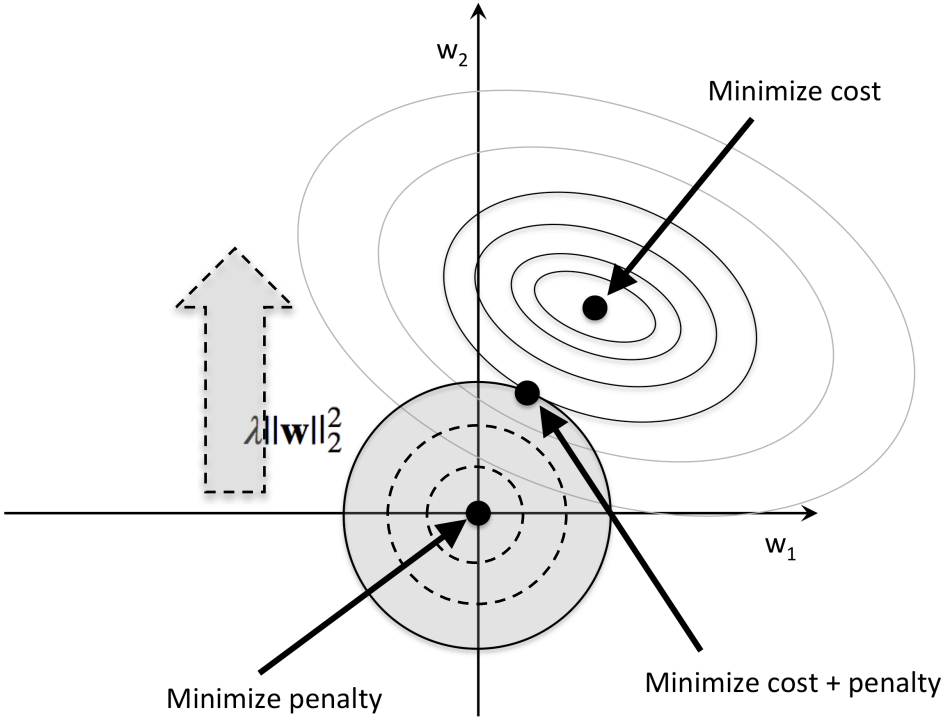
Seleção de caracteristicas

Abordagem embutida

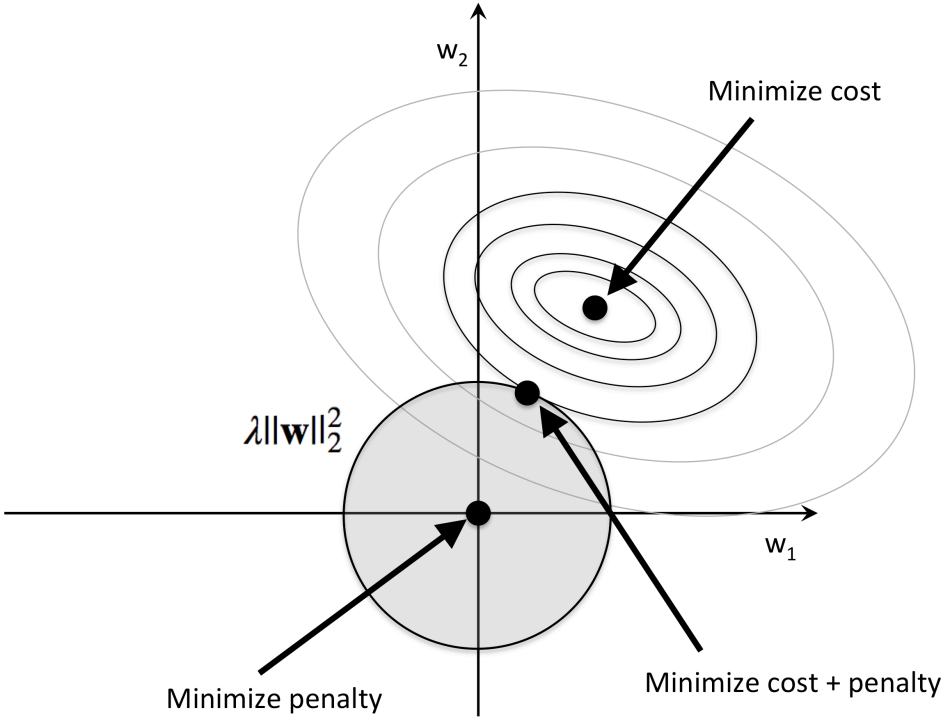
Regularização L2

$$L2 : \|\mathbf{w}\|_2^2 = \sum_{j=1}^{m} w_j^2$$



