Análise de Dados Aplicada à Computação

STATISTICAL MACHINE LEARNING

Prof. M.Sc. Howard Roatti

Aprendizado de Máquina Estatístico

- Diferente dos métodos vistos que criam uma estrutura linear impondo a separação dos dados, veremos algoritmos capazes de identificar novas amostras analisando os dados.
- O **kNN** classifica de acordo com sua vizinhança conhecida
- A **árvore de decisão** aprende regras de relacionamento entre preditoras e resultantes
- Os métodos de ***bagging*** e ***boosting*** unem vários modelos para gerar resultados diferentes de um único modelo

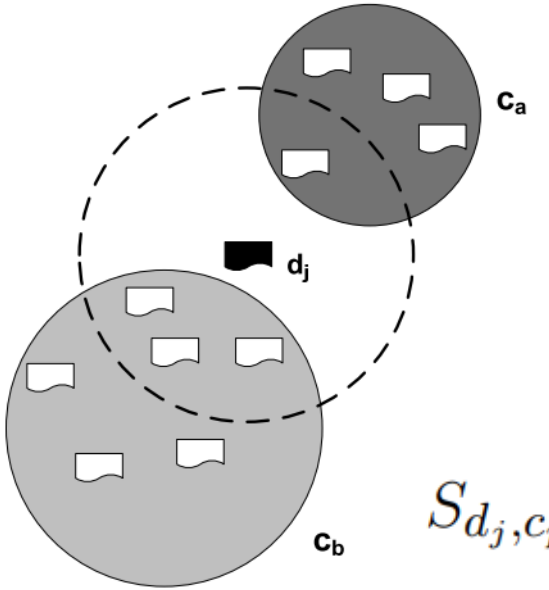
k-Nearest Neighbor

Aprendizado de Máquina Estatístico

- **k-Nearest Neighbor** (**kNN**): classificador sob demanda/preguiçoso
- Classificadores preguiçosos não constroem um modelo a priori
- A classificação é feita quando uma nova amostra d_j é apresentada
- Baseado nos rótulos dos vizinhos mais próximos à d_j
  - determine os k vizinhos mais próximos de d_j no conjunto de treino
  - utilize a classe majoritária desses vizinhos para determinar o rótulo para d_j
  - ***no caso de regressão, calcule a média das amostras vizinhas e associe a d_j***

Aprendizado de Máquina Estatístico

- **k-Nearest Neighbor (kNN)**:



k = 4

C_a = 1

C_b = 3

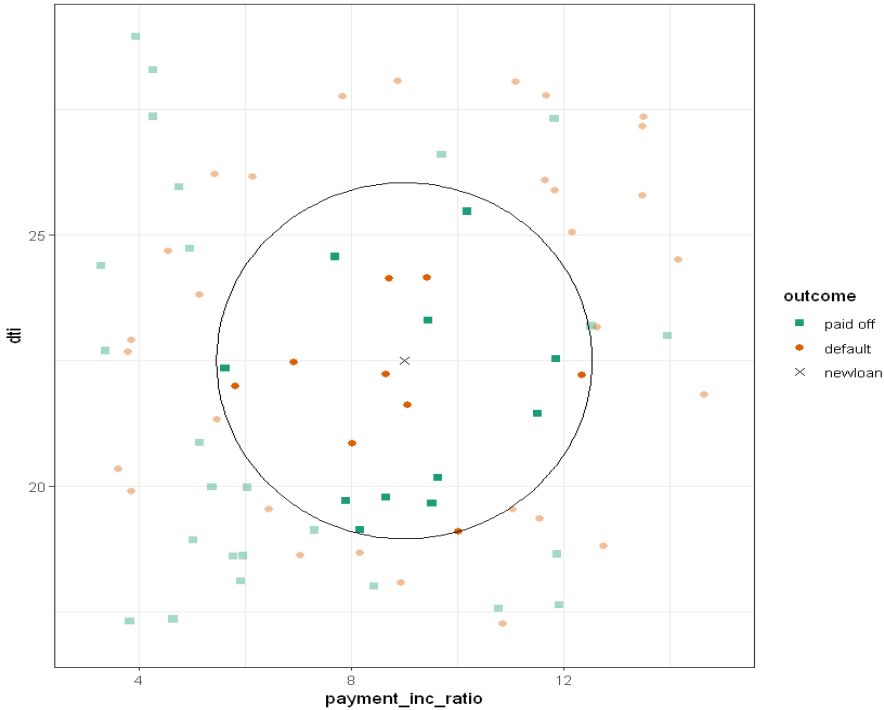
Probabilidade C_a = C_a / k => 1 / 4 => 0.25 => 25%

Probabilidade C_b = C_b / k => 3 / 4 => 0.75 => 75%

$$S_{d_j,c_p} = \sum_{d_t \in N_k(d_j)} similarity(d_j, d_t) \times \mathcal{T}(d_t, c_p)$$

Aprendizado de Máquina Estatístico

- **k-Nearest Neighbor** (**kNN**): utiliza como métricas de similaridade a distância euclidiana, similaridade cosseno, distância Manhattan, distância de Minkowski, distância de Jaccard ou distância de Hamming (LINK)
- Devido a essa natureza de similaridade, kNN só atua com números, portanto, métodos como **one-hot-encoder, LabelEncoder DEVEM** ser utilizados para representar dados categóricos



