

Student Dropout Prediction

Kelompok Fitting 24/7

Agustinus Angelo Christian Fernando - 21/473804/TK/52235 - Programmer + Slide Maker

Aufa Nasywa Rahman - 21/475255/TK/52454 - Programmer + Slide Maker

Ahmad Zaki Akmal - 21/480179/TK/52981 - Programmer + Editor

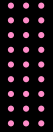


Problem

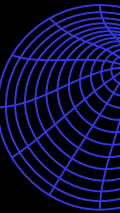


- Sumber dataset:
<https://archive-beta.ics.uci.edu/dataset/697/predict+students+dropout+and+academic+success>
- Dataset ini berisi mengenai data mahasiswa yang terdiri dari berbagai program studi.
- Feature pada dataset ini tentang karakteristik, data diri, rekam jejak akademik, dan faktor sosial-ekonomi mahasiswa.
- Variabel yang diprediksi adalah apakah mahasiswa tersebut lulus atau tidak.
- Data ini bermanfaat untuk tenaga pengajar dalam mencari strategi yang lebih baik agar mahasiswa bisa lulus
- Bermanfaat pula bagi mahasiswa agar bisa mengevaluasi performa akademiknya

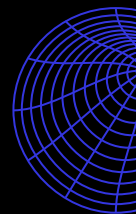
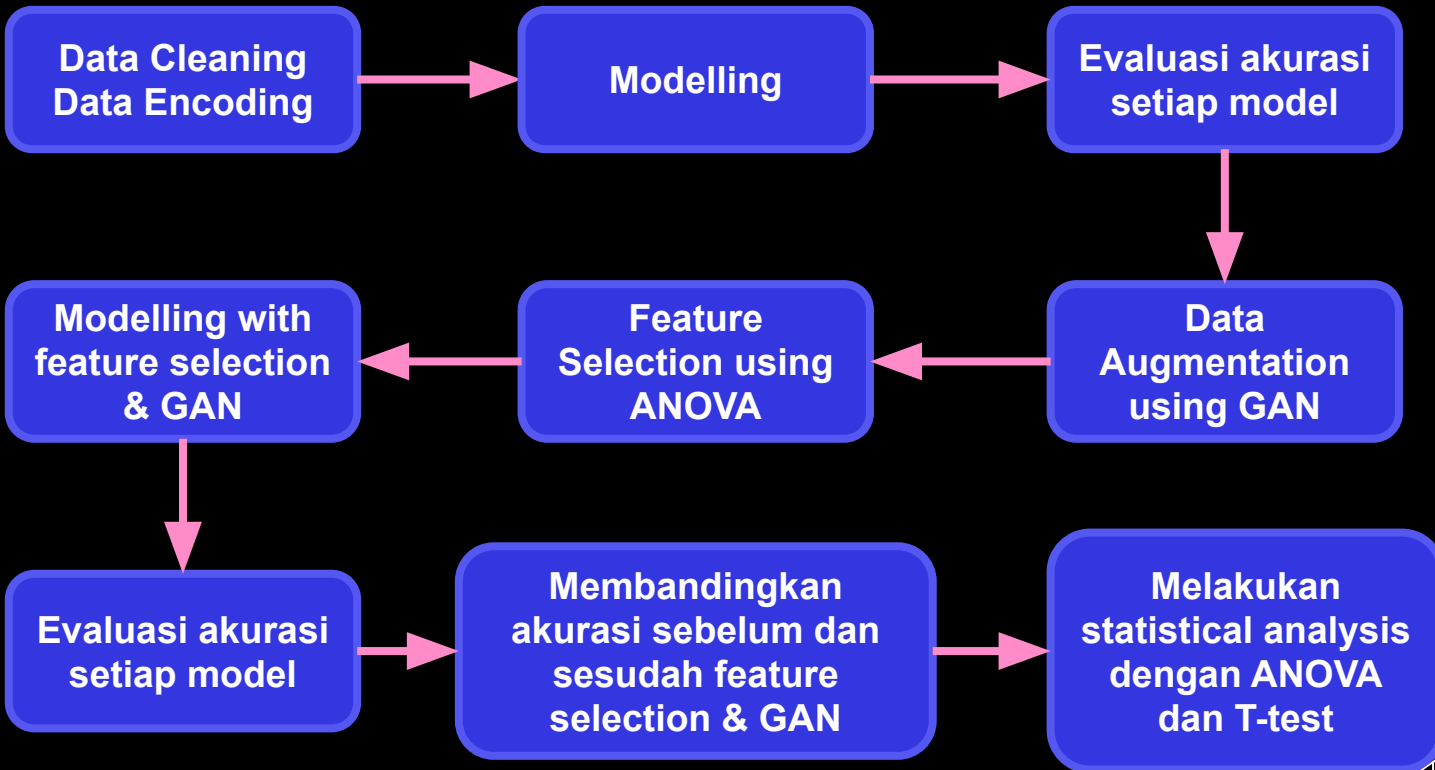
Objective



- Berdasarkan feature-feature tersebut, kita akan memprediksi apakah seorang siswa lulus (graduate) atau tidak lulus kuliah (dropout).
- Prediksi akan dilakukan menggunakan model Logistic Regression, Naive Bayes, Perceptron, dan Support Vector Machine.
- Semua model dibuat dari awal **tanpa** menggunakan library ML apapun
- Kami akan membandingkan hasil prediksi awal terhadap hasil prediksi setelah dilakukan data augmentation menggunakan GAN dan feature selection menggunakan ANOVA F-value.
- Kami juga memvisualisasikan komparasi akurasi dari setiap model.
- Kita akan menjadi tahu model apa yang terbaik
- Kita juga akan mengetahui apa efek dari data augmentation + feature selection terhadap akurasi model.



