**Modelos de clasificacion binaria:**

From previous lectures:

**Hyperparameters**

Parameters that shape the model, but not results of the model

**Decision trees**

only a small number of features are used to make decisions

**kNN**

all features are used equally

**naive Bayes**

Overly strong conditional independence assumption and overestimates the evidence when there are correlated features.

Class 5

**LINEAR CLISSIFIERS**

**Explain the general intuition behind linear classifiers**

Predicting a binary-valued target,

|  |  |
| --- | --- |
|  |  |

This is the same as where is the bias, and are the weights

And there is a threshold ***r****.*

**Specify weights and biases by hand to represent simple linearly separable functions**

Linear regression model: we use predict

Least squares: we use fit

|  |
| --- |
| lr = LinearRegression()  lr.fit(X, y); # the ‘;’ says the model that doesn’t print any result  lr.intercept\_ # returns the beta0  lr.coef\_. # returns the coefficients |

PARAMETERS SHAPE THE MODEL

Use scikit-learn's LogisticRegression classifier

sklearn.linear\_model.LogisticRegression

Use fit, predict, predict\_proba

* use fit, predict, predict\_proba
* use coef\_ to interpret the model weights
* compare logistic regression with naive Bayes

Use coef\_ to interpret the model weights

When the coefficient are the weights (w\_i) and each feature has a weight. While higher is the weight, more important is the feature in the model.

**Generative vs discriminative models**

A **generative model** would have the goal of understanding what dogs look like and what cats look like. You can ask a generative model to draw a dog. During predict, we ask which model fits the given image best.

NAÏVE BAYES

A **discriminative model** is only trying to learn to distinguish the classes. For instance, if all the dogs in the training data are wearing collars and cats aren't, it would tell you that the difference between cats and dogs is that they do not wear collars.

LOGISTIC REGRESSION AND DECISION TREES

**Comparison between models**

#### Naive Bayes vs. Logistic regression

Both are probabilistic classifiers but Naive Bayes is "generative" and Logistic regression is "discriminative"

Naive Bayes is a generative model because it's modeling the joint distribution over the features **x** and labels **y**.

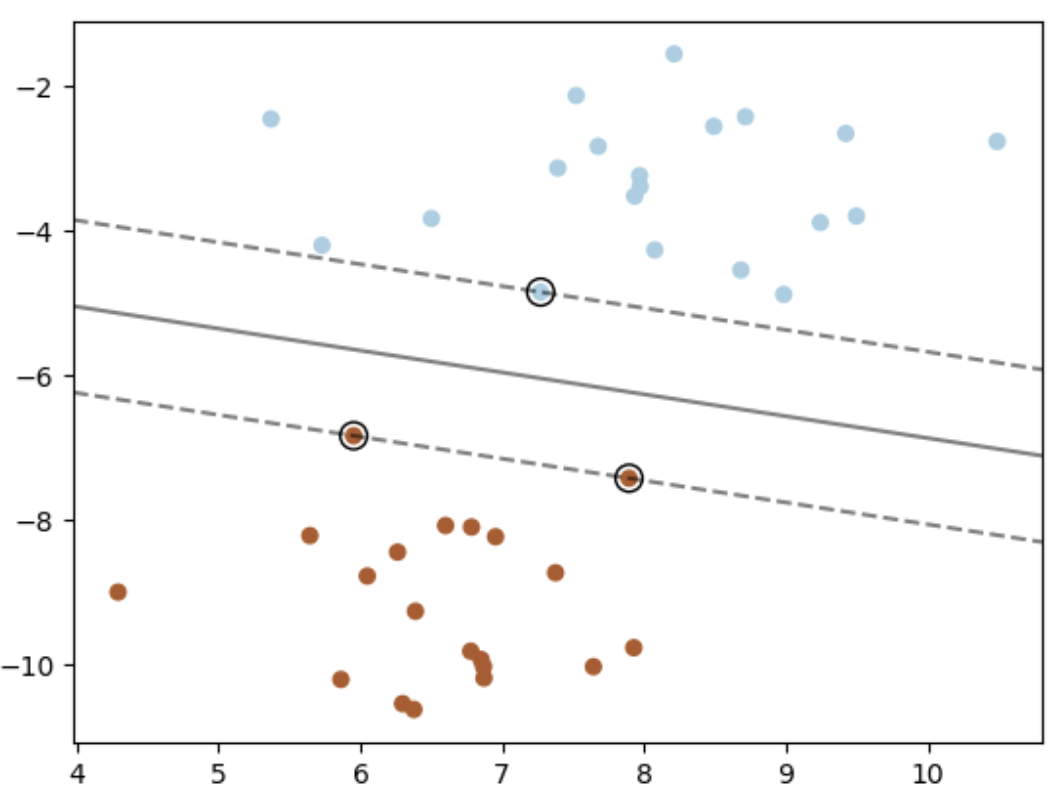
Logistic regression it directly models the probability p(y|x)

