

## 1. Semana 1

### - Hyperparameter tuning:

- Hyperparameter tuning tem um grande impacto, é setado antes, e é importantíssimo pro processo, há uma biblioteca Keras AutoTuner que ajuda com isso
- Keras tuner pode ajudar a determinar o número de hidden layers, por exemplo; Exemplo de código com o keras tuner, onde começa a layer tendo um valor de 16 e em seguida indo de 16 em 16 steps até 512:

```
def model_builder(hp):  
    model = keras.Sequential()  
    model.add(keras.layers.Flatten(input_shape=(28, 28)))  
  
    hp_units = hp.Int('units', min_value=16, max_value=512, step=16)  
    model.add(keras.layers.Dense(units=hp_units, activation='relu'))  
    model.add(tf.keras.layers.Dropout(0.2))  
    model.add(keras.layers.Dense(10))  
  
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',  
                  metrics=['accuracy'])  
    return model
```

## Search strategy

```
tuner = kt.Hyperband(model_builder,  
                     objective='val_accuracy',  
                     max_epochs=10,  
                     factor=3,  
                     directory='my_dir',  
                     project_name='intro_to_kt')
```

- Há várias formas de fazer essa busca, Hyperband, randomsearch, entre outras

- Pode-se fazer um callback quando certa condição foi atingida. Patience=5 indica que após 5 epochs deve ser interrompida a busca

```
stop_early =
    tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                     patience=5)

tuner.search(x_train,
             y_train,
             epochs=50,
             validation_split=0.2,
             callbacks=[stop_early])
```

### Search output

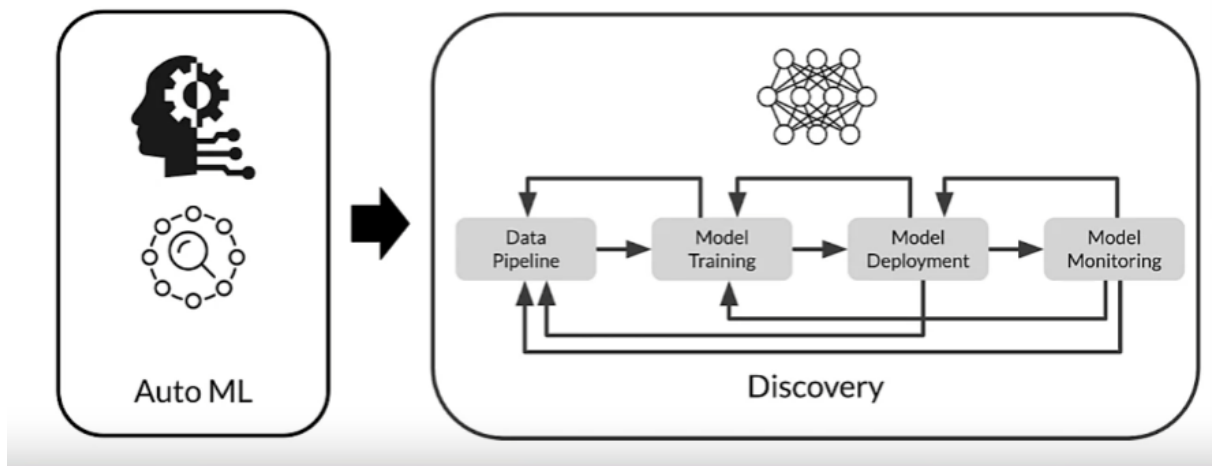
```
Trial 24 Complete [00h 00m 22s]
val_accuracy: 0.3265833258628845
Best val_accuracy So Far: 0.5167499780654907
Total elapsed time: 00h 05m 05s
Search: Running Trial #25
```

Hyperparameter	Value	Best Value So Far
units	192	48
tuner/epochs	10	2
tuner/initial_e...	4	0
tuner/bracket	1	2
tuner/round	1	0
tuner/trial_id	la2ede617bdc476...	None

- **AutoML:** Série de ferramentas para tunar um projeto de ML de fim a fim. Permite desenvolvedores sem muito conhecimento de ML usarem modelos de ML e técnicas. Tenta automatizar o processo de modo a criar soluções isimples, criar soluções rápidas dessas soluções e modelos que talvez performem melhor que

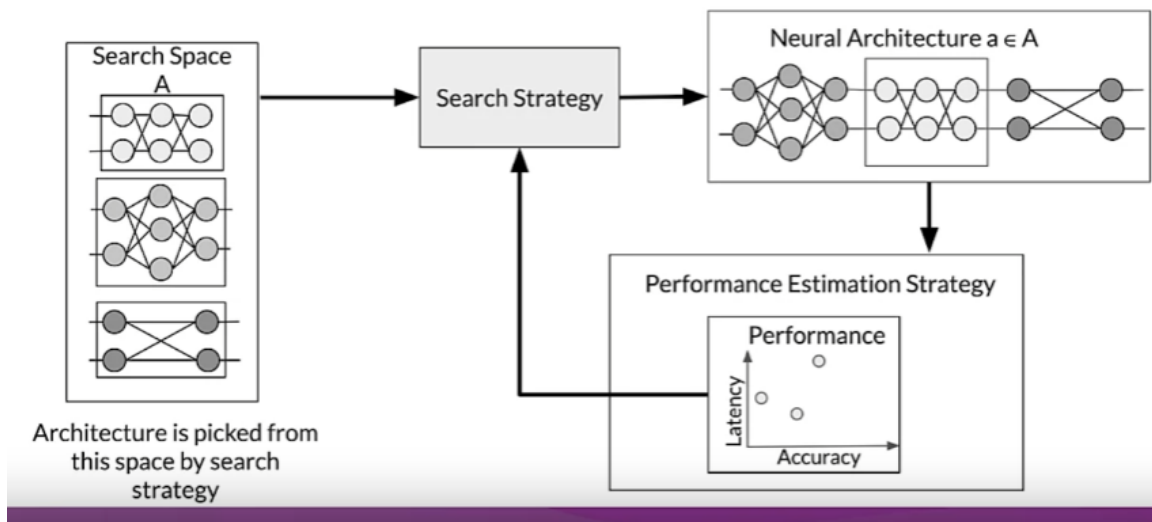
modelos tunados na mão

## Automated Machine Learning (AutoML)



- **Neural Architecture Search (NAS)** técnica para automatizar o design de NN, ajuda a otimizar a arquitetura, faz uma busca em um espaço gigante, usa AutoML para essa busca. Usa o search space: em que uma arquitetura é obtida pelo espaço por uma estratégia de busca, essa estratégia determina como buscamos no espaço. Então se usa essas estratégias em uma estratégia estimativa de performance

