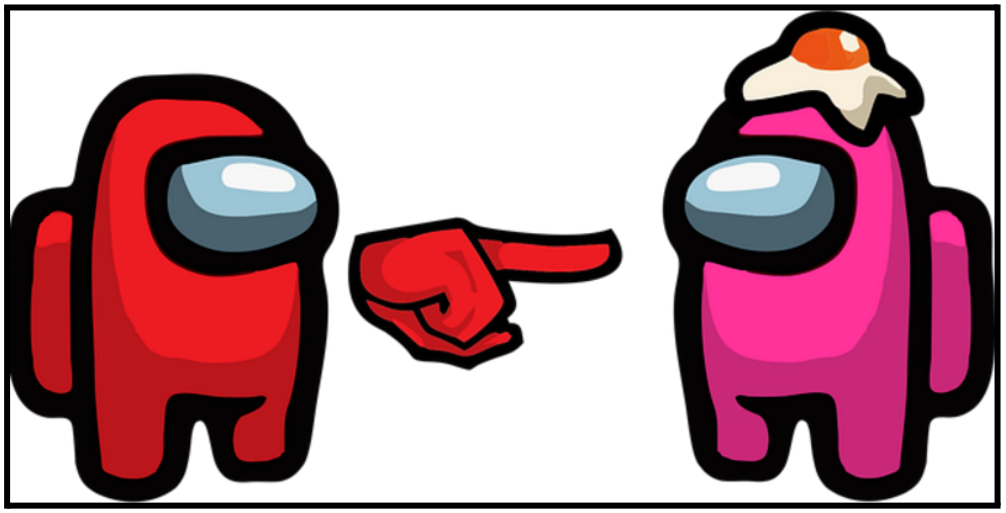
Data mining job

REAL OR FAKE JOBS



Raúl Gamero Ramón - Henok Argudo Vera

MIDA

Índice

Descripción del problema

Siempre que nos aplicamos a ofertas de trabajo, nos fiamos de que la oferta de trabajo sea real. Pero realmente, ese trabajo existe?
Esto fue lo primero que nos preguntamos después de ver el dataset de fake_jobs_postings que encontramos en Kaggle. Y es que realmente hay más 'fake jobs' de los que nos imaginamos.

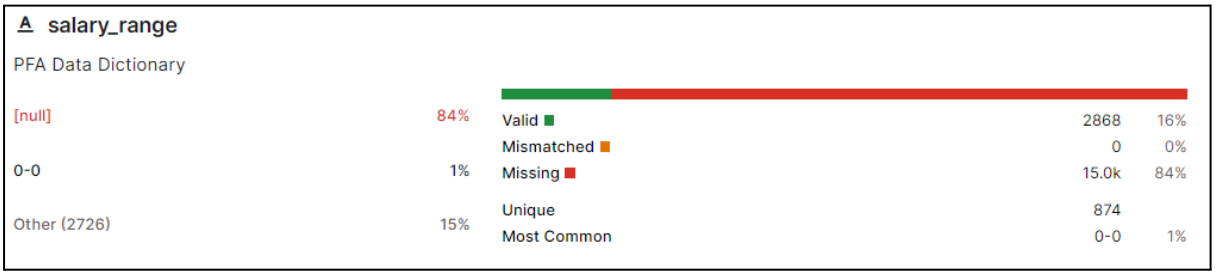
El dataset original contenía 18.000 descripciones de puestos de trabajo, de los cuales 800 eran fake. Tenía 18 columnas, cada una describiendo el puesto de trabajo.
Once de ellas son categóricas, una es numérica, y las 6 restantes son binarias.