Metode - Flow Pengerjaan



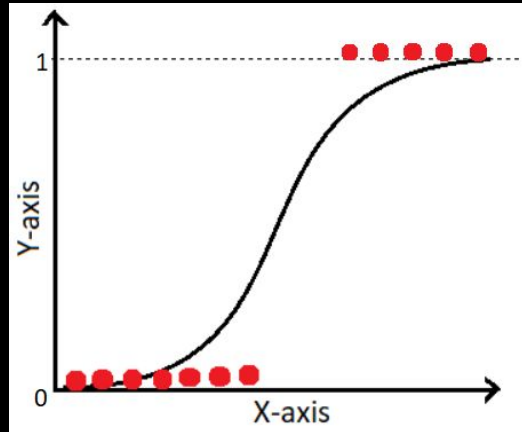
Metode - Overview

- Model yang kami gunakan (Penjelasan di slide berikutnya):
 - Logistic Regression
 - Naive Bayes
 - Perceptron
 - Support Vector Machine
- Data augmentation yang kami gunakan: GAN
- Feature selection yang kami gunakan: ANOVA F-value
- Matriks evaluasi menggunakan: Akurasi
- Validasi menggunakan: K-fold validation
- Statistical analysis menggunakan: ANOVA dan T-test



Metode - Logistic Regression

- Alasan menggunakan logistic regression:
 - Logistic regression adalah klasifikasi biner yang paling dasar dan mudah untuk diimplementasikan.
- Metode ini menggunakan fungsi sigmoid untuk memilih apakah suatu output jatuh pada angka 0 atau angka 1.



Metode - Logistic Regression

```
class ModelLogisticRegression:

    def __init__(self, learning_rate=0.001, n_iters=1000):
        self.lr = learning_rate
        self.n_iters = n_iters
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        n_samples, n_features = X.shape

        # Inisialisasi parameter
        self.weights = np.zeros(n_features)
        self.bias = 0
```


Metode - Logistic Regression

```
# Optimisasi dengan gradient descent
for _ in range(self.n_iters):
    # Memprediksikan output (y) dengan linear kombinasi weight dan x, ditambah dengan bias
    linear_model = np.dot(X, self.weights) + self.bias
    # Menerapkan fungsi sigmoid
    y_predicted = self._sigmoid(linear_model)

    # Komputasi gradient
    dw = (1 / n_samples) * np.dot(X.T, (y_predicted - y)) # Derivative w.r.t weights
    db = (1 / n_samples) * np.sum(y_predicted - y) # Derivative w.r.t bias
    # Update parameter weight dan bias untuk mendapat nilai optimal
    self.weights -= self.lr * dw
    self.bias -= self.lr * db

def predict(self, X):
    linear_model = np.dot(X, self.weights) + self.bias
    y_predicted = self._sigmoid(linear_model)
    y_predicted_cls = [1 if i > 0.5 else 0 for i in y_predicted]
    return np.array(y_predicted_cls)

def _sigmoid(self, x):
    return 1 / (1 + np.exp(-x))
```


Metode - Naive Bayes

- Naive Bayes digunakan karena metode ini mampu menangani sebuah dataset yang tidak lengkap.
- Metode ini mampu “mengabaikan” nilai yang hilang.
- Rumus Naive Bayes

$$P(c | x) = \frac{P(x | c) P(c)}{P(x)}$$

Diagram illustrating the Naive Bayes formula with labels:

- $P(c | x)$ is labeled **Posterior Probability**.
- $P(x | c)$ is labeled **Likelihood**.
- $P(c)$ is labeled **Class Prior Probability**.
- $P(x)$ is labeled **Predictor Prior Probability**.

Metode - Naive Bayes

```
class ModelNaiveBayes:
```

```
def fit(self, X, y):
    n_samples, n_features = X.shape
    self._classes = np.unique(y)
    n_classes = len(self._classes)

    # menghitung mean, var, dan prior untuk masing-masing class
    self._mean = np.zeros((n_classes, n_features), dtype=np.float64)
    self._var = np.zeros((n_classes, n_features), dtype=np.float64)
    self._priors = np.zeros(n_classes, dtype=np.float64)

    for idx, c in enumerate(self._classes):
        X_c = X[y == c]
        self._mean[idx, :] = X_c.mean(axis=0)
        self._var[idx, :] = X_c.var(axis=0)
        self._priors[idx] = X_c.shape[0] / float(n_samples)
```

```
def predict(self, X):
    y_pred = [self._predict(x) for x in X]
    return np.array(y_pred)
```

Menghitung `n_samples` dan `n_features`, mengidentifikasi class yang unik di `y`, inisiasi array untuk menyimpan mean, variance, dan prior, dan melakukan iterasi tiap class untuk mendapatkan nilai mean, variance, dan prior

Digunakan untuk membuat prediksi berdasarkan data baru dengan input data `X`

Metode - Naive Bayes

```
def _predict(self, x):  
    posteriors = []  
  
    # menghitung posterior probability  
    for idx, c in enumerate(self._classes):  
        prior = np.log(self._priors[idx])  
        posterior = np.sum(np.log(self._pdf(idx, x)))  
        posterior = posterior + prior  
        posteriors.append(posterior)  
  
    # return class dengan posterior probability paling besar  
    return self._classes[np.argmax(posteriors)]
```