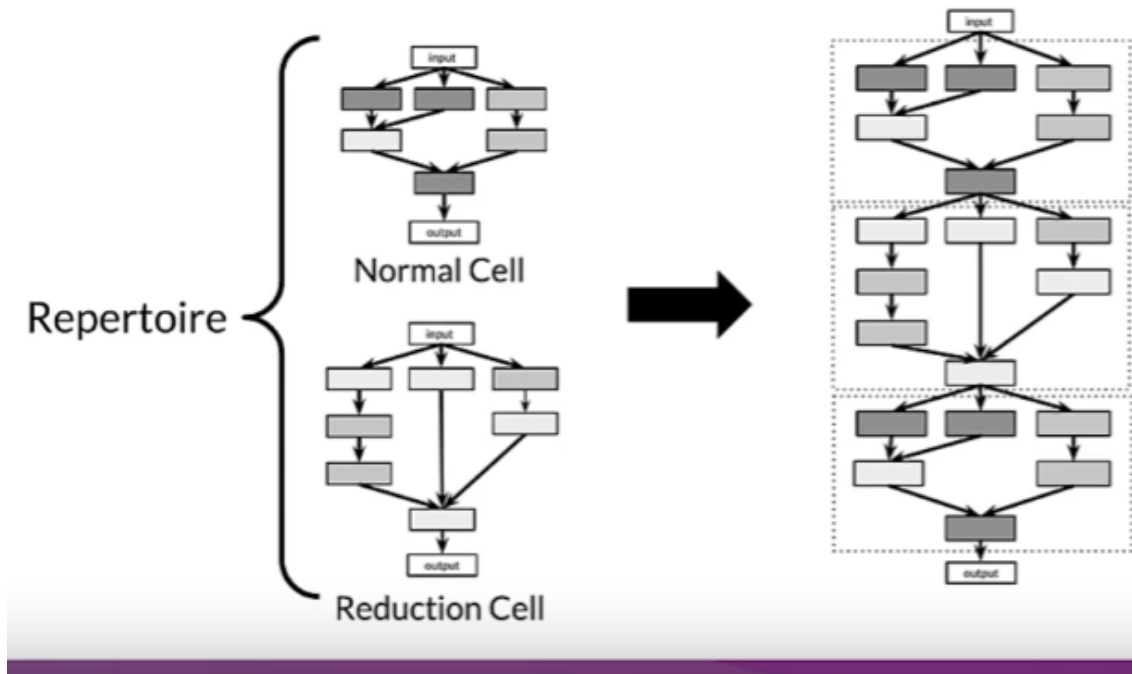
## Neural Architecture Search



- Search spaces: há dois tipos principais de espaços, macro e micro. O Macro contém as layers individuais e os tipos de conexão. Enquanto no micro pega o espaço de células, em que as células são pequenas networks que são empilhadas para gerar a

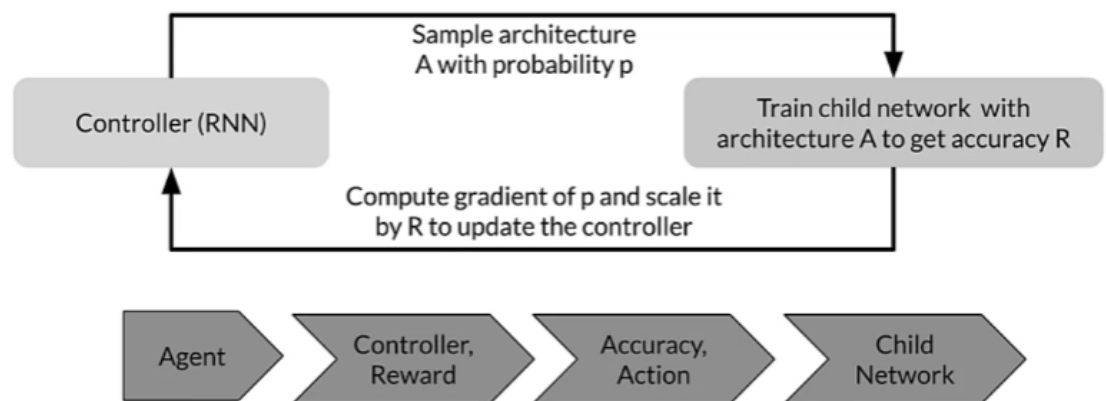
rede final, tem uma performance melhor.

## Micro Architecture Search Space



- Várias estratégias de busca podem ser utilizadas, como gridsearch, randomsearch, bayes, algoritmos genéticos, RL. Grid e random funcionam em datasets pequenos,

## Reinforcement Learning for NAS

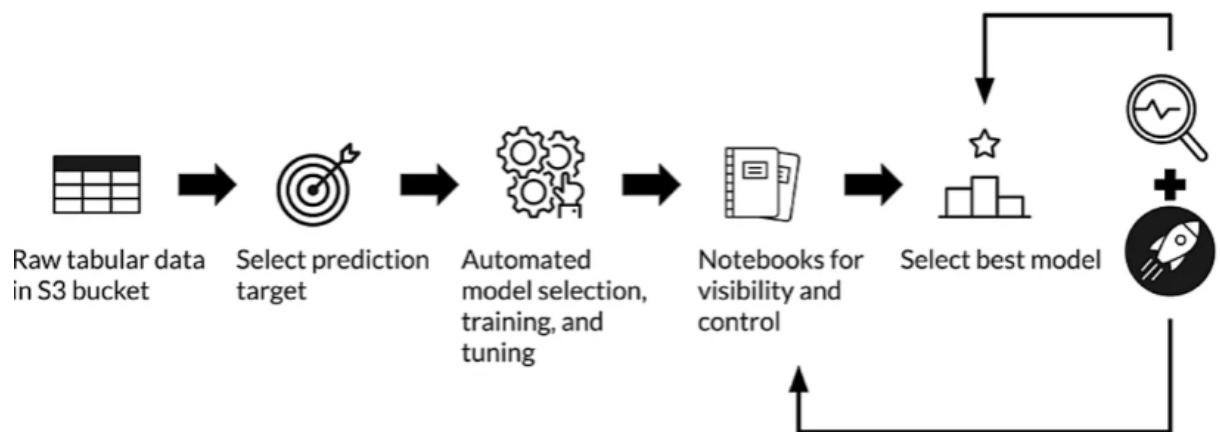


- **Performance Estimation Strategy:** Há diversas formas de obter a performance do algoritmo de busca utilizado, a mais básica é pegar a acurácia da validação, mas isso é computacionalmente custoso demais. **Lower Fidelity Estimates** é uma estratégia que usa partes do dataset ou imagens em menor resolução, ou menos filtros, reduz o custo computacional, entretanto já é comprovado que a arquitetura muda com essas mudanças, sendo não muito recomendado. **Learning Curve Extrapolation** assume que podemos aprender a curva de aprendizado de forma

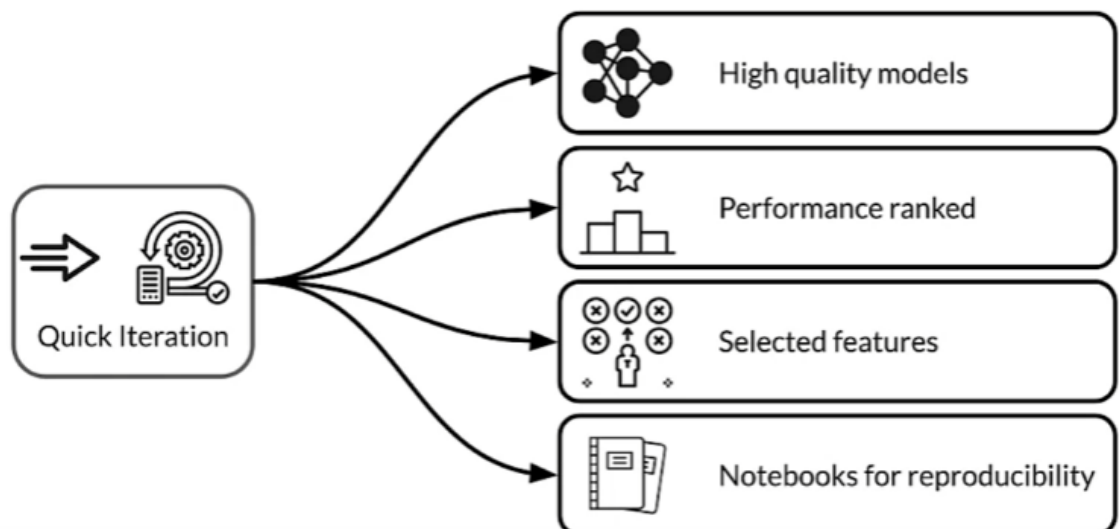
confiável, extrapola todas as curvas de aprendizado inicial e já remove as que tem pior performance. **Weight Inheritance/Network Morphisms** faz algo parecido com o que o transfer learning faz, inicializa os pesos de uma nova arquitetura baseado em arquiteturas previamente treinadas, usa um Network Morphism que é mudar a arquitetura mas sem mudar a função por baixo dos panos, mantém um bom custo computacional e performance.

