

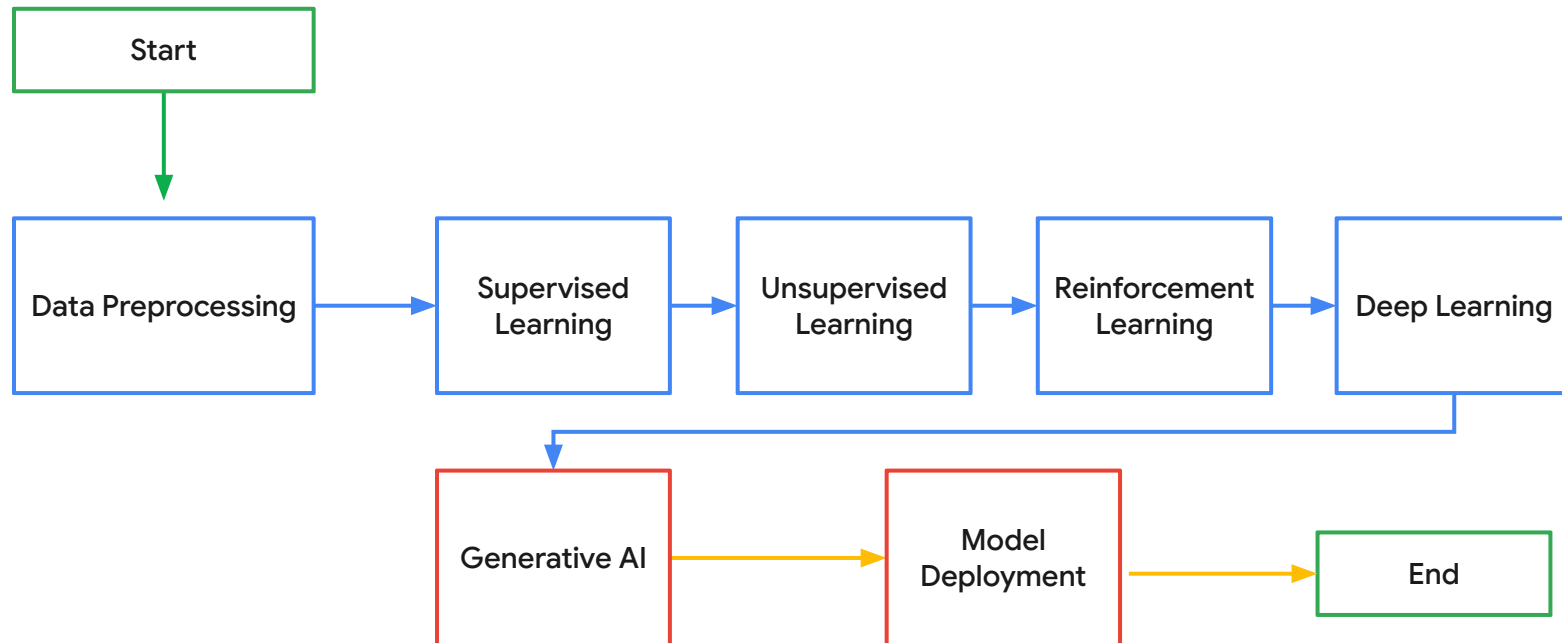


Exploring Unsupervised Learning and Supervised Technique



Ahmad Daffa Abiyyu
Academy Team, GDGoC
UNAIR

The Curriculum



What are we going to learn today:

1. Basic Supervised Models (part 2)
2. Unsupervised Models



Supervised Learning: Part 2

```
lookup.KeyValue  
f.constant(['em  
=tf.constant([G  
lookup.StaticV  
_buckets=5)
```

```
child: Column(  
  crossAxisAlignment: CrossAxisAlignment.  
  children: [  
    /*2*/  
    Conta  
    pad  
    chi  
    '  
    s  
    )  
  ),  
  ),  
  Text(  
    'Ka  
    sty  
    c  
    ),  
    ),
```

Classification

“In the world of classification, you’re either a ‘yes,’ or ‘no,’ answer”

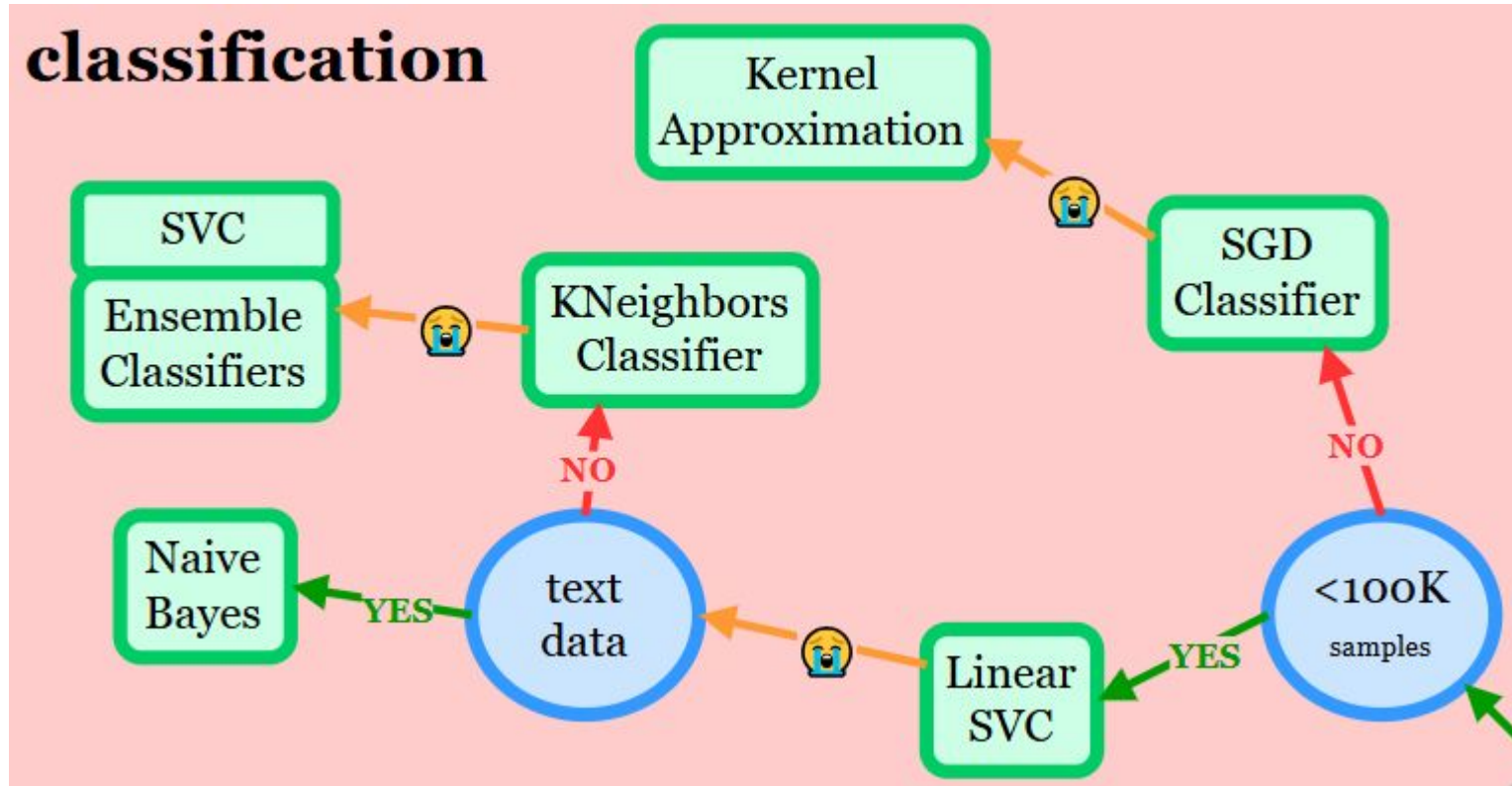


Google Developer Groups

```
er(
ll(32),

/*1*/
child: Column(
    crossAxisAlignment: CrossAxisAlignment.Center,
    children: [
        /*2*/
        Container(
            padding: const EdgeInsets.all(8),
            child: const Text(
                'Oeschinen Lake Campground',
                style: TextStyle(
                    fontWeight: FontWeight.bold,
                ),
            ),
        ),
    ],
),
```