#### SVMs: General idea

Choose the hyperplane which is furthest away from the closest training points.

In other words, **choose the hyperplane which has the largest margin**, where margin is the distance from the boundary to the nearest point(s).

Intuitively, more margin is good because it leaves more "room" before we make an error.



Each training example either is or isn't a "support vector".

* This gets decided during fit.
* It does not apply to test examples.

**Main idea: the decision boundary only depends on the support vectors.**

Let's look at the support vectors

|  |  |
| --- | --- |
|  | The support vectors (SVs) are shown in yellow.  These are the example that "support" the boundary.  Number of support vectors << training set.  If we delete **all non-support** **vector** => decision boundary NOT change  If we delete **1 support vector** => decision boundary WILL change |

**SVM works only with linearly-separable data?**

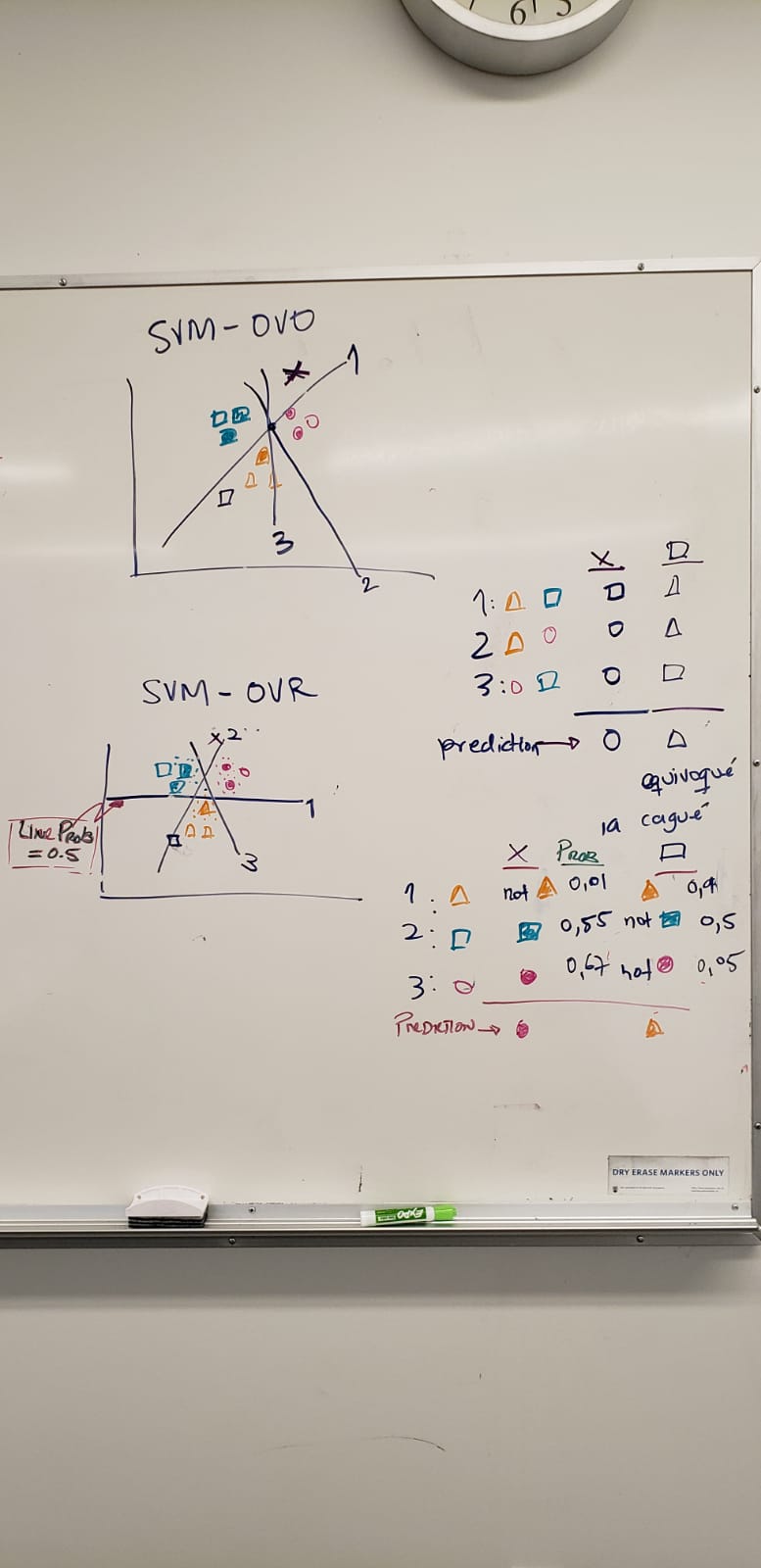
No, we can use the "kernel" trick to deal with non-linearly separable surfaces:

* KERNELS
  + **linear**:
  + **RBF** (default):

OVO (one vs one) is more sensible to compare than the OVR (one vs rest).

Computationalmente OVO is slower/more demanding than the OVR

**The OVO & OVR only works for SVM and logistic regresions.**

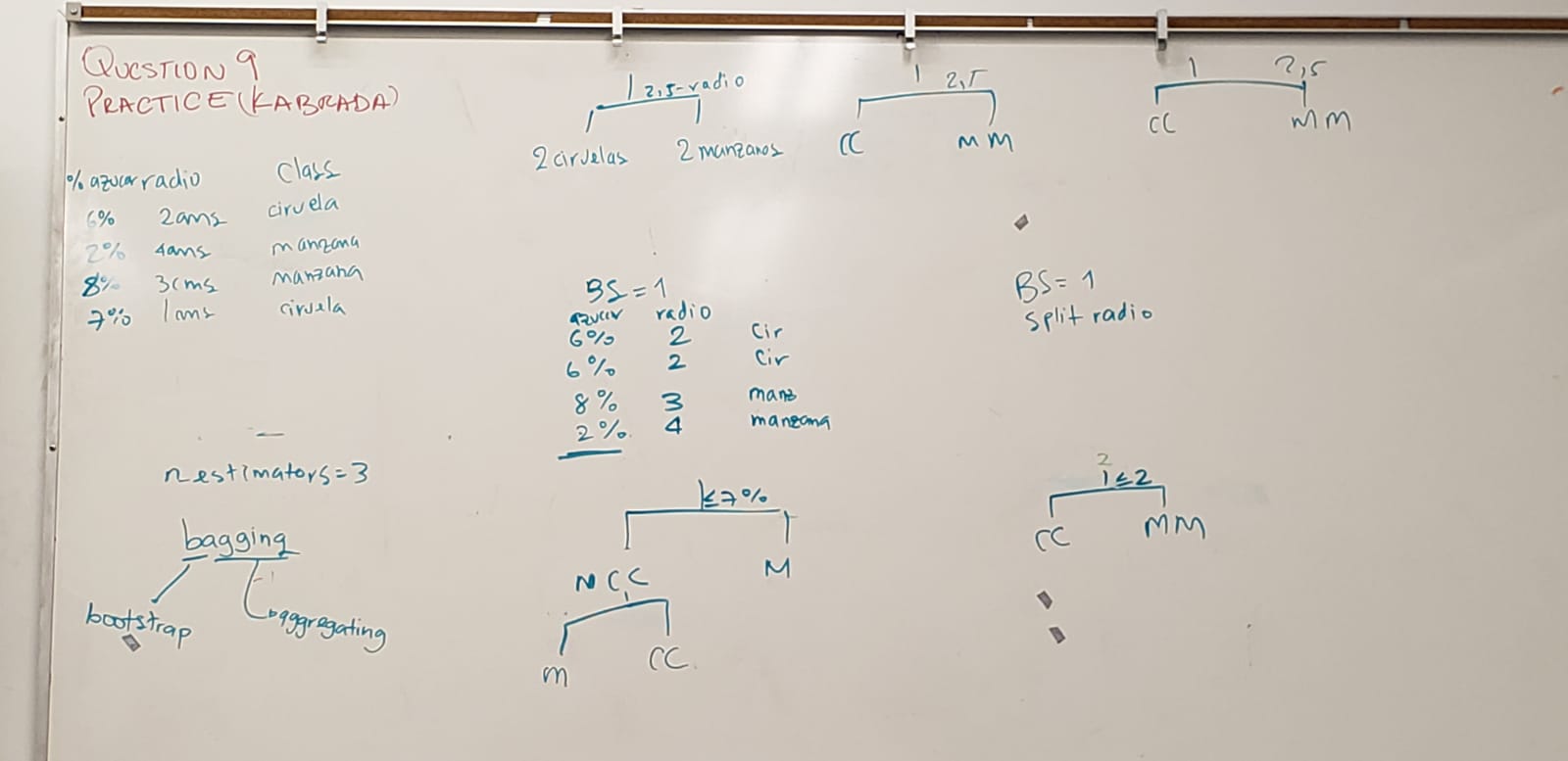


WHEN DATA IS NOT LINEAR SEPARABLE YOU USE KERNELS FOR THE SVM

SVM KERNELLS: LINEAR (BASIC), RBF, POLYNOMIAL

**Random forest**

* fit a diverse set of classifiers by **injecting randomness** in the classifier construction
* predict by taking the average of predictions given by individual classifiers