- Há diversos modelos de AutoML na nuvem
- Amazon autopilot

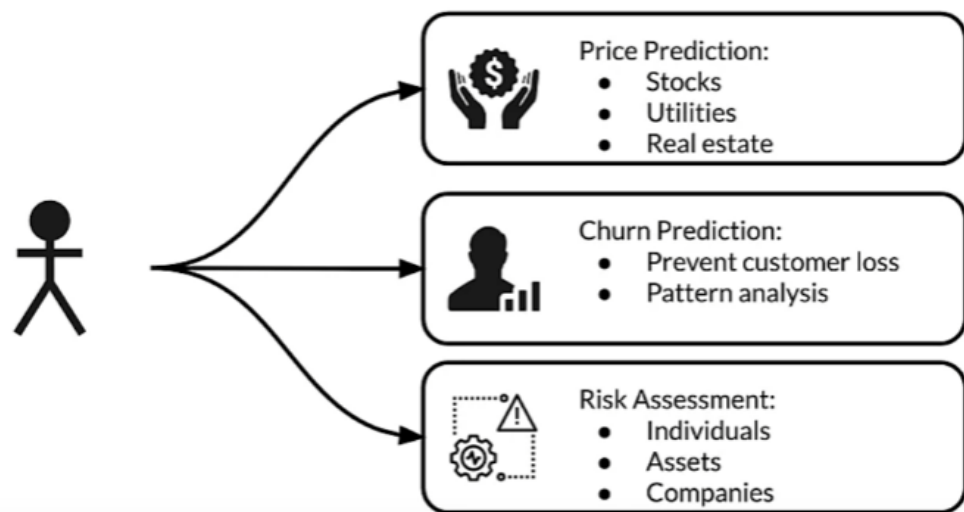
## Amazon SageMaker Autopilot



## Key features

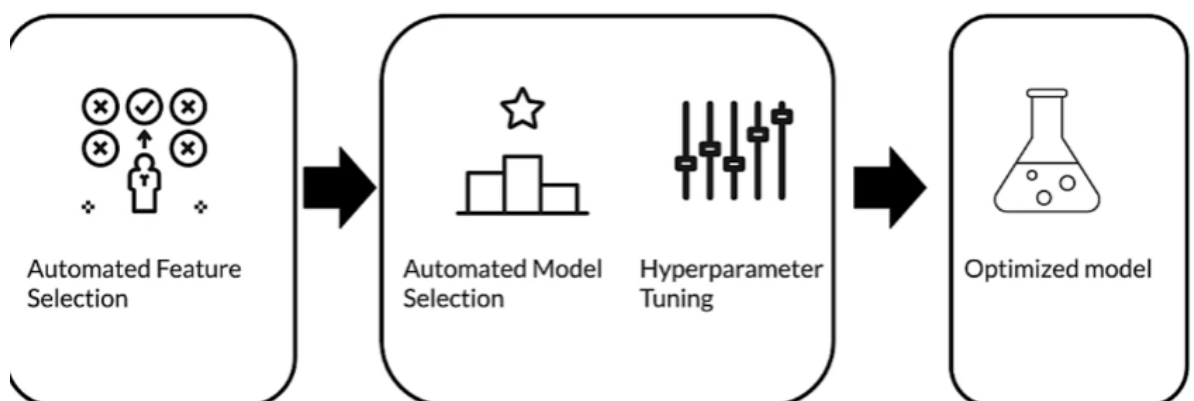


## typical use cases










- DeepLearning.AI

- Microsoft azure  
Microsoft Azure AutoML



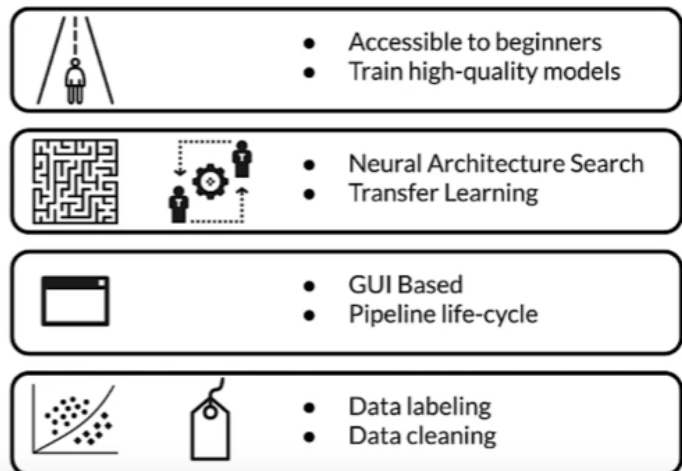
## Key features

-  Quick customization:
  - Model
  - Control settings
-  Automated Feature Engineering
-  Data Visualization
-  Intelligent stopping

- 
  - Experiment summaries
  - Metric visualizations
-  Model Interpretability
-  Pattern Discovery

- Google cloud AutoML

# Google Cloud AutoML



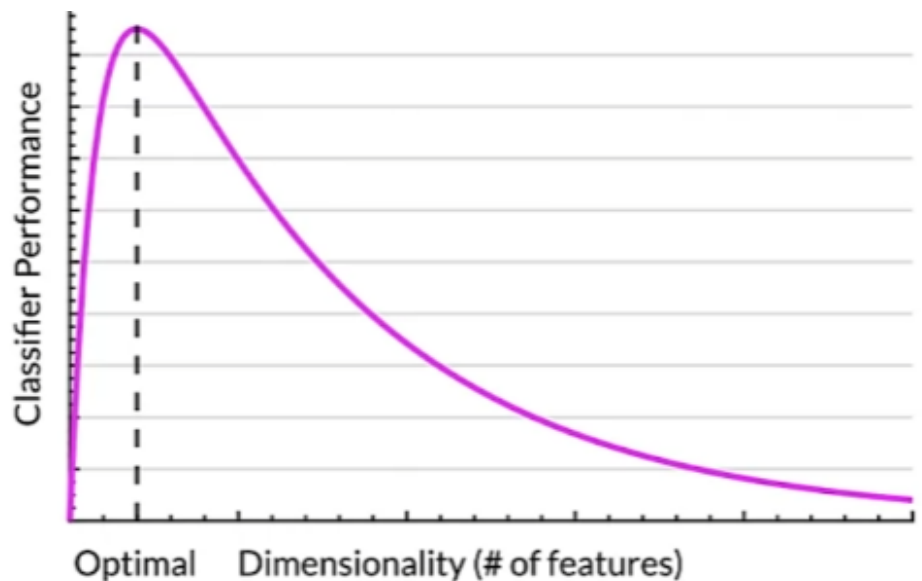
## Cloud AutoML Products

Sight	<b>Auto ML Vision</b> Derive insights from images in the cloud or at the edge.	<b>Auto ML Video Intelligence</b> Enable powerful content discovery and engaging video experiences.
Language	<b>AutoML Natural Language</b> Reveal the structure and meaning of text through machine learning.	<b>Auto ML Translation</b> Dynamically detect and translate between languages.
Structured Data	<b>AutoML Tables</b> Automatically build and deploy state-of-the-art machine learning models on structured data.	

