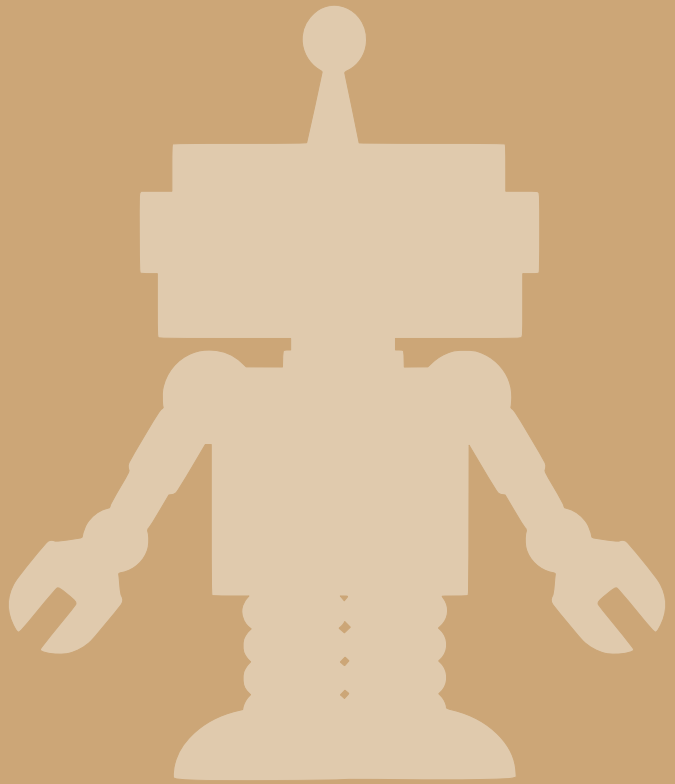
Aprendizado de Máquina 2

Aula 3

Professora: Patrícia Pampanelli

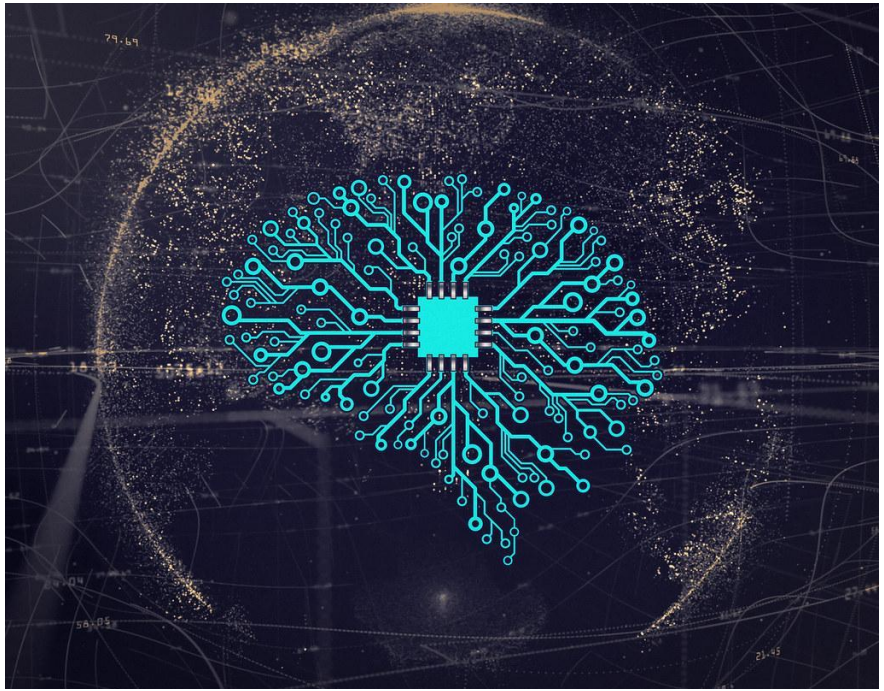
patricia.pampanelli@usp.br



Dúvidas da última aula?