Aprendizado de Máquina Estatístico

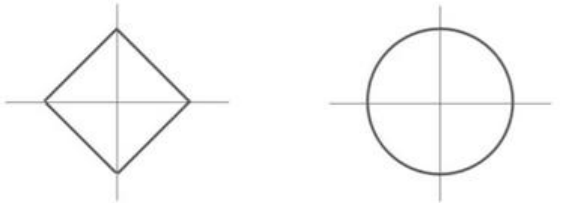
- **k-Nearest Neighbor (kNN):**

(Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

(Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$

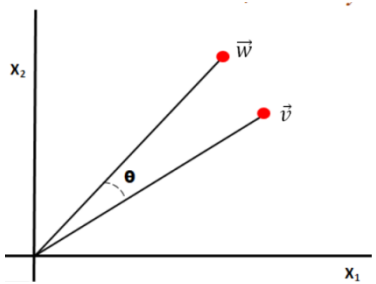


(Minkowski) distance

$$d_3(I_1, I_2) = \sqrt[\lambda]{\sum_P |I_1^P - I_2^P|^\lambda}$$

(cosine) Similarity

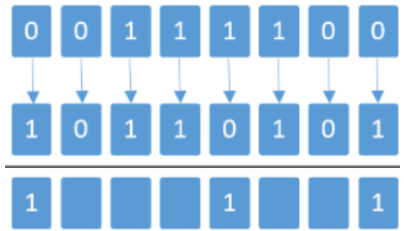
$$D(\vec{w}, \vec{v}) = \frac{\vec{w} \cdot \vec{v}}{\|\vec{w}\| \|\vec{v}\|} = \cos\theta$$



(Hamming) distance

$$D(\vec{w}, \vec{v}) = \sum_i^n I(w_i, v_i)$$

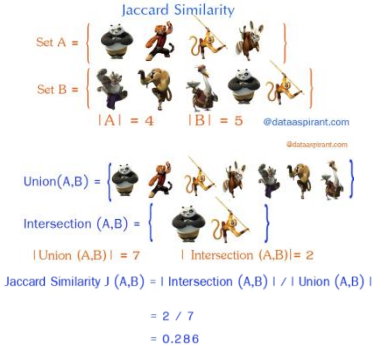
I(x,y) = 0 if identical, 1 if different.



Hamming Distance = 3

(Jaccard) Similarity

$$J(\vec{w}, \vec{v}) = \frac{|\vec{w} \cap \vec{v}|}{|\vec{w} \cup \vec{v}|}$$



https://dataaspirant.com/five-most-popular-similarity-measures-implementation-in-python/
https://www.ufrgs.br/wiki-r/index.php?title=C%C3%A1lculo_de_similaridade

Aprendizado de Máquina Estatístico

- **k-Nearest Neighbor (kNN)**: como a maioria das métricas de similaridade são suscetíveis às grandezas dos atributos, fazendo com que muitas das vezes as características com maiores grandezas tenham mais impactos no resultado a métrica. Sugere-se a utilização do reescalonamento dos atributos utilizado:

- **Padronização** $$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

- **Normalização** $$z = \frac{x_i - \mu}{\sigma}$$

https://www.kdnuggets.com/2020/04/data-transformation-standardization-normalization.html

Aprendizado de Máquina Estatístico

- **k-Nearest Neighbor (kNN)**: Python

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import preprocessing

loan_data = pd.read_csv(LOAN_DATA_CSV)
loan_data = loan_data.drop(columns=['Unnamed: 0', 'status'])
loan_data['outcome'] = pd.Categorical(loan_data['outcome'],
                                      categories=['paid off', 'default'],
                                      ordered=True)

newloan = loan_data.loc[0:0, ['payment_inc_ratio', 'dti', 'revol_bal', 'revol_util']]
print(newloan)
X = loan_data.loc[1:, ['payment_inc_ratio', 'dti', 'revol_bal', 'revol_util']]
y = loan_data.loc[1:, 'outcome']

knn = KNeighborsClassifier(n_neighbors=5).fit(X, y)
nbrs = knn.kneighbors(newloan)
print(X.iloc[nbrs[1][0], :])
```

```
   payment_inc_ratio  dti  revol_bal  revol_util
0             2.3932  1.0       1687         9.4
```

```
       payment_inc_ratio   dti  revol_bal  revol_util
35536            1.47212  1.46       1686        10.0
33651            3.38178  6.37       1688         8.4
25863            2.36303  1.39       1691         3.5
42953            1.28160  7.14       1684         3.9
43599            4.12244  8.98       1684         7.2
```

Aprendizado de Máquina Estatístico

- **k-Nearest Neighbor (kNN)**: Python

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import preprocessing

scaler = preprocessing.StandardScaler()
scaler.fit(X)

X_std = scaler.transform(X)
newloan_std = scaler.transform(newloan)

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_std, y)

nbrs = knn.kneighbors(newloan_std)
print(X.iloc[nbrs[1][0], :])
```

```
   payment_inc_ratio  dti  revol_bal  revol_util
0             2.3932  1.0       1687         9.4
       payment_inc_ratio   dti  revol_bal  revol_util
2080             2.61091  1.03       1218         9.7
1438             2.34343  0.51        278         9.9
30215            2.71200  1.34       1075         8.5
28542            2.39760  0.74       2917         7.4
44737            2.34309  1.37        488         7.2
```

Aprendizado de Máquina Estatístico

- **k-Nearest Neighbor** (**kNN**): selecionar o parâmetro **k** não é simples: **valores pequenos** significam que **ruídos influenciarão os resultados**, **valores maiores** tornam a **resposta lenta**
- Geralmente **opta-se por números ímpares** para que **não** haja **empate** na etapa democrática do método
- *Não foi provado ainda*: mas existem fontes na web que associam a **k = sqrt(n)**

Aprendizado de Máquina Estatístico

- **k-Nearest Neighbor (kNN)**: Python

```
from sklearn.neighbors import KNeighborsClassifier

loan200 = pd.read_csv('./DataSets/loan200.csv‘)

#Remove a primeira amostra utilizada para outro experimento
X = loan200.loc[1:, ['payment_inc_ratio', 'dti']]
y = loan200.loc[1:, 'outcome']
#Train Test Split

knn = KNeighborsClassifier(n_neighbors=20)
knn.fit(X_train, y_train)

pred = knn.predict(X_test)
#Métricas de Avaliação – o kNN retorna o rótulo aplicado
```