## 2. Semana 2

- **Redução de dimensionalidade**
- Mais dimensões gera mais features (podem ser redundantes, podem gerar mais ruído, etc), mais risco de overfitting, a distância entre cluster em um KNN por exemplo tem problemas de serem classificados. O gráfico abaixo mostra o desempenho com mais dimensões em um dataset em que não foram adicionados mais exemplos de treinamento







- 
- PCA e ICA são técnicas utilizadas para diminuir o número de features/dimensões: A diferença significativa entre PCAs e ICAs é que o PCA procura fatores não correlacionados, enquanto o ICA procura fatores independentes. Se dois fatores não estão correlacionados, isso significa que não há relação linear entre eles. Se forem independentes, significa que não dependem de outras variáveis. Por exemplo, a idade de uma pessoa independe do que ela come ou de quanta televisão ela assiste, provavelmente.

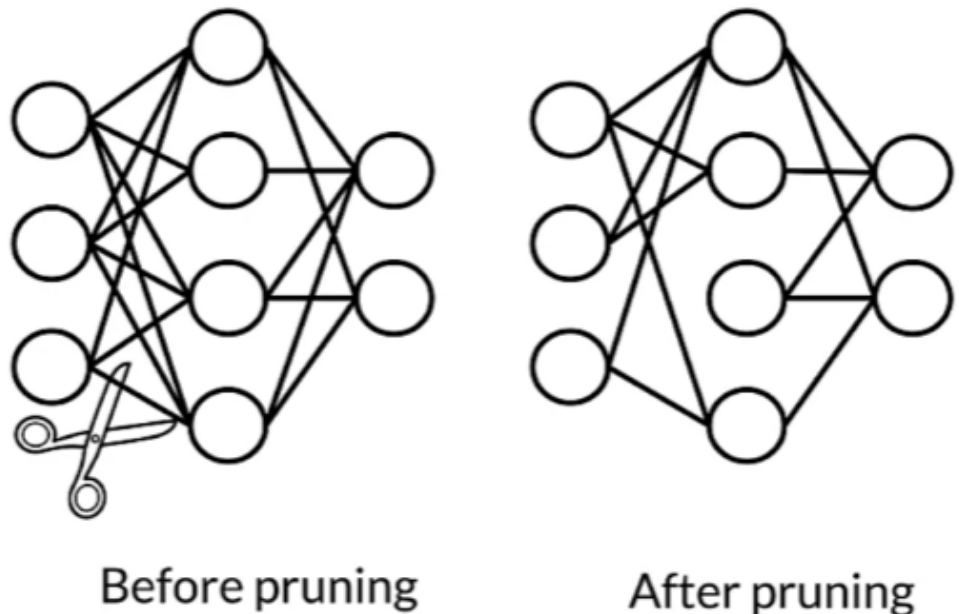
- **Model deployment:** Há diversos serviços para isso

## Model deployment

Options	On-device inference	On-device personalization	On-device training	Cloud-based web service	Pretrained models	Custom models
ML Kit 	✓	✓		✓	✓	✓
Core ML	✓	✓	✓		✓	✓
 TensorFlow Lite *	✓	✓	✓		✓	✓

- \*
- Quantization: mudar o dataset de maneira que cosuma menos recursos computacionais, geralmente resulta em menos acurácia

- Pruning: diminuir o número de conexões/número de neurons, ao mesmo tempo que não comprometa a rede



## Pruning with Keras

```
import tensorflow_model_optimization as tfmot
model = build_your_model()
pruning_schedule = tfmot.sparsity.keras.PolynomialDecay(
    initial_sparsity=0.50, final_sparsity=0.80,
    begin_step=2000, end_step=4000)

model_for_pruning = tfmot.sparsity.keras.prune_low_magnitude(
    model,
    pruning_schedule=pruning_schedule)

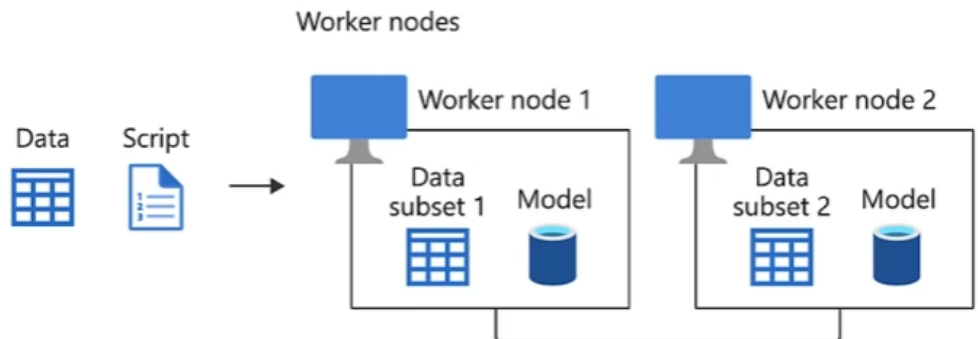
...
model_for_pruning.fit(...)
```

### 3. Semana 3

- **Distributed training** permite treinar grandes modelos ao mesmo tempo que acelera o treinamento. acontece com paralelismo, podendo ser paralelismo de data ou de modelo, de data os modelos são replicados em diferentes processadores e os dados são separados entre essas processadores. Model parallelism é quando os modelos são muito grandes para caber em um único dispositivo, então eles são divididos em diferentes partições em diferentes

aceleradores

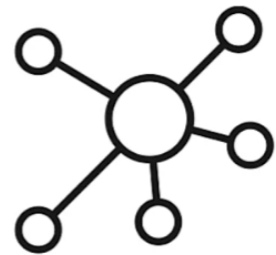
## Data parallelism



- Existe dois estilos para lidar com data parallelism, treino síncrono em que todo mundo treina e atualiza junto aplicando em minibatches e treino assíncrono, em cada trabalhador treina e atualiza em batches separados, é suportado por uma arquitetura de servidor porém é mais difícil de implementar
- Há uma biblioteca no tensorflow para suportar essas distribuições, essa biblioteca permite diversas estratégias

### Distribution Strategies supported by tf.distribute.Strategy

- One Device Strategy
- Mirrored Strategy
- Parameter Server Strategy
- Multi-Worker Mirrored Strategy
- Central Storage Strategy
- TPU Strategy



multi-worker mirrored strategy, central storage strategy, and TPU strategy.

