BAB 1
Pendahuluan

Dalam buku panduan/ tutorial ini, penulis akan memandu Anda melalui proses membangun suatu sistem Deteksi *Website Phising* yang menggunakan teknik *machine learning* dan *model Gradient Boosting Classifier*. Sistem ini akan dibuat menggunakan bahasa pemrograman *python*, yang akan digunakan untuk membuat model *machine learning* dan juga untuk membuat tampilan *website* menggunakan *framework flask*. Tujuan dari sistem ini adalah untuk membantu mengidentifikasi *website phising* yang mungkin merugikan pengguna internet dengan menipu mereka untuk memberikan informasi pribadi atau mengeluarkan dana dengan cara yang tidak sah.

Phishing merupakan tindakan untuk mendapatkan informasi penting seseorang berupa *username, password* dan informasi sensitif lainnya dengan memberikan informasi palsu situs web yang mirip dengan aslinya. Pengelabuan (pemancing informasi penting) adalah sala satu bentuk tindak pidana yang bermaksud untuk mendapatkan rasahasia informasi dari seseorang, seperti nama pengguna, kata sandi dan kartu kredit, dengan menyamar sebagai orang lain atau bisnis terperaya di elektronik resmi komunikasi, seperti surat elektronik dan *instant messages*.[1]

Dengan mengikuti tutorial ini, Anda akan belajar bagaimana menggunakan *python* untuk membuat model *machine learning*, serta bagaimana menggunakan *framework flask* untuk membuat tampilan *website* yang interaktif dan *user friendly*. Setelah menyelesaikan tutorial ini, Anda akan memiliki keterampilan yang diperlukan untuk

membangun sistem Deteksi *Website Phising* sendiri yang dapat membantu melindungi pengguna internet dari ancaman *phishing*.

Sebelum kita memulai pembangunan sistem deteksi *website phishing*, ada beberapa hal yang perlu dipertimbangkan. Pertama, pastikan bahwa Anda memiliki pengetahuan dasar tentang *machine learning* dan pengklasifikasi algoritma, karena kita akan menggunakan teknik ini untuk membangun sistem deteksi *website phishing*. Kedua, siapkan *tools* yang diperlukan untuk mendukung pembangunan sistem ini, seperti python untuk membuat model *machine learning* dan *framework flask* untuk membuat tampilan *website*.

Setelah Anda mempersiapkan hal-hal tersebut, kita dapat mulai membangun sistem deteksi *website phishing* sesuai dengan *tutorial* yang akan kita bahas di buku ini mulai dari perancangan, pembuatan model untuk menentukan model yang terbaik dalam melakukan deteksi *website phising*, lalu melakukan *convert* model dan pembuatan aplikasi *website*.

1.1 Pengenalan Machine Learning

Gambar 1.1 Machine Learning

Pembelajaran mesin adalah aplikasi kecerdasan buatan (AI) yang memberikan sistem kemampuan untuk belajar dan belajar

secara otomatis meningkatkan dari pengalaman tanpa diprogram secara eksplisit. Ini berfokus pada pengembangan program komputer yang bisa mengakses data dan menggunakannya untuk belajar sendiri.

Algoritma pembelajaran mesin sering dikategorikan sebagai diawasi atau tidak diawasi. Algoritma yang diawasi membutuhkan ilmuwan data atau analis data dengan keterampilan pembelajaran mesin untuk memberikan *input* dan yang diinginkan *output*, selain memberikan umpan balik tentang keakuratan prediksi selama pelatihan algoritma [2].

Berikut ini adalah beberapa contoh penerapan *machine learning*:

1. Sistem rekomendasi: Sistem rekomendasi menggunakan *machine learning* untuk mengelompokkan pengguna berdasarkan preferensi dan kebiasaan mereka, dan kemudian menyarankan produk atau layanan yang mungkin mereka sukai.
2. Analisis sentimen: *Machine learning* dapat digunakan untuk menganalisis sentimen terhadap suatu produk atau layanan dengan mempelajari ulasan atau komentar yang diberikan oleh pengguna.
3. Pendeteksi spam: *Machine learning* dapat digunakan untuk mengidentifikasi dan memblokir *email* spam dengan mempelajari pola-pola yang biasanya terdapat pada *email* spam.
4. Pengenalan wajah: *Machine learning* dapat digunakan untuk mengenali wajah orang dalam foto atau video dengan mempelajari wajah-wajah yang telah dikenali terlebih dahulu.
5. Penerjemahan otomatis: Machine learning dapat digunakan untuk menerjemahkan teks dari satu bahasa ke bahasa lain dengan mempelajari terjemahan yang telah dibuat oleh

manusia.

