Building a Fraud Detection Model Using Random Forest and KDD Methodology

Satvik Atmakuri

4 min read · Sep 27, 2024

Introduction

Fraud detection is a crucial task in the finance and payment industries. As the number of online transactions increases, detecting fraudulent transactions in real time has become vital. In this article, I will demonstrate how to build a fraud detection model using the KDD (Knowledge Discovery in Databases) methodology and a Random Forest classifier. We'll also cover hyperparameter tuning to further improve the model's performance.

We'll work with a well-known credit card fraud detection dataset. The dataset contains anonymized credit card transactions, with 492 fraudulent transactions out of 284,807, making it highly imbalanced.

KDD Methodology Overview

The KDD process is a systematic approach used in data mining and knowledge discovery. It involves the following five key steps:

1. **Data Selection:** Identify relevant data that is useful for solving the problem.
2. **Data Preprocessing:** Clean and transform the data, handle missing values, and prepare the features.
3. **Data Transformation:** Perform operations such as feature scaling or dimensionality reduction.
4. **Data Mining (Model Building):** Apply machine learning algorithms to build predictive models.
5. **Interpretation and Evaluation:** Assess the model's performance using various metrics.

Let's go through these steps with the fraud detection problem.

Step 1: Data Selection

We begin by loading the dataset and selecting the relevant features. The dataset contains numerical features (V1 to V28) that are the result of a PCA transformation for privacy. The key columns are:

- `Time` : The time of the transaction.

- `Amount` : The amount spent in the transaction.
- `Class` : The target variable indicating if the transaction is legitimate (`0`) or fraudulent (`1`).

Here's how you can load the dataset:

```
import pandas as pd
# Load the dataset
file_path = 'creditcard.csv'  # Change this path to your dataset location
data = pd.read_csv(file_path)
# Display the first few rows
data.head()
```

Step 2: Data Preprocessing

The next step is to preprocess the data. First, we check for any missing values, which could potentially disrupt the model's training. Fortunately, this dataset has no missing values.

Since the dataset is highly imbalanced (only 492 out of 284,807 transactions are fraudulent), handling this imbalance is crucial. For this, we'll use class weighting in the Random Forest model, which adjusts the weights for each class based on their frequencies.

In addition, we scale the `Time` and `Amount` columns, as their ranges differ significantly.

```
from sklearn.preprocessing import StandardScaler
# Check for missing values
data.isnull().sum()
# Scaling the 'Time' and 'Amount' columns
scaler = StandardScaler()
data[['Time', 'Amount']] = scaler.fit_transform(data[['Time', 'Amount']])
# Verify the scaling
data[['Time', 'Amount']].describe()
```

Step 3: Data Transformation

As the features (V1 to V28) are already PCA-transformed, there's no need for additional feature engineering or dimensionality reduction. We can move straight to model building.

Step 4: Data Mining (Model Building)

We'll build a Random Forest classifier, which is robust and works well with both large datasets and imbalanced classes. We'll use class weighting to handle the imbalance between fraudulent and legitimate transactions.

Initially, we train a model with default parameters and perform cross-validation to validate the model's performance.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
# Separating features and target variable
X = data.drop('Class', axis=1)
```

```
y = data['Class']
# Initialize Random Forest with class_weight set to 'balanced'
rf_model = RandomForestClassifier(class_weight='balanced', n_estimators=50, rand
# Perform cross-validation to assess model performance
cross_val_scores = cross_val_score(rf_model, X, y, cv=3, scoring='accuracy')
print('Cross-Validation Accuracy:', cross_val_scores.mean())
```

Step 5: Model Evaluation

After training the model, we can evaluate its performance by generating classification reports and confusion matrices. This allows us to measure important metrics like precision, recall, and the F1 score for both legitimate and fraudulent transactions.