Aula de Hoje

- XGBoost
- Balanceamento de Datasets
- Exercícios



XGBoost

XGBoost

- O modelo XGBoost foi desenhado para trabalhar com datasets de alta complexidade
- Assim como o Gradient Boosting, ele é um modelo sequencial
- Normalmente, as árvores são limitadas a 6 níveis

XGBoost

- O XGBoost introduz alguns conceitos fundamentais para evitar o overfitting e tornar o modelo mais eficiente:
  - Prune (poda)
  - Regularização

XGBoost - Prune

- São utilizadas **técnicas de prune (poda)** para tornar as árvores mais eficientes. O hiperparâmetro utilizado para o limiar de poda se chama **gama**
- A poda de uma árvore é decidida com base no ganho obtido com um determinado split.
- Se **ganho - gamma < 0**:
  - A sub-árvore é removida
- Se **ganho - gamma >= 0**:
  - A sub-árvore é mantida

XGBoost - Regularização

- Além da técnica de prune, é utilizada a técnica de **regularização**
- A **variável de regularização**, chamada **lambda**, é utilizada para tornar o modelo **menos sensível a observações específicas**
- Isso previne que o modelo tenha overfitting em relação aos dados de treinamento

XGBoost - Regularização

- A **variável de regularização**, chamada **lambda** é utilizada no denominador ao calcularmos a medida de similaridade em uma folha:

$$similaridade = \frac{\sum residuos\ da\ folha}{numero\ de\ amostras\ na\ folha + \mathbf{lambda}}$$

- Quanto maior o valor de lambda, menor a medida de similaridade
- Isso faz com que uma determinada sub-árvore tenha uma chance maior de ser podada

XGBoost - Learning rate (eta)

- Assim como no Gradient Boosting, as contribuições de cada árvore são multiplicadas pelo *learning rate*.
- No contexto do XGBoost, este valor se chama ***eta***
- O valor padrão para ***eta*** é de 0.3

Balanceamento de Datasets

Detecção de fraude

Problema: Detecção de fraude em transações é uma aplicação bastante importante e essencial em diversas indústrias (mercado financeiro, seguros, etc).

