PREDICCIÓN DE VIRALIDAD DE TWEETS

Marc Pascual
Alejandro Lobo

Introducción

Fuente del dataset y objetivo

¿Porque lo hemos escogido?

Introducción

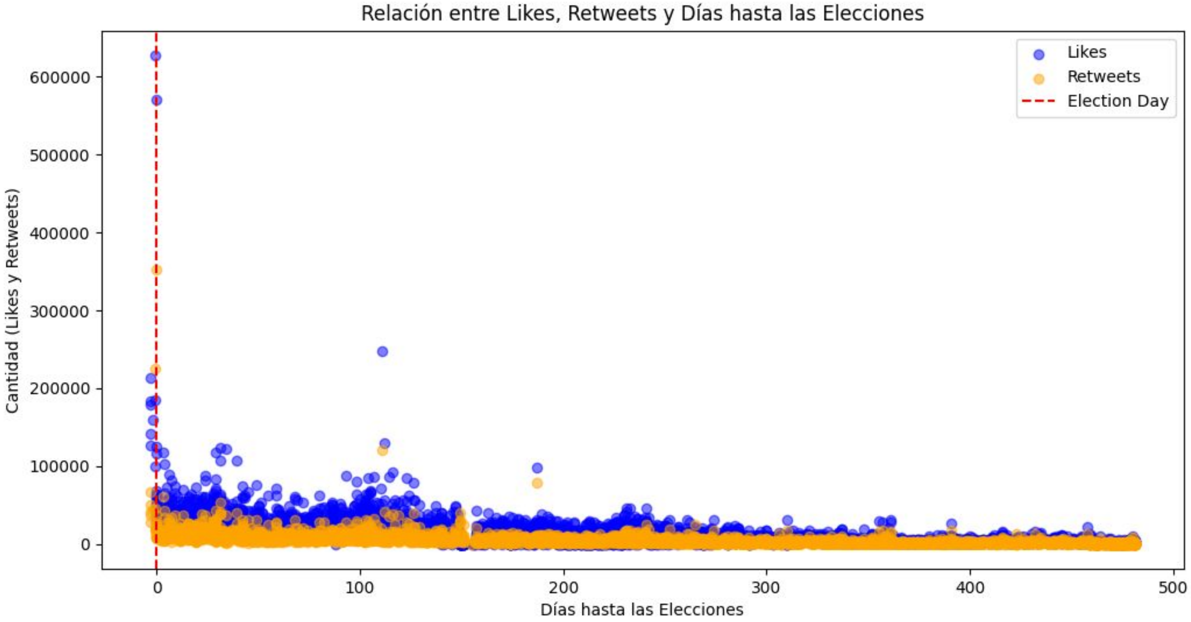
Análisis general

```python
data.isnull().any()
```

```python
Date                                      False
Time                                      False
Tweet_Text                                False
Type                                      False
Media_Type                                 True
Hashtags                                   True
Tweet_Id                                  False
Tweet_Url                                 False
twt_favourites_IS_THIS_LIKE_QUESTION_MARK False
Retweets                                  False
Unnamed: 10                                True
Unnamed: 11                                True
dtype: bool
```

Introducción

Relación entre las elecciones y los likes y RT



- Mas likes y RT cerca de las elecciones
- Decidimos ignorar este hecho

Preprocesamiento

Importación del dataset

CARGAR LOS DATOS

```
pandas.set_option('display.max_columns', None)
pandas.set_option('display.expand_frame_repr', False)
pandas.set_option('display.precision', 3)
df = pandas.read_csv('data.csv', sep=',', na_values="")
print(df.head())
```

```
       Date      Time                                         Tweet_Text  Type Media_Type  Hashtags   Tweet_Id                                           Tweet_Url  twt_favourites_IS_THIS_LIKE_QUESTION_MARK  Retweets  Unnamed: 10  Unnamed: 11
0  16-11-11  15:26:37  Today we express our deepest gratitude to all ...  text      photo  ThankAVet  7.970e+17  https://twitter.com/realDonaldTrump/status/797...                                     127213     41112          NaN          NaN
1  16-11-11  13:33:35  Busy day planned in New York. Will soon be mak...  text        NaN        NaN  7.970e+17  https://twitter.com/realDonaldTrump/status/797...                                     141527     28654          NaN          NaN
2  16-11-11  11:14:20  Love the fact that the small groups of protest...  text        NaN        NaN  7.970e+17  https://twitter.com/realDonaldTrump/status/797...                                     183729     50039          NaN          NaN
3  16-11-11   2:19:44  Just had a very open and successful presidenti...  text        NaN        NaN  7.970e+17  https://twitter.com/realDonaldTrump/status/796...                                     214001     67010          NaN          NaN
4  16-11-11   2:10:46  A fantastic day in D.C. Met with President Oba...  text        NaN        NaN  7.970e+17  https://twitter.com/realDonaldTrump/status/796...                                     178499     36688          NaN          NaN
```

Preprocesamiento

Limpieza de tweets

LIMPIAR EL TEXTO

```python
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
# Preparación de las herramientas de preprocesamiento de texto
nltk.download('stopwords')
stop = set(stopwords.words('english'))
sno = SnowballStemmer('english')

def cleanhtml(sentence):
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext

def cleanpunc(sentence):
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return cleaned

# Procesamiento del texto de los tweets
final_string = []
for sent in df['Tweet_Text'].values:
    filtered_sentence = []
    sent = cleanhtml(sent)
    for w in sent.split():
        cleaned_words = cleanpunc(w)
        if cleaned_words.isalpha() and len(cleaned_words) > 2:
            if cleaned_words.lower() not in stop:
                stemmed_word = sno.stem(cleaned_words.lower())
                filtered_sentence.append(stemmed_word)
    final_string.append(" ".join(filtered_sentence))

# Añadiendo la columna de tweets limpios al DataFrame
df['cleaned_tweet'] = final_string

# Visualización de los primeros registros del DataFrame modificado
#print(df["cleaned_tweet"].head())
print(df['cleaned_tweet'].iloc[0])
```