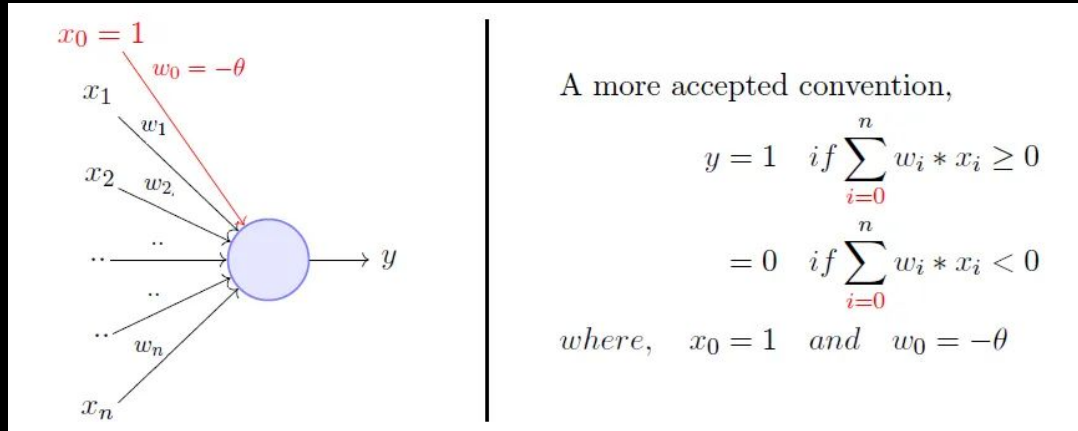
```
def _pdf(self, class_idx, x):  
    mean = self._mean[class_idx]  
    var = self._var[class_idx]  
    numerator = np.exp(-((x - mean) ** 2) / (2 * var))  
    denominator = np.sqrt(2 * np.pi * var)  
    return numerator / denominator
```

_predict adalah helpers function untuk fungsi predict. Fungsi ini akan menginisiasi array kosong untuk menyimpan posterior probability dan menghitung posterior probability dengan rumus Naive Bayes

Menghitung Probability Density Function. Dilakukan dengan mengambil mean dan variance kemudian menghitung numerator dari Gaussian PDF formula menggunakan input sample, mean, dan variance. Dilanjutkan dengan menghitung denominatornya dan diakhiri dengan return hasilnya

Metode - Perceptron

- Perceptron digunakan karena metode ini dapat menangani masalah yang memiliki output biner, yaitu 1 dan 0. Dalam kasus ini, perceptron digunakan untuk memprediksi apakah siswa dropout atau tidak dropout



Metode - Perceptron

- Metode ini terdiri dari berbagai tahapan:
 - 1) Dimulai dengan perhitungan Weighted Sum yang didapatkan dari dot product antara input features dengan weight vector.
 - 2) Proses dilanjutkan dengan melewati weighted sum ke activation function.
 - 3) Proses diakhiri dengan membandingkan predicted output dengan label dari training data, jika prediksi salah, dilakukan update pada weight



Metode - Perceptron

```
def unit_step_func(x):  
    return np.where(x > 0 , 1, 0)
```

Fungsi untuk unit step +
function, dimana akan
return nilai 1 jika $x > 0$ dan
0 jika sebaliknya

```
class ModelPerceptron:
```

```
def __init__(self, learning_rate=0.01, n_iters=1000):  
    self.lr = learning_rate  
    self.n_iters = n_iters  
    self.activation_func = unit_step_func  
    self.weights = None  
    self.bias = None
```

Attribute untuk kalkulasi
menggunakan perceptron,
terdiri dari learning rate,
iterasi, activation function,
weights, dan bias

```
def fit(self, X, y):  
    n_samples, n_features = X.shape  
  
    # inisiasi parameternya  
    self.weights = np.zeros(n_features)  
    self.bias = 0  
  
    y_ = np.where(y > 0 , 1, 0)
```

Fungsi untuk train model
perceptron-nya. Dimulai
dari inisiasi weights dan
bias menjadi zero dan set
y ke binary (0 dan 1)

Metode - Perceptron

```
# menghitung dan evaluasi weights
for _ in range(self.n_iters):
    for idx, x_i in enumerate(X):
        linear_output = np.dot(x_i, self.weights) + self.bias
        y_predicted = self.activation_func(linear_output)

        # update weight
        update = self.lr * (y_[idx] - y_predicted)
        self.weights += update * x_i
        self.bias += update
```

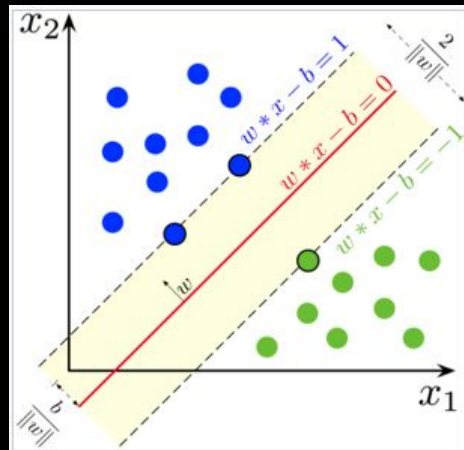
```
def predict(self, X):
    linear_output = np.dot(X, self.weights) + self.bias
    y_predicted = self.activation_func(linear_output)
    return y_predicted
```

Kalkulasi dengan melakukan iterasi tiap input x_i dengan idx yang bersesuaian dengan menghitung dot product x_i dengan weights dan menambahkan bias. Kemudian menghitung output berdasar activation function dan melakukan update ke weights dan bias