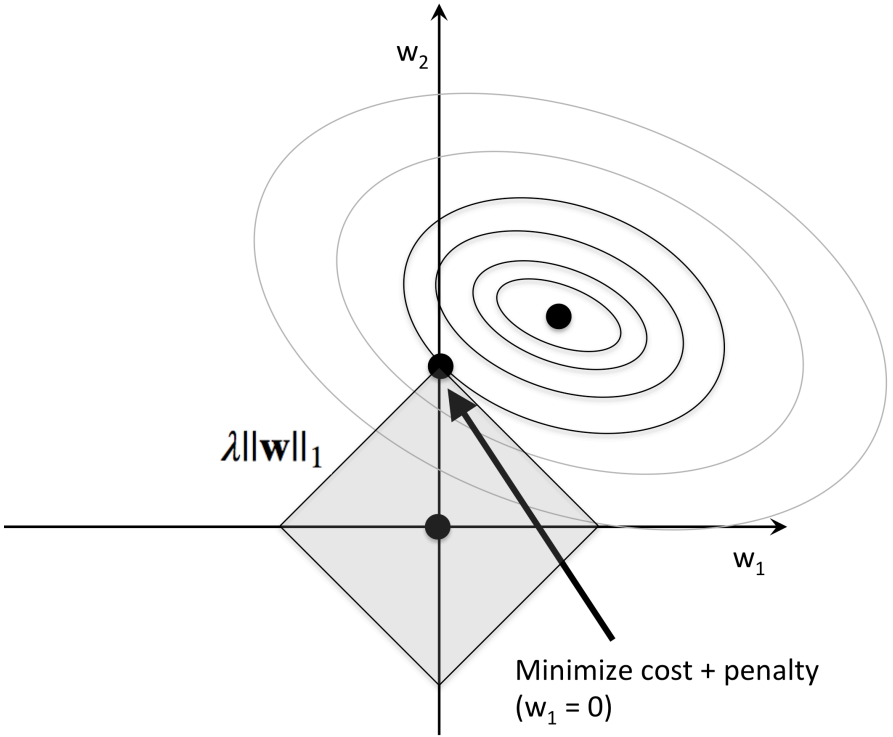
Regularização L2

$$L2 : \|\mathbf{w}\|_2^2 = \sum_{j=1}^{m} w_j^2$$



Regularização L1

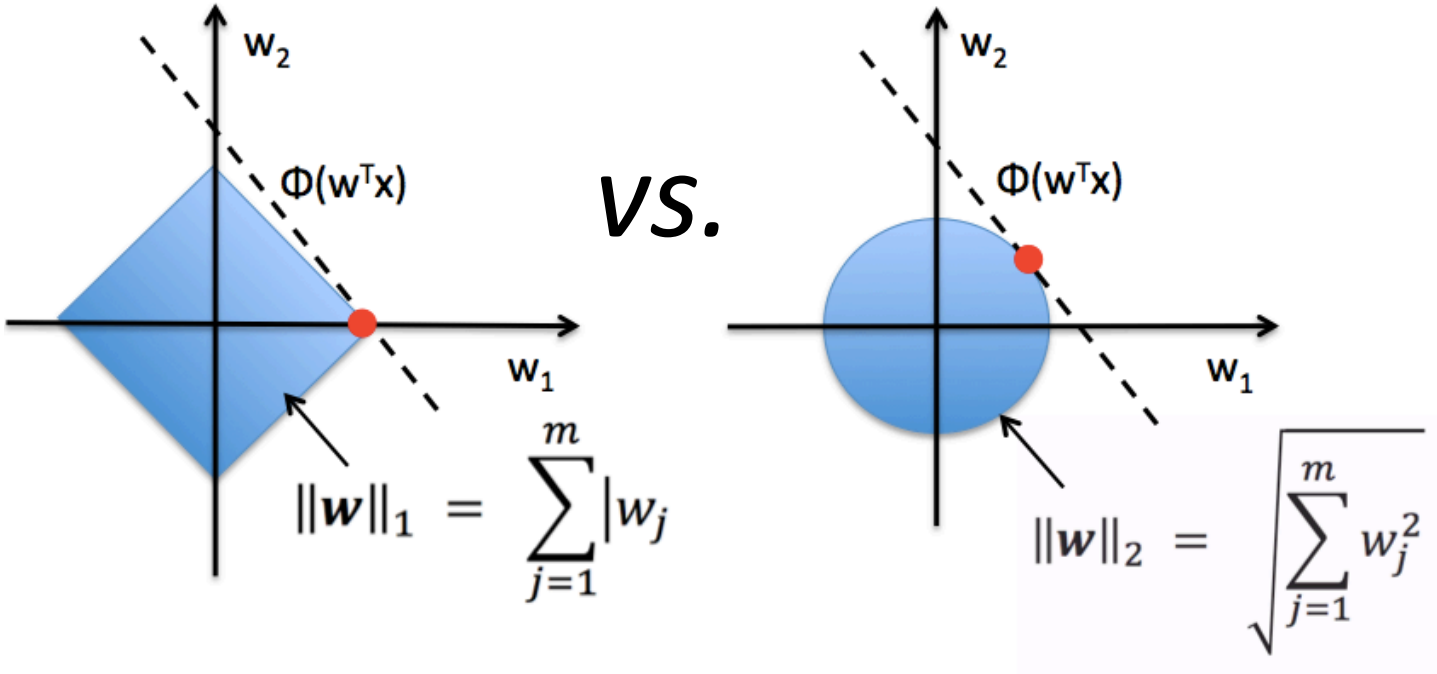
$$L1 : \|\mathbf{w}\|_1 = \sum_{j=1}^{m} |w_j|$$