Aprendizado de Máquina Estatístico

- **k-Nearest Neighbor (kNN)**: podemos utilizar o kNN como motor para gerar uma nova características:
  1. **Executa-se sobre todos os dados realizando previsão para todos, nesse caso o que devemos utilizar como nova feature é a probabilidade obtida**
  2. **Adiciona-se essa nova feature para cada registro e utiliza-se outro classificador sobre os dados.**

Aprendizado de Máquina Estatístico

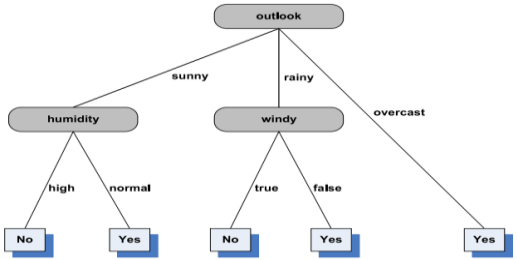
- **k-Nearest Neighbor (kNN)**: Python

```
loan_data = pd.read_csv(LOAN_DATA_CSV)
loan_data = loan_data.drop(columns=['Unnamed: 0', 'status'])
loan_data['outcome'] = pd.Categorical(loan_data['outcome'],
                                      categories=['paid off', 'default'], ordered=True)
predictors = ['dti', 'revol_bal', 'revol_util', 'open_acc', 'delinq_2yrs_zero',
'pub_rec_zero']
X = loan_data[predictors]
y = loan_data['outcome']

knn = KNeighborsClassifier(n_neighbors=20)
knn.fit(X, y)
plt.scatter(range(len(X)), [bs + random.gauss(0, 0.015) for bs in
knn.predict_proba(X)[:,0]],
            alpha=0.1, marker='.‘)

loan_data['borrower_score'] = knn.predict_proba(X)[:, 0]
print(loan_data['borrower_score'].describe())
```

Decision Tree

Aprendizado de Máquina Estatístico

- **Decision Tree** (**DT**): Uma árvore de decisão é um método de classificação/regressão que constrói regras de classificação organizadas como **caminhos em uma árvore**.
- Uma das vantagens da abordagem é que as regras na árvore são **passíveis de interpretação humana**
- As regras são criadas no formato **“if-then-else”**
- Tem a Habilidade de descobrir padrões escondidos



Aprendizado de Máquina Estatístico

- **Decision Tree (DT)**: Python

```
from sklearn.tree import DecisionTreeClassifier

loan3000 = pd.read_csv('./DataSets/loan3000.csv‘)

X = loan3000[predictors]
y = loan3000[outcome]

#Train Test Split

loan_tree = DecisionTreeClassifier(random_state=1, criterion='entropy',
                                   min_impurity_decrease=0.003)
loan_tree.fit(X_train, y_train)

pred = loan_tree.predict(X_test)
#Métricas de Avaliação – a DT retorna o rótulo aplicado
```

Aprendizado de Máquina Estatístico

- **Decision Tree** (**DT**): O Processo de Divisão usa a estratégia de divisão recursiva, por exemplo:
  1. selecione um dos atributos preditores como raiz
  2. use os valores dos atributos para dividir em subconjuntos
  3. para cada subconjunto, selecione outros atributos e divida em subconjuntos
  4. repita o passo anterior até que não seja mais possível obter subconjuntos e o atributo alvo seja alcançado nas folhas

Aprendizado de Máquina Estatístico

- **Decision Tree (DT):**

Impurity Criterion

Gini Index

$$I_G = 1 - \sum_{j=1}^{c} p_j^2$$

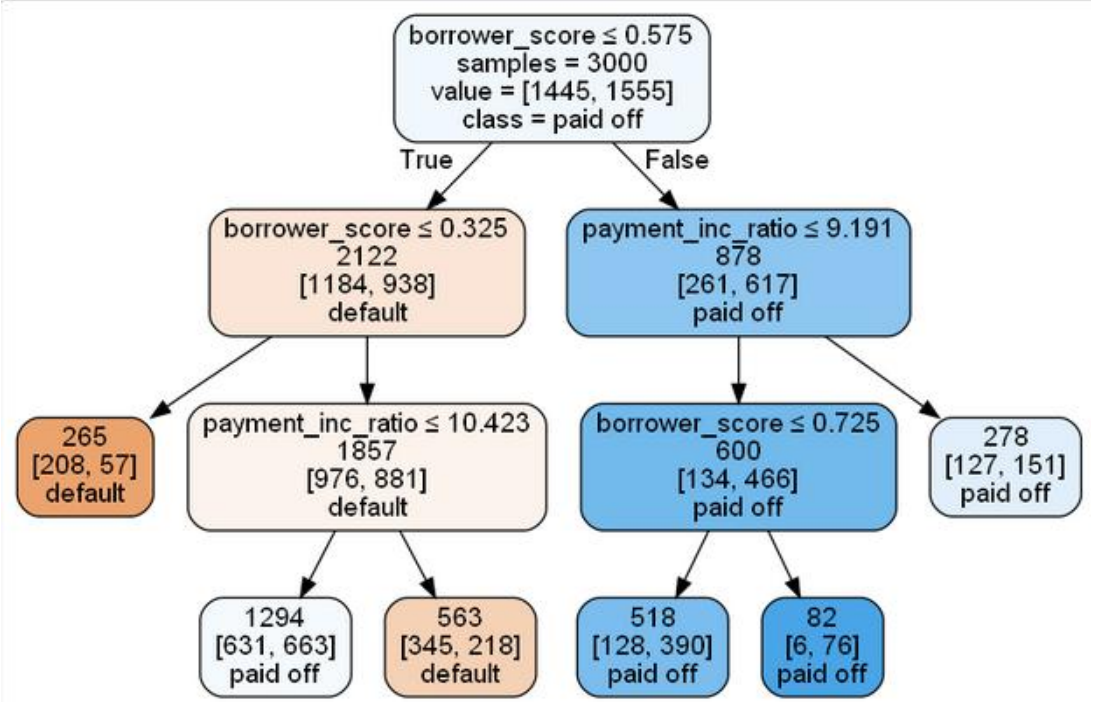
p_j: proportion of the samples that belongs to class c for a particular node