Model Map :



SGD Classifier

```
lookup.KeyValue  
f.constant(['em  
=tf.constant([G  
.lookup.StaticV  
_buckets=5)
```


Definition:

SGDClassifier adalah algoritma pembelajaran mesin yang mengimplementasikan rutinitas pembelajaran sederhana berbasis stochastic gradient descent (SGD). Metode ini bekerja dengan cara memperbarui bobot model secara bertahap untuk meminimalkan kesalahan pada prediksi berdasarkan data pelatihan. SGDClassifier dirancang untuk mendukung berbagai jenis fungsi kerugian (loss functions) seperti hinge loss untuk Support Vector Machines (SVM) dan log loss untuk regresi logistik, sehingga dapat digunakan untuk berbagai masalah klasifikasi. Selain itu, algoritma ini juga mendukung berbagai jenis penalti, seperti L1 (sparse model), L2 (regularisasi ridge), atau kombinasi ElasticNet, yang membantu mengontrol kompleksitas model untuk menghindari overfitting. Karena efisiensinya dalam menangani dataset besar dan pembaruan berbasis batch kecil, SGDClassifier sangat cocok untuk masalah klasifikasi skala besar di mana data tiba secara bertahap.

How it Works:

Bayangkan kita memiliki garis pemisah sederhana (dalam 2D) atau bidang (dalam 3D) untuk membedakan antara dua kelompok data. SGDClassifier memulai dengan tebakan acak untuk garis atau bidang ini.

Alih-alih melihat semua data sekaligus (seperti algoritma lain), SGD melihat satu data atau sekumpulan kecil data (batch kecil) pada satu waktu. Ini seperti belajar sedikit demi sedikit, bukan sekaligus membaca seluruh buku.

Untuk setiap data, algoritma menghitung seberapa jauh prediksi dari model saat ini dibandingkan dengan nilai sebenarnya (kesalahan) menggunakan rumus gradient descent.

Berdasarkan kesalahan ini, model memperbarui parameter (misalnya, kemiringan garis atau berat fitur) agar lebih baik dalam memprediksi data di masa depan. Ini dilakukan menggunakan rumus pembaruan berbasis gradien, yang menunjukkan arah dan seberapa banyak parameter harus diubah.

Proses ini diulangi untuk setiap data atau batch hingga kesalahan menjadi kecil atau model tidak banyak berubah lagi (konvergen).

Kernel Approximation

```
lookup.KeyValue  
f.constant(['em  
=tf.constant([G  
.lookup.StaticV  
_buckets=5)
```

Definition:

Kernel approximation adalah cara untuk meniru efek kernel asli (yang rumit) dengan metode yang bisa memetakan data dengan representasi yang sederhana. Hal ini membuat model pembelajaran mesin bisa bekerja lebih cepat, terutama pada dataset besar, tanpa terlalu banyak mengorbankan performa.

Beberapa teknik yang digunakan untuk kernel approximation, seperti:
Nyström Method: Mengambil sebagian kecil data (sampel) untuk mendekati hasil kernel.

Random Fourier Features: Membuat representasi sederhana data non-linear menggunakan transformasi Fourier acak.

How it Works:

Kamu punya data yang tidak bisa dipisahkan secara lurus.
Kamu ingin mengubah data itu ke bentuk baru agar lebih mudah dipisahkan.

Solusi Normal (Kernel Trick):

Biasanya, kita menghitung hubungan antar data (similaritas) menggunakan fungsi kernel, seperti pola hubungan matematis. Tapi proses ini berat kalau datanya banyak.

Trik Cepat (Kernel Approximation):

Alih-alih menghitung hubungan langsung, kita membuat "versi sederhana" dari data yang bentuknya sudah menyerupai hasil transformasi kernel dengan metode Nyström atau Random Fourier Features. Bayangkan ini seperti "menyulap" data agar terlihat seperti ada di ruang tinggi, tanpa benar-benar memindahkannya.

Setelah data diubah, kamu bisa memakai algoritma sederhana (seperti model linear) untuk memisahkan data tersebut dengan cepat dan efisien.

SVC

```
lookup.KeyValue  
f.constant(['em  
=tf.constant([G  
lookup.StaticV  
_buckets=5)
```

Definition:

SVC adalah algoritma klasifikasi yang termasuk dalam keluarga Support Vector Machines (SVM), yang dirancang untuk menangani masalah klasifikasi dengan menggunakan pendekatan linear. SVC bekerja dengan memisahkan data ke dalam dua kelas dengan cara menemukan garis pemisah (dalam dua dimensi) atau hiperbidang (dalam dimensi lebih tinggi) yang memaksimalkan jarak antara titik data yang paling dekat dari kedua kelas. Proses ini bertujuan untuk menciptakan margin terbesar antara kedua kelas, yang pada gilirannya membantu model membuat prediksi yang lebih baik dan generalisasi yang lebih kuat pada data yang belum pernah dilihat sebelumnya.

How it Works:

Misalkan kamu punya sekumpulan data, seperti titik-titik merah dan biru, yang tersebar di ruang dua dimensi (misalnya, panjang dan lebar). Data ini bisa saja tidak bisa dipisahkan dengan satu garis lurus.

SVC mencoba untuk mencari garis pemisah (disebut hyperplane) yang membagi bola merah dan bola biru. Yang menarik, SVC tidak hanya mencari "garis pemisah", tapi garis yang jaraknya paling jauh dari kedua kelompok tersebut (bola merah dan biru). Garis ini akan meminimalkan kemungkinan kesalahan klasifikasi saat ada data baru.

Dalam mencari garis pemisah, SVC tidak peduli dengan titik data yang berada jauh dari garis. Yang diperhatikan hanya titik-titik yang paling dekat dengan garis dari masing-masing kelompok. Titik-titik ini disebut support vectors dan mereka yang menentukan di mana garis pemisah seharusnya berada.

Kadang, data yang lebih rumit tidak bisa dipisahkan dengan satu garis lurus, misalnya data yang membentuk lingkaran. Di sini, SVC menggunakan apa yang disebut kernel trick untuk "mengubah" data ke ruang yang lebih tinggi, di mana data tersebut bisa dipisahkan dengan garis lurus.

Setelah SVC menemukan garis terbaik yang memisahkan data, model siap untuk memprediksi apakah titik baru akan menjadi bola merah atau bola biru berdasarkan posisinya terhadap garis tersebut.

Naive Bayes

```
lookup.KeyValue  
f.constant(['em  
=tf.constant([G  
lookup.StaticV  
_buckets=5)
```

Definition:

Naive Bayes adalah algoritma klasifikasi yang didasarkan pada teorema Bayes, yang menggunakan probabilitas untuk memprediksi kelas data berdasarkan fitur yang ada. Algoritma ini disebut "naive" karena membuat asumsi sederhana bahwa setiap fitur dalam dataset bersifat independen (terlepas satu sama lain) ketika menghitung probabilitas suatu kelas, meskipun dalam kenyataannya fitur-fitur tersebut seringkali saling berhubungan. Meskipun asumsi independensi ini jarang benar dalam kehidupan nyata, Naive Bayes tetap efektif dalam sekian kasus yang membutuhkan asumsi.

How it Works:

Pertama, sistem akan melihat data yang sudah ada (misalnya, email yang sudah diketahui spam atau tidak spam). Sistem akan melihat kata-kata dalam setiap email, misalnya "diskon", "penawaran", "gratis", dan sebagainya. Naive Bayes akan menghitung seberapa sering setiap kata muncul di dalam email spam dan email tidak spam. Misalnya, kata "gratis" sering muncul di email spam, dan "terima kasih" lebih sering muncul di email tidak spam.