Coulmns	Description
job_id	Unique Job ID
title	The title of the job ad entry
location	Geographical location of the job ad
department	Corporate department (e.g. sales)
salary_range	Indicative salary range (e.g. $50,000-$60,000)
company_profile	A brief company description
description	The details description of the job ad
requirements	Enlisted requirements for the job opening
benefits	Enlisted offered benefits by the employer
telecommuting	True for telecommuting positions
has*company*logo	True if company logo is present
has_questions	True if screening questions are present
employment_type	Full-type, Part-time, Contract, etc
required_experience	Executive, Entry level, Intern, etc
required_education	Doctorate, Master's Degree, Bachelor, etc
industry	Automotive, IT, Health care, Real estate, etc
function	Consulting, Engineering, Research, Sales etc
fraudulent	target - Classification attribute

El dataset contiene gran cantidad de missings, por lo que habría que hacer un gran esfuerzo en el 'preprocessing'. Que haya muchos missing infiere que probablemente obtengamos peores accuracies en nuestros modelos, pero realmente lo que buscamos en este proyecto es entender los diferentes pasos a seguir en la minería de datos y aprender a aplicarlos en un proyecto real.

Pre-processing

Para hacer el preprocessing, vamos a definir dos tipos de datos del dataframe, los numéricos y los categóricos.
Por parte de los numéricos solo existe una columna que es la de ‘salary_range’.



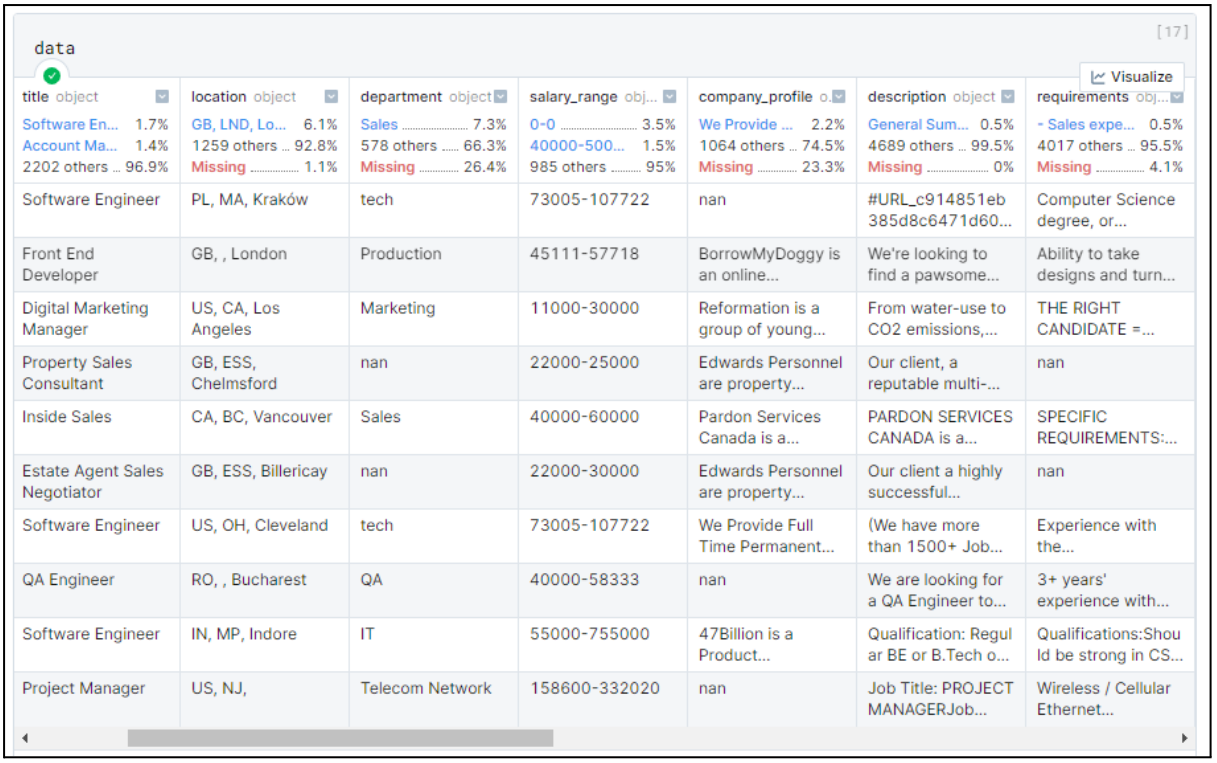
Esta columna tiene muchos valores perdidos, por ello decidimos tratar de “recuperar” los valores perdidos en esta columna tomando como referencia los valores que tengan otras filas con el mismo valor en la columna *title*. Es decir, deducimos el salario que debería de tener un puesto de trabajo en función de las que sí que tienen valor en ‘salary_range’.
Básicamente, cogeremos todos los valores para un mismo puesto de trabajo (mismo ‘title’) y hacemos la media de estos valores. Finalmente asignamos a los valores nulos con ese mismo ‘title’ la media calculada.