DeepLearning.AI

- A estratégia one device vai colocar qualquer variável criada no seu próprio escopo no dispositivo, todas as funções serão executadas naquele dispositivo. É uma boa estratégia de começo antes de passar para mais complicadas
- A estratégia Mirrored suporta treinamento síncrono distribuído em múltiplas GPUs na mesma máquina, cria uma réplica da GPU por dispositivo e coloca cada variável do modelo em cada réplica, aplica todas as mudanças em

todos os dispositivos e da update nos weights usando algoritmos de cross device communication

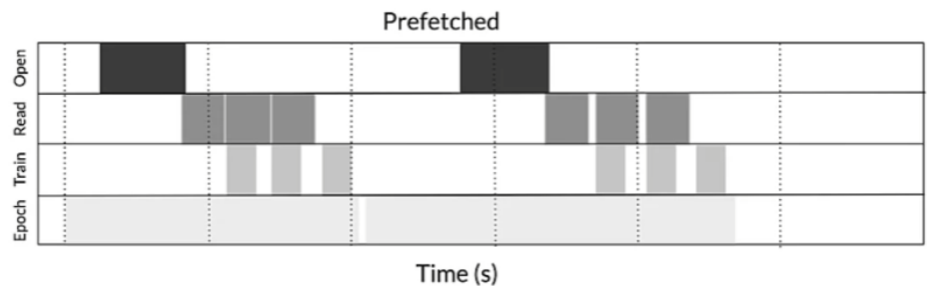
- Parameter Server é uma estratégia que separa em trabalhadores e outros parâmetros. Os parâmetros são armazenados num servidores para que os trabalhadores possam performar em cima dele, é uma estratégia assíncrona.
- **tf.data** é uma biblioteca que ajuda a fazer uma pipeline eficiente para input, é um framework que permite facilitar e aplicar o uso de otimizações

## tf.data: TensorFlow Input Pipeline



- Para otimizar esse pipeline é interessante fazer coisas como um prefetching (pré alocar dado que vai ser lido, por ex), paralisar a extração dos dados e suas transformações, além de armazenar algo em cache

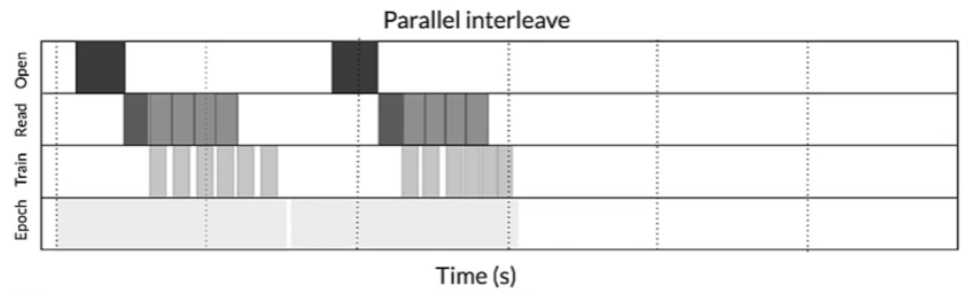
### Optimize with prefetching



```
benchmark(  
  ArtificialDataset()  
  .prefetch(tf.data.experimental.AUTOTUNE)
```

- Paralelizar a extração de dados é uma tarefa que pode ocasionar problemas de sincronia, congestionamento, etc, na comunicação entre os dados locais e os dados online, para resolver isso existem coisas como Time-to-first byte que priorizam os dados locais pois são lidos muito mais rapidamente, ou o Read Throughput que maximiza a largura de banda do armazenamento remoto lendo mais arquivos

## Parallel interleave



```
benchmark(  
  tf.data.Dataset.range(2)  
  .interleave(  
    ArtificialDataset,  
    num_parallel_calls=tf.data.experimental.AUTOTUNE  
  )  
)
```

including interleaving

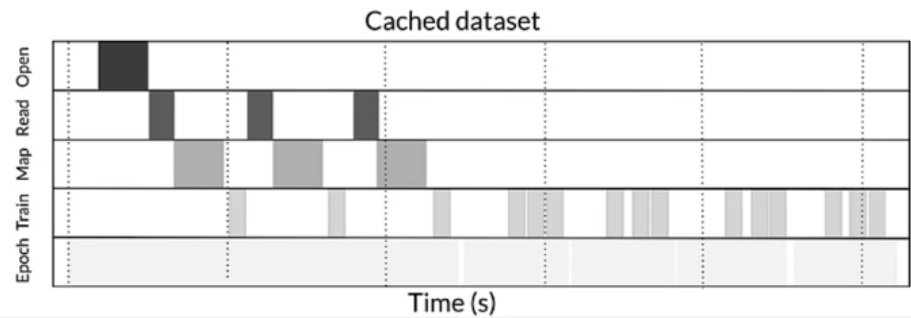
- 
- Paralelismo de dados pode necessitar de pré processamento que pode ser paralizado entre as CPUs, e o valor ótimo de paralelismo depende de fatores como o tamanho dos dados, o custo de mapeamento e a potência da CPU que executa o processo, no tf.data pode se usar o AUTOTUNE para setar o paralelismo automaticamente
- Também pode se armazenar em cache o dataset para adiantar o processo, isso faz com que a leitura de dados seja executada apenas no primeiro epoch enquanto os próximos epochs usam o que está armazenado em cache

## Improve training time with caching

- In-memory: `tf.data.Dataset.cache()`
- Disk: `tf.data.Dataset.cache(filename=...)`

-

# Caching

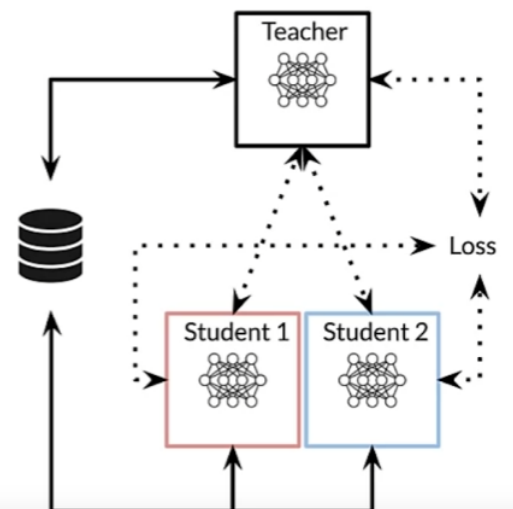


```
benchmark(  
    ArtificialDataset().map(mapped_function).cache(),5  
)
```

- **Knowledge distillation:** duplica a performance de um modelo complexo em um modelo mais simples, a ideia é criar um simples modelo estudante que aprende um modelo complexo “teacher”

## Knowledge distillation

- Duplicate the performance of a complex model in a simpler model
- Idea: Create a simple ‘student’ model that learns from a complex ‘teacher’ model



- Os objetivos do teacher e student podem variar. O teacher é treinado primeiro e buscar atingir o máximo de acurácia ou outra métrica, o student tenta dar match nas distribuições de probabilidades obtida pelas previsões do teacher. A maneira como a destilação do conhecimento funciona é que você transfere conhecimento do professor para o aluno minimizando uma função de perda, na qual o alvo é a distribuição de probabilidades de classe prevista pelo modelo do professor. O que acontece aqui é que os logits dos modelos do professor formam a entrada para a camada softmax final, que é frequentemente usada, pois fornecem mais informações sobre as probabilidades de todas as classes-alvo para cada exemplo. No entanto, em muitos casos, essa distribuição de probabilidade tem a classe correta com uma probabilidade muito alta, com todas as outras probabilidades de classe muito próximas de zero. Realisticamente, às vezes não fornece muita informação além dos ground truth já fornecidos no conjunto de dados. Dark

knowledge é a maneira de distribuir mais informação para o aluno por meio de uma função softmax e um parâmetro T, quanto maior esse parâmetro mais informação o estudante recebe.