Entropy

$$I_H = -\sum_{j=1}^{c} p_j log_2(p_j)$$

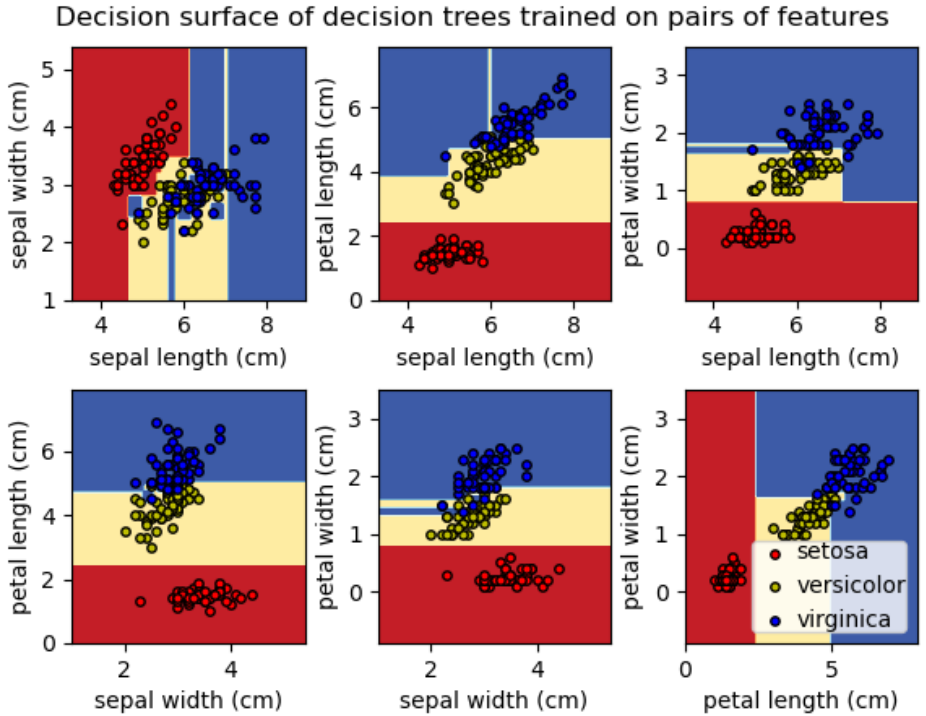
p_j: proportion of the samples that belongs to class c for a particular node.

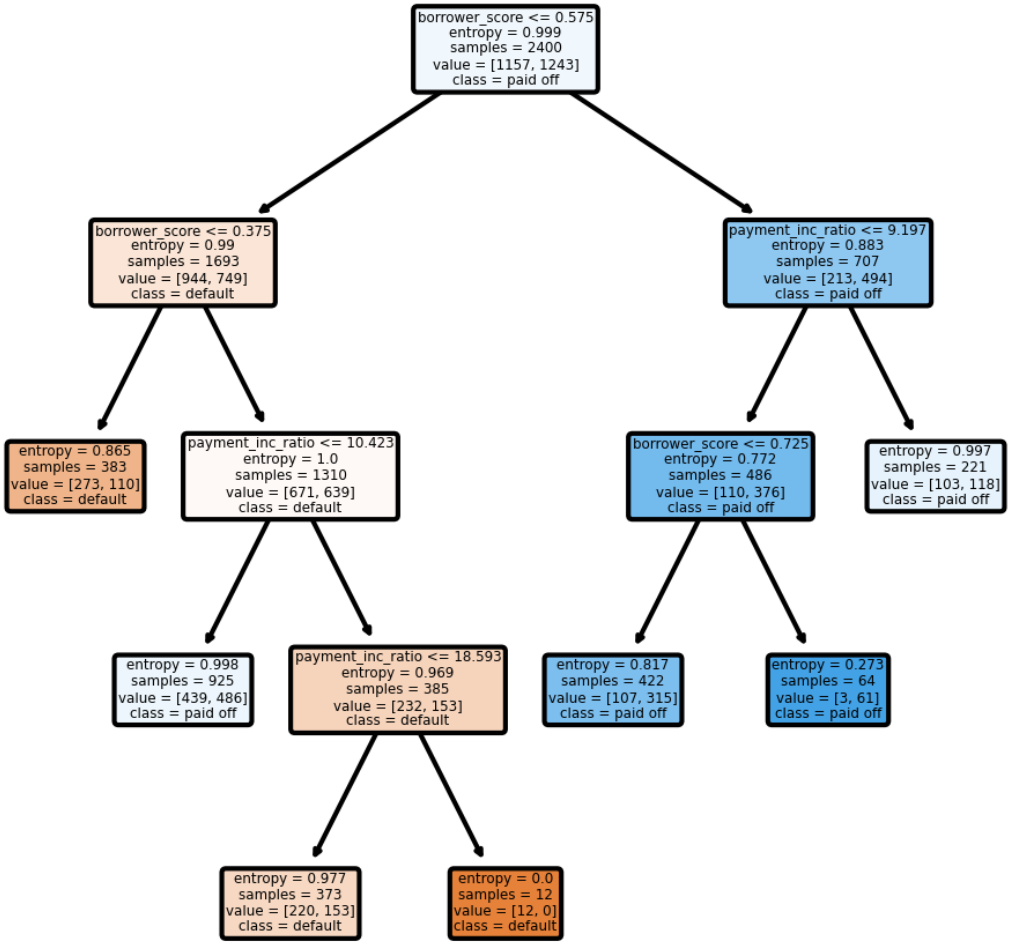
*This is the the definition of entropy for all non-empty classes ($p \neq 0$). The entropy is 0 if all samples at a node belong to the same class.