Sekarang, ketika ada email baru, Naive Bayes akan memeriksa kata-kata yang ada di dalam email tersebut. Misalnya, email baru itu mengandung kata "diskon" dan "penawaran". Naive Bayes akan melihat seberapa sering kata-kata ini muncul dalam email spam dibandingkan dengan email tidak spam. Kemudian, sistem akan menghitung kemungkinan besar bahwa email tersebut adalah spam atau tidak spam berdasarkan kata-kata tersebut.

Setelah melihat kata-kata tersebut, Naive Bayes akan menghitung kemungkinan (probabilitas) menggunakan teori bayesian bahwa email tersebut adalah spam atau tidak spam. Naive Bayes mengasumsikan bahwa kata-kata dalam email itu independen satu sama lain (ini kenapa disebut "naive" atau "sederhana"). Jadi, setiap kata dihitung terpisah dan kemudian digabungkan untuk memprediksi kelas email.

Berdasarkan perhitungan probabilitas, Naive Bayes akan memilih kelas yang memiliki kemungkinan terbesar. Jika probabilitas bahwa email tersebut spam lebih tinggi daripada tidak spam, maka sistem akan mengklasifikasikan email sebagai spam.

K Neighbors Classifier

```
lookup.KeyValue  
f.constant(['em  
=tf.constant([G  
.lookup.StaticV  
_buckets=5)
```

Definition:

K-Nearest Neighbors (KNN) adalah salah satu algoritma pembelajaran terawasi yang digunakan untuk klasifikasi dan regresi. KNN adalah algoritma yang sangat sederhana namun efektif, yang bekerja dengan cara mencari tetangga terdekat dari data yang ingin diprediksi, lalu memberikan prediksi berdasarkan mayoritas kelas atau nilai tetangga terdekat tersebut.

How it Works:

Pertama, kamu punya data yang sudah diketahui klasifikasinya. Misalnya, kamu memiliki data tentang banyak orang yang suka catur atau sepak bola. Setiap orang dalam data ini punya informasi seperti umur, tinggi badan, dan hobi.

Ketika ada data baru yang ingin diprediksi (misalnya, seorang calon peserta pesta yang baru), KNN akan mencari "tetangga terdekat" dari data tersebut, berdasarkan fitur-fitur yang ada (misalnya, umur, tinggi badan). KNN akan melihat orang-orang di sekitar data baru dan memilih beberapa orang terdekat. "K" di KNN merujuk pada berapa banyak tetangga terdekat yang ingin kita lihat.

Setelah menemukan beberapa tetangga terdekat, KNN akan melihat kelas atau kategori apa yang lebih banyak ada di antara tetangga tersebut. Misalnya, jika dari 5 tetangga terdekat, 3 orang suka catur dan 2 orang suka sepak bola, maka data baru akan diprediksi menjadi suka catur, karena mayoritas dari tetangganya suka catur.

Berdasarkan keputusan mayoritas dari tetangga-tetangga terdekat, KNN akan memberi label pada data baru tersebut, apakah dia lebih cocok masuk ke kelompok satu atau yang lain.

Ensemble Classifier

```
lookup.KeyValue  
f.constant(['em  
=tf.constant([G  
.lookup.StaticV  
_buckets=5)
```

Definition:

Ensemble Classifier adalah metode dalam machine learning yang menggabungkan beberapa model (atau classifier) untuk meningkatkan akurasi dan stabilitas prediksi. Ide dasar di balik ensemble learning adalah "berpadu lebih baik" – dengan menggabungkan kekuatan beberapa model yang berbeda, kita dapat memperoleh hasil yang lebih baik daripada jika menggunakan satu model saja.

How it Works:

Alih-alih hanya menggunakan satu model untuk membuat prediksi, Ensemble Classifier melibatkan beberapa model yang bekerja bersama. Model-model ini bisa saja berbeda satu sama lain, misalnya model A bisa menggunakan metode tertentu (misalnya pohon keputusan), model B menggunakan metode lain (misalnya regresi logistik), dan seterusnya.

Setiap model dalam ensemble membuat prediksi berdasarkan data yang diberikan. Jadi, jika ada data baru, setiap model akan memberikan hasil yang berbeda (atau sama).

Setelah semua model memberikan prediksinya, ensemble akan menggabungkan hasil prediksi tersebut untuk membuat keputusan final.

Ada beberapa cara untuk menggabungkan prediksi ini, misalnya:

Voting: Jika sebagian besar model mengatakan data tersebut masuk ke kategori A, maka data tersebut akan diklasifikasikan sebagai kategori A.

Rata-rata: Jika itu adalah masalah regresi, hasilnya bisa berupa rata-rata dari semua prediksi.

Dengan menggabungkan prediksi dari beberapa model, ensemble dapat mengurangi kesalahan yang mungkin terjadi pada model individu, membuat keputusan final menjadi lebih akurat dan andal.

Misalnya, jika satu model membuat kesalahan, model lain mungkin tidak melakukan kesalahan, dan dengan menggabungkan hasilnya, kesalahan itu bisa diminimalisasi.

Matriks Evaluasi Classification

```
lookup.KeyValue  
f.constant(['em  
=tf.constant([G  
lookup.StaticV  
_buckets=5)
```

Definition:

Didalam klasifikasi terdapat 4 evaluasi matriks yaitu accuracy, precision, recall, dan f1 score yang masing masing berdasarkan prediksi dan data asli

		Actual Value	
		Positive	Negative
Predicated Value	Positive	5600	600
	Negative	500	3300

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \times precision \times recall}{precision + recall}$$


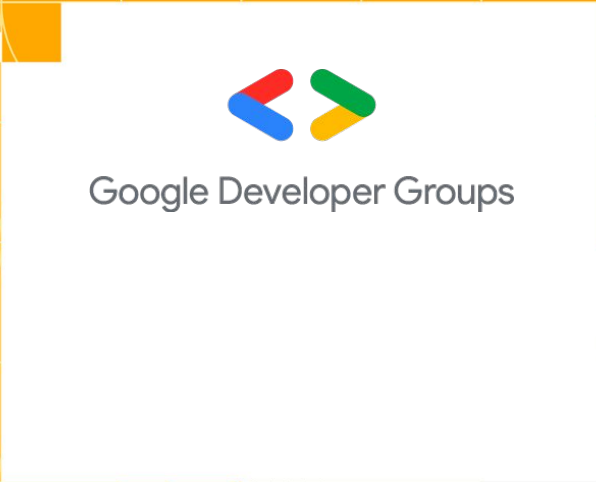
$$accuracy = \frac{TP + TN}{TP + FN + TN + FP}$$

EXAMPLE

```
child: Column(  
  crossAxisAlignment: CrossAxisAlignment.  
  children: [  
    /*2*/  
    Conta  
    pad  
    chi  
    '  
    s  
    )  
  ),  
),  
),  
Text(  
  'Ka  
  sty  
  c  
  ),  
),  
),
```