1.2 Supervised Learning

Algoritma pembelajaran mesin *Supervised Learning* dapat menerapkan apa yang telah dipelajari di masa lalu ke data baru menggunakan contoh berlabel memprediksi peristiwa masa depan. Dimulai dari analisis dataset pelatihan yang diketahui, algoritma pembelajaran menghasilkan kesimpulan berfungsi untuk membuat prediksi tentang nilai output. Sistem mampu memberikan target untuk setiap masukan baru setelah cukup pelatihan. Algoritma pembelajaran juga dapat membandingkan keluarannya dengan keluaran yang benar dan diinginkan serta menemukan kesalahan untuk dimodifikasi sesuai dengan modelnya.[3]

1.2.1 Logistic Regression

Regresi logistik adalah model prediktif yang digunakan untuk mengevaluasi hubungan antara variabel dependen (target) yang merupakan data kategorikal dengan skala nominal atau ordinal dan variabel independen (prediktor) yang merupakan data kategorikal dengan skala interval atau rasio. Algoritma ini juga dapat digunakan untuk pemodelan deret waktu untuk menemukan hubungan antar variabel yang terlibat. Regresi logistik adalah algoritma yang digunakan untuk memprediksi probabilitas variabel dependen kategori. Dalam regresi logistik, variabel dependen ditampilkan sebagai variabel biner yang bernilai 1 (ya) atau 0 (tidak). Model regresi logistik memprediksi sebagai fungsi X. Asumsi yang digunakan dalam regresi Logistik adalah sebagai berikut: regresi logistik biner membutuhkan

variabel dependen biner, untuk regresi biner, tingkat faktor 1 dari variabel dependen harus mewakili hasil yang diinginkan, variabel independen harus independen satu sama lain. Dalam hal ini, model harus memiliki sedikit atau tidak ada multikolinearitas dan berhubungan secara linear dengan peluang log [4].

Berikut ini adalah contoh script sederhana untuk melakukan *logistic regression* menggunakan *Python*:

```python
# Import library yang dibutuhkan
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Persiapkan data
X = [[2, 3], [4, 5], [6, 7], [8, 9]] # data fitur
y = [0, 0, 1, 1] # data kelas

# Split data menjadi data training dan data testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# Buat model logistic regression
model = LogisticRegression()

# Latih model dengan data training
model.fit(X_train, y_train)

# Uji model dengan data testing
score = model.score(X_test, y_test)

# Cetak akurasi model
print("Akurasi model:", score)

# Prediksi kelas suatu objek baru
prediction = model.predict([[5, 6]])
print("Kelas prediksi:", prediction)
```

Gambar 1.2.1 Contoh Script Logistic Regression

Penjelasan *script Logistic Regression* diatas:

Script tersebut terdiri dari beberapa langkah:

1. *Import library* yang dibutuhkan, yaitu ***LogisticRegression*** dan ***train_test_split*** dari *scikit-learn.*
2. Persiapkan data fitur dan kelas yang akan digunakan untuk melatih model.

3. *Split* data menjadi data training dan data *testing* dengan ukuran data *testing* sebesar 30%.
4. Buat model *logistic regression*.
5. Latih model dengan data *training*.
6. Uji model dengan data *testing* dan dapatkan skor akurasi.
7. Cetak akurasi model.
8. Prediksi kelas suatu objek baru.
9. Cetak kelas prediksi.

Setelah melakukan semua langkah tersebut, Anda akan mendapatkan model *logistic regression* yang telah dilatih dan diuji serta dapat digunakan untuk melakukan prediksi kelas suatu objek baru.

1.2.2 K-Nearest Neighbors

K-Nearest Neighbor adalah metode klasifikasi dengan mencari jarak terdekat antara data yang akan dievaluasi dengan *K-Nearest Neighbors* terdekatnya dalam data pelatihan. Model ini dapat digunakan dalam klasifikasi yang akan dilakukan data *training* didalam proses pelatihan tersebut.**[5]** Proses pelatihan KNN menghasilkan k yang memberikan akurasi tertinggi dalam menggeneralisasi data yang akan datang.