They have two aleatory components:

Bagging model -> bootstrap + aggregate

They change the

Ensabling:

* Averaging
* Stacking
* Voting
* Bosting
* Bagging 🡪 Bootstrapping and aggregate

PREDERICT PARADIGM

Para todos los modelos lineales (SVM or logit) tienes que hacer lo mismo: mirar la linea y buscar el signo (es decir calificar la observacion en la categoria A o -A)

GRIDSEARCH:

Meaning

PIPELINES:

Ordered form to automatize steps that are repetitive.

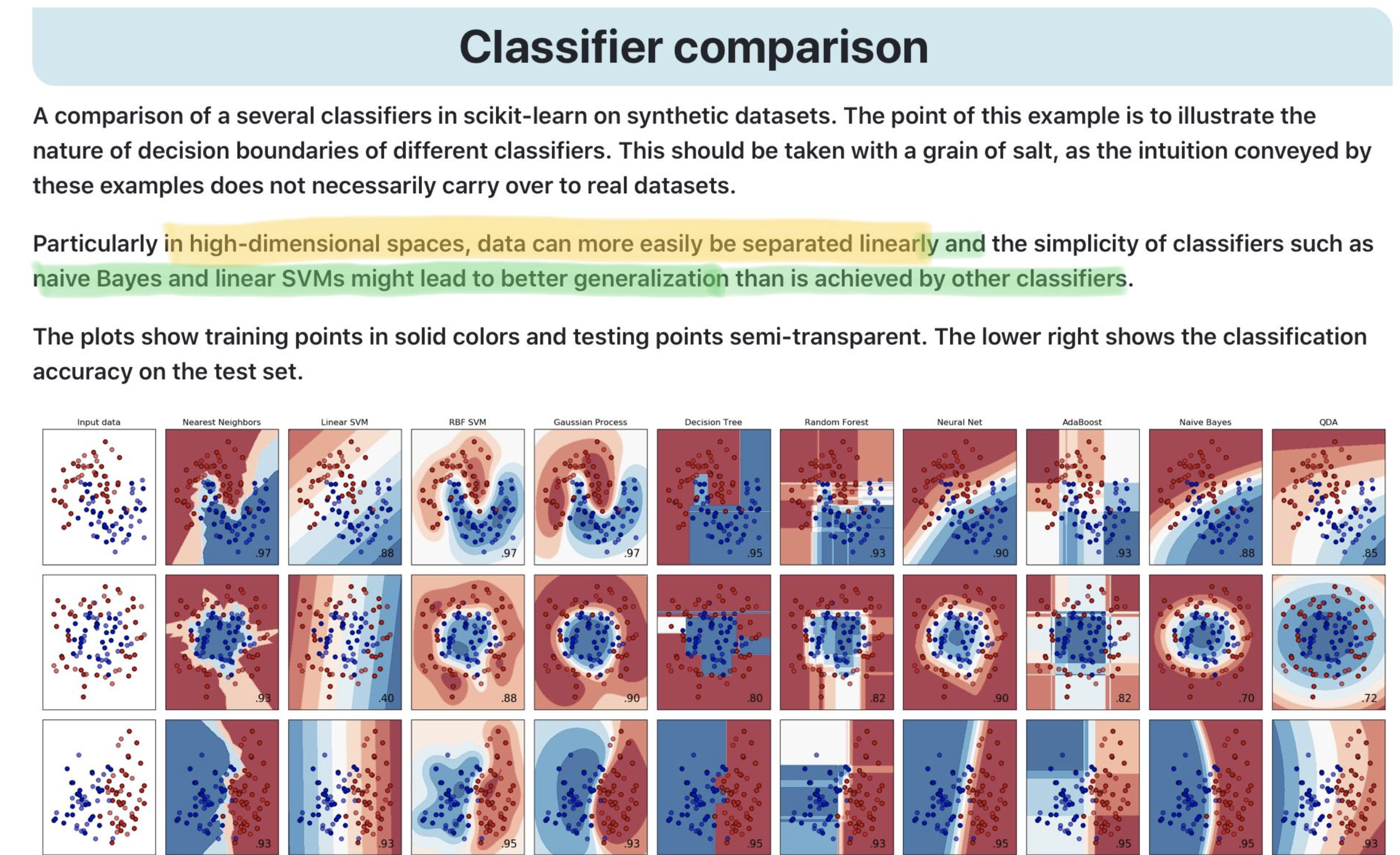
Golden rule.