Regression

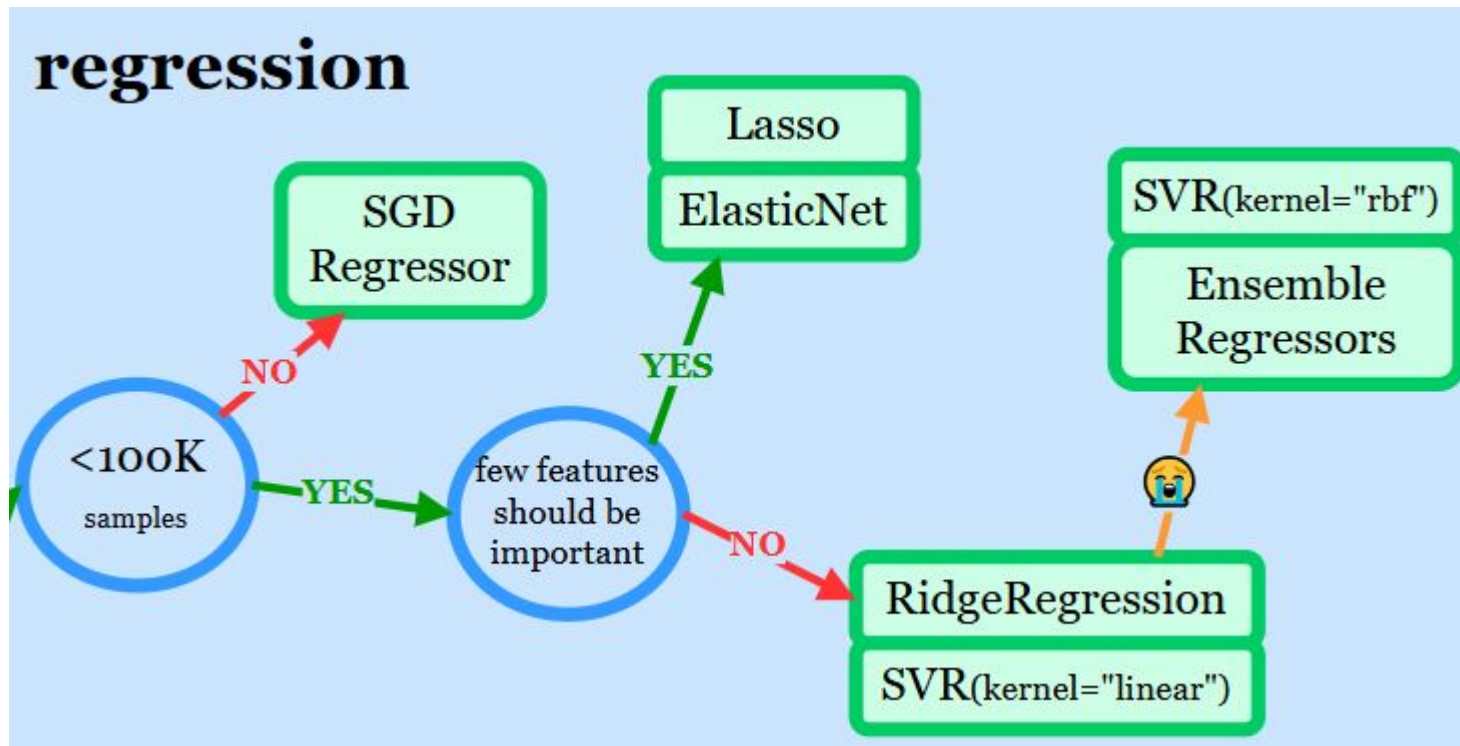
“Because life isn’t binary—it’s a constant struggle to predict just how bad it’s gonna be.”



Google Developer Groups

```
er(
    ll(32),
    // *1*/
child: Column(
    crossAxisAlignment: CrossAxisAlignment.Center,
children: [
    // *2*/
Container(
    padding: const EdgeInsets.all(8),
    child: const Text(
        'Oeschinen Lake Campground',
        style: TextStyle(
            fontWeight: FontWeight.bold,
        ),
    ),
),
),
),
```

Model Map :



SGD Regressor

```
lookup.KeyValue  
f.constant(['em  
=tf.constant([G  
.lookup.StaticV  
_buckets=5)
```


Definition:

SGDRegressor adalah algoritma pembelajaran mesin yang mengimplementasikan rutinitas pembelajaran sederhana berbasis stochastic gradient descent (SGD). Metode ini bekerja dengan cara memperbarui bobot model secara bertahap untuk meminimalkan kesalahan pada prediksi berdasarkan data pelatihan. SGD dirancang untuk mendukung berbagai jenis fungsi kerugian (loss functions) seperti hinge loss untuk Support Vector Machines (SVM) dan log loss untuk regresi logistik, sehingga dapat digunakan untuk berbagai masalah Regressor. Selain itu, algoritma ini juga mendukung berbagai jenis penalti, seperti square error, huber, atau epsilon insensitive, yang membantu mengontrol kompleksitas model untuk menghindari overfitting. Karena efisiensinya dalam menangani dataset besar dan pembaruan berbasis batch kecil, SGDRegressor sangat cocok untuk masalah regresi skala besar atau pembelajaran online di mana data tiba secara bertahap.

How it Works:

Pertama, SGD Regressor mulai dengan tebakan awal untuk hubungan antara jumlah barang yang terjual dan pendapatan toko. Misalnya, dia mulai dengan menebak bahwa setiap barang yang terjual menghasilkan 10 unit uang.

Kemudian, algoritma melihat data yang sebenarnya, misalnya, jumlah barang yang terjual dan pendapatan yang sebenarnya. Dari data ini, SGD Regressor melihat seberapa jauh tebakan awalnya dari kenyataan. Mungkin tebakan awalnya salah, misalnya, seharusnya setiap barang menghasilkan 12 unit uang, bukan 10.

SGD Regressor kemudian mengubah tebakan tersebut sedikit demi sedikit. Setelah melihat data dan mengetahui kesalahan tebakan, dia akan sedikit menyesuaikan tebakannya agar lebih mendekati kenyataan. Proses ini dilakukan bertahap dan terus-menerus.

Yang membedakan SGD dari metode lain adalah dia melakukan pembaruan bertahap. Artinya, dia tidak melihat seluruh data sekaligus, tetapi hanya melihat sebagian kecil data setiap kali dan melakukan penyesuaian berdasarkan itu. Pembaruan ini dilakukan berulang-ulang, dengan setiap langkah membuat tebakan menjadi lebih tepat.

Setelah beberapa kali pembaruan, SGD Regressor akhirnya menemukan hubungan yang lebih baik antara jumlah barang yang terjual dan pendapatan toko, sehingga prediksi yang dihasilkannya lebih akurat.

Setelah belajar dari data yang ada, model ini dapat memprediksi pendapatan toko untuk data baru yang belum pernah dilihat, berdasarkan hubungan yang sudah ditemukan.



Lasso

```
lookup.KeyValue  
f.constant(['em  
=tf.constant([G  
lookup.StaticV  
_buckets=5)
```

Definition:

Lasso (Least Absolute Shrinkage and Selection Operator) adalah sebuah metode dalam regresi linier yang digunakan untuk memilih fitur penting dan mengurangi overfitting dengan menambahkan regularisasi L1 pada fungsi kerugian. Regularisasi L1 ini mengenakan penalti terhadap besarnya koefisien regresi, yang dapat mengarah pada penyusutan beberapa koefisien menjadi nol. Hal ini memungkinkan Lasso untuk secara otomatis melakukan seleksi fitur dan menghasilkan model yang lebih sederhana dan lebih mudah diinterpretasi.

How it Works:

Lasso bekerja dengan memilih fitur (seperti ukuran rumah, jumlah kamar tidur, dll.) yang paling penting untuk membuat prediksi. Misalnya, jika jumlah kamar tidur ternyata tidak berpengaruh banyak terhadap harga rumah, Lasso akan memutuskan untuk mengabaikan faktor ini.

Lasso juga tidak hanya mengabaikan fitur yang tidak penting, tetapi juga dapat mengecilkan pengaruh fitur-fitur yang masih digunakan. Misalnya, jika lokasi rumah berpengaruh besar, Lasso akan mempertahankan pengaruhnya pada prediksi harga, tetapi untuk fitur lainnya yang kurang penting, seperti jumlah kamar tidur, Lasso akan mengurangi pengaruhnya.

Lasso mencari keseimbangan antara membuat model yang sederhana dan akurat. Model yang terlalu rumit dengan banyak fitur bisa jadi tidak terlalu baik, sementara model yang terlalu sederhana mungkin tidak bisa memprediksi dengan baik. Dengan Lasso, kita mendapatkan model yang hanya menggunakan fitur yang benar-benar penting, sehingga lebih sederhana dan mudah dipahami.