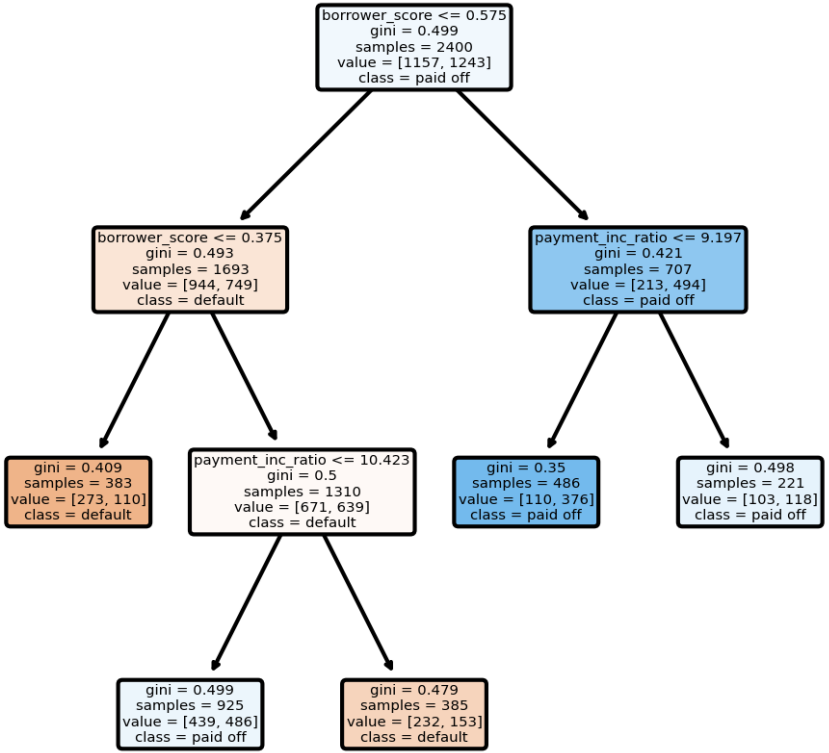


Aprendizado de Máquina Estatístico

- **Decision Tree (DT):**







Criterion: Entropy

Criterion: Gini

Aprendizado de Máquina Estatístico

- **Decision Tree** (**DT**):

- O critério de Gini e o critério de entropia são medidas diferentes para calcular a impureza de um nó em uma árvore de decisão. Embora ambos sejam usados para tomar decisões sobre como dividir os dados, eles têm abordagens diferentes para avaliar a pureza dos nós e, portanto, podem levar a árvores de decisão diferentes.

- No geral, as diferenças podem ocorrer na estrutura da árvore, na ordem em que os recursos são selecionados para fazer divisões e nos limiares de divisão escolhidos em cada nó.

- Embora possam ser diferentes, não significa que uma seja melhor do que a outra em todos os casos. A escolha entre os critérios depende do problema, dos dados disponíveis e de outras considerações.

Aprendizado de Máquina Estatístico

- **Decision Tree** (**DT**): Quando se trata de previsão, é interessante a utilização de diversas árvores que tornam o modelo mais potente...
- ***Random Forest*** (RF) → Modelo Ensemble Bagging que agrupa diversas árvores para classificação
- ***XGBoost*** (XGB) → Modelo Ensembe Boosting que, além de agrupar diversas árvores, utiliza um modelo de otimização para acelerar o processamento e o desempenho

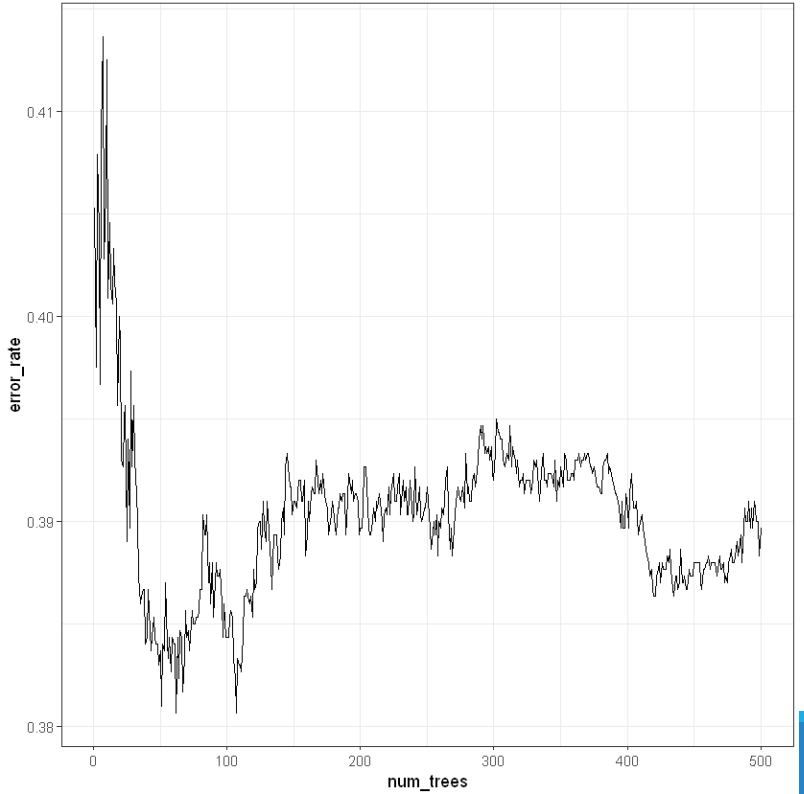
Random Forest

Aprendizado de Máquina Estatístico

- **Random Forest** (**RF**): Utiliza o conceito de ***bagging***, ***b****ootstrap* ***agg****regat****ing***.
- É um algoritmo de agrupamentos, porém, ao invés de criar várias árvores com os mesmos dados, cada árvore é gerada com um conjunto de amostras distintas (reamostras boostrap).
- O algoritmo envolve a definição do número de estimadores a serem criados, a reamostragem boostrap e o treinamento dos modelos. Gerando uma floresta aleatória

Aprendizado de Máquina Estatístico

- **Random Forest** (**RF**): a proporção que o número de árvores aumenta, a taxa de erro tende a diminuir ou estabilizar...