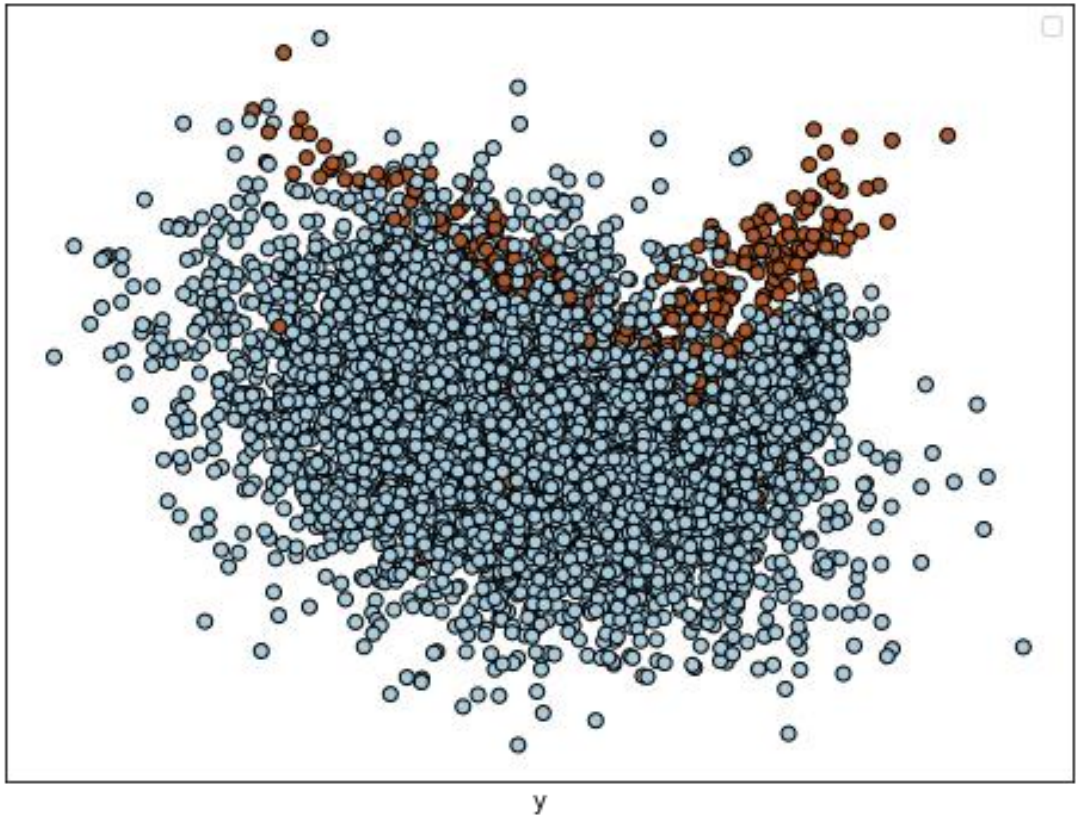
Dataset

Vamos utilizar os seguintes dados com 10.000 amostras, sendo:

- 337 de transações fraudulentas
- 9663 de transações legítimas

3% das transações são fraude!

Este dataset é bastante desbalanceado, mas nos aproxima de situações reais.



Baseline

Antes de avaliarmos as estratégias de balanceamentos vamos um modelo sem nenhuma estratégia de balanceamento. O modelos *"mais simples"* que podemos obter.

Nós temos com este baseline:

- 36 fraudes identificadas corretamente (total de 116)
- 3169 transações normais identificadas corretamente (total de 3184)

Em outras palavras:

- **31% das fraudes foram identificadas**
- **71% das fraudes identificadas eram realmente fraudes**

```
[ ]  1 logreg_baseline = LogisticRegression(max_iter=200)
     2 logreg_baseline.fit(X_train, y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=200,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
[ ]  1 y_pred_baseline = logreg_baseline.predict(X_test)
```

```
[ ]  1 from sklearn.metrics import confusion_matrix
     2 confusion_matrix(y_test, y_pred_baseline)
```

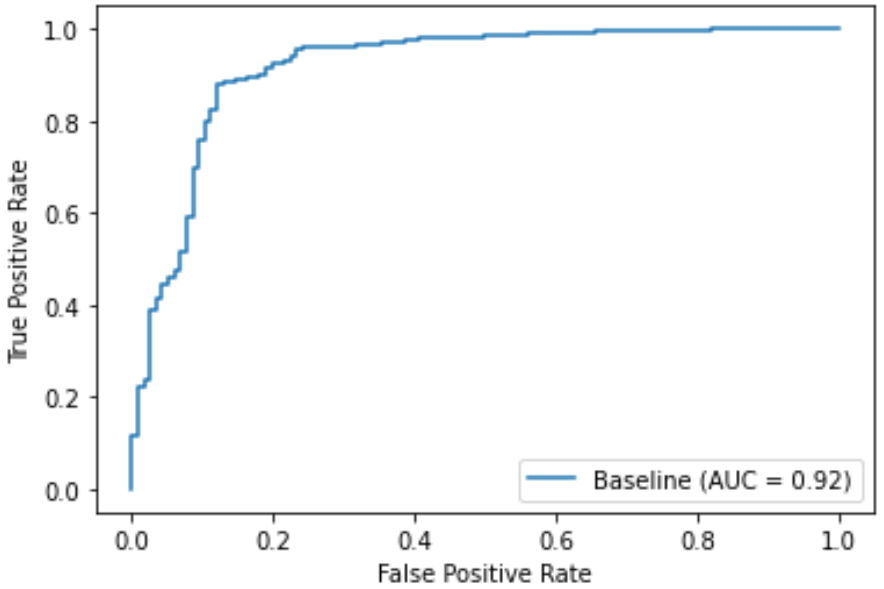
```
array([[  36,   80],
       [  15, 3169]])
```

```
[ ]  1 from sklearn.metrics import classification_report
     2 print(classification_report(y_test, y_pred_baseline, target_names=['Fraud', 'Not Fraud']))
```

```
              precision    recall  f1-score   support

       Fraud       0.71      0.31      0.43       116
   Not Fraud       0.98      1.00      0.99      3184

    accuracy                           0.97      3300
   macro avg       0.84      0.65      0.71      3300
weighted avg       0.97      0.97      0.97      3300
```