Dengan memilih hanya fitur-fitur yang paling relevan, Lasso membuat model yang lebih sederhana dan lebih mudah diinterpretasikan. Ini juga mencegah model dari overfitting, yaitu kondisi ketika model terlalu terfokus pada data latih dan gagal bekerja baik pada data baru.

Ridge Regression

```
lookup.KeyValue  
f.constant(['em  
=tf.constant([G  
.lookup.StaticV  
_buckets=5)
```

Definition:

Ridge Regression adalah salah satu teknik dalam regresi linier yang digunakan untuk mengurangi overfitting dengan menambahkan penalti L2 regularization pada fungsi kerugian regresi linier. Tujuan utama dari Ridge Regression adalah untuk menghasilkan model yang lebih stabil dan lebih mampu menggeneralisasi, terutama saat data memiliki multikolinearitas (ketika beberapa fitur saling berkorelasi) atau saat jumlah fitur lebih banyak daripada jumlah sampel.

How it Works:

Ridge Regression tetap menggunakan semua fitur (seperti ukuran rumah, jumlah kamar tidur, dan lokasi), tetapi memberikan pembatas pada seberapa besar pengaruh masing-masing fitur. Tujuan dari Ridge adalah agar fitur-fitur yang kurang penting tidak "berlebihan" mempengaruhi hasil prediksi.

Misalnya, jika kamu memiliki beberapa fitur yang sangat mirip atau saling terkait (seperti jumlah kamar tidur dan luas rumah), Ridge Regression akan memberikan pengaruh yang lebih kecil pada fitur-fitur yang terlalu berkorelasi satu sama lain. Ini membantu mencegah model menjadi terlalu sensitif terhadap data tertentu, yang bisa membuat prediksi jadi tidak stabil.

Ridge Regression membantu menjaga model agar tidak terlalu rumit atau overfit. Overfitting terjadi ketika model terlalu menyesuaikan diri dengan data latih, sehingga tidak bisa generalisasi dengan baik pada data baru. Dengan menambahkan pembatas pada pengaruh fitur, Ridge Regression membuat model lebih sederhana dan lebih stabil, meskipun menggunakan banyak fitur.

Ridge Regression mencari keseimbangan antara akurasi dan kesederhanaan. Dia mencoba untuk membuat prediksi yang akurat sambil menjaga agar model tidak terlalu kompleks dan terlalu menyesuaikan diri dengan data yang ada.

Elastic Net

```
lookup.KeyValue  
f.constant(['em  
=tf.constant([G  
.lookup.StaticV  
_buckets=5)
```

Definition:

Elastic Net adalah metode dalam regresi linier yang menggabungkan dua teknik regularisasi populer: Lasso (L1 regularization) dan Ridge (L2 regularization). Elastic Net dirancang untuk mengatasi beberapa keterbatasan yang ada pada Lasso dan Ridge. Dengan demikian, Elastic Net mendapatkan manfaat dari seleksi fitur (dari Lasso) dan pengurangan koefisien yang lebih halus (dari Ridge).

How it Works:

Lasso membantu memilih hanya fitur yang paling relevan dan mengabaikan fitur yang tidak penting. Namun, kadang-kadang Lasso bisa menghapus terlalu banyak fitur. Ridge membantu menjaga semua fitur dan memberikan pembatas untuk menghindari overfitting, tetapi bisa membuat model tetap rumit.

ElasticNet menggabungkan kedua metode ini: dia memilih fitur yang penting (seperti Lasso) dan menjaga fitur yang penting sambil memberikan pembatas agar model tetap stabil (seperti Ridge).

Seperti Lasso, ElasticNet akan mencoba mengurangi atau bahkan menghapus fitur yang tidak terlalu berpengaruh. Jadi, jika ada fitur yang tidak penting, ElasticNet akan memilih untuk mengabaikannya.

Seperti Ridge, ElasticNet juga akan menghindari overfitting dengan memberikan pembatas pada pengaruh fitur. Dengan ini, model akan lebih stabil dan tidak terlalu terfokus pada data latih.

ElasticNet mencari keseimbangan yang baik antara memilih fitur yang relevan (seperti Lasso) dan menjaga stabilitas model (seperti Ridge). Dengan cara ini, ElasticNet memberikan model yang lebih akurat dan efisien, terutama ketika ada banyak fitur yang saling berkorelasi.



SVR

```
lookup.KeyValue  
f.constant(['em  
=tf.constant([G  
.lookup.StaticV  
_buckets=5)
```

Definition:

SVR (Support Vector Regression) adalah varian dari Support Vector Machine (SVM) yang digunakan untuk regresi (prediksi nilai kontinu) daripada untuk klasifikasi. SVM, yang awalnya dirancang untuk masalah klasifikasi, diadaptasi menjadi SVR untuk menangani tugas regresi, dengan tujuan memprediksi suatu nilai kontinu berdasarkan data input.

How it Works:

Seperti regresi biasa, SVR mencoba menemukan garis atau kurva yang paling baik yang dapat memprediksi nilai output (misalnya, harga rumah) berdasarkan input (misalnya, ukuran rumah, lokasi, dll.). Bedanya, SVR mencoba menemukan garis yang berada dalam batas toleransi tertentu, bukan yang benar-benar melalui semua titik data. Ini artinya, sedikit kesalahan dalam prediksi masih bisa diterima, selama kesalahan tersebut kecil.

SVR memiliki batas toleransi atau margin kesalahan yang disebut epsilon. Artinya, jika prediksi model sedikit meleset dari nilai yang sebenarnya, itu tidak masalah selama kesalahannya dalam batas yang sudah ditentukan.

Batas ini membantu model agar tidak terlalu terpengaruh oleh data yang sangat jauh dari nilai prediksi yang benar (misalnya, jika ada beberapa harga rumah yang sangat tinggi atau sangat rendah).

SVR bekerja dengan memilih titik-titik data yang paling penting yang disebut support vectors. Titik-titik ini adalah data yang paling berpengaruh dalam menentukan posisi garis atau kurva prediksi.

Jadi, SVR tidak memperhatikan semua titik data secara merata, tetapi lebih fokus pada titik-titik yang ada di dekat batas toleransi, yang benar-benar membantu dalam membuat prediksi yang akurat.

Dengan memilih garis atau kurva yang berada dalam batas toleransi dan fokus pada titik data yang penting, SVR membuat model yang lebih stabil dan robust. Model ini tidak mudah terpengaruh oleh data yang aneh atau outlier (misalnya, harga rumah yang sangat mahal atau sangat murah yang tidak sesuai dengan mayoritas data).

Ensemble Regressor

```
lookup.KeyValue  
f.constant(['en  
=tf.constant([G  
.lookup.StaticV  
_buckets=5)
```

Definition:

Ensemble Regressor adalah teknik dalam machine learning yang menggabungkan prediksi dari beberapa model regresi untuk meningkatkan akurasi dan stabilitas prediksi dibandingkan dengan menggunakan satu model regresi saja. Konsep dasar dari ensemble methods adalah bahwa kombinasi model-model yang berbeda dapat mengurangi kesalahan dan meningkatkan performa model secara keseluruhan. Tujuan dari teknik ini adalah untuk menghasilkan prediksi yang lebih baik dengan mengurangi overfitting, meningkatkan generalisasi, dan memperbaiki ketepatan prediksi.

How it Works:

Ensemble Regressor bekerja dengan menggabungkan beberapa model prediksi yang berbeda, seperti model regresi linear, decision tree, atau SVR. Setiap model memberikan prediksi yang berbeda berdasarkan data yang sama. Dengan menggabungkan hasil dari beberapa model ini, Ensemble mencoba mendapatkan prediksi yang lebih akurat.