Regularização L1

Provoca maior esparsidade: maioria dos pesos terão valor 0 (zero).

Útil para datasets de alta dimensionalidade onde muitas features são irrelevantes



UCI

Machine Learning Repository

Center for Machine Learning and Intelligent Systems

Wine Data Set

Download: Data Folder, Data Set Description

Abstract: Using chemical analysis determine the origin of wines

Data Set Characteristics:	Multivariate	**Number of Instances:**	178	**Area:**	Physical
Attribute Characteristics:	Integer, Real	**Number of Attributes:**	13	**Date Donated**	1991-07-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	698776

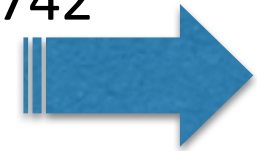
	Class label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	OD280/OD315 of diluted wines	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

```
Leitura de dados...
Separação do dataset em treinamento e teste...
Estandartização...
```

```python
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(penalty='l1', C=0.1)
lr.fit(X_train_std, y_train)
print('Acuracia do treinamento:', lr.score(X_train_std, y_train))
print('Acuracia do teste:', lr.score(X_test_std, y_test))
```

Acurácia do treinamento: 0.983870967742

Acurácia do teste: 0.981481481481



Ausência de overfitting!

```
1 lr.coef_
```

```
array([[ 0.2798866 ,  0.        ,  0.        , -0.02777116,  0.        ,
         0.        ,  0.70991877,  0.        ,  0.        ,  0.        ,
         0.        ,  0.        ,  1.23672005],
       [-0.64399277, -0.06878079, -0.05720482,  0.        ,  0.        ,
         0.        ,  0.        ,  0.        ,  0.        , -0.92679578,
         0.06015547,  0.        , -0.37103282],
       [ 0.        ,  0.06151626,  0.        ,  0.        ,  0.        ,
         0.        , -0.63594727,  0.        ,  0.        ,  0.49806351,
        -0.35818593, -0.57130706,  0.        ]])
```