Fungsi untuk membuat prediksi berdasar data baru dengan input data X

Metode - SVM

- SVM adalah metode klasifikasi biner yang membagi 2 kategori dengan cara mengambil garis (dalam 2 dimensi) atau bidang (dalam 3 dimensi) yang memisahkan semua data menjadi 2 kelompok.
- Metode ini cocok digunakan karena mampu memisahkan semua siswa menjadi 2 kelompok yaitu, siswa yang drop out dan tidak



Metode - SVM

```
class ModelSVM():

    # Inisialisasi parameter
    def __init__(self, learning_rate, no_of_iterations, lambda_parameter):
        self.learning_rate = learning_rate
        self.no_of_iterations = no_of_iterations
        self.lambda_parameter = lambda_parameter

    # Menyesuaikan data untuk SVM Classifier
    def fit(self, X, Y):

        # m --> banyaknya data points --> banyaknya baris (rows)
        # n --> banyaknya input features --> banyaknya kolom (columns)
        self.m, self.n = X.shape

        # Inisialisasi nilai-nilai weight dan bias
        self.w = np.zeros(self.n)
        self.b = 0
        self.X = X
        self.Y = Y
```

Metode - SVM

```
# Optimisasi dengan cara implementasi algoritma gradient descent
for i in range(self.no_of_iterations):
    self.update_weights()

# Function untuk meng-update nilai-nilai weights dan bias
def update_weights(self):
    # Pelabelan klasifikasi pada output
    y_label = np.where(self.Y <= 0, -1, 1)

    # Menghitung gradient ( dw, db)
    for index, x_i in enumerate(self.X):
        condition = y_label[index] * (np.dot(x_i, self.w) - self.b) >= 1

        if (condition == True):
            dw = 2 * self.lambda_parameter * self.w
            db = 0

        else:
            dw = 2 * self.lambda_parameter * self.w - np.dot(x_i, y_label[index])
            db = y_label[index]

    self.w = self.w - self.learning_rate * dw
    self.b = self.b - self.learning_rate * db
```

Metode - SVM

```
# Prediksikan label untuk input yang diberikan
def predict(self, X):

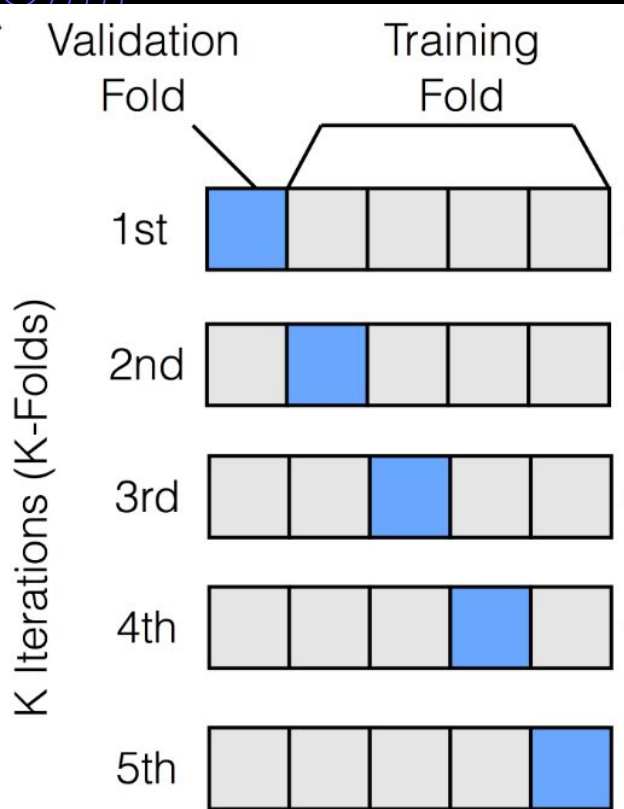
    output = np.dot(X, self.w) - self.b

    predicted_labels = np.sign(output)

    y_hat = np.where(predicted_labels <= -1, 0, 1)

    return y_hat
```

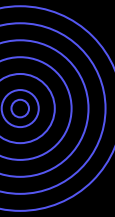
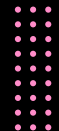


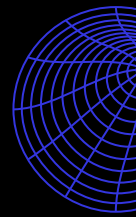

Metode - Validasi with K-fold



- K-fold validation adalah teknik untuk meminimalisir overfitting dengan cara membagi data ke k-subset dengan ukuran yang sama.
- Kita menggunakan $k = 5$
- Data dilatih dengan 4 subset dan diuji dengan 1 subset tersisa.
- Dengan begitu, maka semua data memiliki kesempatan untuk digunakan sebagai bahan pelatihan dan bahan validasi
- Maka, model tidak hanya bagus untuk data tertentu saja, tapi juga bagus untuk data-data yang belum diketahui sebelumnya



Metode - GAN Data Augmentation

- 
- 
- 
- Kami menggunakan Keras untuk melakukan GAN Data Augmentation.
 - Ada 2 proses, yaitu generator dan discriminator.
 - **Pada tahap generator**, sistem akan menghasilkan data sintetis yang dibuat semirip mungkin dengan data asli, kemudian akan diperiksa oleh diskriminator.
 - **Pada tahap diskriminator**, sistem akan membandingkan dengan data asli.
 - Jika sistem masih bisa membedakan antara data asli dan palsu, maka sistem akan mengulangi proses generator dan diskriminator sampai sistem bisa tertipu dengan data palsu.
 - Setelah itu, kita berhasil mengenerate 1000 data baru yang menyerupai data awal
- 
- 
- 