Por otra parte tenemos los datos categóricos, que són prácticamente todos los demás.
Para sustituir estos valores perdidos, lo haremos de la misma manera que hicimos con ‘salary-range’, y definiremos una función para ello.

```
def preprocessing_function (dic, reference, missing):
```

Dónde ‘dic’ contiene los nombres de todos los trabajos del dataset, ‘reference’ indica la columna que se toma como referencia para obtener los valores y ‘missing’ indica la columna sobre la que se quieren insertar nuevos valores.
El primer paso es recorrer todas las filas del dataset y por cada nuevo valor (diferente de null) que se encuentre en la columna de ‘missing’ se asociará al valor que corresponda a la columna de ‘reference’.
Realizamos un último recorrido del dataset en el que asignaremos un valor a todas las casillas de la columna ‘missing’ que tengan valor null y con un valor en la columna ‘reference’ igual a alguna de las filas que se han recorrido anteriormente.

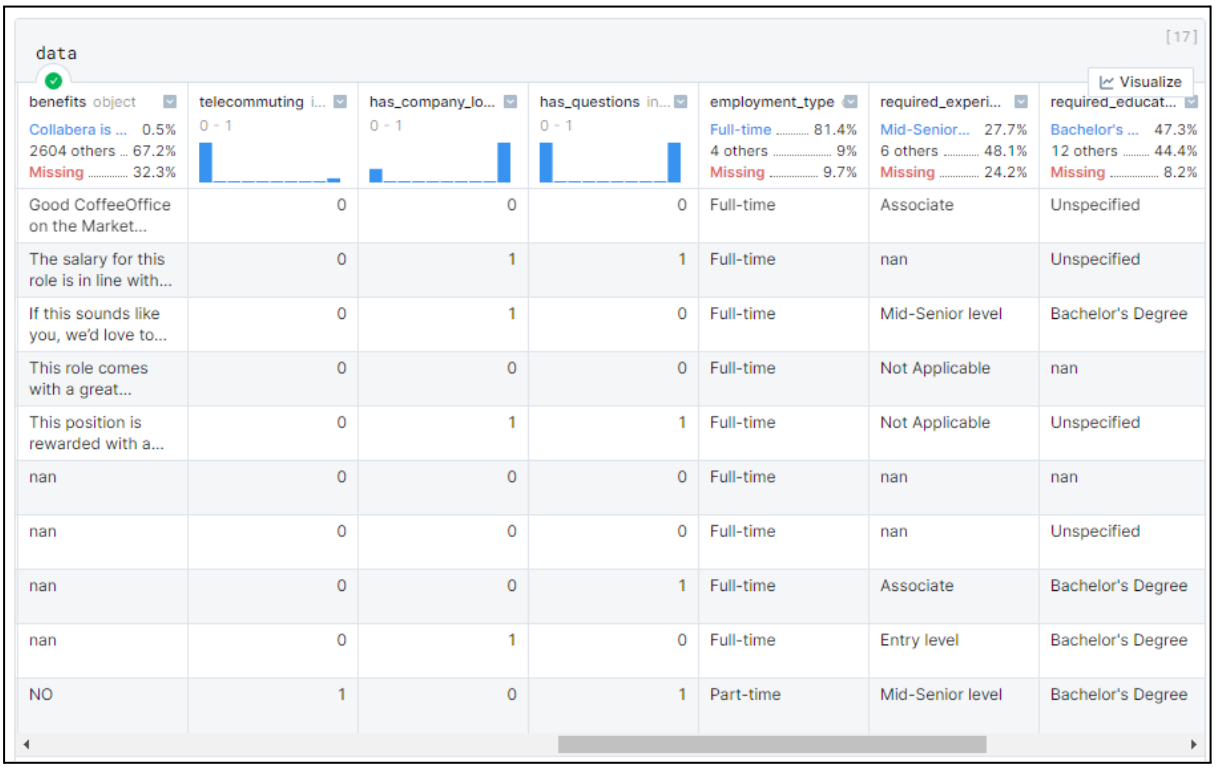
Una vez intentado sustituir todos los valores perdidos, nos queda arreglar todos los missing restantes.