✓ 6.8s

Antes

```
Today we express our deepest gratitude to all those who have served in our armed forces. #ThankAVet https://t.co/wPk7QWpK8Z
```

Después

```
today express deepest gratitud serv arm thankavet
```

Preprocesamiento

Creación de Viralidad

CREAR PORCENTAJE DE MUCHO Y POCO

```
media_likes= df["twt_favourites_IS_THIS_LIKE_QUESTION_MARK"].median()
media_retweets=df["Retweets"].median()
df['Viral'] = df.apply(lambda x: 'mucho' if (x['twt_favourites_IS_THIS_LIKE_QUESTION_MARK'] + x['Retweets']) >= (media_likes + media_retweets) * 1.05
                                 else ('poco'), axis=1)
print(df["Viral"].value_counts())
```

[13]

```
Viral
poco      3773
mucho     3602
Name: count, dtype: int64
```

Preprocesamiento

Creación de Viralidad

CREAR PORCENTAJE DE MUCHO Y POCO

```
media_likes= df["twt_favourites_IS_THIS_LIKE_QUESTION_MARK"].median()
media_retweets=df["Retweets"].median()
df['Viral'] = df.apply(lambda x: 'mucho' if (x['twt_favourites_IS_THIS_LIKE_QUESTION_MARK'] + x['Retweets']) >= (media_likes + media_retweets) * 1.05
                            else ('poco'), axis=1)
print(df["Viral"].value_counts())
```

[13]

```
Viral
poco     3773
mucho    3602
Name: count, dtype: int64
```

Preprocesamiento

Eliminación de columnas

ELIMINAR COLUMNAS INECESAREAS

```
df.drop('Tweet_Text', axis=1, inplace=True)
df.drop('Tweet_Id', axis=1, inplace=True)
df.drop('Tweet_Url', axis=1, inplace=True)
df.drop('Date', axis=1, inplace=True)
df.drop('Time', axis=1, inplace=True)
df.drop('Media_Type', axis=1, inplace=True)
df.drop('Type', axis=1, inplace=True)
df.drop('Hashtags', axis=1, inplace=True)
df.drop('twt_favourites_IS_THIS_LIKE_QUESTION_MARK', axis=1, inplace=True)
df.drop('Retweets', axis=1, inplace=True)
df.drop('Unnamed: 10', axis=1, inplace=True)
df.drop('Unnamed: 11', axis=1, inplace=True)
y=df["Viral"].values
# Definir X como todas las columnas excepto 'Viral'
X= df.drop('Viral', axis=1).values
print(df[0:5])
```

Dataset Limpio

	cleaned_tweet	Viral
0	today express deepest gratitud serv arm thankavet	mucho
1	busi day plan new soon make import decis peopl...	mucho
2	love fact small group protest last night passi...	mucho
3	open success presidenti profession incit unfair	mucho
4	fantast day met presid obama first realli good...	mucho

Preprocesamiento

Creación de columnas

GENERAR COLUMNAS POR PALABRAS I FILTRAR POR UMBRAL

```
from sklearn.feature_extraction.text import CountVectorizer

count_vectorizer = CountVectorizer()
X = count_vectorizer.fit_transform( X[:, -1])
feature_names = count_vectorizer.get_feature_names_out()
X = pandas.DataFrame(X.toarray(), columns=feature_names)
word_frequencies = X.sum(axis=0)
filtered_words = word_frequencies[word_frequencies >= 5]
X = X[filtered_words.index]
X.to_csv('X.csv', index=False)
df.drop('cleaned_tweet', axis=1, inplace=True)
df.to_csv('y.csv', index=False)
print(X[0:5])
print(y[0:5])
print(X.shape)
```

Dataset X e y

```
   abc  abl  abolish  absolut  accept  accord  a
0    0    0        0        0       0       0
1    0    0        0        0       0       0
2    0    0        0        0       0       0
3    0    0        0        0       0       0
4    0    0        0        0       0       0
['mucho' 'mucho' 'mucho' 'mucho' 'mucho']
(7375, 1477)
```

Criterios de evaluación

Evaluar División Fija

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Ratios de Training, Validación y Test
split_ratios = [(0.7, 0.1), (0.6, 0.1), (0.5, 0.2)]  # Train, Validation ratios
results = []

for train_ratio, val_ratio in split_ratios:
    # Primer paso: dividir en entrenamiento y temporal
    X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=1-train_ratio, random_state=42)

    # Segundo paso: dividir el conjunto temporal en validación y prueba
    X_val, X_test, y_val, y_test = train_test_split(
        X_temp, y_temp, test_size=val_ratio/(val_ratio + (1-train_ratio-val_ratio)), random_state=42
    )

    # Crear y entrenar el modelo
    model = DecisionTreeClassifier()
    model.fit(X_train, y_train)

    # Evaluar el modelo
    val_accuracy = accuracy_score(y_val, model.predict(X_val))
    test_accuracy = accuracy_score(y_test, model.predict(X_test))

    # Guardar resultados
    results.append({
        'train_ratio': train_ratio,
        'val_ratio': val_ratio,
        'test_ratio': 1-train_ratio-val_ratio,
        'val_accuracy': val_accuracy,
        'test_accuracy': test_accuracy
    })
# Imprimir resultados
for res in results:
    print(f"Train: {res['train_ratio']*100}%, Validation: {res['val_ratio']*100}%, Test: {res['test_ratio']*100}%")
    print(f"Validation Accuracy: {res['val_accuracy']:.4f}, Test Accuracy: {res['test_accuracy']:.4f}")
    print()
```

Valores

Train: 70.0%, Validation: 10.0%, Test: 20.0%
Validation Accuracy: 0.6807, Test Accuracy: 0.6707

Train: 60.0%, Validation: 10.0%, Test: 30.0%
Validation Accuracy: 0.6786, Test Accuracy: 0.6504

Train: 50.0%, Validation: 20.0%, Test: 30.0%
Validation Accuracy: 0.6655, Test Accuracy: 0.6551

Criterios de evaluación

División de los datos

Python
```python
(X_train, X_test,  y_train, y_test) = train_test_split(X, y2, test_size=.3, 
random_state=1)
```

Criterios de evaluación

Single vs K fold cross validation

