped to streamline the dataset.

```
# Dropping redundant columns
df_final = df_encoded.drop(['medical_history', 'intermediate_column_1', 'intermediate_column_2'], axis=1)
```

Ensure you replace `'intermediate_column_1'` and `'intermediate_column_2'` with actual column names used in your dataset.

5. Correlation Analysis and Feature Selection

We use a correlation matrix to visualize the relationships between features. High correlations may indicate redundancy. However, a more robust approach is to calculate the Variance Inflation Factor (VIF) for each feature. VIF quantifies the multicollinearity between a feature and all other features. Features with a VIF greater than 10 are typically considered highly correlated and are candidates for removal. The process is iterative, removing one feature at a time and recalculating VIF until all remaining features have acceptable VIF values.

Correlation Matrix Visualization:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Calculate correlation matrix
corr_matrix = df_final.corr()

# Plot heatmap
plt.figure(figsize=(10,8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.show()
```

Variance Inflation Factor (VIF) Calculation:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Preparing DataFrame for VIF calculation
X = df_final.drop('annual_premium_amount', axis=1)
y = df_final['annual_premium_amount']

# Scaling features
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
X_scaled = pd.DataFrame(X_scaled, columns=X.columns)

# Function to calculate VIF
def calculate_vif(df):
    vif = pd.DataFrame()
    vif["Feature"] = df.columns
    vif["VIF"] = [variance_inflation_factor(df.values, i) for i in range(df.shape[1])]
    return vif

vif_df = calculate_vif(X_scaled)
print(vif_df)
```

Removing Features with High VIF:

Iteratively remove features with the highest VIF and recalculate until all VIF values are below 10.

```
# Example: Dropping 'income_level' due to high VIF
X_reduced = X_scaled.drop('income_level', axis=1)
vif_reduced = calculate_vif(X_reduced)
print(vif_reduced)
```

Rationale:

- **Income Level vs. Income in Lacks:** If `income_level` is a binned version of `income_lacks`, they are highly correlated. Dropping one prevents redundancy.
- **Iterative Process:** Always drop the feature with the highest VIF first to effectively reduce multicollinearity.

Summary

This comprehensive feature engineering process prepares the data for model training by:

- **Transforming Categorical Features:** Converting complex and categorical data into numerical formats suitable for machine learning models.
- **Handling Redundancies:** Dropping irrelevant or redundant features to streamline the dataset.
- **Addressing Multicollinearity:** Using correlation analysis and VIF to identify and remove highly correlated features, ensuring model stability.
- **Scaling Features:** Normalizing numerical features to ensure balanced influence across all predictors.

By meticulously engineering features, we enhance the model's ability to learn meaningful patterns, ultimately leading to more accurate and reliable insurance premium predictions.