Classe

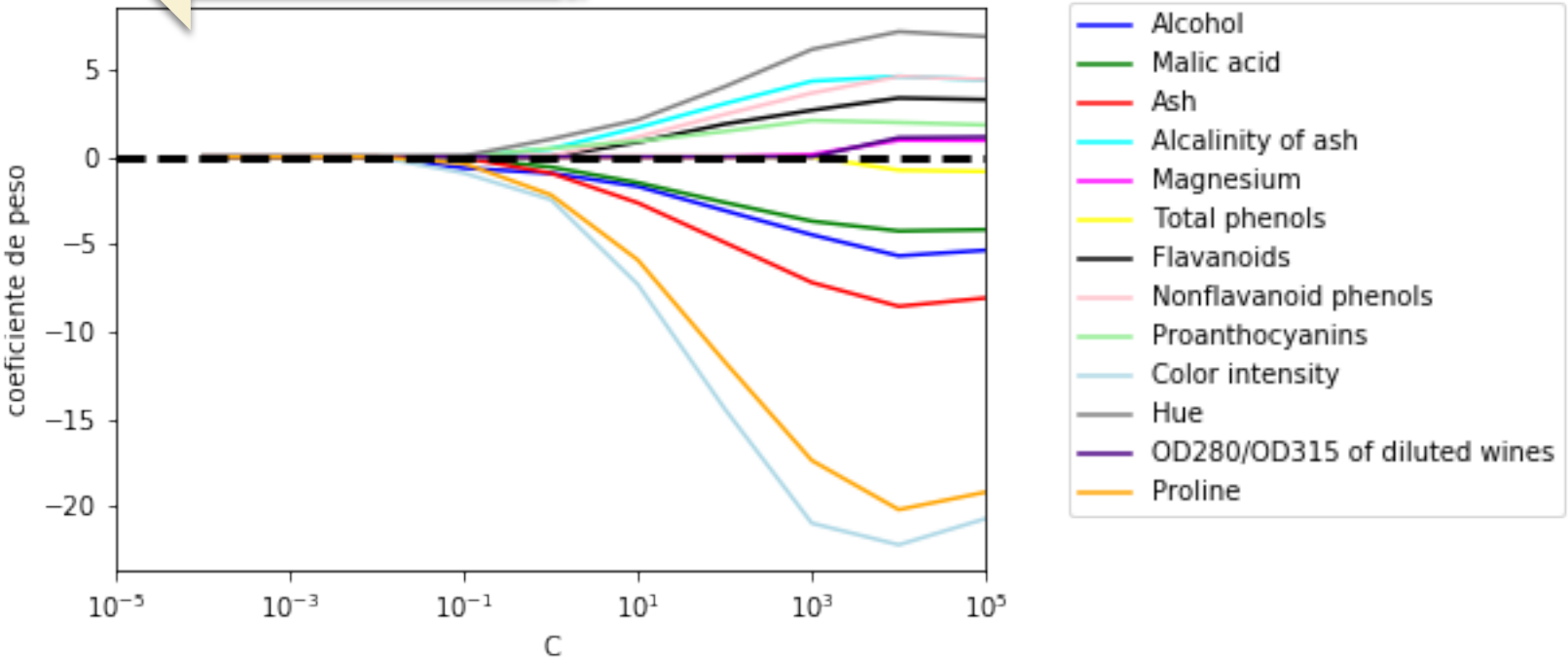
1 vs. All

2 vs. All

3 vs. All

Esparsidade!!!

Esparsidade!!!



Conclusão

É possível utilizar regularização L1 para anular o efeito de *features* irrelevantes!

Entretanto...

Há modelos que não suportam regularização!

Abordagem baseada em Wrapper

Performance do classificador é a função objetivo

- Algoritmos de seleção sequencial (ex: greedy search)
- Algoritmos de busca heurística (ex: uso de GA)
- Drawbacks: esforço computacional de treinar o modelo repetidas vezes

Chandrashekar, G., & Sahin, F. (2014). A survey on feature selection methods. *Computers & Electrical Engineering*, Elsevier, *40*(1), 16-28.

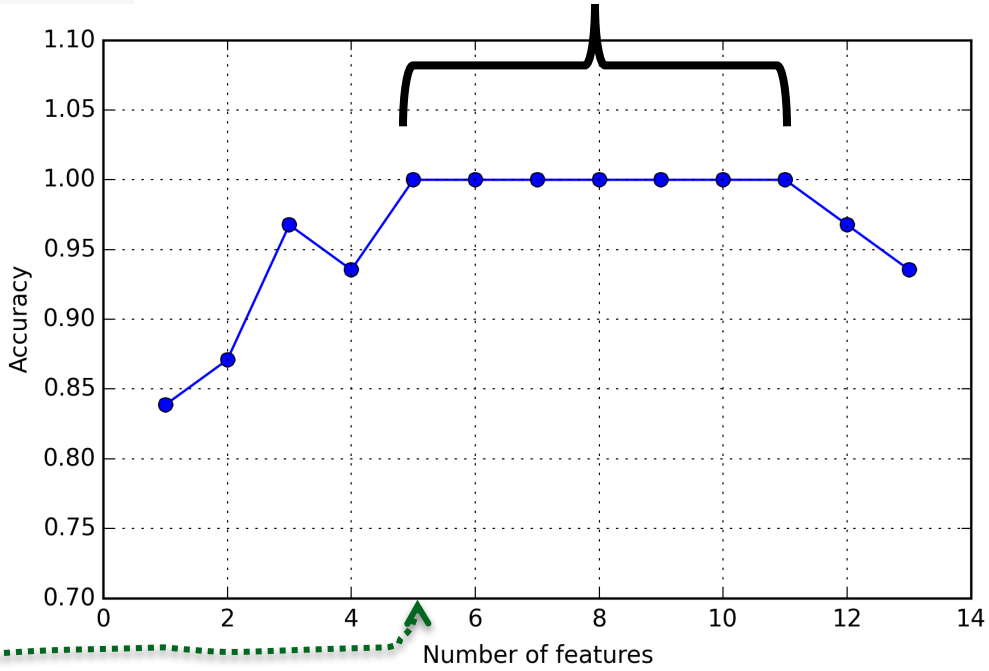
Sequential Backward Selection (SBS)

