BAB 2

LANDASAN TEORI

2.1 Phising

Phishing adalah upaya untuk mendapatkan informasi sensitif seperti *username*, *password*, dan rincian kartu kredit dan umumnya untuk alasan berbahaya, dengan menyamar sebagai entitas terpercaya dalam sebuah komunikasi elektronik. Menurut Whittaker, Ryner & Nazif, *phishing* adalah kejahatan rekayasa sosial (*social engineering crime*) yang umumnya didefinisikan sebagai kejahatan yang berkedok atau menyamar sebagai pihak ketiga terpercaya untuk mendapatkan akses ke data pribadi. Dapat diambil kesimpulan bahwa *phishing* merupakan tindak kriminal untuk mendapatkan informasi sensitif pribadi seperti *username* dan *password* dengan cara melakukan penyamaran sebagai entitas yang terpercaya dan membuat korban tidak sadar telah memberikan informasi sensitif ke penipu.

2.1.1 Phising Website Feature

Pada penelitian yang dilakukan oleh Rami M. Mohammad dan kawan-kawan yang berjudul "*An Assesment of Features Related to Phising Websites using an Automated Technique*" diusulkan *rules* atau kriteria untuk mengambil fitur secara otomatis, Struktur dari *rules* ini terdiri dari empat kelas utama dimana fitur diklasifikasikan dan ditempatkan di kelas yang tepat. Keempat kelas tersebut adalah fitur pada *address bar,* fitur yang abnormal, fitur yang diambil dari komponen HTML dan *JavaScript,* dan fitur berdasarkan domain. Pada fitur yang berasal dari

address bar, contohnya adalah penggunaan *IP Address*, panjang dari *url,* penggunaan simbol tertentu, penggunaan *subdomain.* Untuk fitur yang abnormal, contohnya adalah *request url,* URl *of anchor, server form handler,* dan *abnormal url.* Untuk fitur yang diambil dari komponen HTML dan *JavaScript,* terdapat *redirect page, disabling right click, using mouseover to hide link.* Sedangkan untuk fitur berdasarkan domain, terdapat *age of domain, dns record,* dan *website traffic.*

2.2 Decision Tree

Decision tree merupakan suatu metode klasifikasi yang menggunakan struktur pohon, dimana setiap node merepresentasikan atribut dan cabangnya merepresentasikan nilai dari atribut, sedangkan daunnya digunakan untuk merepresentasikan kelas. Node teratas dari decision tree ini disebut dengan root. Breiman et al. (1984) menyatakan bahwa metode ini merupakan metode yang sangat populer untuk digunakan karena hasil dari model yang terbentuk mudah untuk dipahami. Dinamai pohon keputusan karena aturan yang dibentuk mirip dengan bentuk pohon. Pohon dibentuk dari proses penyortiran rekursif *biner* dalam kelompok data, sehingga nilai variabel respons di setiap kelompok data membuat hasil penyortiran menjadi lebih homogen (Akbar Asfihan, 2021).

Konsep dari pohon keputusan adalah mengubah data menjadi decision tree dan aturan-aturan keputusan. Manfaat utama dari menggunakan pohon keputusan adalah kemampuan untuk menyederhanakan proses pengambilan keputusan yang kompleks sehingga pembuat keputusan dapat menafsirkan solusi untuk masalah.

2.3 Gradient Boosted Trees

Gradient Boosting pertama kali diperkenalkan oleh J.H. Friedman. GBT mampu membangun *decision tree* berdasarkan peningkatan dalam struktur pohon pada pembelajaran yang lemah untuk memperbaiki kesalahan pohon dan mencegah terjadinya potensi *overfitting*. Dalam membangun *decision tree*, dapat dilakukan penambahan jumlah iterasi yang sangat konservatif yang dapat menghasilkan dan meningkatkan kinerja model yang lebih baik. GBT mampu memecahkan masalah dengan menyesuaikan pembelajaran lemah dengan gradien negatif dari fungsi kerugian (*loss function*) dan meningkatkan pohon (*trees*) dengan parameter yang mewakili variabel *split* yang dipasang pada setiap *node* terminal pohon (Achmad Bisri, 2019).