Metode - GAN Data Augmentation

- Selanjutnya, kita meng-concat data yang digenerate oleh GAN ke dataset awal agar bisa kita lanjutkan ke tahap feature selection dan modelling.

```
# Menyimpan data yang digenerate menggunakan GAN agar tidak perlu run ulang cell di atas, mengingat waktu run yang cukup lama
df_synthetic.to_csv("generated_by_GAN.csv", index=False)
```

```
# Menggabungkan df_gan ke df menggunakan concat
df = pd.concat([df, df_gan])
df = df.reset_index(drop=True)
```

df

| | Marital status | Application mode | Application order | Course | Daytime/evening attendance\% | Previous qualification | Previous qualification (grade) | Nationality | Mother's qualification | Father's qualification | ... |
|------|----------------|------------------|-------------------|-------------|------------------------------|------------------------|--------------------------------|-------------|------------------------|------------------------|-----|
| 0 | 1.000000 | 17.000000 | 5.000000 | 171.000000 | 1.0 | 1.000000 | 122.000000 | 1.000000 | 19.000000 | 12.000000 | ... |
| 1 | 1.000000 | 15.000000 | 1.000000 | 9254.000000 | 1.0 | 1.000000 | 160.000000 | 1.000000 | 1.000000 | 3.000000 | ... |
| 2 | 1.000000 | 1.000000 | 5.000000 | 9070.000000 | 1.0 | 1.000000 | 122.000000 | 1.000000 | 37.000000 | 37.000000 | ... |
| 3 | 1.000000 | 17.000000 | 2.000000 | 9773.000000 | 1.0 | 1.000000 | 122.000000 | 1.000000 | 38.000000 | 37.000000 | ... |
| 4 | 2.000000 | 39.000000 | 1.000000 | 8014.000000 | 0.0 | 1.000000 | 100.000000 | 1.000000 | 37.000000 | 38.000000 | ... |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 4625 | 1.231963 | 41.489559 | 0.963845 | 9510.637298 | 0.0 | 1.351659 | 123.747029 | 1.000299 | 37.023604 | 39.325381 | ... |
| 4626 | 1.000365 | 1.000000 | 2.100440 | 9798.324351 | 1.0 | 1.000006 | 160.448055 | 1.013908 | 1.448802 | 37.698158 | ... |
| 4627 | 1.007148 | 1.000274 | 0.966551 | 9345.020090 | 1.0 | 1.000175 | 142.221854 | 1.029042 | 2.446310 | 19.289242 | ... |
| 4628 | 1.001638 | 19.534708 | 3.072585 | 9291.957341 | 1.0 | 1.091216 | 136.159591 | 1.021562 | 1.357073 | 2.458007 | ... |
| 4629 | 1.000075 | 17.851196 | 2.959118 | 9350.612749 | 1.0 | 1.000529 | 144.159477 | 1.014072 | 1.417123 | 1.377772 | ... |

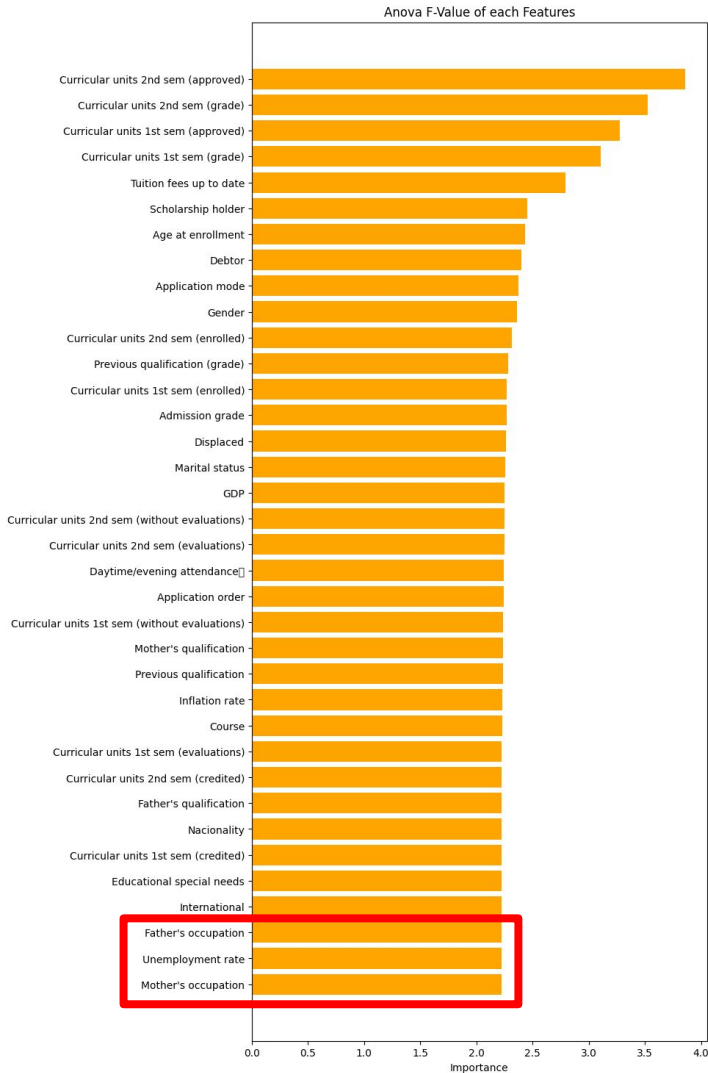
4630 rows x 37 columns

Metode - Feature Selection ANOVA

- Kami menggunakan ANOVA untuk melakukan feature selection
- Tujuannya untuk mencari F-value dari setiap feature.
- Code feature selection ANOVA ini kami buat dari awal tanpa library ML apapun