Pode ser a diferença em performance do modelo antes e após a remoção da feature

1. Initialize the algorithm with $k = d$, where d is the dimensionality of the full feature space $\mathbf{X}_d$
2. Determine the feature x^- that maximizes the criterion $x^- = \arg\max J(\mathbf{X_k} - x)$, where $x \in \mathbf{X}_k$.
3. Remove the feature x^- from the feature set: $\mathbf{X}_{k-l} := \mathbf{X}_k - x^-; \quad k := k-1.$
4. Terminate if k equals the number of desired features, if not, got to step 2.

```
%matplotlib inline
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt

knn = KNeighborsClassifier(n_neighbors=2)

# selecting features
sbs = SBS(knn, k_features=1)
sbs.fit(X_train_std, y_train)

# plotting performance of feature subsets
k_feat = [len(k) for k in sbs.subsets_]

plt.plot(k_feat, sbs.scores_, marker='o')
```

Acurácia do conjunto de *validação* = 100%




```
k5 = list(sbs.subsets_[8])
print(k5)
print(df_wine.columns[1:][k5])
```

```
[0, 1, 3, 10, 12]
Index(['Alcohol', 'Malic acid', 'Alcalinity of ash', 'Hue', 'Proline'], dtype='object')
```

Conjunto de teste original (”não contaminado”)

Com todas as d=13 features originais

```
knn.fit(X_train_std, y_train)
print('Training accuracy:', knn.score(X_train_std, y_train))
print('Test accuracy:', knn.score(X_test_std, y_test))
```

```
Training accuracy: 0.983870967742
Test accuracy: 0.944444444444
```

Acurácia do conjunto de teste 0.04 menor sugere *overfitting*

Com as k=5 features selecionadas pelo SBS

```
knn.fit(X_train_std[:, k5], y_train)
print('Training accuracy:', knn.score(X_train_std[:, k5], y_train))
print('Test accuracy:', knn.score(X_test_std[:, k5], y_test))
```

```
Training accuracy: 0.959677419355
Test accuracy: 0.962962962963
```