```python
%matplotlib inline
import pandas
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import sklearn
import sklearn.datasets as ds
import sklearn.model_selection as cv
import sklearn.neighbors as nb
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import f1_score, classification_report, confusion_matrix
from sklearn.feature_selection import SelectKBest, f_classif
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Cargar los datos (esto ya lo tienes)
X = pandas.read_csv('X.csv', sep=',', na_values="")
y = pandas.read_csv('y.csv', sep=',', na_values="")

# 1. **Single Fold Validation** - División entre datos de entrenamiento y test (70-30)
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3, random_state=42)

# Inicializar el clasificador KNN
knn = KNeighborsClassifier(n_neighbors=5)

# Entrenamiento del modelo
knn.fit(X_train, y_train)

# Predicciones
y_pred = knn.predict(X_val)

# Calcular la precisión y otras métricas
accuracy = accuracy_score(y_val, y_pred)
print("\nSingle Fold Validation Results:")
print(f"Accuracy: {accuracy}")

# 2. **K-Fold Cross-Validation** - Usando 5 pliegues
cv = 10

# Inicializar KNN nuevamente
knn = KNeighborsClassifier(n_neighbors=5)

# Realizar K-Fold Cross-Validation
scores = cross_val_score(knn, X, y, cv=cv, scoring='accuracy')

# Mostrar los resultados de la validación cruzada
print("\nK-Fold Cross-Validation Results:")
print(f"Average accuracy: {scores.mean()}")
```

Valores

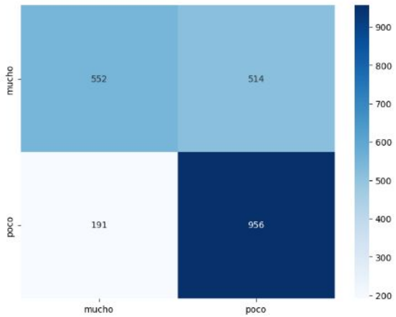
Single Fold Validation Results:
Accuracy: 0.6037053773158608

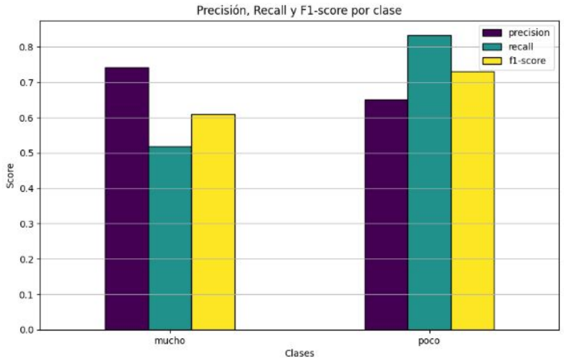
K-Fold Cross-Validation Results:
Average accuracy: 0.5919848282607657

Criterios de evaluación

Métricas de evaluación

```python
print(sklearn.metrics.accuracy_score(y_test, pred))
print(classification_report(y_test, pred))
cm = (confusion_matrix(y_test, pred))
plt.figure(figsize=(6, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Poco",
"Mucho"], yticklabels=["Poco", "Mucho"])
plt.show()
# Reporte de clasificación
report = sklearn.metrics.classification_report(y_test, pred,
output_dict=True)
# Visualización de métricas globales del reporte
metrics = ['precision', 'recall', 'f1-score']
report_df = pandas.DataFrame(report).T
report_df = report_df[metrics]
# Plot de métricas
report_df.iloc[:-3].plot(kind='bar', figsize=(10, 6), colormap='viridis',
edgecolor='black')
plt.title("Precisión, Recall y F1-score por clase")
plt.ylabel("Score")
plt.xlabel("Clases")
plt.xticks(rotation=0)
plt.grid(axis='y')
plt.show()
```





Naive Bayes

Independencia de Variables en Naïve Bayes

```python
from sklearn.feature_selection import mutual_info_classif
from sklearn.metrics import pairwise_distances
word_correlations = np.corrcoef(X, rowvar=False)
top_correlated_pairs = []
threshold = 0.5
for i in range(word_correlations.shape[0]):
    for j in range(i + 1, word_correlations.shape[1]):
        if abs(word_correlations[i, j]) > threshold:
            top_correlated_pairs.append((i, j, word_correlations[i, j]))
print("Top Correlated Pairs")
print(len(top_correlated_pairs))
```