2.4 Extreme Gradient Boosting

Extreme Gradient Boosting atau XGBoost merupakan kumpulan *decission tree* yang saat membangun pohon berikutnya bergantung pada pohon sebelumnya. Pohon pertama dalam XGBoost lemah dalam melakukan klasifikasi dengan inisialisasi kemungkinan yang ditentukan oleh peneliti dan kemudian dilakukan *update* bobot pada setiap pohon yang dibangun sehingga menghasilkan kumpulan pohon klasifikasi yang kuat. Prediksi dilakukan dengan menjumlahkan seluruh bobot yang ada di setiap pohon dan kemudian memasukkan nilai tesebut ke fungsi logistik (Muhamad Syukron, 2020).

Algoritma XGBoost bekerja dengan menggunakan *loss function* untuk membangun tree dengan menimalisir nilai. Dalam hal ini dataset memiliki n data

dengan n row, dengan i untuk merepresentasikan setiap data didalamnya. (Sydney Chen, 2020)

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \ where\ \Omega(f_k) = \gamma T + \frac{1}{2}\lambda\|\omega\|^2 \qquad (2.1)$$

Pada persamaan diatas pada bagian pertama merepresentasikan *loss function* yang menghitung pseudo residual dari nilai yi yang diprediksi dengan *hat* dan *true value* dari yi di setiap *leaf*, pada bagian kedua berisi dua bagian. Pada bagian terakhir di formula *omega* berisi *regularization term lambda* dimana ditunjukkan untuk mengurangi insensitifitas prediksi terhadap observasi individual dan w merepresentasikan *weight* dari *leaf* yang dapat dijadikan sebagai nilai *output* dari *leaf*. T merepresentasikan angka dari *terminal nodes* atau *leaves* di pohon dan *gamma* merepresentasikan *user-definable penalty* yang yang mendorong *pruning.*

Tujuannya adalah untuk mencari nilai output yang teroptimisasi untuk leaf untuk meminimalisir *equation*. Dikarenakan dimulai dengan value awal y0 dengan hat, prediksi baru selalu sama dengan prediksi ke i-1th ditambah dengan nilai hasil keluaran dari pohon ke ith. Sehingga bagian pertama disubtitusi menjadi sebagai berikut.

$$\mathcal{L}^t \simeq \sum_{i=1}^{n} \left[l\left(y_i, \hat{y}_i^{(t-1)}\right) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \qquad (2.2)$$

Lalu pada bagian pertama dari persamaan. Dengan menggunakan *second order taylor approximation*, didapatkan persamaan sebagai berikut. Dikarenakan turunan pertama dari *loss function* berhubungan dengan g*radient*, digunakan gi

untuk merepresentasikan *gradient*, dan turunan kedua dari *loss function* sebagai *hessian* dengan hi untuk merepresentasikan.

$$\mathcal{L}^t \simeq \sum_{i=1}^{n} \left[l\left(y_i, \hat{y}_i^{(t-1)}\right) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \tag{2.3}$$

Dengan mengembangkan sigma, dari setiap i sama dengan L(yi,y*hat*(i-1)) ditambah dengan bagian gi dan hi. Dikarenakan L(yi,y*hat*(i-1)) tidak terkait dengan ft(xi), tidak ada efek pada keluaran final sehingga dapat diabaikan untuk menyederhanakan perhitungan. Sehingga didapatkan persamaan sebagai berikut.

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} \omega_j^2 \right]$$

$$= \sum_{j=1}^{T} \left[\left(\sum_{i \epsilon I_j} g_i\right) \omega_j + \frac{1}{2} \left(\sum_{i \epsilon I_j} h_i + \lambda\right) \omega_j^2 \right] + \gamma T \tag{2.4}$$

$$\omega_j^* = -\frac{\sum_{i \epsilon I_j} g_i}{\sum_{i \epsilon I_j} h_i + \lambda}$$

Nilai keluaran yang didapat adalah sebagai berikut.