| Source of Variation | Sum of Square Jumlah Kuadrat | Degrees of Freedom Derajat Bebas | Mean Squares Kuadrat Rata-Rata | F Value F hitung |
|----------------------------|--|-------------------------------------|-----------------------------------|-----------------------|
| Between groups Kelompok | $SSB = \sum n_i(\bar{x}_i - \bar{x})^2$ JKK | $df_1 = k - 1$ | $MSB = \frac{SSB}{df_1}$ KRK | $f = \frac{MSB}{MSE}$ |
| Error Galat | $SSE = \sum \sum (x - \bar{x}_i)^2$ JKG | $df_2 = N - k$ | $MSE = \frac{SSE}{df_2}$ KRG | |
| Total | $SST = SSB + SSE$ JKT | $df_3 = N - 1$ | | |

```
def anova_f_value(X, y):  
    # Number of distinct classes  
    classes = np.unique(y)  
  
    # Overall mean  
    overall_mean = np.mean(X, axis=0)  
  
    # Between class variability  
    S_B = 0  
    for c in classes:  
        class_data = X[y == c]  
        class_mean = np.mean(class_data, axis=0)  
        S_B += len(class_data) * np.sum((class_mean - overall_mean)**2)  
  
    # Within class variability  
    S_W = 0  
    for c in classes:  
        class_data = X[y == c]  
        class_mean = np.mean(class_data, axis=0)  
        S_W += np.sum((class_data - class_mean)**2)  
  
    # Compute ANOVA F-value  
    F = S_B / S_W  
  
    return F
```



- Dari perhitungan ANOVA, kami menampilkan visualisasi F-value dari setiap feature
- Kita akan membuang feature yang F-valuenya rendah.
- **Mengapa?** Karena F-value rendah menandakan bahwa feature tersebut tidak terlalu berpengaruh terhadap target yang kita prediksi.
- Tapi, tampak bahwa F-value dari setiap feature sangat mirip
- Artinya, setiap feature memiliki pengaruh penting untuk proses prediksi
- Maka, kita hanya akan meng-drop 3 feature dengan F-value terkecil di plot tersebut

Result - Overview

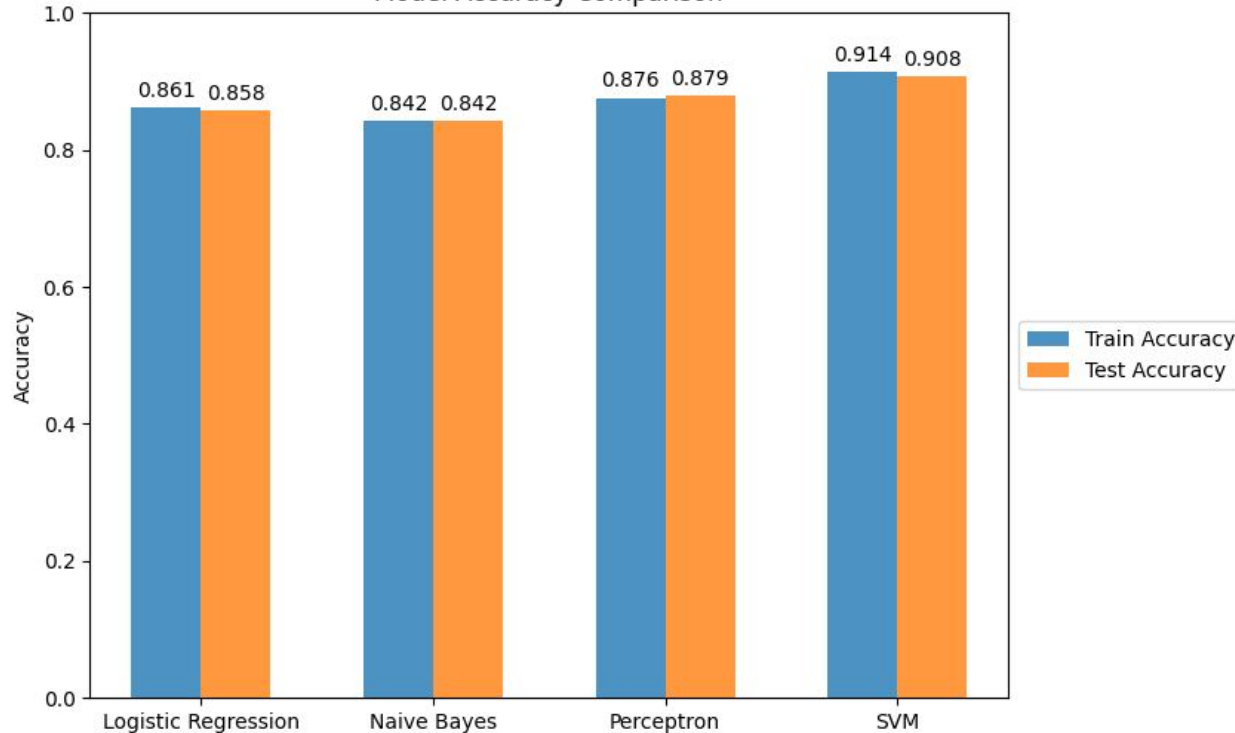
- Karena ini adalah data classification, maka kami menggunakan metrik akurasi untuk mengevaluasi model.
- Berikut adalah rumusnya:

$$\text{accuracy} = \frac{\text{jumlah baris hasil prediksi yang sama dengan value target aslinya}}{\text{jumlah baris target}}$$

- Kami memvisualisasikan akurasi hasil prediksi dengan barplot agar lebih mudah dipahami
- Kemudian, hasil prediksi kita validasi menggunakan K-fold sebanyak 5 kali.
- Yuk cek slide berikutnya untuk mengetahui hasil akurasi setiap model...