**THE STEPS FOR USING A MODEL**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| # Defining your model  model = LogisticRegression() /  Here you use:   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | **Type** | Logistic regression | SVC Suppoert Vector Clasifier (SVM) | KNN |  |  |  | | **Python sklearn** | LogisticRegression() | SVC(kernel='linear') | KNeighborsClassifier() |  |  |  |   # Fit the model to your training data  lr.fit(X\_lin\_sep, y\_lin\_sep);  # Accuracy  print("Training error: %.2f" % ( model.score(X\_train, y\_train) ))  print("Validation error: %.2f" % ( model.score(X\_test, y\_test) ))  # or Error  print("Training error: %.2f" % ( 1 - model.score(X\_train, y\_train) ))  print("Validation error: %.2f" % ( 1 - model.score(X\_test, y\_test) )) |

**THE MODELS**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Model | Key words | Advantage / Strengths | Weakness | Hyperparam. | Things done in **fit** | Things done in **predict** | Python sklearn |
| Decision tree |  | Easy to understand and interpret, very visual. Easy to implement. Doesn't requiere normalization or scaling of data. They are very flexible: it can use a mixture type of variables (contiues, integers or cathegorical)  only a small number of features are used to make decisions | Number of threes, max depth It is easy to overfit and underfit, and difficult to obtain a good accuracy of the model. Unstable. Decision bias towards categorical atributtes with more levels. Higher time to train the model. | Max depth | Construye arbol | Calcula las probabilidades | from sklearn import tree  from sklearn.tree import DecisionTreeClassifier |
| KNN |  | Algorithm easy to implement and interpret. It is very intuitive and simple. It uses few hyperparameters. | No parametric (when adding an observation, we need to recalculate the distances). Lazy learner. Scale sensitive (needs scaling or normalization).  All features are used equally | K-neighbours | Le avuentas todo | Hace los calculos | from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor |
| Naïve Bayes |  | When the independence assumption is true, naive bayes performs better to other classification.  Easy for implementation. Works well with few data. | Assumption of conditional independency between variables.  When test data set has not a feature non-observed in the training data, it assign a probability of zero, and we need to apply Laplace smoothing.  Overly strong conditional independence assumption and overestimates the evidence when there are correlated features. | Alpha (when we use Laplace) | Calcula probabilities  P(X|Y) | Plug in the data | from sklearn.naive\_bayes import BernoulliNB, GaussianNB, MultinomialNB |
| Logistic regressions |  | Used EVERYWHERE.  Fast training and testing.   * Training on huge datasets * Testing is just computing 𝑤𝑇𝑥𝑖   Easy to understand.  Coefficients of a linear classifier similar interpretation to that of linear regression: if coefficient 𝑗 is large, that means a change in feature j has a large impact on the prediction.  Works better on larger datasets.  It is much more robust to correlated features: If two features are correlated, regression will assign part of the weight to one and part to another. | Generally doesn't work good on small datasets. It needs to assume linear relationship To perform multi-class, it needs to assume OVR or OVO. Doesn't work well if data isn't "linearly separable". | Multi-class classification (OVR and OVO)  Max number of iterations | Calculate the support vector. The line  + gde | Choose which data belong to each group | from sklearn.linear\_model import LogisticRegression, LinearRegression |
| SVM | Hyperplane | Works better when the data is clear separable between classes. Effective in high dimensional spaces. Stable model, a small change on the data doesn't affect as much the line (hyperplane) that divide the groups. | Doesn't work well with large datasets. Results difficult to understand and interpret. It requires much more time to train compared with other models. | Kernel = linear, poly, RDF (Radial basis function)  Gamma  C (cost of missclasiffication). | Calculate the support vector. The line  + gde | Choose which data belong to each group | from sklearn.svm import SVC, SVR, LinearSVC  from sklearn.multiclass import OneVsRestClassifier, OneVsOneClassifier |