Ada beberapa cara untuk menggabungkan model-model ini, misalnya dengan cara rata-rata hasil prediksi (misalnya, model-model membuat prediksi yang berbeda dan hasil akhir diambil dari rata-rata mereka).

Dalam beberapa metode, model-model yang lebih baik atau lebih kuat bisa diberikan bobot lebih besar, sehingga prediksi dari model-model tersebut lebih berpengaruh.

Salah satu keuntungan besar dari Ensemble Regressor adalah kemampuannya untuk mengurangi kesalahan dan menghindari overfitting. Jika salah satu model salah dalam memprediksi harga rumah, model lainnya mungkin memberikan hasil yang lebih baik.

Dengan menggabungkan berbagai model, Ensemble Regressor menghasilkan prediksi yang lebih stabil dan akurat, karena kesalahan dari satu model bisa diperbaiki oleh model lainnya.

Dengan menggunakan beberapa model sekaligus, Ensemble Regressor mampu memberikan hasil yang lebih handal dan akurat dibandingkan jika hanya menggunakan satu model saja. Ini sangat bermanfaat jika data yang kamu gunakan sangat bervariasi atau kompleks.

Matriks Evaluasi Regression

```
lookup.KeyValue  
f.constant(['em  
=tf.constant([G  
.lookup.StaticV  
_buckets=5)
```

Definition:

Mean Squared Error: Mean Squared Error (MSE) mengukur rata-rata kesalahan prediksi dalam model regresi, dengan memberi bobot lebih besar pada kesalahan yang besar karena setiap selisih dipangkatkan dua.

Misal: Jika Anda memprediksi harga rumah dan hasil prediksi jauh dari harga sebenarnya, MSE akan besar karena kesalahan besar dihukum lebih berat (dipangkatkan dua).

R²: R² Score mengukur seberapa baik model regresi menjelaskan variasi dalam data. Ini memberi tahu Anda seberapa banyak variabel target dapat dijelaskan oleh fitur yang digunakan dalam model.

Misal: Jika Anda memprediksi konsumsi energi berdasarkan suhu, $R^2 = 0.85$ berarti 85% variasi konsumsi energi dapat dijelaskan oleh suhu, dan 15% sisanya disebabkan oleh faktor lain.

EXAMPLE




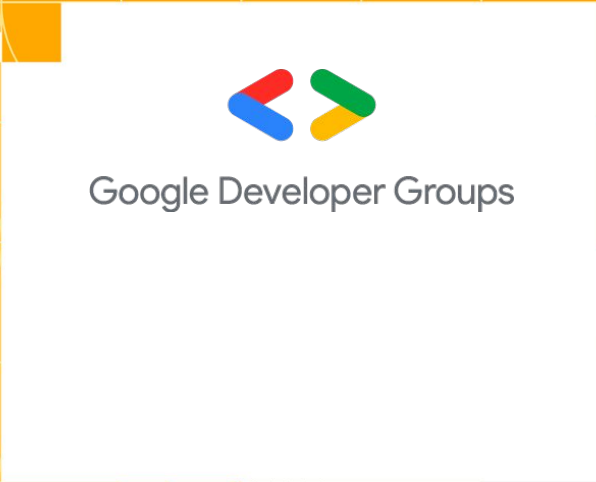
Unsupervised Learning

```
lookup.KeyValue  
f.constant(['em  
=tf.constant([G  
.lookup.StaticV  
_buckets=5)
```

```
child: Column(  
  crossAxisAlignment: CrossAxisAlignment.  
  children: [  
    /*2*/  
    Conta  
    pad  
    chi  
    '  
    s  
    )  
  ),  
),  
Text(  
  'Ka  
  sty  
  c  
  ),  
),  
),
```

Clustering

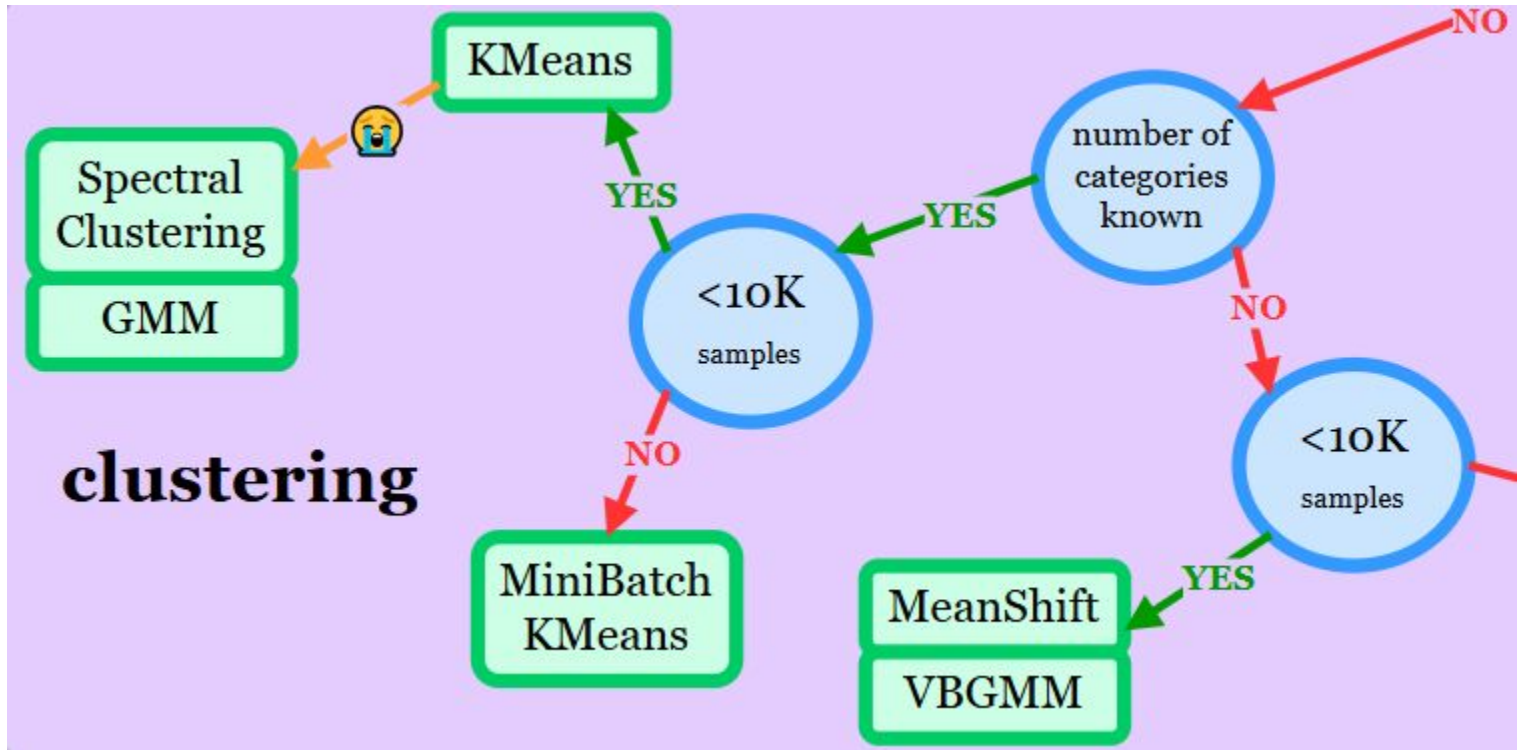
“In unsupervised learning, you’re not wrong. You’re just... uniquely clustered.”



Google Developer Groups

```
er(
    ll(32),
/*1*/
child: Column(
    crossAxisAlignment: CrossAxisAlignment,
children: [
    /*2*/
Container(
padding: const EdgeInsets,
child: const Text(
'Oeschinen Lake Campg
style: TextStyle(
fontWeight: FontWeight
),
),
),
```

Model Map:



K-Means

```
lookup.KeyValue  
f.constant(['em  
=tf.constant([G  
.lookup.StaticV  
_buckets=5)
```

Definition:

K-Means adalah algoritma clustering yang populer dalam machine learning. Algoritma ini digunakan untuk membagi data ke dalam sejumlah kelompok atau cluster berdasarkan kemiripan antar data. K-Means termasuk dalam unsupervised learning, yang berarti algoritma ini bekerja tanpa label target atau output yang telah ditentukan sebelumnya.

