Bintang Nabiil Lukman
1103223126
TK-46-06

Analisa – KlasifikasiUTS

1. **Jika model Machine Learning menunjukkan AUC-ROC tinggi (0.92) tetapi Presisi sangat rendah (15%) pada dataset tersebut, jelaskan faktor penyebab utama ketidaksesuaian ini! Bagaimana strategi tuning hyperparameter dapat meningkatkan Presisi tanpa mengorbankan AUC-ROC secara signifikan? Mengapa Recall menjadi pertimbangan kritis dalam konteks ini, dan bagaimana hubungannya dengan cost false negative?**

   Jawaban:

   Ketika sebuah model machine learning menunjukkan AUC-ROC yang tinggi (misalnya 0.92) tetapi presisi sangat rendah (15%), hal ini biasanya terjadi karena ketidakseimbangan kelas (imbalanced dataset). AUC-ROC mengukur kemampuan model dalam memisahkan kelas positif dan negatif secara keseluruhan di berbagai threshold. Artinya, meskipun secara global model tampak bagus (AUC tinggi), saat threshold default (biasanya 0.5) digunakan untuk klasifikasi, model mungkin menghasilkan banyak prediksi positif yang salah (false positives). Inilah yang menyebabkan presisi menjadi sangat rendah. Jadi, penyebab utamanya adalah model bisa membedakan kelas secara umum, tetapi threshold atau kebijakan prediksinya belum tepat.

   Untuk mengatasi hal ini, ada beberapa strategi yang bisa dilakukan melalui tuning hyperparameter, yaitu:

   - **Menyesuaikan threshold:** Dengan menaikkan threshold dari 0.5 ke angka yang lebih tinggi (misalnya 0.7–0.8), model akan lebih hati-hati sebelum memutuskan suatu kasus sebagai positif. Ini akan menekan jumlah false positive dan otomatis meningkatkan presisi.
   - **Memberi bobot khusus pada kelas minoritas:** Algoritma seperti Logistic Regression, Random Forest, dan SVM memungkinkan pengaturan class_weight untuk menyeimbangkan kelas, agar model lebih memperhatikan kelas yang minoritas.
   - **Tuning hyperparameter inti:** Misalnya, pada Random Forest kita bisa menyesuaikan max_depth atau min_samples_leaf agar model tidak terlalu overfit. Pada SVM, mengatur C dan gamma bisa membantu memperhalus margin klasifikasi, sehingga keputusan model lebih tajam.

Sementara itu, Recall menjadi pertimbangan yang sangat kritis dalam konteks seperti fraud detection atau diagnosa medis. Recall mengukur kemampuan model untuk menangkap seluruh kasus positif. Jika recall rendah, artinya banyak kasus penting yang lolos begitu saja (false negative), yang dalam dunia nyata bisa berakibat fatal, seperti transaksi fraud yang tidak terdeteksi atau pasien yang penyakitnya terabaikan. Oleh karena itu, dalam kasus-kasus seperti ini, kita sering memprioritaskan recall yang tinggi meskipun presisi sedikit dikorbankan. Ini karena biaya atau risiko dari false negative jauh lebih besar dibandingkan false positive (misalnya lebih baik memeriksa manual beberapa transaksi yang salah dikira fraud daripada membiarkan fraud yang sesungguhnya lolos).

Kesimpulannya, masalah presisi rendah ini biasanya karena threshold yang belum optimal dan dataset yang timpang. Dengan tuning threshold dan hyperparameter yang tepat, presisi bisa ditingkatkan tanpa mengorbankan AUC secara signifikan. Namun, dalam skenario high-stakes seperti fraud detection, fokus tetap harus diberikan pada recall karena dampak false negative lebih berat daripada false positive.

2. **Sebuah fitur kategorikal dengan 1000 nilai unik (high-cardinality) digunakan dalam model machine learning. Jelaskan dampaknya terhadap estimasi koefisien dan stabilitas Presisi! Mengapa target encoding berisiko menyebabkan data leakage dalam kasus dataset tersebut, dan alternatif encoding apa yang lebih aman untuk mempertahankan AUC-ROC?**

Jawaban:

Jika kita menggunakan fitur kategorikal dengan 1000 nilai unik (high-cardinality) dalam sebuah model machine learning, ada beberapa dampak yang cukup besar yang harus diperhatikan:

- Estimasi Koefisien jadi tidak stabil

  Misalnya dalam Logistic Regression, setiap kategori akan diubah menjadi fitur dummy (one-hot encoding). Dengan 1000 nilai unik, ini berarti model harus memperkirakan 1000 koefisien yang berbeda. Akibatnya, banyak kategori yang hanya muncul sedikit (bahkan mungkin hanya 1–2 kali), sehingga koefisiennya tidak stabil dan cenderung overfit terhadap data pelatihan. Ini bikin model sulit untuk generalisasi ke data baru.