Berikut ini adalah contoh *script* sederhana untuk melakukan *K-Nearest Neighbors* (KNN) menggunakan *Python*:

```python
# Import library yang dibutuhkan
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

# Persiapkan data
X = [[2, 3], [4, 5], [6, 7], [8, 9]] # data fitur
y = [0, 0, 1, 1] # data kelas

# Split data menjadi data training dan data testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# Buat model KNN dengan jumlah tetangga sebanyak 3
model = KNeighborsClassifier(n_neighbors=3)

# Latih model dengan data training
model.fit(X_train, y_train)

# Uji model dengan data testing
score = model.score(X_test, y_test)

# Cetak akurasi model
print("Akurasi model:", score)

# Prediksi kelas suatu objek baru
prediction = model.predict([[5, 6]])
print("Kelas prediksi:", prediction)
```

Gambar 1.2.2 Contoh Script KNN

Penjelasan *script* KNN diatas:

Script tersebut terdiri dari beberapa langkah:

1. *Import library* yang dibutuhkan, yaitu ***KNeighborsClassifier*** dan ***train_test_split*** dari *scikit-learn.*
2. Persiapkan data fitur dan kelas yang akan digunakan untuk melatih model.
3. *Split* data menjadi data training dan data *testing* dengan ukuran data *testing* sebesar 30%.
4. Buat model KNN.
5. Latih model dengan data *training*.
6. Uji model dengan data *testing* dan dapatkan skor akurasi.
7. Cetak akurasi model.
8. Prediksi kelas suatu objek baru.

9. Cetak kelas prediksi.

Setelah melakukan semua langkah tersebut, Anda akan mendapatkan model KNN yang telah dilatih dan diuji serta dapat digunakan untuk melakukan prediksi kelas suatu objek baru.

1.2.3 Support Vector Machine

Mesin vektor pendukung adalah algoritma kuat lainnya di teknologi pembelajaran mesin. Dalam mendukung mesin vektor algoritma setiap item data diplot sebagai titik dalam n-dimensi ruang dan mendukung konstruksi algoritma mesin vector garis pemisah untuk klasifikasi dua kelas, pemisahan ini garis dikenal sebagai *hyperplane*.

SVM adalah teknik pembelajaran mesin berdasarkan *Supervised Learning* dan sesuai untuk kedua regresi dan klasifikasi. SVM dianggap sebagai pencapaian teknik modern penerimaan cepat karena hasil yang baik dicapai dalam banyak bidang masalah data mining, berdasarkan fondasi yang kuat dalam teori belajar statistik. SVM adalah teknik klasifikasi berdasarkan pembelajaran statistik, yang berhasil dimanfaatkan dalam banyak aplikasi klasifikasi nonlinier dan besar dataset dan masalah. [6].

Berikut ini adalah contoh script sederhana untuk melakukan *Support Vector Machine* (SVM) menggunakan *Python*:

```python
# Import library yang dibutuhkan
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split

# Persiapkan data
X = [[2, 3], [4, 5], [6, 7], [8, 9]] # data fitur
y = [0, 0, 1, 1] # data kelas

# Split data menjadi data training dan data testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# Buat model SVM dengan kernel linear
model = SVC(kernel='linear')

# Latih model dengan data training
model.fit(X_train, y_train)

# Uji model dengan data testing
score = model.score(X_test, y_test)

# Cetak akurasi model
print("Akurasi model:", score)

# Prediksi kelas suatu objek baru
prediction = model.predict([[5, 6]])
print("Kelas prediksi:", prediction)
```

Gambar 1.2.3 Contoh Script SVM

Penjelasan *script* SVM diatas:

Script tersebut terdiri dari beberapa langkah:

1. *Import library* yang dibutuhkan, yaitu ***SVC*** dan ***train_test_split*** dari *scikit-learn.*
2. Persiapkan data fitur dan kelas yang akan digunakan untuk melatih model.
3. *Split* data menjadi data training dan data *testing* dengan ukuran data *testing* sebesar 30%.
4. Buat model SVM.
5. Latih model dengan data *training*.
6. Uji model dengan data *testing* dan dapatkan skor akurasi.
7. Cetak akurasi model.
8. Prediksi kelas suatu objek baru.
9. Cetak kelas prediksi.

Setelah melakukan semua langkah tersebut, Anda akan mendapatkan model SVM yang telah dilatih dan diuji serta dapat digunakan untuk melakukan prediksi kelas suatu objek baru.

1.2.4 Naïve Bayes

Naive bayes termasuk ke dalam pembelajaran *supervised*, sehingga pada tahapan pembelajaran dibutuhkan data awal berupa data pelatihan untuk dapat mengambil keputusan. Pada tahapan pengklasifikasian akan dihitung nilai probabilitas dari masing-masing label kelas yang ada terhadap masukan yang diberikan.