How it Works:

Pertama, kamu harus memilih berapa banyak grup (atau cluster) yang ingin kamu buat. Misalnya, kamu ingin membagi data rumah menjadi 3 grup: rumah kecil, rumah sedang, dan rumah besar. Jumlah cluster ini adalah K dalam K-Means. Kemudian, K-Means akan memilih secara acak beberapa titik dalam data sebagai titik awal untuk setiap cluster. Ini disebut centroid (titik pusat) dari cluster tersebut. Bayangkan ini seperti memilih titik awal yang mewakili lokasi "pusat" setiap grup rumah.

Setelah memilih titik-titik pusat cluster, K-Means akan mengelompokkan semua data berdasarkan kedekatannya dengan titik pusat tersebut. Setiap rumah akan masuk ke dalam grup yang pusatnya paling dekat. Misalnya, jika ada rumah yang dekat dengan titik pusat rumah besar, maka rumah itu akan dikelompokkan ke dalam grup rumah besar.

Setelah semua rumah dikelompokkan, K-Means akan menghitung ulang titik pusat (centroid) untuk masing-masing grup berdasarkan rata-rata posisi rumah yang ada di dalam grup tersebut. Ini berarti titik pusat cluster akan bergeser ke posisi yang lebih tepat berdasarkan rumah-rumah yang ada di dalam grup itu.

Setelah menghitung ulang titik pusat, K-Means akan kembali menugaskan rumah-rumah ke cluster baru berdasarkan titik pusat yang sudah diperbarui. Proses ini diulang terus-menerus hingga titik pusat tidak lagi berubah secara signifikan, yang berarti kelompok-kelompok tersebut sudah stabil.

Mini Batch K-means

```
lookup.KeyValue  
f.constant(['em  
=tf.constant([G  
lookup.StaticV  
_buckets=5)
```

Definition:

Mini-Batch K-Means adalah algoritma clustering yang efisien untuk dataset besar, dengan menggunakan subset kecil data pada setiap iterasi untuk memperbarui centroid. Meskipun kurang akurat dibandingkan K-Means standar, algoritma ini menawarkan waktu eksekusi yang jauh lebih cepat dan kebutuhan memori yang lebih rendah, menjadikannya pilihan ideal untuk aplikasi skala besar dan data streaming.

How it Works:

Sama seperti K-Means, langkah pertama adalah menentukan jumlah cluster yang ingin kamu buat, misalnya 3 cluster (untuk rumah kecil, rumah sedang, dan rumah besar). Mini-Batch K-Means memilih beberapa titik acak sebagai titik pusat untuk masing-masing cluster. Ini adalah titik-titik awal, mirip dengan K-Means.

Alih-alih menggunakan seluruh data untuk memperbarui titik pusat cluster seperti pada K-Means, Mini-Batch K-Means hanya menggunakan sebagian kecil data (mini-batch) pada setiap iterasi. Misalnya, jika datasetmu berisi ribuan rumah, alih-alih menghitung posisi titik pusat dengan semua rumah, Mini-Batch K-Means hanya akan memilih sebagian kecil dari rumah-rumah tersebut untuk memperbarui titik pusat cluster.

Mini-Batch K-Means akan mengelompokkan data dari mini-batch ke dalam cluster yang paling dekat dengan titik pusat yang sudah ada. Jadi, rumah-rumah dalam mini-batch ini akan masuk ke dalam grup berdasarkan kedekatannya dengan titik pusat.

Setelah mengelompokkan data, Mini-Batch K-Means memperbarui titik pusat cluster berdasarkan mini-batch yang digunakan. Namun, pembaruan titik pusat dilakukan dengan menggunakan hanya sebagian kecil data, bukan seluruh dataset. Ini membuat prosesnya lebih cepat.

Proses ini diulang beberapa kali dengan memilih mini-batch data yang berbeda pada setiap iterasi. Secara bertahap, titik pusat cluster akan bergerak menuju posisi yang lebih tepat, meskipun dengan pembaruan yang lebih cepat karena hanya menggunakan sebagian kecil data.

Mean Shift

```
lookup.KeyValue  
f.constant(['em  
=tf.constant([G  
.lookup.StaticV  
_buckets=5)
```

Definition:

Mean Shift adalah algoritma clustering berbasis density yang tidak memerlukan asumsi tentang jumlah cluster sebelumnya. Berbeda dengan algoritma seperti K-Means yang memerlukan jumlah cluster K , Mean Shift bekerja dengan mencari peaks (puncak) dalam distribusi densitas data, dan cluster diidentifikasi berdasarkan puncak-puncak tersebut.

How it Works:

Langkah pertama adalah memilih beberapa titik data awal. Titik-titik ini bisa dipilih secara acak atau dipilih dari data yang sudah ada. Titik-titik ini akan digunakan sebagai tempat awal untuk mencari area dengan kepadatan tinggi di sekitar mereka.

Setelah memilih titik-titik awal, algoritma akan mencari pusat kepadatan sekitar titik tersebut. Titik-titik di sekitarnya yang lebih dekat akan dihitung, dan algoritma akan bergerak menuju titik pusat (mean) dari titik-titik tersebut.

Algoritma akan memindahkan titik yang ada ke arah pusat kepadatan yang lebih besar, yaitu titik rata-rata dari titik-titik terdekat tersebut. Proses ini diulang, di mana titik bergerak ke pusat baru berdasarkan rata-rata titik yang berada di sekitarnya.

Titik-titik yang bergerak terus mendekati area dengan kepadatan titik yang lebih tinggi. Ini akan berlanjut hingga titik-titik tersebut stabil dan tidak bergerak lagi karena mereka sudah berada di pusat kepadatan yang maksimal.

Setelah titik-titik mencapai stabilitas, maka data-data yang berada dekat dengan pusat-pusat yang sudah ditemukan akan dikelompokkan bersama. Setiap kelompok titik yang terpusat pada pusat kepadatan ini akan membentuk sebuah cluster.



GMM

```
lookup.KeyValue  
f.constant(['em  
=tf.constant([G  
.lookup.StaticV  
_buckets=5)
```

Definition:

Gaussian Mixture Model (GMM) adalah algoritma clustering yang fleksibel dan probabilistik. GMM memodelkan data sebagai campuran beberapa distribusi Gaussian, menjadikannya sangat cocok untuk data dengan cluster berbentuk kompleks atau data dengan overlap yang tinggi. Meskipun memerlukan asumsi jumlah cluster dan lebih kompleks secara komputasi dibandingkan K-Means, GMM sering kali menghasilkan hasil clustering yang lebih akurat dan informatif.

How it Works:

GMM berasumsi bahwa data kita berasal dari beberapa grup atau cluster, dan masing-masing grup mengikuti distribusi normal (bentuk lonceng). Langkah pertama adalah menebak berapa banyak distribusi normal (atau cluster) yang ada di data. Misalnya, kita bisa memulai dengan menganggap ada 3 cluster dalam data kita.

Setelah menentukan jumlah cluster, GMM memilih posisi awal untuk setiap distribusi normal (pusatnya) dan bentuknya (seberapa lebar atau sempit distribusinya). Ini berarti GMM memilih titik tengah dan sebaran data untuk setiap cluster yang ada.

Setelah menentukan posisi dan bentuk distribusi, GMM kemudian mencoba untuk menetapkan setiap titik data ke salah satu distribusi normal yang ada. Setiap titik data memiliki kemungkinan untuk berada di salah satu distribusi, dan GMM menghitung kemungkinan ini berdasarkan posisi data dan distribusi normal yang sudah ditentukan.