Acurácia do conjunto de teste = treinamento sugere melhor desempenho

Abordagem baseada em filtro

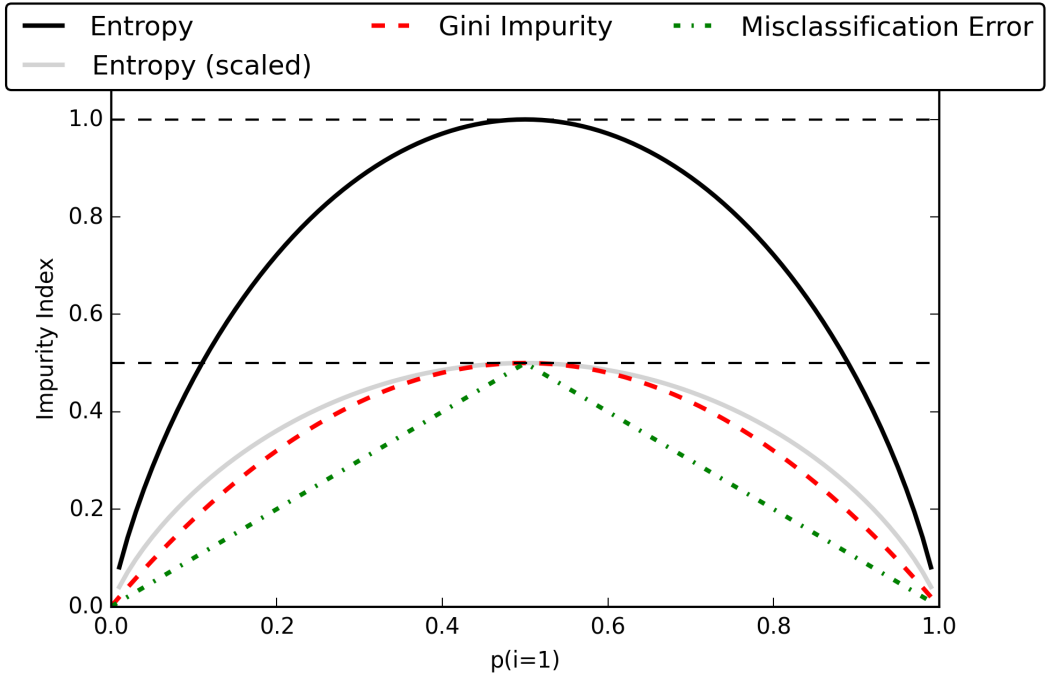
Árvores de decisão

$$IG(D_p, f) = I(D_p) - \sum_{j=1}^{m} \frac{N_j}{N_p} I(D_j)$$

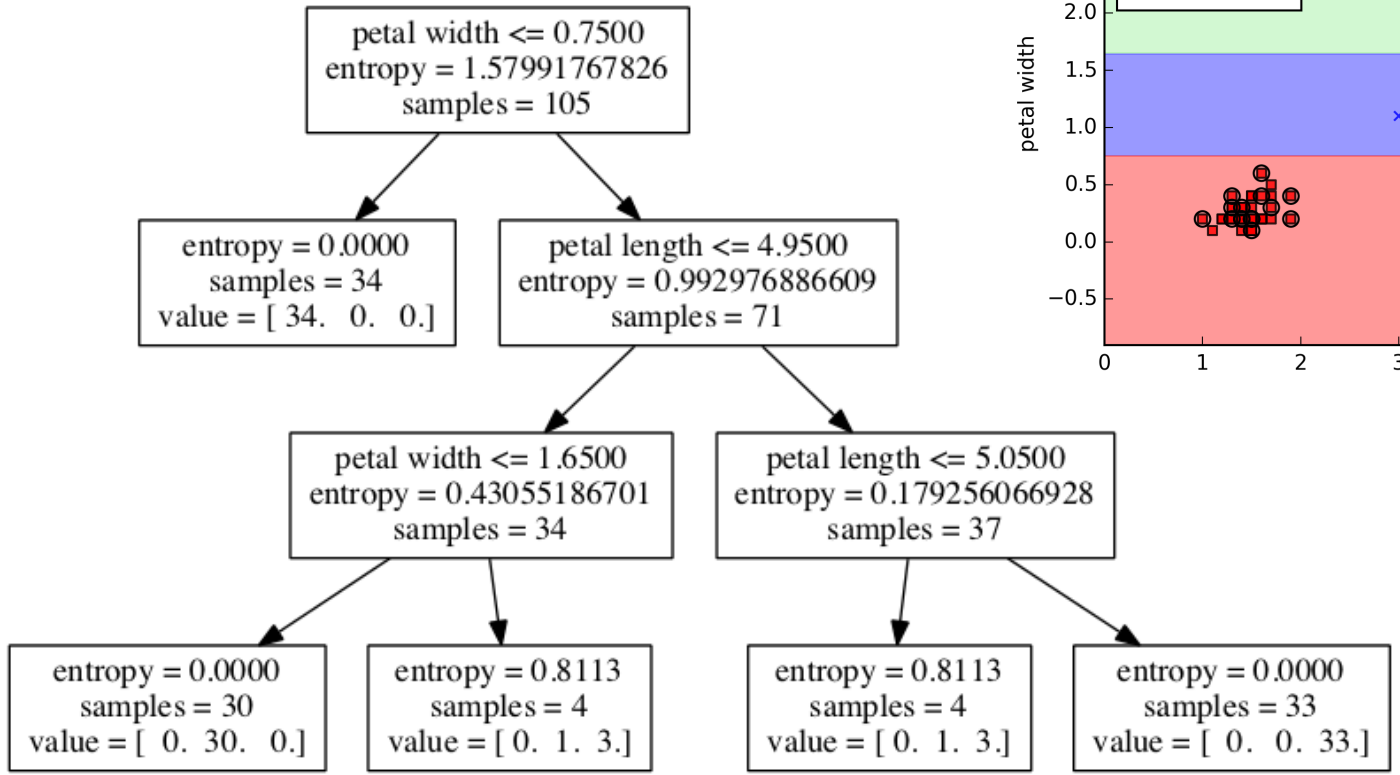
(Impureza)

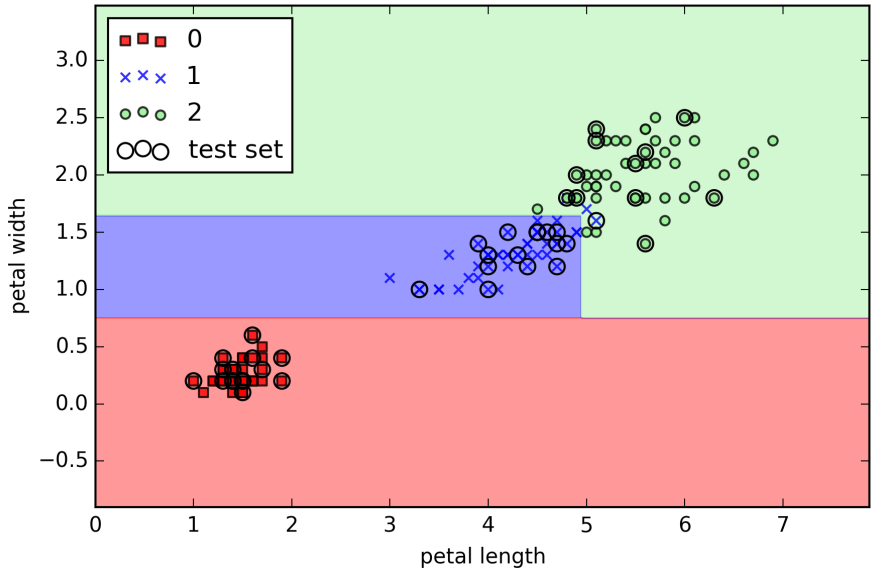
$$I_H(t) = -\sum_{i=1}^{c} p(i|t) \log_2 p(i|t)$$