Modelos

Estratégia	Acurácia	Precision	Recall	F_1 Score	AUC
Baseline	0.97	0.71	0.31	0.43	0.92



Pesos entre as classes

No sklearn nós temos duas possibilidades para fazer esse ajuste de pesos através do parâmetro *class_weight*:

- *'balanced'*
- Customizada (dicionário)

class_weight : *dict or 'balanced', default=None*

Weights associated with classes in the form `{class_label: weight}`. If not given, all classes are supposed to have weight one.

The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as `n_samples / (n_classes * np.bincount(y))`.

Note that these weights will be multiplied with sample_weight (passed through the fit method) if sample_weight is specified.

New in version 0.17: class_weight='balanced'

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Show me the code!

- 88% das fraudes foram identificadas
- 17% das fraudes identificadas eram realmente fraudes

Quantas transações normais foram identificadas como fraudes?

```
1 logreg = LogisticRegression(class_weight='balanced')
2 logreg.fit(X_train, y_train)
3 y_pred = logreg.predict(X_test)
```

```
1 from sklearn.metrics import confusion_matrix
2 confusion_matrix(y_test, y_pred)
```

```
array([[ 102,   14],
       [ 504, 2680]])
```

```
1 from sklearn.metrics import classification_report
2 print(classification_report(y_test, y_pred, target_names=['Fraud', 'Not Fraud']))
```

```
              precision    recall  f1-score   support

       Fraud       0.17      0.88      0.28       116
   Not Fraud       0.99      0.84      0.91      3184

    accuracy                           0.84      3300
   macro avg       0.58      0.86      0.60      3300
weighted avg       0.97      0.84      0.89      3300
```

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Modelos

Estratégia	Acurácia	Precision	Recall	F_1 Score	AUC
Baseline	0.97	0.71	0.31	0.43	0.92
Balanced classes	0.84	0.17	0.88	0.28	0.92

O **balanceamento forçado** das classes **não teve uma performance muito boa**. Observamos uma piora nas métricas de avaliação do modelo.

A quantidade de **Falsos Positivos** faz com que o **volume de transações bloqueadas seja muito alto**. Isso prejudica muito a experiência dos usuários e, portanto, também não resolve o problema de detecção de fraude.

Show me the code!

Vamos experimentar o rebalanceamento manual, com ele obtemos:

- 65% das fraudes foram identificadas
- 44% das fraudes identificadas eram realmente fraudes

```
1 logreg = LogisticRegression(class_weight={0: 0.85,1: 0.15})
2 logreg.fit(X_train, y_train)
3 y_pred = logreg.predict(X_test)
```

```
1 from sklearn.metrics import confusion_matrix
2 confusion_matrix(y_test, y_pred)
```

```
array([[  75,   41],
       [  94, 3090]])
```