Setelah data ditetapkan ke cluster, GMM memperbarui posisi (pusat) dan bentuk distribusi normal berdasarkan data yang telah dimasukkan ke dalam cluster tersebut. Artinya, GMM menghitung ulang titik tengah dan sebaran distribusi normal untuk setiap cluster, sehingga lebih sesuai dengan data yang ada.

Langkah-langkah ini (menetapkan data ke cluster dan memperbarui distribusi) diulang berkali-kali. Proses ini berlanjut sampai distribusi normal stabil, artinya posisi dan bentuk distribusi tidak lagi berubah secara signifikan.

VBGMM

```
lookup.KeyValue  
f.constant(['em  
=tf.constant([G  
lookup.StaticV  
_buckets=5)
```

Definition:

VBGMM adalah versi lebih canggih dari Gaussian Mixture Model yang menggunakan inferensi Bayesian untuk menentukan jumlah cluster secara otomatis. Dengan kemampuan regularisasi melalui ARD, algoritma ini sangat berguna untuk dataset dengan jumlah cluster yang tidak diketahui sebelumnya. Meskipun lebih lambat dibandingkan GMM, VBGMM memberikan solusi yang lebih fleksibel dan tahan terhadap overfitting, menjadikannya pilihan yang kuat untuk clustering berbasis probabilitas.

How it Works:

Sama seperti GMM, langkah pertama adalah menentukan berapa banyak cluster yang ingin kita cari dalam data. Misalnya, kita ingin mencari 3 kelompok data. VBGMM mulai dengan mengasumsikan bahwa data kita berasal dari beberapa distribusi normal yang berbeda (seperti GMM). Kita mulai dengan memilih posisi dan sebaran distribusi normal secara acak untuk masing-masing cluster.

VBGMM menggunakan pendekatan Bayesian untuk memperkirakan parameter (posisi dan bentuk distribusi) cluster, yang berbeda dengan GMM yang menggunakan estimasi titik tengah dan sebaran secara langsung. Dengan pendekatan Bayesian, kita bukan hanya mencoba menentukan satu nilai untuk setiap parameter, tetapi kita memperkirakan kemungkinan berbagai nilai untuk parameter tersebut berdasarkan data yang ada.

Variational Inference adalah teknik yang digunakan oleh VBGMM untuk memperkirakan distribusi parameter dengan cara yang lebih efisien. Alih-alih mencari nilai pasti untuk setiap parameter, VBGMM mencari distribusi kemungkinan (distribution of possibilities) untuk parameter tersebut. Ini memungkinkan model untuk menangani ketidakpastian lebih baik dan memberikan gambaran yang lebih jelas tentang seberapa yakin kita dengan hasil cluster yang ditemukan.

Seperti GMM, VBGMM akan menetapkan setiap titik data ke salah satu cluster berdasarkan distribusi normal yang telah diperkirakan. Namun, dalam VBGMM, proses ini lebih probabilistik dan bergantung pada distribusi parameter yang dihitung menggunakan pendekatan Bayesian.

Setelah menetapkan data ke cluster, parameter distribusi (seperti posisi dan sebaran) diperbarui berdasarkan distribusi kemungkinan parameter yang ditemukan. Proses ini diulang beberapa kali, memperbarui parameter dan probabilitas setiap titik data untuk bergabung dengan cluster yang berbeda. Proses ini diulang hingga konvergen, artinya parameter distribusi dan pembagian data ke cluster sudah stabil, dan tidak ada perubahan besar lagi.

Spectral Clustering

```
lookup.KeyValue  
f.constant(['em  
=tf.constant([G  
.lookup.StaticV  
_buckets=5)
```


Definition:

Spectral Clustering adalah algoritma berbasis graf yang sangat kuat untuk menemukan cluster dengan bentuk kompleks atau non-linier. Algoritma ini bekerja dengan memanfaatkan analisis eigen pada matriks afinitas data, lalu menggunakan algoritma seperti K-Means untuk menemukan cluster. Meskipun komputasinya lebih mahal dibanding algoritma lain seperti K-Means, Spectral Clustering sangat cocok untuk data dengan struktur non-trivial.

How it Works:

Langkah pertama dalam Spectral Clustering adalah mengonversi data menjadi graf. Data yang ingin dikelompokkan dianggap sebagai titik-titik pada graf, dan hubungan antar titik (seberapa mirip atau dekat titik-titik tersebut) digambarkan sebagai tepi (edge). Biasanya, jarak antar titik data digunakan untuk menentukan apakah dua titik tersebut terhubung dalam graf. Misalnya, jika dua titik cukup dekat, mereka akan terhubung.

Setelah graf dibangun, langkah selanjutnya adalah membangun matriks kedekatan (atau similarity matrix). Matriks ini menggambarkan seberapa kuat hubungan (kedekatan) antara setiap pasangan titik data. Di dalam matriks ini, semakin besar nilai antara dua titik, semakin dekat atau mirip titik tersebut satu sama lain.

Spectral Clustering kemudian menghitung Matriks Laplacian, yang mencerminkan bagaimana data terhubung satu sama lain dalam graf. Matriks ini membantu mengidentifikasi struktur global data, seperti kelompok-kelompok data yang terpisah atau terhubung dengan kuat.

Selanjutnya, algoritma Spectral Clustering menghitung eigenvectors dan eigenvalues dari Matriks Laplacian. Ini adalah langkah yang lebih matematis, tetapi intinya adalah untuk menemukan arah dan skala dari struktur data dalam graf tersebut. Eigenvectors ini akan digunakan untuk menentukan posisi data dalam ruang yang lebih rendah (dimensi yang lebih rendah).

Setelah memperoleh eigenvectors, data diproyeksikan ke dalam ruang dimensi rendah berdasarkan nilai-nilai ini. Langkah ini memungkinkan kita untuk melihat pola atau kelompok data yang mungkin tidak terlihat pada dimensi asli yang lebih tinggi.

Setelah data diproyeksikan ke ruang dimensi rendah, langkah terakhir adalah melakukan pengelompokan menggunakan metode yang lebih sederhana, seperti K-Means. Data yang berada dekat satu sama lain dalam ruang dimensi rendah akan dikelompokkan menjadi satu cluster,

Matriks Evaluasi Clustering

```
lookup.KeyValue  
f.constant(['em  
=tf.constant([G  
.lookup.StaticV  
_buckets=5)
```

Definition:

Within-Cluster Sum of Squares (WCSS): WCSS adalah jumlah kuadrat jarak antara setiap titik data dalam cluster ke centroid (pusat cluster)-nya. Ini mengukur seberapa kompak data dalam suatu cluster.

Misal: Misalkan Anda mengelompokkan pelanggan berdasarkan pembelian mereka. Jika WCSS kecil, ini berarti pelanggan dalam setiap cluster memiliki pola pembelian yang sangat mirip.

Silhouette Score : Silhouette Score adalah metrik yang digunakan untuk mengevaluasi kualitas pengelompokan data dengan mempertimbangkan dua aspek utama: kompak dan terpisah. Kompak berarti data dalam satu cluster harus saling dekat, menunjukkan bahwa anggota cluster memiliki kesamaan yang kuat, sedangkan terpisah berarti cluster yang berbeda harus terpisah dengan jelas untuk memastikan bahwa setiap cluster memiliki identitas yang unik dan tidak saling tumpang tindih.

Misal: Jika Anda mengelompokkan jenis pelanggan berdasarkan kebiasaan belanja mereka, skor Silhouette tinggi menunjukkan bahwa setiap jenis pelanggan memiliki kebiasaan unik yang jelas berbeda dari jenis lainnya.

EXAMPLE

Now it's time for...

Hands on project

In today hands on:

Classification and Regression:

<https://www.kaggle.com/datasets/taweilo/loan-approval-classification-data>

Clustering:

<https://www.kaggle.com/datasets/valakhorasani/bank-transaction-dataset-for-fraud-detection/data>