```python
Top Correlated Pairs
53
```

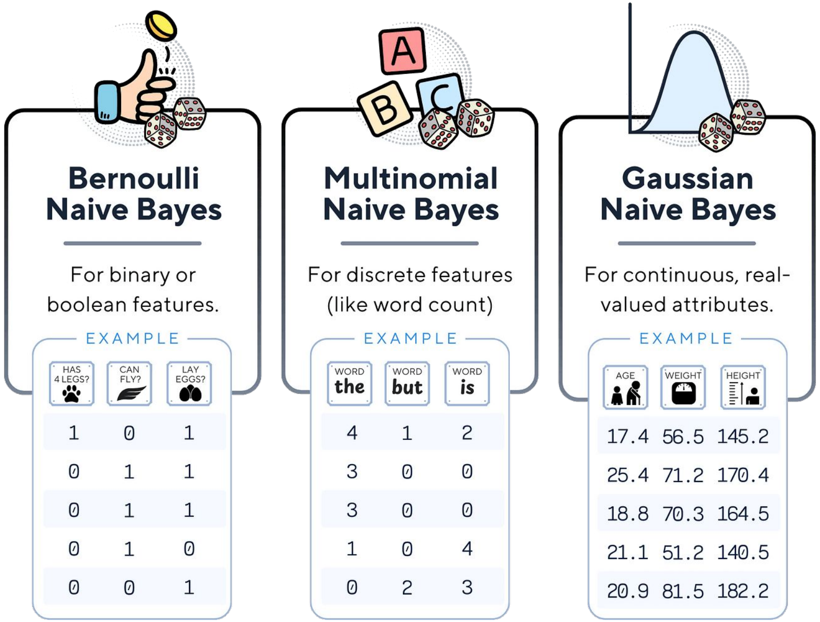
Tenemos suficientes datos

```
    print(X.shape)
    print(y.shape)
[2]

... (7375, 1477)
    (7375, 1)
```

poco 3773
mucho 3602

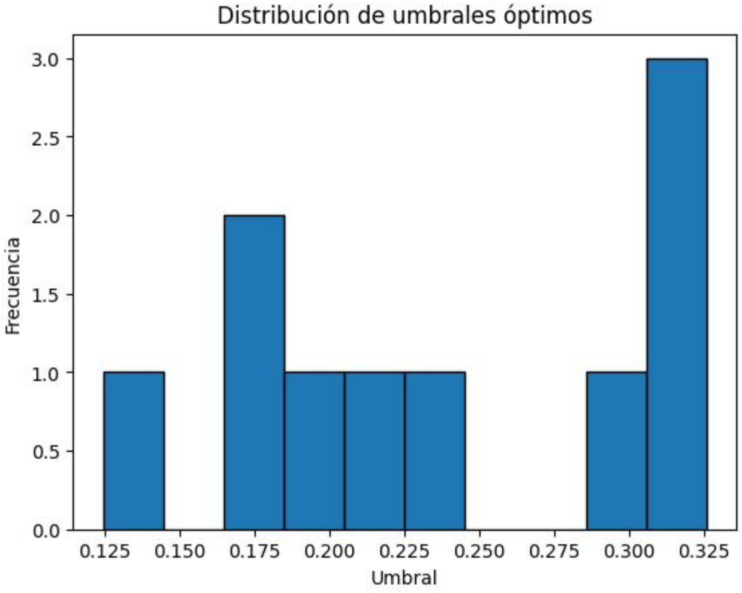
Naive Bayes

Decisión de utilizar **MultinomialNB**



Naive Bayes

Optimización del umbral



- Sensible a partición de datos
- umbral menor a 0,5

Naive Bayes

Resultados

Con umbral

Python

```
              precision    recall  f1-score   support

  0.0(mucho)       0.50      0.63      1066
   1.0(poco)       0.66      0.92      0.77      1147

    accuracy                           0.72      2213
   macro avg       0.76      0.71      0.70      2213
weighted avg       0.75      0.72      0.70      2213
```

Sin umbral

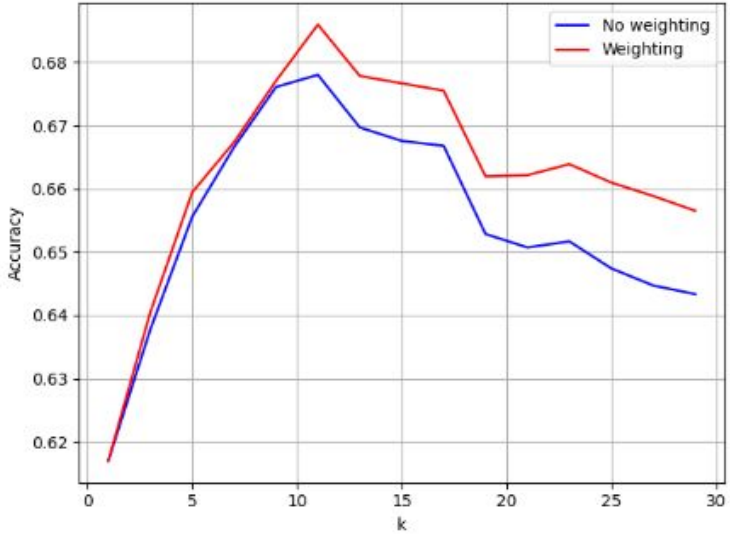
Python

```
0.7144148215092635
              precision    recall  f1-score   support

         0.0       0.72      0.67      0.69      1066
         1.0       0.71      0.76      0.73      1147

    accuracy                           0.71      2213
   macro avg       0.71      0.71      0.71      2213
weighted avg       0.71      0.71      0.71      2213
```

KNN

Weighting" o "No weighting

```
lr = []
for ki in range(1,30,2):
    cv_scores = cross_val_score(nb.KNeighborsClassifier(n_neighbors=ki), X=X_train, y=y_train, cv=10)
    lr.append(np.mean(cv_scores))
plt.plot(range(1,30,2),lr,'b',label='No weighting')

lr = []
for ki in range(1,30,2):
    cv_scores = cross_val_score(nb.KNeighborsClassifier(n_neighbors=ki,weights='distance'), X=X_train, y=y_train, cv=10)
    lr.append(np.mean(cv_scores))
plt.plot(range(1,30,2),lr,'r',label='Weighting')
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.legend(loc='upper right')
plt.grid()
plt.tight_layout()