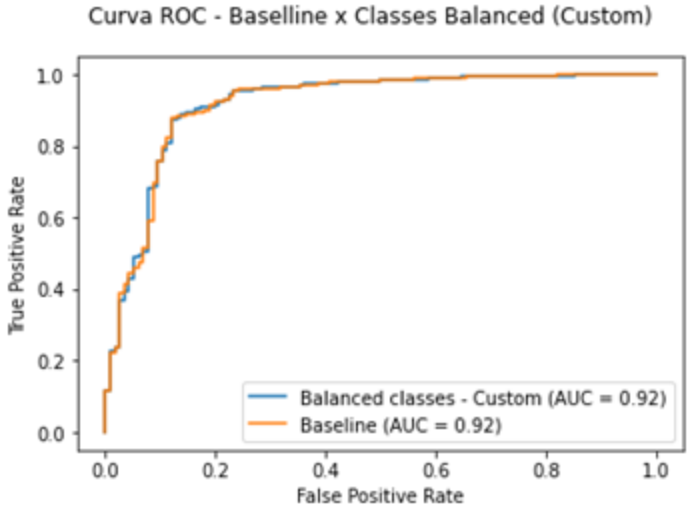
```
1 from sklearn.metrics import classification_report
2 print(classification_report(y_test, y_pred, target_names=['Fraud', 'Not Fraud']))
```

```
              precision    recall  f1-score   support

       Fraud       0.44      0.65      0.53       116
   Not Fraud       0.99      0.97      0.98      3184

    accuracy                           0.96      3300
   macro avg       0.72      0.81      0.75      3300
weighted avg       0.97      0.96      0.96      3300
```

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Modelos

Estratégia	Acurácia	Precision	Recall	F_1 Score	AUC
Baseline	0.97	0.71	0.31	0.43	0.92
Balanced classes	0.84	0.17	0.88	0.28	0.92
Custom balance	0.96	0.44	0.65	0.53	0.92



Imbalanced learn

Para fazer o processo de balanceamento dos datasets de treinamento, nós vamos utilizar um pacote do Python bastante parecido com o Scikit Learn.

O Imbalanced learn implementa diversas metodologias para balanceamento de datasets. Além disso, ele implementa alguns métodos que encapsulam o balanceamento e a classificação.

Por fim, ele também tem integração com outros Frameworks como Tensorflow e Keras.

Over-sampling

A abordagem de *over-sampling* consiste em **duplicar de forma randômica** as **amostras mais raras** no dataset. Uma outra forma bastante popular é utilizando um algoritmo de **geração sintética de dados** (Ex: Smote, ADASYN, etc).

No caso da detecção de fraude, **duplicar** as amostras que representam **transações fraudadas**.

Over-sampling - SMOTE

O algoritmo **Smote (*Synthetic Minority Over-sampling*)** gera novas amostras através de interpolação linear a partir das amostras já existentes no dataset. Esta operação é feita selecionando os vizinhos mais próximos de uma amostras e criando novos pontos entre elas.

https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html#imblearn.over_sampling.SMOTE

Over-sampling - ADASYN

Já o algoritmo **ADASYN (*Adaptive Synthetic (ADASYN) algorithm*)** funciona de forma similar ao Smote com a diferença de que ele gera novas amostras no dataset levando em consideração a distribuição da classe minoritária.

https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.ADASYN.html#imblearn.over_sampling.ADASYN

SMOTE x ADASYN

A principal diferença entre estes dois algoritmos é que o ADASYN leva em **consideração as amostras** da classe minoritária que foram **mais difíceis de aprender**. Essa análise é feita através do classificador K-Nearest Neighbors.

Ponto de atenção: quando os dados da classe minoritária são esparsos.

https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.ADASYN.html#imblearn.over_sampling.ADASYN

Show me the code: The imbalanced-learn package

```
1 from imblearn.over_sampling import ADASYN
2
3 sm = ADASYN(random_state=42, sampling_strategy=0.22)
4 X_over, y_over = sm.fit_resample(X_train, y_train)
5 print('Resultado após o oversampling %s' % Counter(y_over))
```

```
Resultado após o oversampling Counter({1: 6479, 0: 1399})
```

```
[125] 1 from sklearn.metrics import confusion_matrix
      2 confusion_matrix(y_test, y_pred)
```

```
array([[  77,   39],
       [ 102, 3082]])
```

```
[126] 1 from sklearn.metrics import classification_report
      2 print(classification_report(y_test, y_pred, target_names=['Fraud', 'Not Fraud']))
```

	precision	recall	f1-score	support
Fraud	0.43	0.66	0.52	116
Not Fraud	0.99	0.97	0.98	3184
accuracy			0.96	3300
macro avg	0.71	0.82	0.75	3300
weighted avg	0.97	0.96	0.96	3300

https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.ADASYN.html#imblearn.over_sampling.ADASYN