```
# Fitting the Random Forest model
rf_model.fit(X, y)
# Predicting on the training set to evaluate performance
y_pred = rf_model.predict(X)
from sklearn.metrics import classification_report, confusion_matrix
# Generating classification report
print('Classification Report:')
print(classification_report(y, y_pred))
# Confusion Matrix
print('Confusion Matrix:')
print(confusion_matrix(y, y_pred))
```

Hyperparameter Tuning

To improve the model's performance, we can fine-tune its hyperparameters. Specifically, we'll adjust the max_depth parameter, which controls the depth of the trees in the Random Forest. Limiting the depth can prevent the model from overfitting.

Here, we test `max_depth=10` , which proved to be effective during tuning:

```
# Tuning the Random Forest with max_depth=10
rf_model_depth_10 = RandomForestClassifier(class_weight='balanced', n_estimators
# Cross-va
lidation with max_depth=10
cross_val_scores_depth_10 = cross_val_score(rf_model_depth_10, X, y, cv=3, scori
# Fit the model
rf_model_depth_10.fit(X, y)
# Evaluate performance
y_pred_depth_10 = rf_model_depth_10.predict(X)
print('Classification Report (max_depth=10):')
print(classification_report(y, y_pred_depth_10))
print('Confusion Matrix (max_depth=10):')
print(confusion_matrix(y, y_pred_depth_10))
```

Results

After tuning, the model achieved high accuracy with the following metrics:

- Precision for Class 0 (legitimate transactions): 1.00
- Recall for Class 1 (fraudulent transactions): 1.00
- Overall Accuracy: 99.93%

This shows that the model is highly effective at detecting fraud without misclassifying too many legitimate transactions.

Conclusion

In this article, we followed the KDD methodology to build a fraud detection model using Random Forest. By addressing the class imbalance with class weighting and tuning hyperparameters, we achieved a highly accurate and reliable model. This approach can be applied to various other classification problems where class imbalance is a challenge.

Feel free to adapt this approach to other datasets and explore more advanced techniques such as SMOTE for resampling or ensemble methods to further boost performance.

Written by Satvik Atmakuri

Edit profile

0 Followers

More from Satvik Atmakuri

Satvik Atmakuri

Building an Income Prediction Model with Machine Learning: A...

Machine learning projects are all about exploration, analysis, and iteration. In this...

Sep 27

Satvik Atmakuri

Building a Wine Quality Prediction Model Using the SEMMA...

In this article, we'll explore how to use the SEMMA methodology to build a machine...

Sep 27

Satvik Atmakuri

Predicting Heart Attack Risk Using Machine Learning: A CRISP-DM...

Introduction

Sep 27

See all from Satvik Atmakuri

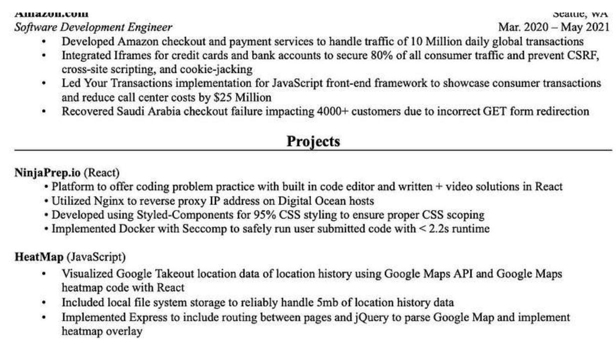