plt.show()
```



KNN

Mejor parámetro k

```
from sklearn.model_selection import GridSearchCV
params = {'n_neighbors':list(range(1,30,2)), 'weights':('distance','uniform')}
knc = nb.KNeighborsClassifier()
clf = GridSearchCV(knc, param_grid=params,cv=10,n_jobs=-1) # If cv is integer, by default is Stratifyed
clf.fit(X_train, y_train)
print("Best Params=",clf.best_params_, "Accuracy=", clf.best_score_)
```

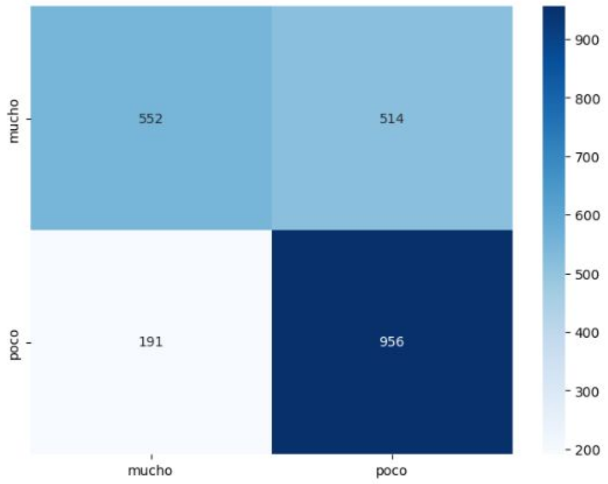
```
Best Params= {'n_neighbors': 11, 'weights': 'distance'} Accuracy= 0.6859681675738083
```

KNN

Resultados

Python

	precision	recall	f1-score	support
mucho	0.74	0.52	0.61	1066
poco	0.65	0.83	0.73	1147
accuracy			0.68	2213
macro avg	0.70	0.68	0.67	2213
weighted avg	0.69	0.68	0.67	2213



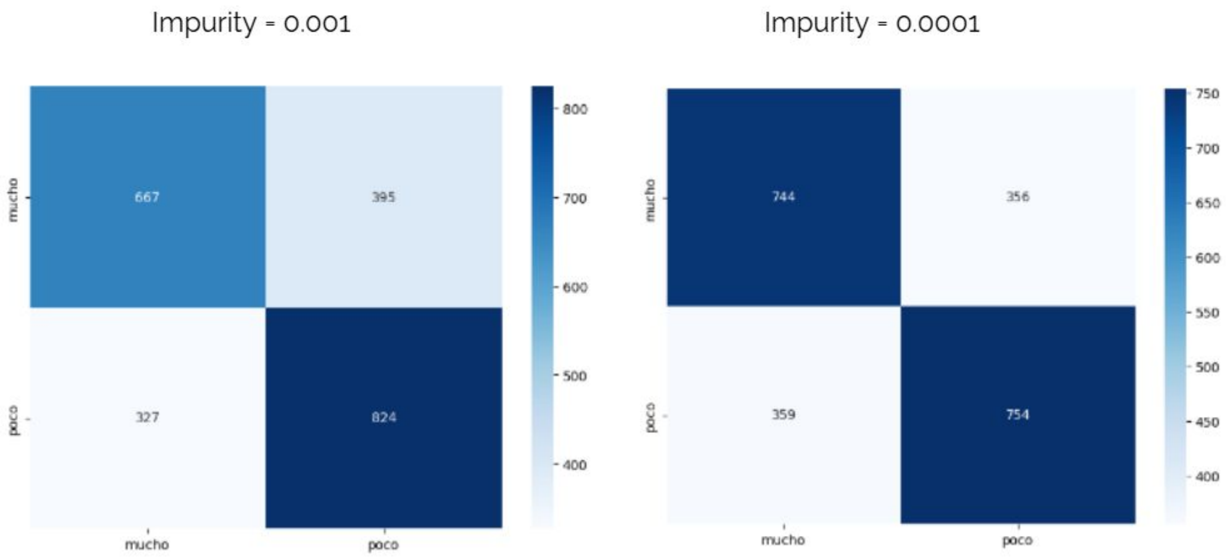
Decision Trees

Evaluación del Modelo

```
from sklearn import tree

clf = tree.DecisionTreeClassifier(criterion='entropy', min_impurity_decrease=0.0001)
pred = clf.fit(X_train, y_train).predict(X_test)
print("Accuracy on test set:", sklearn.metrics.accuracy_score(y_test, pred))
epsilon = sklearn.metrics.accuracy_score(y_test, pred)
print(sklearn.metrics.classification_report(y_test, pred))
conf_matrix = sklearn.metrics.confusion_matrix(y_test, pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y), yticklabels=np.unique(y))
plt.show()

# Reporte de clasificación
report = sklearn.metrics.classification_report(y_test, pred, output_dict=True)
# Visualización de métricas globales del reporte
metrics = ['precision', 'recall', 'f1-score']
report_df = pandas.DataFrame(report).T
report_df = report_df[metrics]
# Plot de métricas
report_df.iloc[:-3].plot(kind='bar', figsize=(10, 6), colormap='viridis', edgecolor='black')
plt.title("Precisión, Recall y F1-score por clase")
plt.ylabel("Score")
plt.xlabel("Clases")
plt.xticks(rotation=0)
plt.grid(axis='y')
plt.show()