| Random Forrest |  | Can handle large data sets with higher dimensionality. Reduce overfitting. They are very flexible: it can use a mixture type of variables (contiues, integers or cathegorical) | A complex model because it creates and combines the result of several decision trees. It requires much more time to train compared with other models. | N-estimators (number of trees) Max depths | More time to train |  | from sklearn.ensemble import RandomForestClassifier |



TO COMPARE A MODEL WE SHOULD FOLLOW THE NEXT STEPS:

DOWNLOAD DATA

SPLIT DATA

EDA - EXPLANATORY DATA ANALYSIS

|  |
| --- |
| # load dataset  data\_abalone = pd.read\_csv("abalone\_age.csv")  # EDA: Check for data errors, outliers and erroneous values  print('- - - TYPE - - -')  print(type(data\_abalone))  print('\n- - - VARIABLES - - -')  print(list(data\_abalone))  print('\n- - - GENERAL INFO - - -')  print(data\_abalone.info())  print('\n- - - DESCRIBE VARIABLES - - -')  print(data\_abalone.describe())  print('\n- - - NaN OBSERVATIONS - - -')  print(data\_abalone[data\_abalone.isnull().any(axis=1)])  # Define and split the data  features = data\_abalone.drop(['Rings'], axis=1)  target = data\_abalone['Rings']  X\_train, X\_test, y\_train, y\_test = train\_test\_split(features,  target,  test\_size=0.2)  # Pre-process the data  numeric\_features = ['Length', 'Diameter', 'Height', 'Whole weight',  'Shucked weight', 'Viscera weight', 'Shell weight']  categorical\_features = ['Sex']  print("\n- - - COUNT CASES (categ. variables): 'Sex' - - -")  print(data\_abalone['Sex'].value\_counts())  print('\n')  preprocessor = ColumnTransformer(transformers=[  ('scale', StandardScaler(), numeric\_features),  ('ohe', OneHotEncoder(drop="first"), categorical\_features)])  # # Choose hyperparameters  # parameters\_RF = {'max\_depth':np.arange(1, 5)} #, 'n\_estimators':np.arange(1, 151, 25)}  # model\_RF = RandomForestClassifier()  # model\_RF\_param = GridSearchCV(model\_RF, param\_grid = parameters\_RF, cv=5)  # model\_RF\_param.fit(X\_train, y\_train)  # Models: LinearRegression, SVR, KNeighborsRegressor and RandomForestRegressor  models = {'lin\_mod': LinearRegression(),  'svr\_mod': SVR(),  'knn\_mod': KNeighborsRegressor(),  'rf\_mod': RandomForestRegressor() }  # parameters for the gridsearch  parameters = {  'lin\_mod': {},  'svr\_mod': {'classifier\_\_kernel': ['linear', 'rbf']},  'knn\_mod': {'classifier\_\_n\_neighbors': range(1, 10)},  'rf\_mod':{'classifier\_\_max\_depth': range(1, 30, 5)}  }  results\_dict = {}  xxx = pd.DataFrame()  for model\_name, model in models.items():  t = time.time()  #print(model\_name, ":")    # I create the models  clf = Pipeline(steps=[('preprocessor', preprocessor),  ('classifier', model)])  # gridsearch  gs = GridSearchCV(clf, parameters[model\_name], cv=5)    # and now I fit the model  gs.fit(X\_train, y\_train);  train\_error, test\_error, root\_mse = get\_scores(gs, X\_train, y\_train,  X\_test, y\_test, show = False)  elapsed\_time = time.time() - t  results\_dict[model\_name] = [round(train\_error,3), round(test\_error,3), round(elapsed\_time,4),  root\_mse, gs.best\_params\_]  #print("Elapsed time: %.1f s" % elapsed\_time)    #prediction of each model  xxx[model\_name] = gs.predict(X\_test)  results\_df = pd.DataFrame(results\_dict).T  results\_df.columns = ["Train error", "Validation error", "Time in seconds",  "Root of Mean-Square-Error (MSE)", "Best parameters"]  results\_df  # table to graph  new\_table = pd.concat([xxx, y\_test.reset\_index()], axis = 1).drop(columns='index').melt(id\_vars="Rings")  new\_table  #graph  import altair as alt  from vega\_datasets import data  # Uncomment this line if you are in a Jupyter Notebook (not JupyterLab) environment  alt.renderers.enable('notebook')  alt.Chart(new\_table).mark\_circle(size=60).encode(  x='Rings',  y='value',  color='variable',  tooltip=['Rings', 'value', 'variable']  ).interactive() |