Pengklasifikasi *Naïve Bayes* adalah salah satu deteksi tinggi pendekatan untuk mempelajari klasifikasi dokumen teks. Diberikan satu set sampel pelatihan rahasia, sebuah aplikasi dapat belajar dari sampel tersebut, sehingga dapat memprediksi kelas sampel yang tidak terpenuhi. [7].

Berikut ini adalah contoh *script* sederhana untuk melakukan *Naive Bayes* menggunakan *Python*:

```python
# Import library yang dibutuhkan
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split

# Persiapkan data
X = [[2, 3], [4, 5], [6, 7], [8, 9]] # data fitur
y = [0, 0, 1, 1] # data kelas

# Split data menjadi data training dan data testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# Buat model Naive Bayes
model = GaussianNB()

# Latih model dengan data training
model.fit(X_train, y_train)

# Uji model dengan data testing
score = model.score(X_test, y_test)

# Cetak akurasi model
print("Akurasi model:", score)

# Prediksi kelas suatu objek baru
prediction = model.predict([[5, 6]])
print("Kelas prediksi:", prediction)
```

Gambar 1.2.4 Contoh Script Naïve Bayes

Penjelasan *script Naïve Bayes* diatas:

Script tersebut terdiri dari beberapa langkah:

1. *Import library* yang dibutuhkan, yaitu ***GaussianNB*** dan ***train_test_split*** dari *scikit-learn.*
2. Persiapkan data fitur dan kelas yang akan digunakan untuk melatih model.
3. *Split* data menjadi data training dan data *testing* dengan ukuran data *testing* sebesar 30%.
4. Buat model *Naïve Bayes*.
5. Latih model dengan data *training*.
6. Uji model dengan data *testing* dan dapatkan skor akurasi.
7. Cetak akurasi model.
8. Prediksi kelas suatu objek baru.
9. Cetak kelas prediksi.

Setelah melakukan semua langkah tersebut, Anda

akan mendapatkan model *Naïve Bayes* yang telah dilatih dan diuji serta dapat digunakan untuk melakukan prediksi kelas suatu objek baru.

1.2.5 Decision Tree

Pohon keputusan (DT). DT mengklasifikasikan barang berdasarkan pembuatan keputusan pada setiap cabang untuk mendapatkan sebanyak-banyaknya keuntungan entropi sebanyak mungkin. Sebuah pohon keputusan terdiri dari a simpul akar, beberapa simpul internal, dan simpul daun. Daun *node* menunjukkan hasil dari *classifier*, dan lainnya *node* menunjukkan setiap atribut. Setiap rute dari simpul akar ke simpul daun sesuai dengan penentuan urutan pengujian. Ini mengikuti aturan dari memecah dan menaklukkan [8]. Setiap pohon memiliki cabang, cabang mewakili suatu atribut yang harus dipenuhi untuk menuju cabang selanjutnya hingga berakhir di daun (tidak ada cabang lagi). Konsep data dalam *decision tree* adalah data dinyatakan dalam bentuk tabel yang terdiri dari atribut dan *record*. Atribut digunakan sebagai parameter yang dibuat sebagai kriteria dalam pembuatan pohon.

Berikut ini adalah contoh script sederhana untuk melakukan *decision tree* menggunakan *Python*:

```python
# Import library yang dibutuhkan
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

# Persiapkan data
X = [[2, 3], [4, 5], [6, 7], [8, 9]] # data fitur
y = [0, 0, 1, 1] # data kelas

# Split data menjadi data training dan data testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# Buat model decision tree dengan kriteria gini
model = DecisionTreeClassifier(criterion='gini')

# Latih model dengan data training
model.fit(X_train, y_train)

# Uji model dengan data testing
score = model.score(X_test, y_test)

# Cetak akurasi model
print("Akurasi model:", score)

# Prediksi kelas suatu objek baru
prediction = model.predict([[5, 6]])
print("Kelas prediksi:", prediction)
```

Gambar 1.2.5 Contoh Script Decision Tree

Penjelasan *script Decision Tree* diatas:

Script tersebut terdiri dari beberapa langkah:

1. *Import library* yang dibutuhkan, yaitu ***DecissionTreeClassifier*** dan ***train_test_split*** dari *scikit-learn.*
2. Persiapkan data fitur dan kelas yang akan digunakan untuk melatih model.
3. *Split* data menjadi data training dan data *testing* dengan ukuran data *testing* sebesar 30%.
4. Buat model *Decision Tree*.
5. Latih model dengan data *training*.
6. Uji model dengan data *testing* dan dapatkan skor akurasi.
7. Cetak akurasi model.
8. Prediksi kelas suatu objek baru.
9. Cetak kelas prediksi.