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{\left(\sum_{i \epsilon I_j} g_i\right)^2}{\sum_{i \epsilon I_j} h_i + \lambda} + \gamma T \tag{2.5}$$

2.5 Holdout Validation dan K-Fold Validation

Data yang ada biasanya tidak seluruhnya digunakan dalam proses pelatihan. Untuk menghindari *overfitting* maka model harus dibangun sedemikian rupa

sehingga ketika ada data baru bisa diprediksi sama baiknya dengan menggunakan data pada proses pelatihan. Pengujian model bisa dilakukan dengan data *test* yang telah memiliki label kelas respon. Pembagian data *train* dan data *test* bisa dilakukan dengan *holdout validation* yaitu membagi data menjadi 2 bagian dengan proporsi tertentu yang ditentukan oleh peneliti. Proporsi yang biasa digunakan oleh peneliti adalah 60/40, 70/30, atau 80/20. Selain *holdout validation*, data juga bisa dibagi menjadi data latih dan data uji dengan metode *K-fold cross validation*. Metode ini membagi data *train* dan data *test* sebanyak "k" kelompok, sehingga proses pelatihan menjadi sebanyak "k" kemudian *performance* dari model merupakan rata-rata dari semua proses pelatihan tersebut (Muhamad Syukron, 2020).

2.6 Browser Extension

Ekstensi browser adalah program mini khusus yang memungkinkan untuk menambahkan fungsional baru yang berguna untuk memenuhi kebutuhan pengguna yang dalam hal yang spesifik. Ekstensi browser pada chrome dibangun menggunakan HTML dan *JavaScript*.

Sebuah ekstensi harus dapat memenuhi sebuah kebutuhan yang spesifik dan mudah dipahami. Sebuah ekstensi dapat memiliki beberapa komponen dan beragam fungsionalitas, yang berkontribusi untuk mencapai tujuan utama (Google Developers, 2020).

2.7 Evaluasi Performa

Evaluasi dilakukan dengan tujuan untuk mengetahui nilai performa dari sistem ini dilakukan dengan perhitungan *precision*, *recall* dan F1 score.

Precision digunakan untuk mengukur ketepatan hasil *classifier*, yang dapat dihitung dengan TP dibagi dengan total dari penjumlahan TP dengan FP. Persamaan *Precision* dapat dilihat pada persamaan 2.1.

$$Precision = \frac{TP}{(TP + FP)} \tag{2.6}$$

Recall untuk digunakan mengukur kelengkapan hasil *classifier*, yang dapat diukur dengan perhitungan TP dibagi dengan total penjumlahan TP dan FN. Persamaan *recall* dapat dilihat pada persamaan 2.2.

$$Recall = \frac{TP}{(TP + FN)} \tag{2.7}$$

F-*Measure* digunakan untuk mengukur rata-rata harmonic dari *precision* dan *recall*. Pengukuran F-*Measure* dapat dilihat pada persamaan 2.3.

$$F1 = 2 . \frac{(Precision\ x\ Recall)}{(Precision + Recall)} \tag{2.8}$$

2.8 Recursive Feature Elimination

Recursive Feature Elimination (RFE) merupakan sebuah algoritma *feature selection* (pemilihan fitur). *Feature selection* merupakan teknik yang memilih sebagian dari fitur yang paling relevan dari *dataset*. Dengan fitur yang lebih sedikit, algoritma pembelajaran mesin dapat berjalan lebih efisien dan efektif. Tahapan dalam RFE adalah sebagai berikut:

1. Pertama, inisialisasi sebuah set dari fitur F.
2. Pilih *classifier* C
3. Hitung *weight* dari setiap fitur fi di dalam F (pemilihannya berdasarkan akurasi dari prediksi *classifier*)
4. Hapus fitur yang memiliki *weight* paling minimum atau terkecil dan update F

5. Ulangi langkah 3 dan 4 hingga F hanya memiliki 1 fitur tersisa.
6. Lakukan *ranking* dari *feature importance*