Aprendizado de Máquina Estatístico

- **Random Forest** (**RF**): Python

```
from sklearn.ensemble import RandomForestClassifier

loan3000 = pd.read_csv('./DataSets/loan3000.csv‘)

X = loan3000[['borrower_score', 'payment_inc_ratio']]
y = loan3000['outcome']

#Train Test Split

rf = RandomForestClassifier(n_estimators=500, random_state=1)
rf.fit(X_train, y_train)

pred = rf.predict(X_test)
#Métricas de Avaliação – a DT retorna o rótulo aplicado
```

Aprendizado de Máquina Estatístico

- **Random Forest** (**RF**): o modelo de florestas aleatórias constrói árvores determinando quais preditoras são importantes e descobre relacionamentos complexos entre as preditoras. Podemos utilizar esse poder do modelo para determinar quais atributos devem ser mantidos para criação de um modelo de alto desempenho.

Aprendizado de Máquina Estatístico

- **Random Forest** (**RF**): Python

```
predictors = ['loan_amnt', 'term', 'annual_inc', 'dti',
              'payment_inc_ratio', 'revol_bal', 'revol_util',
              'purpose', 'delinq_2yrs_zero', 'pub_rec_zero',
              'open_acc', 'grade', 'emp_length', 'purpose_',
              'home_', 'emp_len_', 'borrower_score']
outcome = 'outcome'

X = pd.get_dummies(loan_data[predictors], drop_first=True)
y = loan_data[outcome]

rf_all = RandomForestClassifier(n_estimators=500, random_state=1)
rf_all.fit(X, y)

rf_all_entropy = RandomForestClassifier(n_estimators=500, random_state=1,
                                        criterion='entropy')
print(rf_all_entropy.fit(X, y))
```

Aprendizado de Máquina Estatístico

- **Random Forest** (**RF**): Python

```
rf = RandomForestClassifier(n_estimators=500)
scores = defaultdict(list)

# crossvalidate the scores on a number of different random splits of the data
for _ in range(3):
    train_X, valid_X, train_y, valid_y = train_test_split(X, y,
                                                          test_size=0.3)
    rf.fit(train_X, train_y)
    acc = metrics.accuracy_score(valid_y, rf.predict(valid_X))
    for column in X.columns:
        X_t = valid_X.copy()
        X_t[column] = np.random.permutation(X_t[column].values)
        shuff_acc = metrics.accuracy_score(valid_y, rf.predict(X_t))
        scores[column].append((acc-shuff_acc)/acc)
print('Features sorted by their score:')
print(sorted([(round(np.mean(score), 4), feat) for
              feat, score in scores.items()], reverse=True))
```

```
Features sorted by their score:
[(0.0717, 'borrower_score'), (0.0407, 'grade'), (0.0274, 'term_60 months'), (0.0127, 'annual_inc'), (0.0123, 'payment_inc_
ratio'), (0.0052, 'revol_util'), (0.0049, 'dti'), (0.0039, 'purpose_small_business'), (0.0028, 'purpose__small_business'),
(0.0028, 'open_acc'), (0.0026, 'emp_length'), (0.0011, 'emp_len__ > 1 Year'), (0.001, 'revol_bal'), (0.001, 'purpose__othe
r'), (0.001, 'home__OWN'), (0.0007, 'purpose_credit_card'), (0.0007, 'delinq_2yrs_zero'), (0.0006, 'home__RENT'), (0.0004,
'purpose__home_improvement'), (0.0003, 'purpose_medical'), (0.0003, 'pub_rec_zero'), (0.0002, 'purpose_other'), (0.0001, '
purpose_wedding'), (0.0001, 'purpose_vacation'), (0.0001, 'purpose_debt_consolidation'), (-0.0, 'purpose_moving'), (0.0, '
purpose_major_purchase'), (-0.0, 'purpose__debt_consolidation'), (-0.0001, 'purpose__medical'), (-0.0003, 'purpose_house
'), (-0.0004, 'purpose_home_improvement'), (-0.0004, 'purpose__major_purchase'), (-0.0018, 'loan_amnt')]
```