- Presisi menjadi tidak konsisten

  High-cardinality cenderung membuat model terlalu spesifik terhadap kategori tertentu. Ketika kategori baru atau jarang muncul di data uji, model tidak punya cukup pengalaman untuk memprediksi dengan tepat. Ini bisa membuat Presisi turun drastis pada data nyata meskipun terlihat baik saat training.

Target encoding (juga dikenal sebagai mean encoding) mengganti setiap kategori dengan rata-rata nilai target untuk kategori tersebut. Meskipun ini efektif mereduksi dimensi dan bisa mempertahankan performa AUC-ROC, target encoding sangat berisiko menyebabkan data leakage jika diterapkan sembarangan. Masalahnya adalah kalau kita menghitung rata-rata target pakai seluruh dataset (termasuk data uji), informasi label dari data uji "bocor" masuk ke data latih. Ini membuat model terlihat sangat bagus saat training, tapi performanya drop parah ketika diaplikasikan ke data nyata karena sudah terkontaminasi informasi yang seharusnya tersembunyi.

Agar aman dan tetap mempertahankan AUC-ROC yang baik, beberapa strategi lebih disarankan:

- K-Fold Target Encoding
  Alih-alih menghitung mean target dari seluruh dataset, kita bagi data ke dalam beberapa fold (misalnya 5). Untuk setiap fold, kita hitung mean target hanya dari data di fold lain (out-of-fold), jadi tidak ada informasi dari data yang sedang diproses itu sendiri. Ini mengurangi risiko data leakage secara signifikan.
- Frequency Encoding
  Kita ubah kategori menjadi frekuensi kemunculannya di dataset. Ini aman karena hanya menggunakan informasi yang ada pada fitur itu sendiri, tanpa menyentuh target.
- Embedding (Untuk Model Non-Linear)
  Kalau pakai model seperti neural network atau gradient boosting, kita bisa memanfaatkan categorical embedding atau leave-one-out encoding, yang secara otomatis mempelajari representasi yang efisien tanpa harus membuat dummy 1000 kolom.

Kalau pakai model seperti neural network atau gradient boosting, kita bisa memanfaatkan categorical embedding atau leave-one-out encoding, yang secara otomatis mempelajari representasi yang efisien tanpa harus membuat dummy 1000 kolom.

3. **Setelah normalisasi Min-Max, model SVM linear mengalami peningkatan Presisi dari 40% ke 60% tetapi Recall turun 20%. Analisis dampak normalisasi terhadap decision boundary dan margin kelas minoritas! Mengapa scaling yang sama mungkin memiliki efek berlawanan jika diterapkan pada model Gradient Boosting?**

Jawaban:

Ketika kita melakukan **normalisasi Min-Max**, data diubah skalanya supaya semua fitur berada dalam rentang tertentu, biasanya antara 0 dan 1. Ini punya dampak besar terhadap model seperti **SVM Linear**, yang sangat sensitif terhadap skala fitur.

- Sebelum Normalisasi

Jika ada fitur yang skalanya jauh lebih besar dibanding fitur lain, SVM akan "lebih peduli" ke fitur besar itu dalam menentukan decision boundary. Akibatnya, garis pemisah (hyperplane) mungkin tidak optimal karena didominasi fitur yang skala besar, padahal fitur lain juga penting.

- Setelah Normalisasi

  Semua fitur disetarakan kontribusinya karena sudah dalam skala yang sama. Ini membuat decision boundary lebih seimbang dan tidak condong ke satu fitur saja. Hasilnya, presisi naik dari 40% ke 60%, model jadi lebih selektif dalam memutuskan kelas positif, sehingga lebih sedikit kesalahan positif palsu. Namun, Recall turun karena lebih ketat, model malah melewatkan lebih banyak kasus positif, yang menjelaskan penurunan Recall 20%.

Gradient Boosting (misalnya XGBoost atau LightGBM) tidak bergantung pada jarak antar titik data seperti SVM. Dia membangun pohon keputusan (decision tree) secara bertahap dan fokus pada pembagian (split) berdasarkan nilai fitur, bukan seberapa jauh satu titik dengan yang lain. Karena itu:

- Min-Max scaling tidak terlalu berpengaruh pada Gradient Boosting. Bahkan kalau skalanya diubah, selama urutan atau distribusi fitur tetap sama, model tetap menemukan split yang optimal.
- Kadang, scaling yang memaksa semua fitur ke skala kecil bisa mengurangi variasi alami yang justru berguna untuk membagi data secara tajam dalam pohon keputusan. Ini membuat Gradient Boosting kurang efektif dan justru bisa menurunkan performa.