- Há diversas formas e técnicas de usar esses conhecimentos, entre elas: pesos ou objetivos dos 2 e combinar numa backprop, ou então comparar a distribuição das predições, usando KL divergence.

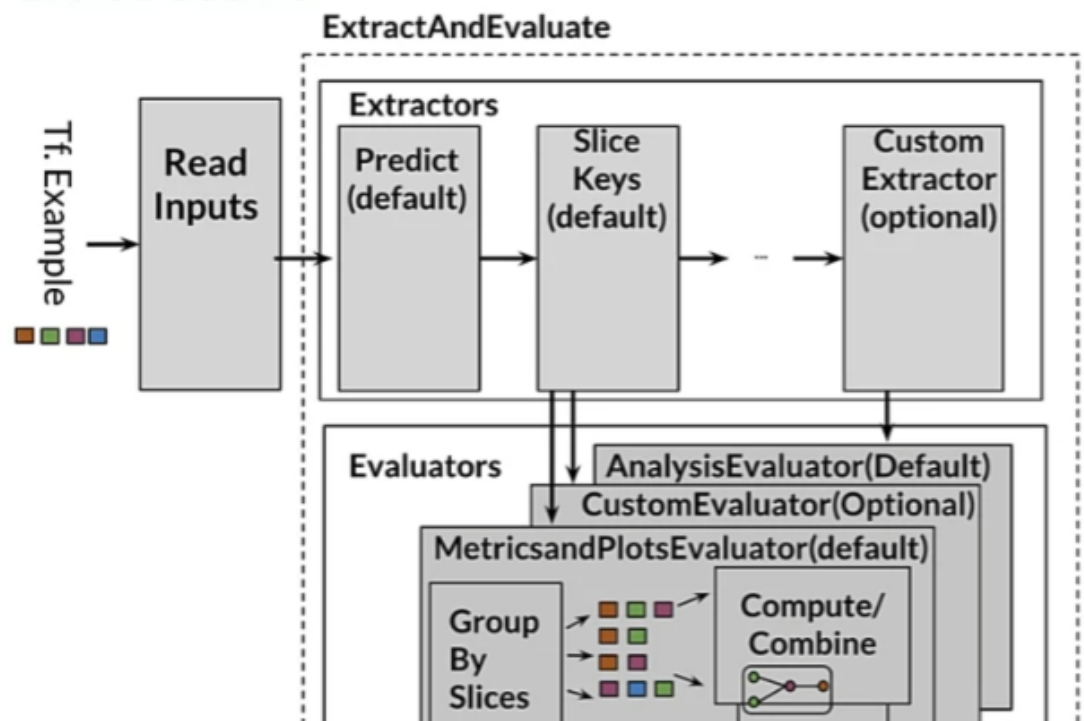
$$L = (1 - \alpha) L_H + \alpha L_{KL}$$

- Essa distribuição tenta aproximar as probabilidades de predição entre professor e student.

#### 4. Semana 4

- **Model performance analysis:** existem duas maneiras principais de se avaliar a performance de um modelo, pode ser por black box, que é ver a acurácia de cada run do model, como no tensorboard, ou podemos avaliar por model introspection, que é entender cada layer do model
- **TensorFlow Model Analysis:** é um framework para analisar a performance do modelo, é escalável e open source. É usável em frameworks maiores como no TFX

## Architecture



## Step 1: Export EvalSavedModel for TFMA

```
import tensorflow as tf
import tensorflow_transform as tft
import tensorflow_model_analysis as tfma

def get_serve_tf_examples_fn(model, tf_transform_output):
    # Return a function that parses a serialized tf.Example and applies TFT

    tf_transform_output = tft.TFTransformOutput(transform_output_dir)
    signatures = {
        'serving_default': get_serve_tf_examples_fn(model, tf_transform_output)
                           .get_concrete_function(tf.TensorSpec(...)),
    }

    model.save(serving_model_dir_path, save_format='tf', signatures=signatures)
```

## Step 2: Create EvalConfig

```
# Specify slicing spec
slice_spec = [slicer.SingleSliceSpec(columns=['column_name']), ...]

# Define metrics
metrics = [tf.keras.metrics.Accuracy(name='accuracy'),
           tfma.metrics.MeanPrediction(name='mean_prediction'), ...]
metrics_specs = tfma.metrics.specs_from_metrics(metrics)

eval_config = tfma.EvalConfig(
    model_specs=[tfma.ModelSpec(label_key=features.LABEL_KEY)],
    slicing_specs=slice_spec,
    metrics_specs=metrics_specs, ...)
```

## Step 3: Analyze model

```
# Specify the path to the eval graph and to where the result should be written
eval_model_dir = ...
result_path = ...

eval_shared_model = tfma.default_eval_shared_model(
    eval_saved_model_path=eval_model_dir,
    eval_config=eval_config)

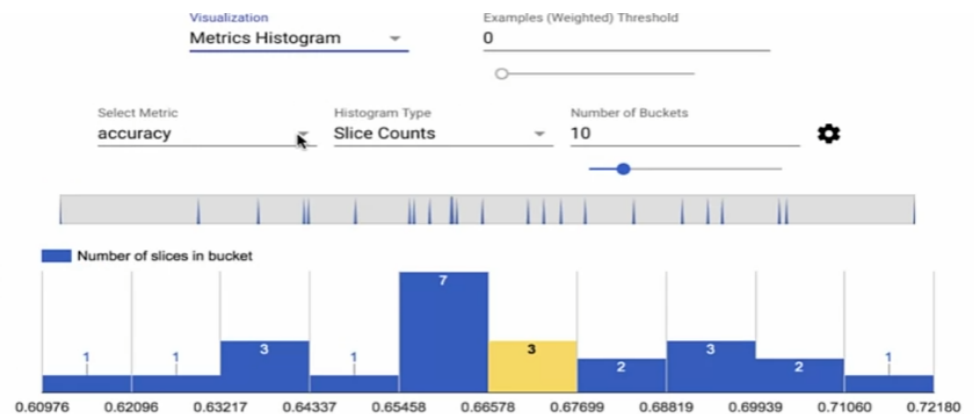
# Run TensorFlow Model Analysis
eval_result = tfma.run_model_analysis(eval_shared_model=eval_shared_model,
    output_path=result_path,
    ...)
```



## Step 4: Visualizing metrics

```
# render results  
tfma.viewer.render_slicing_metrics(result)
```

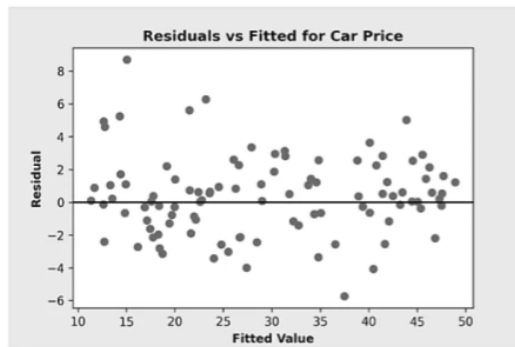
- 
- Resultado:



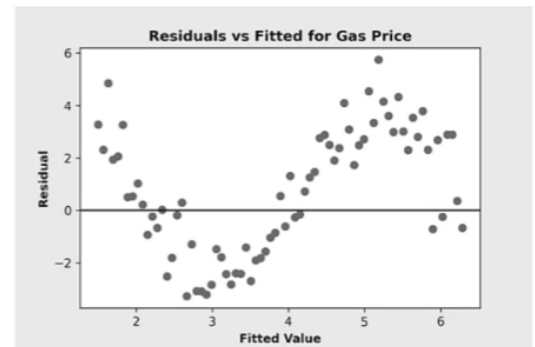
feature	accuracy	accuracy_baseline	auc	auc_precision_recall	average_loss
trip_start_hour:19	0.63582	0.59104	0.64311	0.56092	0.64626
trip_start_hour:14	0.67117	0.65766	0.63793	0.49112	0.61667
trip_start_hour:2	0.66102	0.63559	0.58527	0.47002	0.65236
trip_start_hour:12	0.69643	0.65625	0.68270	0.54122	0.59538

- Para **debugar** um modelo, pode-se usar coisas modelos de benchmark que são modelos uteis para comparação e debug de etapas do processo, pois já são bem consolidados.
- Sensitivity Analysis analisa o impacto que uma feature tem na análise do modelo, simula os dados da minha escolha e vê o que o modelo prevê. Vê como o modelo reage a data que não foi vista antes, pode ser feito com random attacks, expondo o modelo pra muitos dados randômicos, também pode ser feito partial dependence plots pra plotar os efeitos de mudar uma ou mais variáveis no modelo. PDPbox e PyCEBOX, a sensibilidade pode ser usada para ver o quão vulnerável o modelo é ataques, por exemplo
- Residual analysis: mede a diferença entre as previsões do modelo e o ground truth. resíduos não devem ser correlacionados com outra feature e não devem ser correlacionados entre si

## Residual analysis



Random = Good



Systematic = Bad

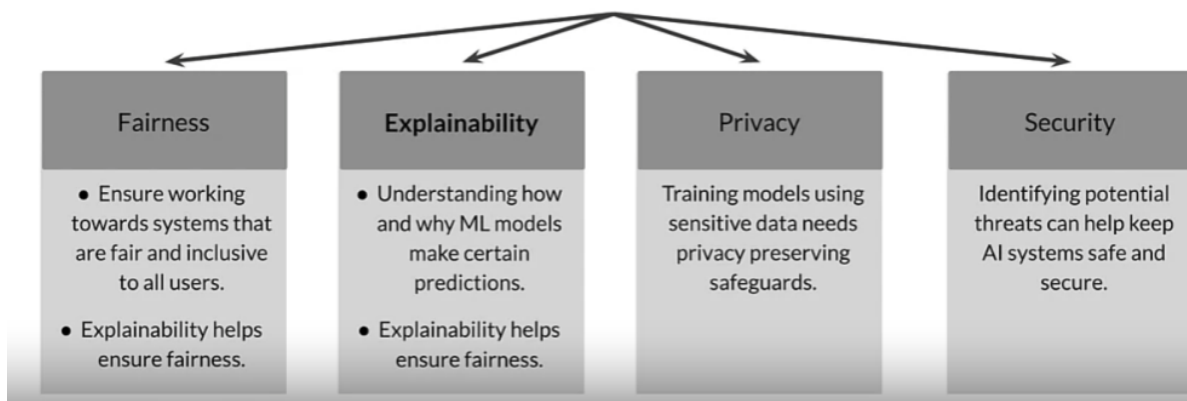
- Fairness: há uma biblioteca no tf para computar a fairness em modelos, deve-se computar a performance em partes dos dados, avaliar as métricas por diversos thresholds e se a margem de decisão for pequena reportar em detalhe
- Para medir fairness pode-se usar coisas como Positive Rate/ Negative Rate para calcular a porcentagem de data points classificados como positivos ou negativos, vê a igualdade dos parâmetros, também o True Positive Rate e FNR para medir a quantidade de dados corretamente preditos como positivo ou negativo, out também falsos positivos e falsos negativos. Também podemos usar AUC (accuracy and area under the curve) que pega métricas como a acurácia e a porcentagem de data points que são corretamente classificados para cada classe dado peso independente

### 5. Semana 5

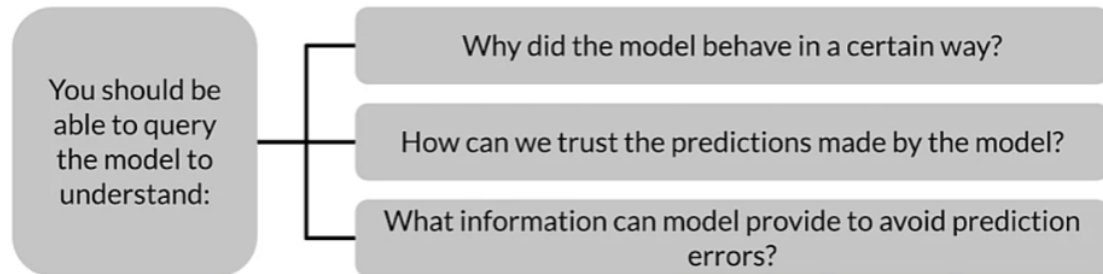
- **Explainable AI:** Garante fairness, identifica bias e problemas, debugar o modelo, etc.

## Responsible AI

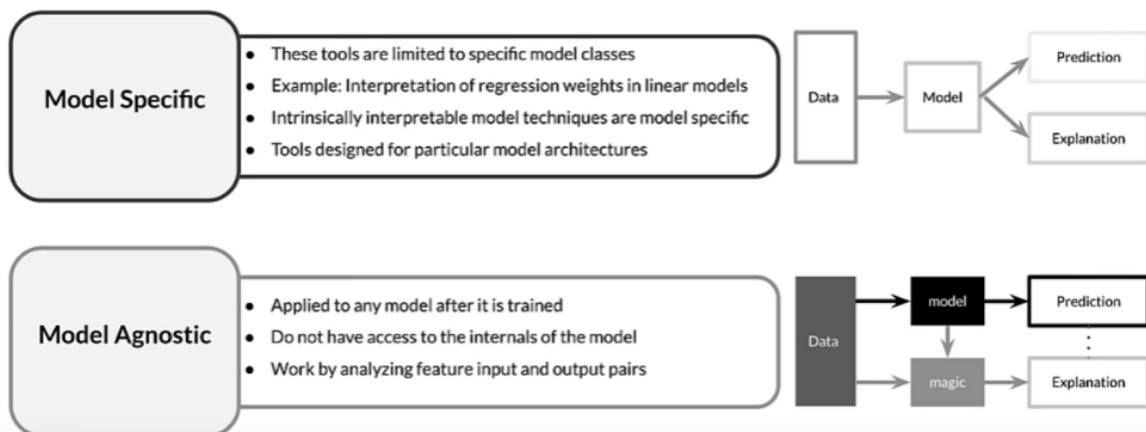
- Development of AI is creating new opportunities to improve lives of people
- Also raises new questions about the best way to build the following into AI systems:



## What are the requirements?

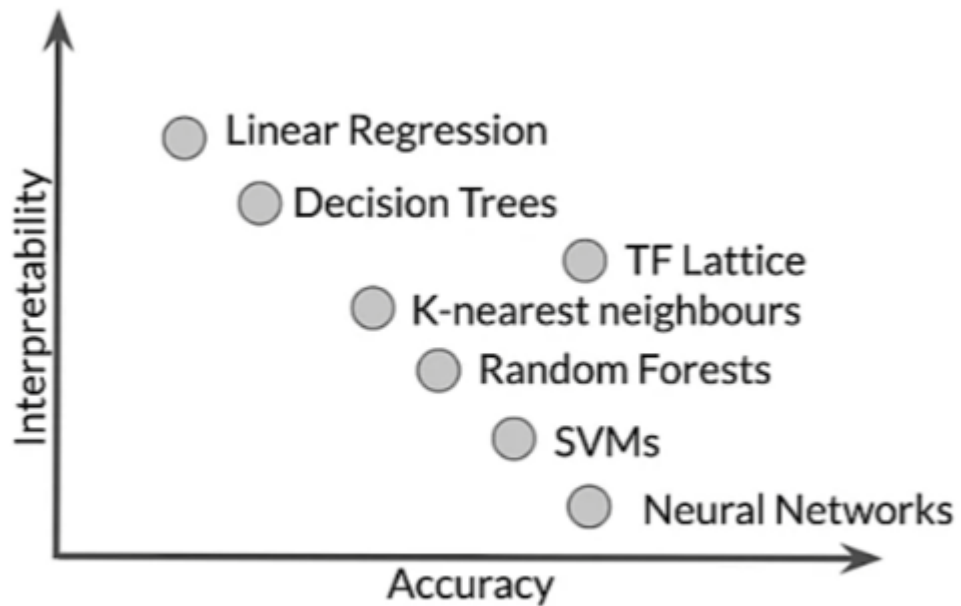


## Model Specific or Model Agnostic



## Interpretable Models

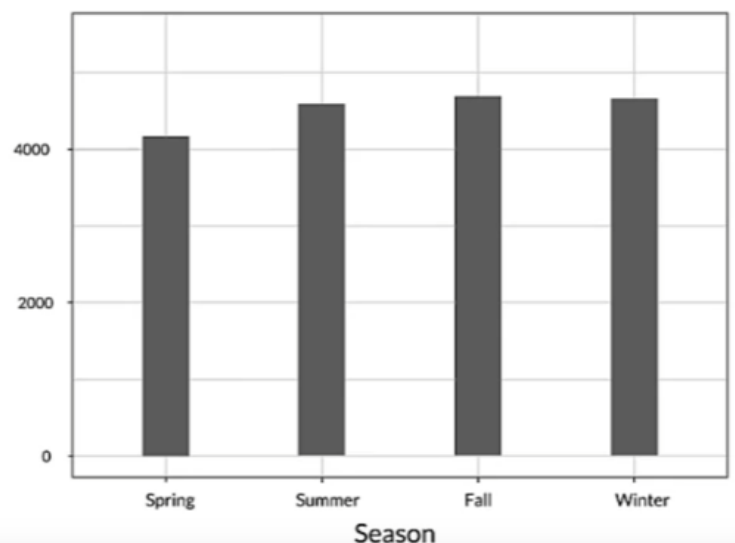
Algorithm	Linear	Monotonic	Feature Interaction	Task
Linear regression	Yes	Yes	No	regr
Logistic regression	No	Yes	No	class
Decision trees	No	Some	Yes	class, regr
RuleFit	Yes*	No	Yes	class, regr
K-nearest neighbors	No	No	No	class, regr
TF Lattice	Yes*	Yes	Yes	class, regr



**Interpretability vs Accuracy Trade off**

- 
- Model agnostic methods: Métodos que separam as explicações do modelo de ML, busca ter flexibilidade de modelo, explicabilidade e representação
- **PDP. Partial dependence plots** ajudam a entender os efeitos que recursos específicos têm nos resultados do modelo que você está vendo e o tipo de relacionamento entre esses recursos e os destinos ou rótulos em seus dados de treinamento. Normalmente concentra-se no impacto marginal causado por um ou dois recursos nos resultados do modelo. PDP é interessante pois é intuitivo e representa como uma feature influencia numa predição, além de ser fácil de implementar, entretanto assume que features não tem interações entre si, o que nem sempre é verdade

## PDP for Categorical Features



- A importância do recurso de permutação é apenas uma maneira de medir a importância de um recurso. Permutar um recurso é apenas uma maneira de quebrar a relação entre um recurso e o resultado do modelo. Essencialmente, atribuindo um valor quase aleatório ao recurso. Para a importância do recurso de permutação, medimos a importância de um recurso medindo o aumento no erro de previsão após a permuta desse recurso. Um recurso é importante se embaralhar seus valores aumenta o erro do modelo. A importância do recurso de permutação tem uma boa interpretação porque a importância do recurso é o aumento no erro do modelo quando as informações do recurso são destruídas. É uma visão global altamente compactada do comportamento do modelo. Como ao permutar o recurso você também destrói os efeitos de interação com outros recursos, ele também mostra as interações entre os recursos. Isso significa que ele leva em conta tanto os efeitos de características principais quanto os efeitos de interação no desempenho do modelo. Uma grande vantagem é que não requer retreinamento do modelo. Alguns outros métodos sugerem excluir um recurso, treinar novamente o modelo e comparar o erro do modelo. Já que retreinar um modelo pode levar muito tempo, não exigir isso é uma grande vantagem. uma desvantagem é não saber se usa teste ou treinamento e também a correlação de features
- **Shapley value** , em teoria de jogos, é um método para designar pagamentos para jogadores dependendo da sua contribuição para o total.
- Em ML o player é uma feature, diz o quanto uma feature contribui para o resultado

## Shapley Value

Term in Game Theory	Relation to ML	Relation to House Prices Example
Game	Prediction task for single instance of dataset	Prediction of house prices for a single instance
Gain	Actual prediction for instance - Average prediction for all instances	Prediction for house price (€300,000) - Average Prediction(€310,000) = -€10,000
Players	Feature values that contribute to prediction	'Park=nearby', 'cat=banned', 'area=50m²', 'floor=2nd'

- É computacionalmente caro, pode ser facilmente mal interpretado, é útil quando se tem poucas features, existe hoje em dia frameworks que

extendem o shapley value

## SHAP

- SHAP (SHapley Additive exPlanations) is a framework for Shapley Values which assigns each feature an importance value for a particular prediction
- Includes extensions for:
  - TreeExplainer: high-speed exact algorithm for tree ensembles
  - DeepExplainer: high-speed approximation algorithm for SHAP values in deep learning models
  - GradientExplainer: combines ideas from Integrated Gradients, SHAP, and SmoothGrad into a single expected value equation
  - KernelExplainer: uses a specially-weighted local linear regression to estimate SHAP values for any model

- Shapley values podem ser vistos como forças em um dataset e nos seus resultados
- **Concept Activation Vectors (CAV)** explica o estado interno de uma rede em termos de fácil entendimento para humanos
- **Local interpretable Model agnostic Explanations (LIME)**: Implementa modelos que são usados para explicar previsões individuais, usa datapoints perto de previsões individuais para aproximar as previsões do modelo real
- 
- 
- 
- 
- 
- 
- 
- 
-