print("Confidence interval: ",proportion_confint(count=epsilon*X_test.shape[0], nobs=X_test.shape[0], alpha=0.05, method='binom_test'))
```

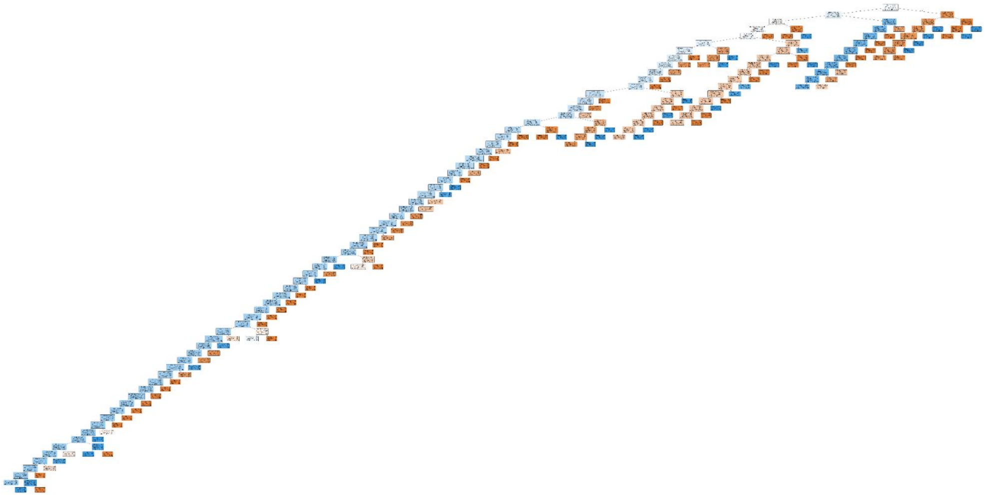
Decision Trees

min_impurity_decrease



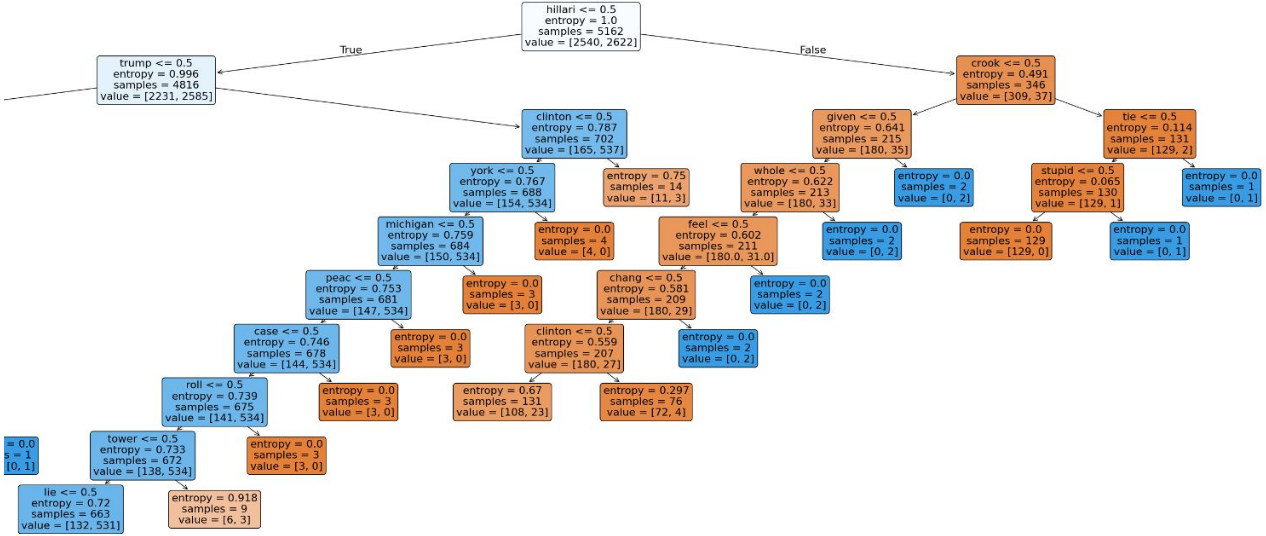
Decision Trees

Interpretación del Arbol



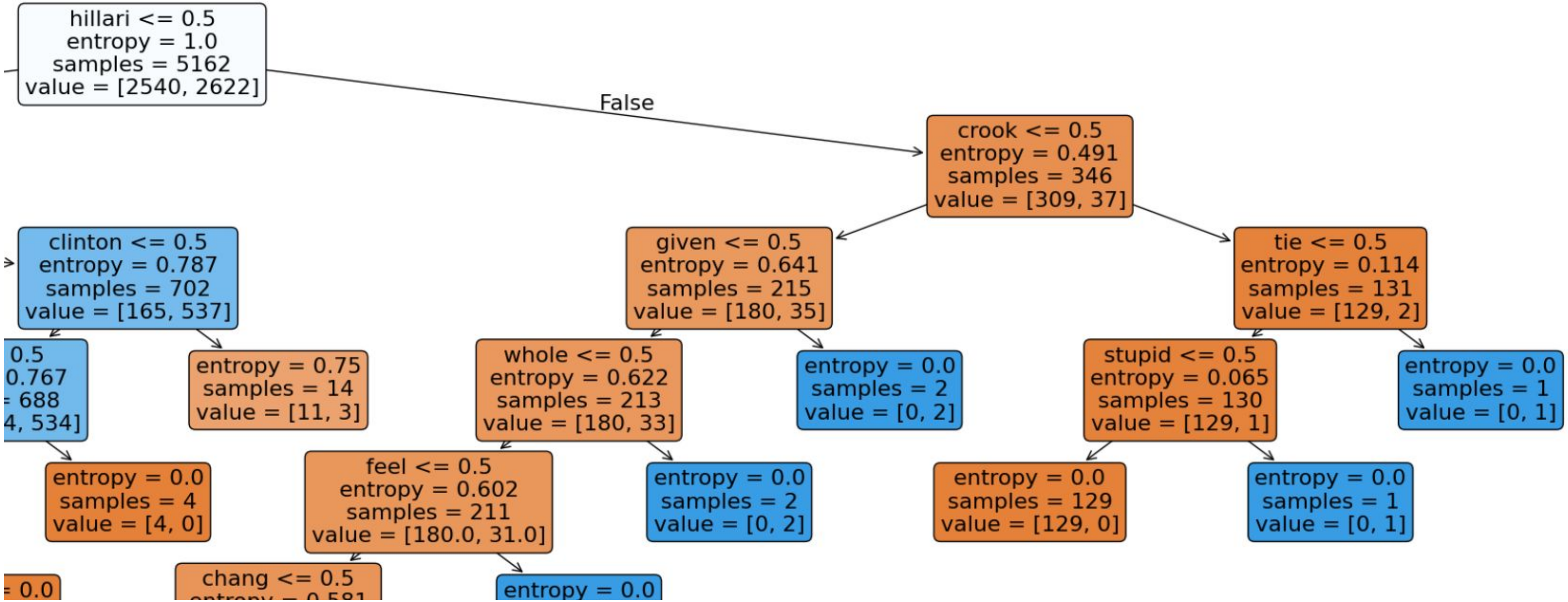
Decision Trees

Interpretación del Arbol



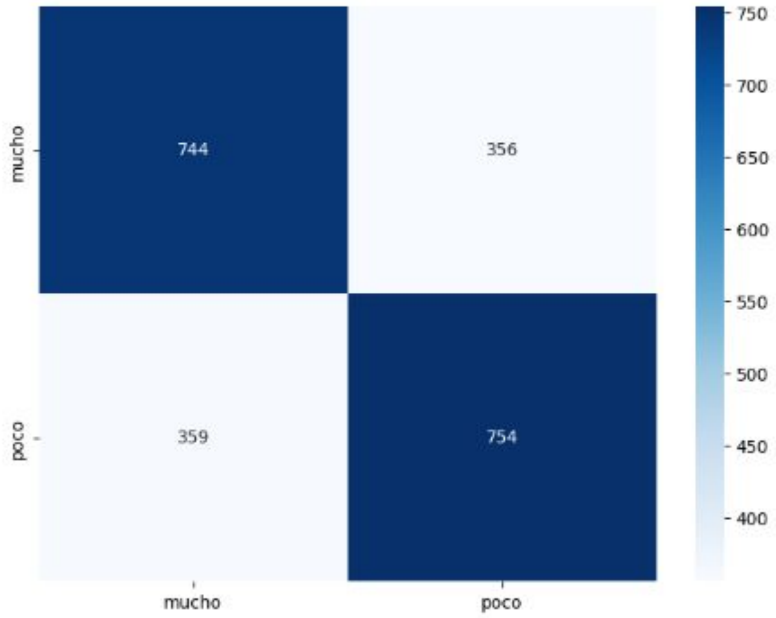
Decision Trees

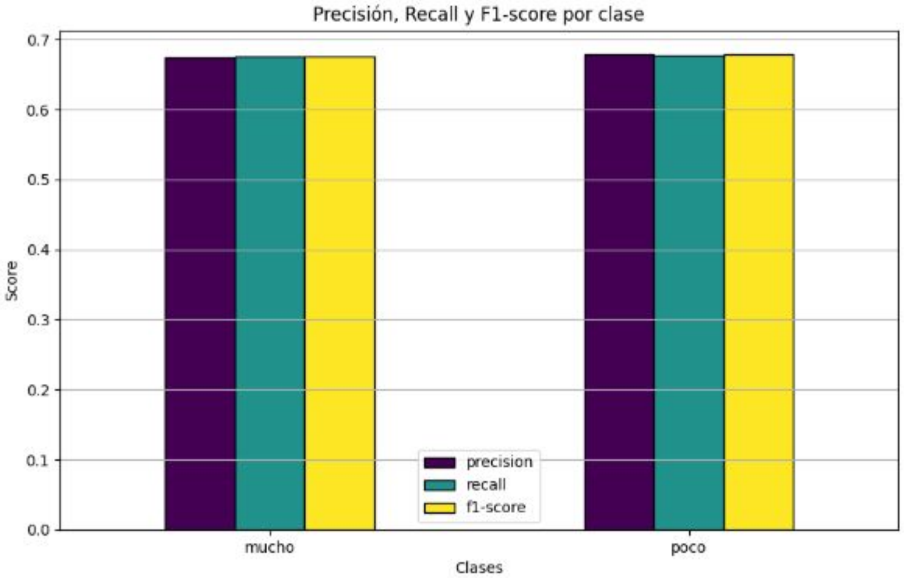
Interpretación del Arbol



Decision Trees

Resultados

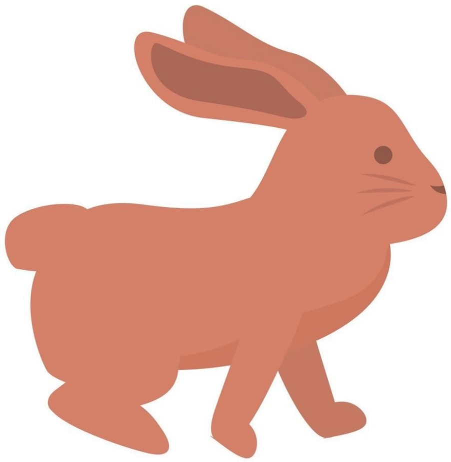




Support Vector Machines

Elección del Kernel y parámetros

Kernel lineal



Kernel Polinomial y RBF

Support Vector Machines

Método para acelerar el training

Reducción número de características