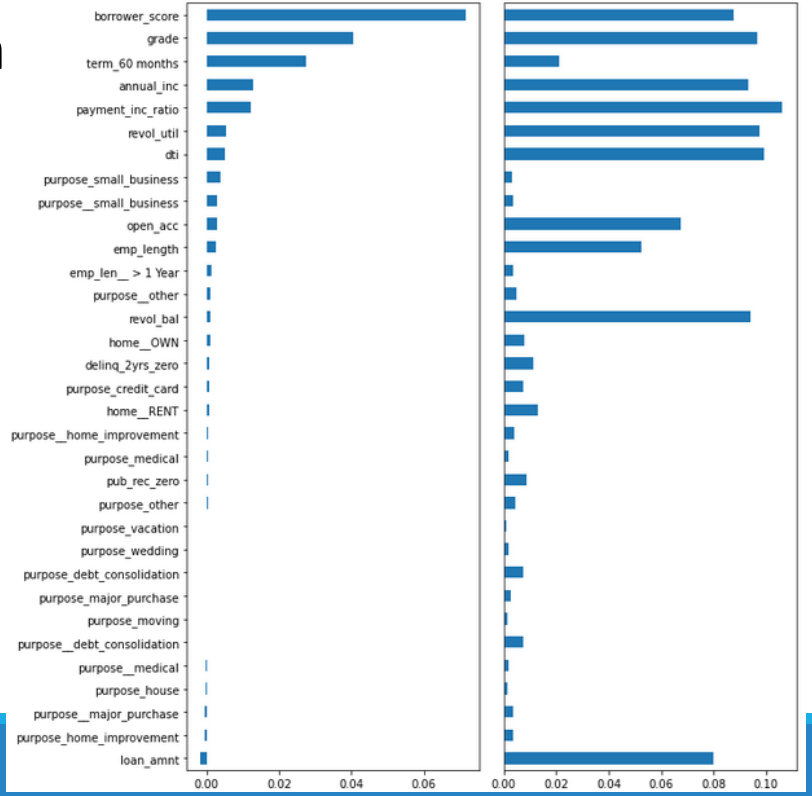
Aprendizado de Máquina Estatístico

- **Random Forest** (**RF**): Python

```
importances = rf_all.feature_importances_

df = pd.DataFrame({'feature': X.columns,
                   'Accuracy decrease': [np.mean(scores[column]) for column in X.columns],
                   'Gini decrease': rf_all.feature_importances_,
                   'Entropy decrease': rf_all_entropy.feature_importances_,})
df = df.sort_values('Accuracy decrease')

fig, axes = plt.subplots(ncols=2, figsize=(10, 10))
ax = df.plot(kind='barh', x='feature', y='Accuracy decrease', legend=False, ax=axes[0])
ax.set_ylabel('')

ax = df.plot(kind='barh', x='feature', y='Gini decrease', legend=False, ax=axes[1])
ax.set_ylabel('')
ax.get_yaxis().set_visible(False)

plt.tight_layout()
plt.show()
```

Aprendizado de Máquina Estatístico

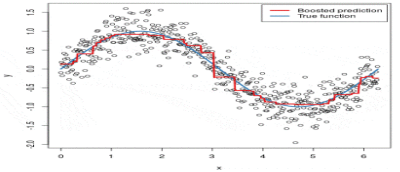
- **Random Forest** (**RF**): Python



XGBoost

Aprendizado de Máquina Estatístico

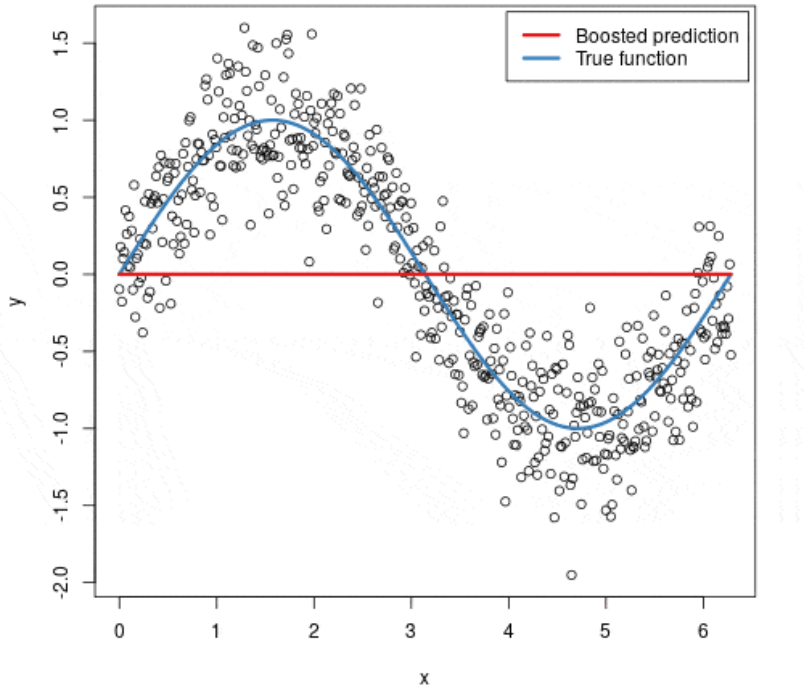
- Os métodos de ***Boosting***, ao criar novos estimadores, utilizam o feedback de avaliação dos estimadores criados anteriormente para melhorar a eficiência.

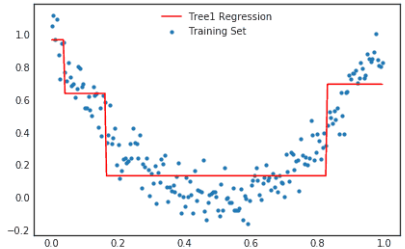


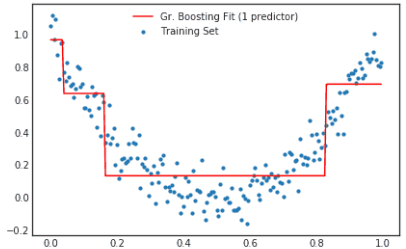
- Os ***Gradient Boosting*** minimizam os erros produzidos pelos estimadores anteriores aplicando um algoritmo de gradiente descendente, assim é possível eliminar àqueles estimadores que não tiveram desempenho muito bom no treinamento.