(Entropia)

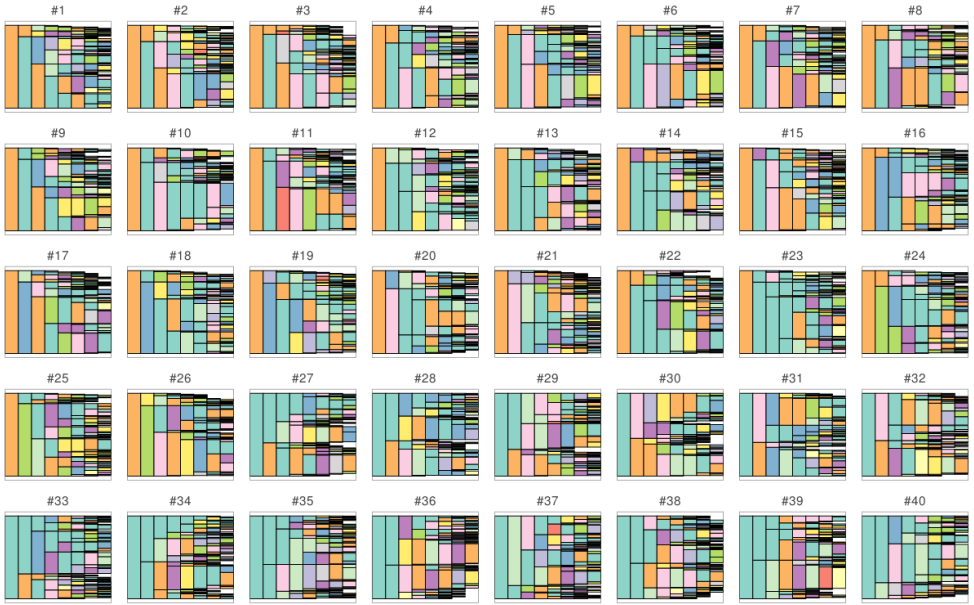


Árvores de decisão





Árvores de decisão --> Random forests



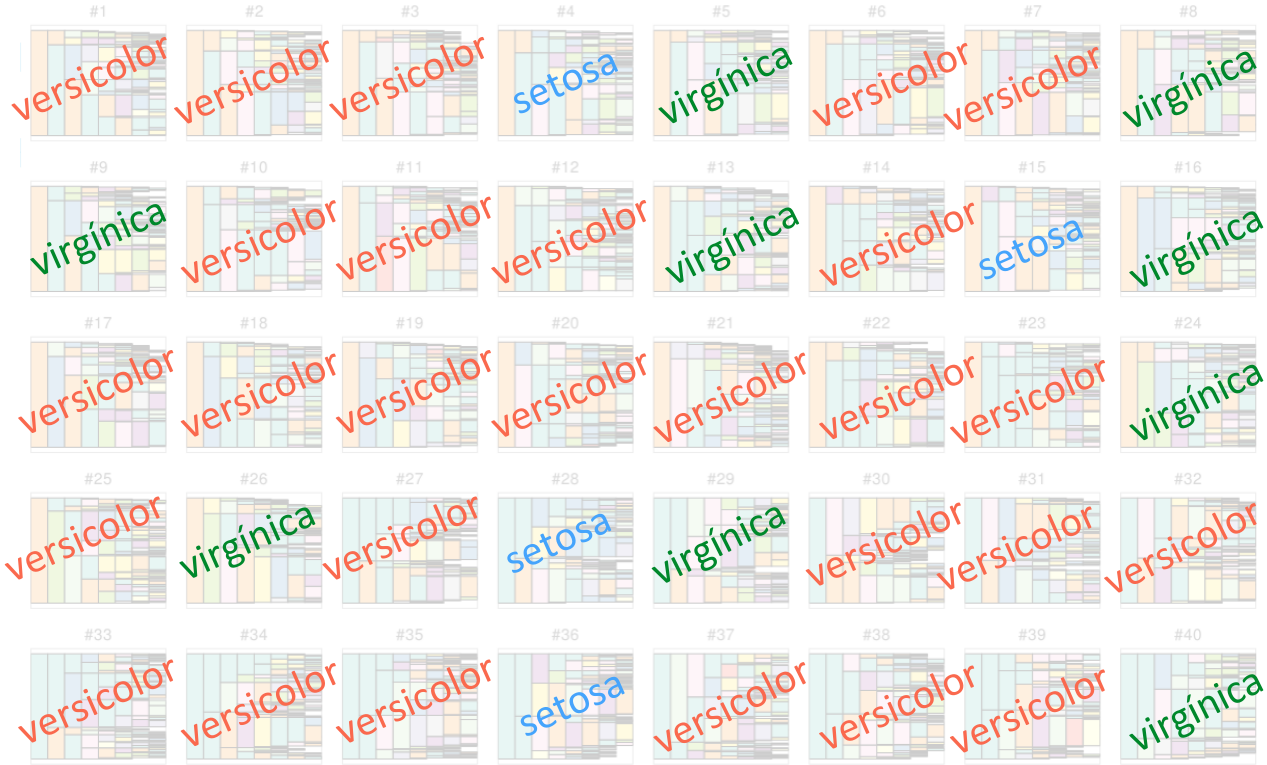
(ensemble learning)