|  |
| --- |
| # LIBRARIES  # Lecture 1  from sklearn import tree  from sklearn.tree import DecisionTreeClassifier  # Lecture 2  from sklearn.model\_selection import train\_test\_split  # Lecture 3  from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor  from statsmodels.nonparametric.smoothers\_lowess import lowess  # Lecture 4  from sklearn.naive\_bayes import BernoulliNB, GaussianNB, MultinomialNB  from sklearn.preprocessing import normalize, scale, Normalizer, StandardScaler, OneHotEncoder  # Lecture 5  from sklearn.linear\_model import LogisticRegression, LinearRegression  # Lecture 6  from sklearn.svm import SVC, SVR, LinearSVC  from sklearn.multiclass import OneVsRestClassifier, OneVsOneClassifier  from sklearn.ensemble import RandomForestClassifier  # Lecture 7  from sklearn.ensemble import VotingClassifier, AdaBoostClassifier, GradientBoostingClassifier, RandomForestRegressor  from sklearn.tree import DecisionTreeRegressor  from lightgbm import LGBMClassifier  from xgboost import XGBClassifier  # Lecture 8  from sklearn.compose import ColumnTransformer  from sklearn.impute import SimpleImputer  from sklearn.model\_selection import GridSearchCV  from sklearn.pipeline import Pipeline, FeatureUnion, make\_pipeline  from sklearn.preprocessing import FunctionTransformer  # Other  import numpy as np  import pandas as pd  import pickle  import graphviz  from sklearn import datasets  from sklearn.datasets import make\_blobs, make\_hastie\_10\_2  from sklearn.datasets import fetch\_20newsgroups  from sklearn.feature\_extraction.text import TfidfVectorizer, TfidfTransformer  from sklearn.feature\_extraction.text import CountVectorizer  # pip install git+git://github.com/mgelbart/plot-classifier.git  from plot\_classifier import plot\_classifier  import matplotlib.pyplot as plt  %matplotlib inline  import pandas as pd  pd.set\_option("display.max\_colwidth", 200)  import altair as alt  import time  # pip install ipython-autotime  import autotime  alt.renderers.enable('notebook')  #alt.data\_transformers.enable('json')  ## Attribution: Many visualization code snippets are from Mike's lecture from last year. |