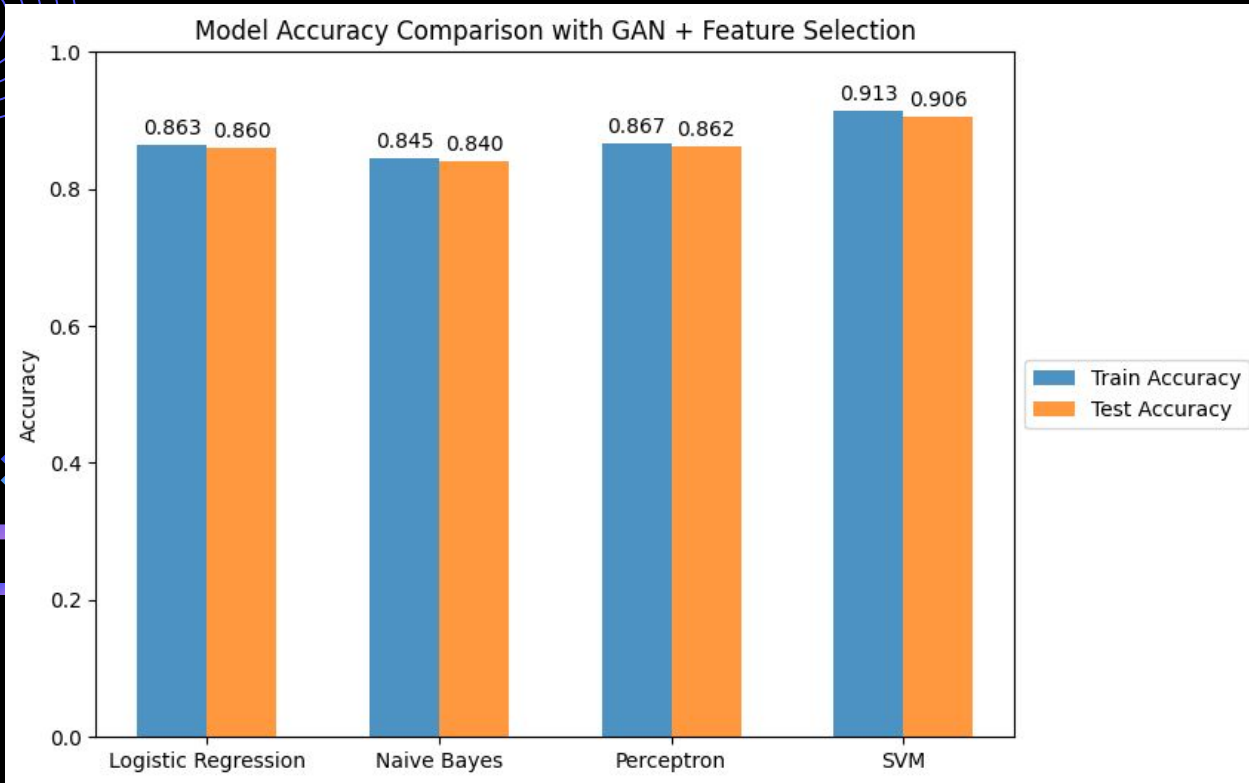
Result - Tanpa Feature Selection & GAN

Model Accuracy Comparison

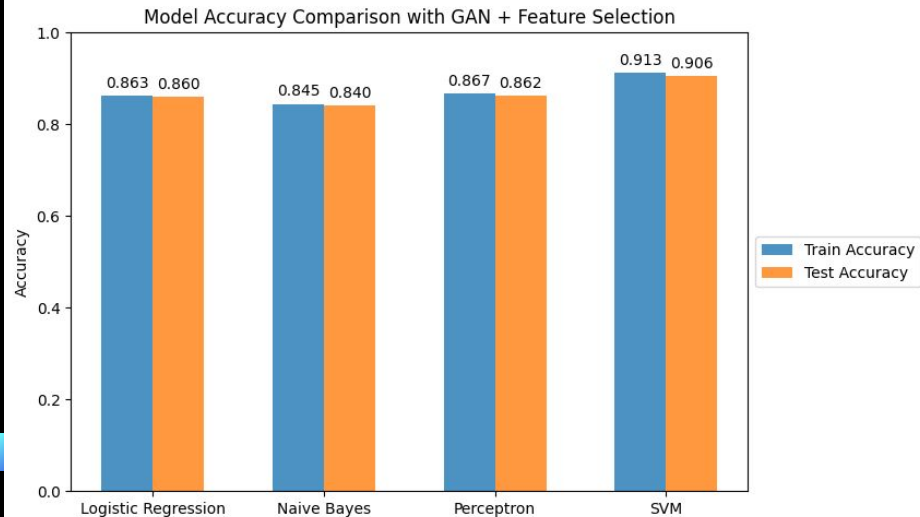
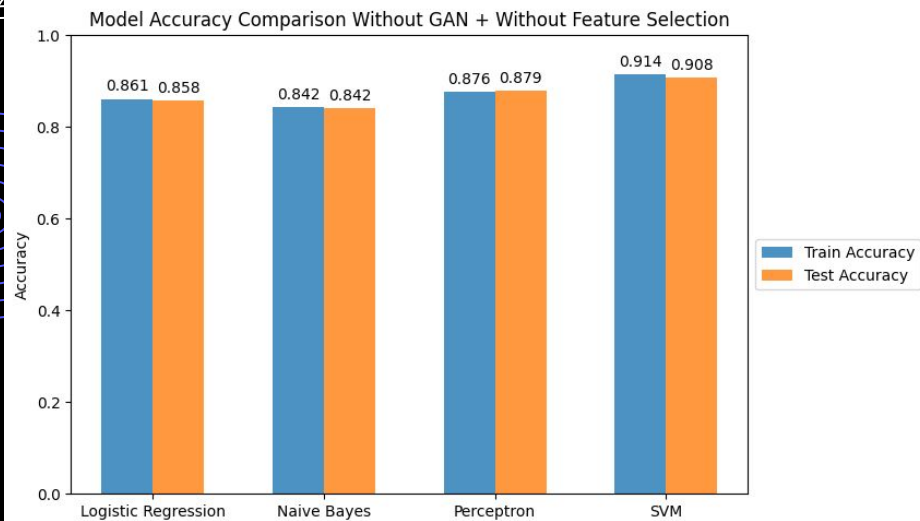