4. **Eksperimen feature interaction dengan menggabungkan dua fitur melalui perkalian meningkatkan AUC-ROC dari 0.75 ke 0.82. Jelaskan mekanisme matematis di balik peningkatan ini dalam konteks decision boundary non-linear! Mengapa uji statistik seperti chi-square gagal mendeteksi interaksi semacam ini, dan metode domain knowledge apa yang dapat digunakan sebagai alternatif?**

Jawaban:

Contoh mekanismenya misal kita punya dua fitur X_1 dan X_2, awalnya model hanya belajar pola linear seperti:

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2$$

Setelah kita buat interaksi $X_1 \times X_2$, model bisa belajar:

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 (X_1 \times X_2)$$

Di sini $(X_1 \times X_2)$ memungkinkan model untuk menyesuaikan decision boundary yang non-linear. Misalnya, kasus tertentu hanya muncul jika keduanya tinggi, tapi tidak

cukup kalau hanya salah satunya. Fitur interaksi ini menangkap pola semacam itu dan membuat model lebih fleksibel dalam memisahkan kelas, sehingga AUC-ROC meningkat dari 0.75 ke 0.82.

Chi-square test hanya mengukur hubungan langsung antara satu fitur dengan target secara univariat (satu per satu). Artinya:

- Dia tidak memperhitungkan kombinasi dua fitur sekaligus.
- Jika X_1 dan X_2 secara individu tidak kuat, tapi kombinasi mereka penting, chi-square tidak akan melihat kekuatan ini.

Karena itu, uji statistik seperti chi-square cenderung gagal mendeteksi interaksi antar fitur yang secara sendiri-sendiri tampak lemah tapi bersama-sama sangat informatif.

Supaya interaksi yang relevan bisa ditemukan, kita bisa:

- Menggunakan pengetahuan domain, misalnya di bidang medis, kombinasi dua gejala tertentu mungkin menjadi indikator yang kuat untuk penyakit tertentu.
- Menggunakan metode otomatis, seperti decision tree atau random forest, yang secara alami sudah mencari split yang bersifat interaktif antar fitur.
- Menggunakan teknik statistik lanjutan, seperti mutual information atau partial dependence plot, yang memvisualisasikan hubungan antar fitur dan target dalam bentuk 2D atau lebih.

5. **Dalam pipeline preprocessing, penggunaan oversampling sebelum pembagian train-test menyebabkan data leakage dengan AUC-ROC validasi 0.95 tetapi AUC-ROC testing 0.65. Jelaskan mengapa temporal split lebih aman untuk fraud detection, dan bagaimana stratified sampling dapat memperparah masalah ini! Bagaimana desain preprocessing yang benar untuk memastikan evaluasi metrik Presisi/Recall yang realistis?**

Jawaban:

Dalam fraud detection, data biasanya memiliki komponen waktu (time-dependent)—misalnya transaksi yang terjadi hari ini bisa sangat berbeda polanya dengan yang terjadi 3 bulan lalu. Kalau kita hanya pakai split acak (random split), kita berisiko menggunakan informasi dari masa depan untuk memprediksi masa lalu yang tidak realistis dalam dunia nyata. Dengan temporal split, kita memisahkan data berdasarkan waktu, misalnya:

- Train: dara Januari-Mei
- Test: data Juni-Juli

Ini jauh lebih aman karena mensimulasikan skenario nyata, yaitu memprediksi data yang *belum pernah terjadi*.

Stratified sampling memastikan distribusi kelas tetap seimbang di train dan test. Tapi dalam konteks oversampling sebelum split, stratifikasi bisa memperburuk:

- Data sintetis (hasil oversampling**) ikut ter-stratifikasi** ke test set.
- Test set akhirnya terkontaminasi data palsu/sintetis, yang membuat evaluasi tidak valid dan memberi kesan bahwa model performanya tinggi padahal tidak.

Supaya metrik seperti Presisi/Recall realistis, pipeline yang benar adalah:

- Lakukan train-test split dulu, *baru* lakukan oversampling di training set saja. Ini memastikan test set tetap benar-benar murni.
- Ini lebih baik dari random split karena mencerminkan skenario prediksi nyata.
- Lakukan scaling, encoding, dan semua transformasi hanya berdasarkan training set (fit di train, transform di train & test). Ini mencegah kebocoran informasi statistik.
- Pastikan test set tidak mengandung data sintetis atau informasi yang bocor. Semua metrik yang diukur harus berdasarkan data asli untuk mendapatkan gambaran performa yang sebenarnya.