Inúmeras árvores de decisão

Random forests

1. Draw a random **bootstrap** sample of size n (randomly choose n samples from the training set with replacement).
2. Grow a decision tree from the bootstrap sample. At each node:
    1. Randomly select d features without replacement.
    2. Split the node using the feature that provides the best split according to the objective function, for instance, by maximizing the information gain.
3. Repeat the steps 1 to 2 k times.
4. Aggregate the prediction by each tree to assign the class label by **majority vote.**

Random forests



Majority vote

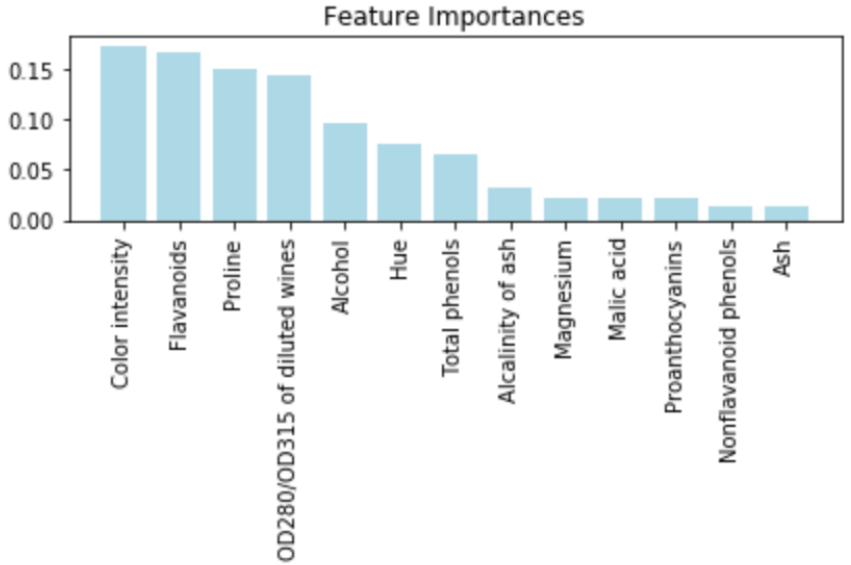
versicolor

```python
from sklearn.ensemble import RandomForestClassifier

feat_labels = df_wine.columns[1:]

forest = RandomForestClassifier(n_estimators=10000,
                                criterion='entropy',
                                max_features='sqrt',
                                random_state=0,
                                n_jobs=-1)

forest.fit(X_train, y_train)
importances = forest.feature_importances_

indices = np.argsort(importances)[::-1]

for f in range(X_train.shape[1]):
    print("%2d) %-*s %f" % (f + 1, 30,
                            feat_labels[indices[f]],
                            importances[indices[f]]))

plt.title('Feature Importances')
plt.bar(range(X_train.shape[1]),
        importances[indices],
        color='lightblue',
        align='center')

plt.xticks(range(X_train.shape[1]),
           feat_labels[indices], rotation=90)
plt.xlim([-1, X_train.shape[1]])
plt.tight_layout()
#plt.savefig('./random_forest.png', dpi=300)
plt.show()
```

k árvores

d features

```
 1) Color intensity                 0.174273
 2) Flavanoids                      0.167982
 3) Proline                         0.149730
 4) OD280/OD315 of diluted wines    0.145087
 5) Alcohol                         0.096237
 6) Hue                             0.076172
 7) Total phenols                   0.065276
 8) Alcalinity of ash               0.032060
 9) Magnesium                       0.022632
10) Malic acid                      0.022628
11) Proanthocyanins                 0.021253
12) Nonflavanoid phenols            0.013354
13) Ash                             0.013316
```



Divulgação científica

saense

http://www.saense.com.br
https://www.facebook.com/saense/

Hendrik Macedo
Escreve sobre Inteligência Artificial no Saense.
http://www.saense.com.br/autores/artigos-publicados-por-hendrik-macedo/