Modelos

Estratégia	Acurácia	Precision	Recall	F_1 Score	AUC
Baseline	0.97	0.71	0.31	0.43	0.92
Balanced classes	0.84	0.17	0.88	0.28	0.92
Custom balance	0.96	0.44	0.65	0.53	0.92
Over sampling ADASYN (22:100)	0.95	0.43	0.66	0.52	0.92

Under-sample ou *Down-*

A abordagem de *under-sampling* consiste em **remover** as **amostras abundantes** no dataset. Uma das abordagens é fazer isso de forma **randômica**. Nós podemos também fazer isso utilizando um algoritmo de reamostragem baseado no **K-means (algoritmo de clusterização)**.

No caso da detecção de fraude, **remover** as amostras que representam **transações legítimas**.

Show me the code: The imbalanced-learn package

```
[947]  1 from imblearn.under_sampling import ClusterCentroids
       2
       3 cc = ClusterCentroids(random_state=42, sampling_strategy=0.35)
       4 X_under, y_under = cc.fit_resample(X_train, y_train)
       5 print('Resultado após o under sampling %s' % Counter(y_under))
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py
  warnings.warn(msg, category=FutureWarning)
Resultado após o under sampling Counter({1: 631, 0: 221})
```

```
[956]  1 from sklearn.metrics import classification_report
       2 print(classification_report(y_test, y_pred, target_names=['Fraud', 'Not Fraud']))
```

	precision	recall	f1-score	support
Fraud	0.39	0.58	0.47	116
Not Fraud	0.98	0.97	0.98	3184
accuracy			0.95	3300
macro avg	0.69	0.77	0.72	3300
weighted avg	0.96	0.95	0.96	3300

```
1 from sklearn.metrics import confusion_matrix
2 confusion_matrix(y_test, y_pred)
```

```
array([[  67,   49],
       [ 104, 3080]])
```

Modelos

Estratégia	Acurácia	Precision	Recall	F_1 Score	AUC
Baseline	0.97	0.71	0.31	0.43	0.92
Balanced classes	0.84	0.17	0.88	0.28	0.92
Custom balance	0.96	0.44	0.65	0.53	0.92
Over sampling ADASYN (22:100)	0.95	0.43	0.66	0.52	0.92
Under sampling	0.95	0.39	0.58	0.47	0.91

Under sampling + Over-sampling
Proporção de 1:10

Nós podemos também utilizar estas duas técnicas combinadas para obtermos um dataset de treinamento mais balanceado.

Nesta estratégia combinada são reamostradas ambas as classes (fraudes e não fraudes).

Modelos

Estratégia	Acurácia	Precision	Recall	F_1 Score	AUC
Baseline	0.97	0.71	0.31	0.43	0.92
Balanced classes	0.84	0.17	0.88	0.28	0.92
Custom balance	0.96	0.44	0.65	0.53	0.92
Over sampling ADASYN (22:100)	0.95	0.43	0.66	0.52	0.92
Under sampling	0.95	0.39	0.58	0.47	0.91
Over sampling e Under sampling (1:10)	0.95	0.51	0.59	0.55	0.92

Exercícios

Exercícios

- Exercício em sala:
  - **Tópicos:** Árvores de Decisão, Random Forest, XGBoost, Gini Impurity, Inferência e Deploy de modelos
  - **Objetivos:**
    - Treinamento de modelos (Decision Tree Classifier e Random Forest)
    - Visualizar as árvores que foram construídas
    - Métricas de classificação (matriz de confusão)
    - Verificar o cálculo do da medida de impureza
  - **Dataset:** MNIST

Trabalho

- Exercício: XGboost
- **Tópicos:**
  - XGBoost
  - Deploy e Inferência
- **Dataset:** livre escolha
- **Entrega:** 13/08 até 11:59

Resumo da Aula de Hoj

- XGBoost
- Balanceamento de Datasets
- Exercícios

Dúvidas?

Obrigada!