Recommended from Medium

Stephen Echessa

Stacking Ensembles: Combining XGBoost, LightGBM and CatBoost…

In the ever-evolving world of machine learning, where numerous algorithms vie for…

Jul 29 24 1



Software Development Engineer Mar. 2020 – May 2021

- Developed Amazon checkout and payment services to handle traffic of 10 Million daily global transactions
- Integrated Iframes for credit cards and bank accounts to secure 80% of all consumer traffic and prevent CSRF, cross-site scripting, and cookie-jacking
- Led Your Transactions implementation for JavaScript front-end framework to showcase consumer transactions and reduce call center costs by $25 Million
- Recovered Saudi Arabia checkout failure impacting 4000+ customers due to incorrect GET form redirection

Projects

NinjaPrep.io (React)

- Platform to offer coding problem practice with built in code editor and written + video solutions in React
- Utilized Nginx to reverse proxy IP address on Digital Ocean hosts
- Developed using Styled-Components for 95% CSS styling to ensure proper CSS scoping
- Implemented Docker with Seccomp to safely run user submitted code with < 2.2s runtime

HeatMap (JavaScript)

- Visualized Google Takeout location data of location history using Google Maps API and Google Maps heatmap code with React
- Included local file system storage to reliably handle 5mb of location history data
- Implemented Express to include routing between pages and jQuery to parse Google Map and implement heatmap overlay

Alexander Nguyen in Level Up Coding

The resume that got a software engineer a $300,000 job at Google.

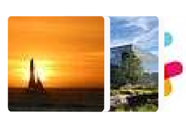
1-page. Well-formatted.

May 31 25K 478

Lists

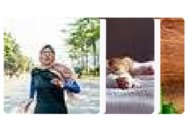
Staff Picks

755 stories · 1416 saves



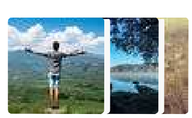
Stories to Help You Level-Up at Work

19 stories · 852 saves



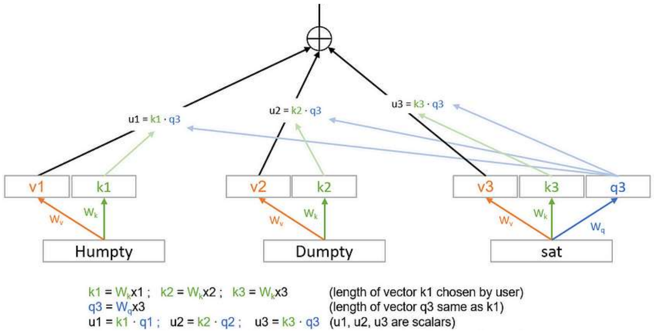
Self-Improvement 101

20 stories · 2957 saves



Productivity 101

20 stories · 2505 saves

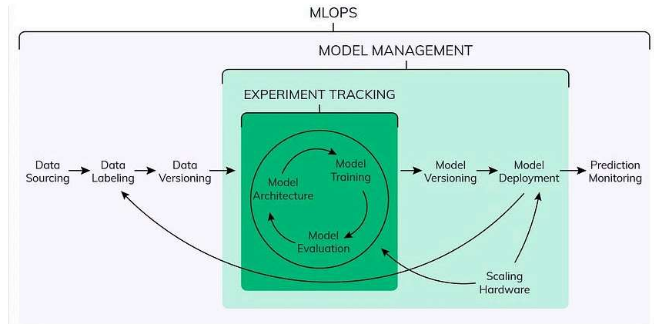


Rohit Patel in Towards Data Science

Understanding LLMs from Scratch Using Middle School Math

In this article, we talk about how LLMs work, from scratch—assuming only that you know...

Oct 19 2.1K 24

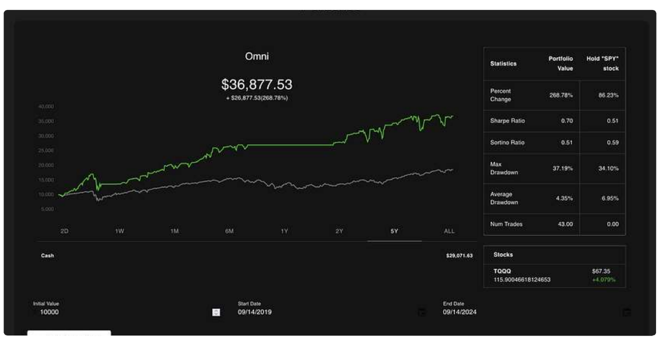


Prashant Shinde in Data And Beyond

From Zero to MLOps Hero: Your First Steps in Building an MLOps...

Just a few months ago, I was where you are—thrilled by the possibilities of machine...

Sep 15 56



Austin Starks in DataDrivenInvestor

I used OpenAI's o1 model to develop a trading strategy. It is...

It literally took one try. I was shocked.

Sep 15 5.3K 138

Ignacio de Gregorio

Apple Speaks the Truth About AI. It's Not Good.

Are We Being Lied To?

Oct 23 4.4K 136