```
print(X.shape)
print(y.shape)

(295, 34)
(295, 1)
```

Reducción de parámetros

```
# Lista de valores de C a probar. Usualmente
Cs = np.logspace(-3, 5, num=3, base=10.0)
```

Support Vector Machines

Resultados

Kernel Lineal

- Mejor parámetro C encontrado: 10.0
- Exactitud en el conjunto de test: 64.04%
- Número de soportes: **152** (73.78% de los datos de entrenamiento)
- Intervalo de confianza en validación cruzada para el mejor C:
- Precisión promedio: 58.28%
- Intervalo de confianza (95%): [50.86%, 65.71%]

Kernel Polinomial

- Mejor combinación de parámetros: C = 10000.0, Grado = 3
- Exactitud en el conjunto de test: 58.43%
- Número de soportes: **137** (66.50% de los datos de entrenamiento)
- Intervalo de confianza en validación cruzada para el mejor C:
- Precisión promedio: 55.34%
- Intervalo de confianza (95%): [54.49%, 56.19%]

Kernel RBF (Radial Basis Function)

- Mejor combinación de parámetros: C = 10000.0, Gamma = 1.0
- Exactitud en el conjunto de test: 61.80%
- Número de soportes: **178** (86.41% de los datos de entrenamiento)
- Intervalo de confianza en validación cruzada para el mejor C y Gamma:
- Precisión promedio: 52.92%
- Intervalo de confianza (95%): [52.05%, 53.78%]

Meta Methods

Majority Voting

Hard

Soft

Meta Methods

Resultados

HARD