- No overfitting
- No underfitting
- Terbaik: SVM
- Terendah: Naive Bayes
- **Mengapa SVM bisa menjadi yang terbaik?**
- Tenang, nanti akan kami jelaskan

Result - Dengan Feature Selection & GAN



- No overfitting
- No underfitting
- Terbaik: SVM
- Terendah: Naive Bayes



- Apa yang terjadi setelah melakukan data augmentation dengan GAN dan melakukan ANOVA feature selection?
- Akurasi Logistic Regression meningkat sebesar 0.002
- Akurasi Naive Bayes pada data train meningkat 0.003 dan pada data test menurun 0.002
- Akurasi Perceptron berkurang 0.009
- Akurasi SVM hanya berkurang 0.001
- Tampak perubahannya hanya sedikit.
- **Bagaimana dengan T-test nya?**

Statistikal Analysis - T-test

Hasil T-test Logistic Regression:
p-value: 0.5146053879988526

Hasil T-test Naive Bayes:
p-value: 0.4929033879685083

Hasil T-test Perceptron:
p-value: 0.41980927804975443

Hasil T-test SVM:
p-value: 0.4847373184401299

- Kita menggunakan confidence interval 95%
- Maka, $\alpha = 0.05$
- Jika $p\text{-value} < \alpha$, maka tidak ada perbedaan signifikan
- Tampak bahwa semua hasil p-value dari 4 model tersebut menunjukkan bahwa tidak ada perbedaan signifikan antara sebelum dan sesudah GAN + Feature Selection.
- **Mengapa? Yuk cek slide berikutnya...**

Result - Analisis

- Alasan perubahan sebelum dan sesudah feature selection + GAN hanya sedikit:
 - Jumlah data awal sudah cukup banyak (3500++ data)
 - Semua feature memiliki peran yang penting
- Jika jumlah data awal sudah cukup banyak, maka penggunaan GAN menjadi tidak terlalu berefek.
- Jika semua feature memiliki peran penting, maka feature selection justru dapat membuat akurasi menurun karena model kehilangan informasi penting dari feature-feature yang dihilangkan tersebut.



Statistical Analysis - ANOVA

- Kami juga melakukan statistical analysis menggunakan ANOVA antara akurasi sebelum dan sesudah menggunakan GAN + feature selection.
- H_0 : Tidak ada perbedaan signifikan antara penggunaan GAN dengan tidak.
 H_1 : Ada perbedaan signifikan antara penggunaan GAN dengan tidak.
- Hasil ANOVA adalah p-value dengan nilai 0.123
- Dengan tingkat keyakinan 95% ($\alpha = 5\%$), maka p-value $> \alpha$.
- Karena p-value $> \alpha$, maka keputusan yang diambil adalah fail to reject null hypothesis, berarti tidak ada cukup bukti untuk mendukung bahwa penggunaan GAN mengakibatkan perubahan yang signifikan.
- Alasannya sama seperti yang dijelaskan di slide sebelumnya

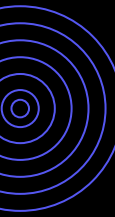
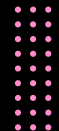





Hasil ANOVA:

F-value: 0.1033986863738109

p-value: 0.12365457019052761



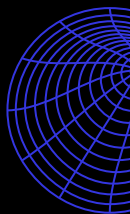
Result - Komparasi antar model

- 
- 
- 
- Tadi kita sudah tahu bahwa urutan dari akurasi tertinggi adalah:
SVM - Perceptron - Logistic Regression - Naive Bayes
 - **Mengapa SVM dan Perceptron bisa menghasilkan akurasi tinggi?**
 - Hal tersebut karena kita tahu bahwa SVM dan Perceptron mampu menangani dataset yang memiliki banyak feature secara lebih baik dibandingkan model lain.
 - Naive Bayes menghasilkan akurasi terendah karena Naive Bayes menganggap semua featurenya benar-benar independen.
 - Hal tersebut cukup rawan karena dataset yang kita gunakan memiliki cukup banyak feature.
- 
- 
- 
- 

Kesimpulan



- Terbukti bahwa 4 model yang dibuat dari scratch, yaitu Logistic Regression, Naive Bayes, Perceptron, dan Support Vector Machine mampu menghasilkan prediksi lulus atau tidaknya mahasiswa dengan akurat. Hal tersebut sudah dibuktikan dengan evaluasi yang kami lakukan.
- Akurasi tertinggi didapatkan oleh model SVM sebesar 0.91 dan yang terendah oleh model Naive Bayes sebesar 0.86
- Statistical analysis menggunakan ANOVA dan T-test menunjukkan bahwa tidak ada perbedaan signifikan antara sebelum dan sesudah data augmentation + feature selection karena berbagai alasan yang sudah kami jelaskan tadi.



Sekian dan Terima Kasih