Setelah melakukan semua langkah tersebut, Anda akan mendapatkan model *Decision Tree* yang telah dilatih dan diuji serta dapat digunakan untuk melakukan prediksi kelas suatu objek baru.

1.2.6 Random Forest

Algoritma *Random Forest* dapat mencapai akurasi tertinggi sebelum dan sesudah pemilihan fitur dan peningkatan bangunan secara dramatis. Hasil percobaan menunjukkan bahwa menggunakan pendekatan seleksi dengan mesin algoritma pembelajaran dapat meningkatkan efektivitas model klasifikasi untuk deteksi *phishing* tanpa mengurangi kinerja mereka. [9]. Dalam *random forest*, banyak pohon ditumbuhkan sehingga terbentuk hutan (*forest*), kemudian analisis dilakukan pada kumpulan pohon tersebut.

Berikut ini adalah contoh *script* sederhana untuk melakukan random forest menggunakan *Python*:

```python
# Import library yang dibutuhkan
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

# Persiapkan data
X = [[2, 3], [4, 5], [6, 7], [8, 9]] # data fitur
y = [0, 0, 1, 1] # data kelas

# Split data menjadi data training dan data testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# Buat model random forest dengan jumlah pohon sebanyak 10
model = RandomForestClassifier(n_estimators=10)

# Latih model dengan data training
model.fit(X_train, y_train)

# Uji model dengan data testing
score = model.score(X_test, y_test)

# Cetak akurasi model
print("Akurasi model:", score)

# Prediksi kelas suatu objek baru
prediction = model.predict([[5, 6]])
print("Kelas prediksi:", prediction)
```

Gambar 1.2.6 Contoh Script Random Forest

Penjelasan *script Random Forest* diatas:

Script tersebut terdiri dari beberapa langkah:

1. *Import library* yang dibutuhkan, yaitu ***RandomForestClassifier*** dan ***train_test_split*** dari *scikit-learn.*
2. Persiapkan data fitur dan kelas yang akan digunakan untuk melatih model.
3. *Split* data menjadi data training dan data *testing* dengan ukuran data *testing* sebesar 30%.
4. Buat model *Random Forest*.
5. Latih model dengan data *training*.
6. Uji model dengan data *testing* dan dapatkan skor akurasi.
7. Cetak akurasi model.
8. Prediksi kelas suatu objek baru.
9. Cetak kelas prediksi.

Setelah melakukan semua langkah tersebut, Anda akan mendapatkan model *Random Forest* yang telah dilatih dan diuji serta dapat digunakan untuk melakukan prediksi kelas suatu objek baru.

1.2.7 Gradient Boosting

Tujuan utama dari *Boosting* adalah menggabungkan semua train yang lemah bersama-sama untuk membentuk model yang kuat.

- *Gradient boosting* adalah suatu teknik yang sangat kuat untuk mengembangkan model prediktif. Ini berlaku untuk beberapa fungsi risiko dan mengoptimalkan akurasi prediksi model. Ini juga menyelesaikan masalah multikolinearitas di mana korelasi antar variabel prediktor tinggi.
- *Gradient Boosting* adalah algoritma pembelajaran mesin ansambel dan biasanya digunakan untuk menyelesaikan klasifikasi dan regresi [10].

Berikut ini adalah contoh *script* sederhana untuk melakukan *gradient boosting* menggunakan *Python*:

```python
# Import library yang dibutuhkan
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split

# Persiapkan data
X = [[2, 3], [4, 5], [6, 7], [8, 9]] # data fitur
y = [0, 0, 1, 1] # data kelas

# Split data menjadi data training dan data testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# Buat model gradient boosting dengan jumlah pohon sebanyak 10
model = GradientBoostingClassifier(n_estimators=10)

# Latih model dengan data training
model.fit(X_train, y_train)

# Uji model dengan data testing
score = model.score(X_test, y_test)

# Cetak akurasi model
print("Akurasi model:", score)

# Prediksi kelas suatu objek baru
prediction = model.predict([[5, 6]])
print("Kelas prediksi:", prediction)
```

Gambar 1.2.7 Contoh Script Gradient Boosting

Penjelasan *script Gradient Boosting* diatas:

Script tersebut terdiri dari beberapa langkah:

1. *Import library* yang dibutuhkan, yaitu ***GradientBoostingClassifier*** dan ***train_test_split*** dari *scikit-learn.*
2. Persiapkan data fitur dan kelas yang akan digunakan untuk melatih model.
3. *Split* data menjadi data training dan data *testing* dengan ukuran data *testing* sebesar 30%.
4. Buat model *Gradient Boosting*.
5. Latih model dengan data *training*.
6. Uji model dengan data *testing* dan dapatkan skor akurasi.
7. Cetak akurasi model.
8. Prediksi kelas suatu objek baru.
9. Cetak kelas prediksi.

Setelah melakukan semua langkah tersebut, Anda akan mendapatkan model *Gradient Boosting* yang telah dilatih dan diuji serta dapat digunakan untuk melakukan prediksi kelas suatu objek baru.

1.2.8 Catboost

Algoritma *CatBoost* membutuhkan lebih banyak waktu untuk pelatihan dan menguji kumpulan data yang menggunakan lebih banyak komputasi sumber daya. Efektivitas algoritma *CatBoost* memiliki telah ditunjukkan melalui kinerjanya yang lebih tinggi algoritma yang bersaing. Dalam hal pekerjaan masa depan, *Apache* Kerangka kerja Spark dapat digunakan untuk meningkatkan *Sickit-learn library* yang disebut *Sk-dist. Sk-dist* telah mengatasi batasan tersebut perpustakaan *Sickit-learn* seperti memakan waktu dan lagging pelatihan model [11].

Peneliti dapat mengatur pengaturan untuk jumlah maksimum iterasi yang digunakan *CatBoost*, kedalaman maksimum Pohon Keputusan konstituen, dan jumlah maksimum kombinasi fitur kategorikal untuk meningkatkan performa model. Nilai-nilai itu peneliti menggunakan *hyper-parameter* ini dapat menjelaskan perbedaan dalam kinerja *CatBoost*.

Berikut ini adalah contoh script sederhana untuk melakukan *CatBoost* menggunakan *Python*:

```python
# Import library yang dibutuhkan
from catboost import CatBoostClassifier
from sklearn.model_selection import train_test_split

# Persiapkan data
X = [[2, 3], [4, 5], [6, 7], [8, 9]] # data fitur
y = [0, 0, 1, 1] # data kelas

# Split data menjadi data training dan data testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# Buat model CatBoost
model = CatBoostClassifier()

# Latih model dengan data training
model.fit(X_train, y_train)

# Uji model dengan data testing
score = model.score(X_test, y_test)

# Cetak akurasi model
print("Akurasi model:", score)

# Prediksi kelas suatu objek baru
prediction = model.predict([[5, 6]])
print("Kelas prediksi:", prediction)
```

Gambar 1.2.8 Contoh Script Catboost

Penjelasan *script Catboost* diatas:

Script tersebut terdiri dari beberapa langkah:

1. *Import library* yang dibutuhkan, yaitu ***CatBoostClassifier*** dan ***train_test_split*** dari *scikit-learn.*
2. Persiapkan data fitur dan kelas yang akan digunakan untuk melatih model.
3. *Split* data menjadi data training dan data *testing* dengan ukuran data *testing* sebesar 30%.
4. Buat model *Cat Boost*.
5. Latih model dengan data *training*.
6. Uji model dengan data *testing* dan dapatkan skor akurasi.
7. Cetak akurasi model.
8. Prediksi kelas suatu objek baru.
9. Cetak kelas prediksi.

Setelah melakukan semua langkah tersebut, Anda akan mendapatkan model *Cat Boost* yang telah dilatih dan diuji serta dapat digunakan untuk melakukan prediksi kelas suatu objek baru.

1.3 Python

Python adalah bahasa yang dirancang dengan baik yang dapat digunakan secara nyata pemrograman dunia. *Python* adalah tingkat yang sangat tinggi, dinamis, berorientasi objek, bahasa pemrograman tujuan umum yang menggunakan juru bahasa dan dapat digunakan dalam domain yang luas aplikasi. *Python* dirancang agar mudah dimengerti dan gunakan. *Python* disebut sebagai sangat *user-friendly* dan bahasa yang ramah pemula belakangan ini. *Python* punya memperoleh popularitas karena menjadi bahasa yang ramah bagi pemula, dan telah menggantikan Java sebagai pengantar paling popular bahasa [12].