```
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)

eclf = VotingClassifier(estimators=[('nb', clf1), ('knn3', clf2), ('dt', clf3)], voting='hard')
scores = cross_val_score(eclf, X, y, cv=cv, scoring='accuracy')
print("Accuracy: %0.3f [%s]" % (scores.mean() , "Majority Voting"))
```

```
Accuracy: 0.636 [Majority Voting]
```

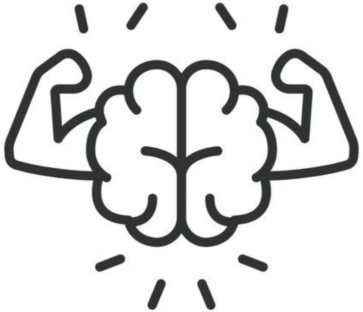
SOFT

```
eclf = VotingClassifier(estimators=[('nb', clf1), ('knn3', clf2), ('dt', clf3)],voting='soft', weights=[2.5,1.5,2])
scores = cross_val_score(eclf, X, y, cv=cv, scoring='accuracy')
print("Accuracy: %0.3f [%s]" % (scores.mean(), "Weighted Voting"))
```

```
Accuracy: 0.635 [Weighted Voting]
```

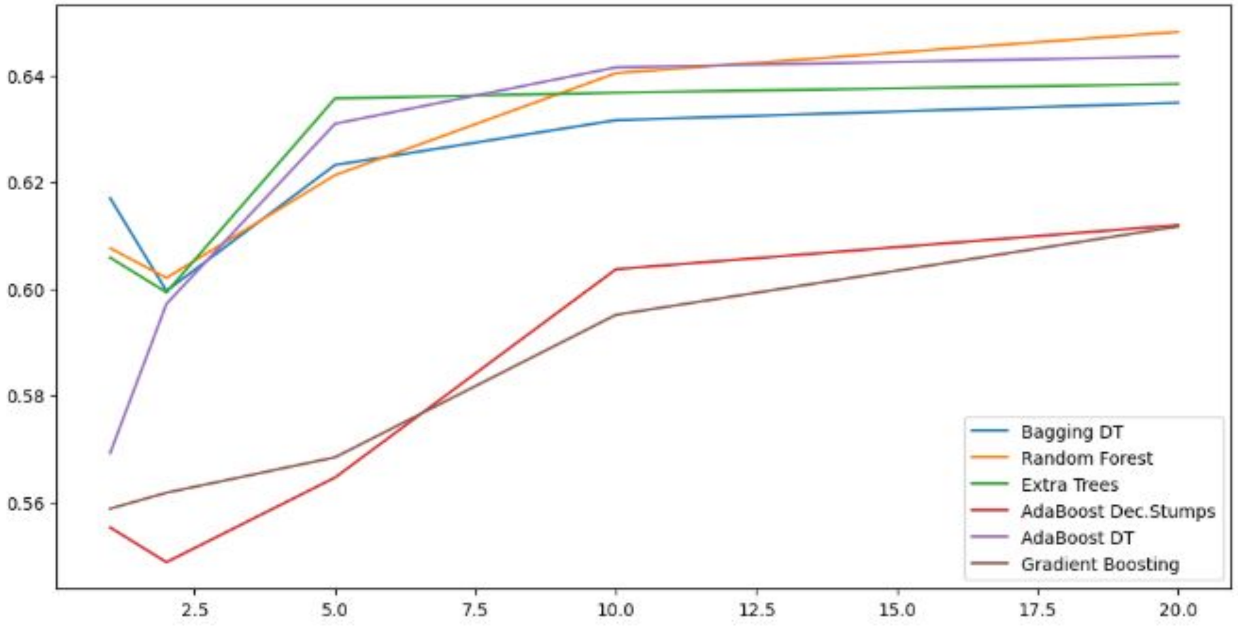
Meta Methods

Bagging, Boosting, Random Forest



Meta Methods

Resultados



Conclusiones

Cross Validation

Cross Validation	Accuracy
Single-Fold CV	60.37%
K-Fold CV	59.20%

Conclusiones

Learning Methods

Model/Classificador	Accuracy
Naïve Bayes	72%
KNN	68%
Decision Tree	67%
SVM	58%
Meta-learning (Random Forest)	71%

Model/Classificador	Class	precision	recall	f1-score
Naïve Bayes	Mucho	0.72	0.67	0.69
	Poco	0.71	0.76	0.73
KNN	Mucho	0.74	0.52	0.61
	Poco	0.65	0.83	0.73
Decision Tree	Mucho	0.67	0.68	0.68
	Poco	0.68	0.68	0.68
SVM	Mucho	0.68	0.40	0.51
	Poco	0.54	0.79	0.64
Meta-learning (Random Forest)	Mucho	0.72	0.70	0.71
	Poco	0.71	0.73	0.72

Conclusiones

Intervalos de confianza

Model/Classificador	Intervalos de Confianza
Naïve Bayes	(0.696, 0.733)
KNN	(0.662, 0.701)
Decision Tree	(0.657, 0.696)
SVM	(0.509, 0.657)
Meta-learning (Random Forest)	(0.636, 0.658)