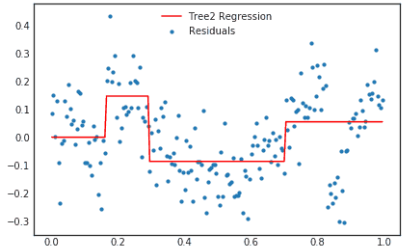
- Já o ***eXtreme Grandient Boosting***, utiliza técnicas de otimização algorítmica e de hardware para produzir resultados superiores e utilizar menos recursos computacionais

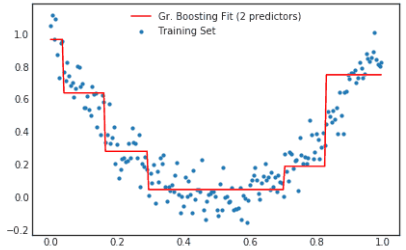
Aprendizado de Máquina Estatístico

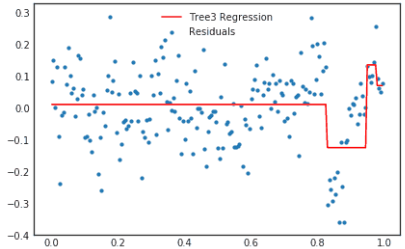


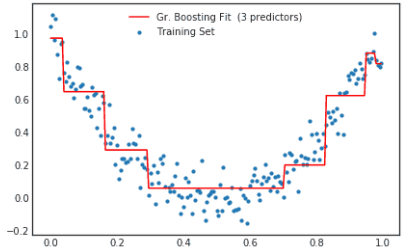












Aprendizado de Máquina Estatístico

- **XGBoost** (**XGB**): A função do XGBoost tem diversos parâmetros que devem ser ajustados. Dois parâmetros muito importantes são subsample (controla a fração de amostras que devem ser utilizadas) e eta (fator de aprendizagem utilizado em cada iteraçao).
- Ao usar **subsample** o algoritmo se comporta como o RF exceto por não realizar reposição de amostras.
- O **eta** auxilia o modelo a não fazer Overfitting.

Aprendizado de Máquina Estatístico

- **XGBoost** (**xgb**): Python

```
from xgboost import XGBClassifier

loan3000 = pd.read_csv('./DataSets/loan3000.csv‘)

X = loan3000[['borrower_score', 'payment_inc_ratio']]
y = pd.Series([1 if o == 'default' else 0 for o in loan3000[outcome]])

#Train Test Split

xgb = XGBClassifier(objective='binary:logistic', subsample=.63, use_label_encoder=False)
xgb.fit(X_train, y_train)

pred = xgb.predict(X_test)
#Métricas de Avaliação – a DT retorna o rótulo aplicado
```

Aprendizado de Máquina Estatístico

- **XGBoost** (**XGB**): A aplicação do xgboost sem conhecimento pode levar ao Overfitting, isso pode levar a um modelo que possui muito erro ao investigar amostras que não estão no conjunto de treino e os resultados de suas previsões se tornam instáveis.

- Para evitar podemos utilizar o subsample e o eta, porém em alguns casos somente eles não será o suficiente. Então podemos utilizar os parâmetros alpha e lambda. Penalizações para modelos complexos, diminuindo a produção de árvores ajustadas.
  - alpha: relacionado diretamente a distância Manhattan.
  - lambda: corresponde a penalização relacionada a distância euclidiana.

Aprendizado de Máquina Estatístico

- **XGBoost (XGB)**: Uma outra forma de se evitar o Overfitting e selecionar os melhores parâmetros para o modelo, é aplicação do CrossValidation

Aprendizado de Máquina Estatístico

- **Random Forest** (**RF**): Python

```
idx = np.random.choice(range(5), size=len(X), replace=True); error = []
for eta, max_depth in product([0.1, 0.5, 0.9], [3, 6, 9]):
    xgb = XGBClassifier(objective='binary:logistic', n_estimators=250, max_depth=max_depth,
                        learning_rate=eta, use_label_encoder=False, eval_metric='error')
    cv_error = []
    for k in range(5):
        fold_idx = idx == k
        train_X = X.loc[~fold_idx]; train_y = y[~fold_idx]
        valid_X = X.loc[fold_idx]; valid_y = y[fold_idx]

        xgb.fit(train_X, train_y)
        pred = xgb.predict_proba(valid_X)[:, 1]
        cv_error.append(np.mean(abs(valid_y - pred) > 0.5))
    error.append({ 'eta': eta,
                   'max_depth': max_depth,
                   'avg_error': np.mean(cv_error)})
errors = pd.DataFrame(error)
print(errors)
```

	eta	max_depth	avg_error
0	0.1	3	0.327931
1	0.1	6	0.335278
2	0.1	9	0.343994
3	0.5	3	0.340838
4	0.5	6	0.367693
5	0.5	9	0.374152
6	0.9	3	0.355849
7	0.9	6	0.388467
8	0.9	9	0.384625

Aprendizado de Máquina Estatístico

- **Resumo:**
  - **O kNN é simples e procura amostras semelhantes para aplicar a previsão ou regressão democrática na nova amostra.**
  - **O modelos em árvore estimam diversas regras de cortes dividindo interativamente o conjunto de treino em seções e subseções que sejam homogêneas em relação as classes. As regras formam um caminho para classificação ou regressão.**
  - **Os modelos em árvore são muito populares e eficientes, originando a partir deles métodos de agrupamentos (boosting e bagging) que melhoram o poder de predição em árvores.**

Referências

- Bruce, P.; Bruce, A.; **Estatística Prática para Cientista de Dados**: 50 Conceitos Essenciais; Rio de Janeiro; Alta Books; 2019.
- Morettin, P. A.; Bussab, W. O.; **Estatística Básica**. 8 ed. São Paulo: Saraiva, 2013.
- https://dataaspirant.com/five-most-popular-similarity-measures-implementation-in-python/ Acessado em: 30/05/2021
- https://www.kdnuggets.com/2020/04/data-transformation-standardization-normalization.html

FAESA
CENTRO UNIVERSITÁRIO

Análise de Dados Aplicada à Computação

PROF. M.SC HOWARD ROATTI