data [17]

Visualize

title object	location object	department object	salary_range obj...	company_profile o...	description object	requirements obj...
Software En... 1.7% Account Ma... 1.4% 2202 others ... 96.9%	GB, LND, Lo... 6.1% 1259 others ... 92.8% Missing ... 1.1%	Sales ... 7.3% 578 others ... 66.3% Missing ... 26.4%	0-0 ... 3.5% 40000-500... 1.5% 985 others ... 95%	We Provide ... 2.2% 1064 others ... 74.5% Missing ... 23.3%	General Sum... 0.5% 4689 others ... 99.5% Missing ... 0%	- Sales expe... 0.5% 4017 others ... 95.5% Missing ... 4.1%
Software Engineer	PL, MA, Kraków	tech	73005-107722	nan	#URL_c914851eb 385d8c6471d60...	Computer Science degree, or...
Front End Developer	GB, , London	Production	45111-57718	BorrowMyDoggy is an online...	We're looking to find a pawsome...	Ability to take designs and turn...
Digital Marketing Manager	US, CA, Los Angeles	Marketing	11000-30000	Reformation is a group of young...	From water-use to CO2 emissions,...	THE RIGHT CANDIDATE =...
Property Sales Consultant	GB, ESS, Chelmsford	nan	22000-25000	Edwards Personnel are property...	Our client, a reputable multi-...	nan
Inside Sales	CA, BC, Vancouver	Sales	40000-60000	Pardon Services Canada is a...	PARDON SERVICES CANADA is a...	SPECIFIC REQUIREMENTS:...
Estate Agent Sales Negotiator	GB, ESS, Billericay	nan	22000-30000	Edwards Personnel are property...	Our client a highly successful...	nan
Software Engineer	US, OH, Cleveland	tech	73005-107722	We Provide Full Time Permanent...	(We have more than 1500+ Job...	Experience with the...
QA Engineer	RO, , Bucharest	QA	40000-58333	nan	We are looking for a QA Engineer to...	3+ years' experience with...
Software Engineer	IN, MP, Indore	IT	55000-755000	47Billion is a Product...	Qualification: Regular BE or B.Tech o...	Qualifications:Should be strong in CS...
Project Manager	US, NJ,	Telecom Network	158600-332020	nan	Job Title: PROJECT MANAGERJob...	Wireless / Cellular Ethernet...



data [17]

Visualize

benefits object	telecommuting i...	has_company_lo...	has_questions in...	employment_type	required_experi...	required_educat...
Collabera is ... 0.5% 2604 others ... 67.2% Missing ... 32.3%	0 - 1	0 - 1	0 - 1	Full-time ... 81.4% 4 others ... 9% Missing ... 9.7%	Mid-Senior... 27.7% 6 others ... 48.1% Missing ... 24.2%	Bachelor's ... 47.3% 12 others ... 44.4% Missing ... 8.2%
Good CoffeeOffice on the Market...	0	0	0	Full-time	Associate	Unspecified
The salary for this role is in line with...	0	1	1	Full-time	nan	Unspecified
If this sounds like you, we'd love to...	0	1	0	Full-time	Mid-Senior level	Bachelor's Degree
This role comes with a great...	0	0	0	Full-time	Not Applicable	nan
This position is rewarded with a...	0	1	1	Full-time	Not Applicable	Unspecified
nan	0	0	0	Full-time	nan	nan
nan	0	0	0	Full-time	nan	Unspecified
nan	0	0	1	Full-time	Associate	Bachelor's Degree
nan	0	1	0	Full-time	Entry level	Bachelor's Degree
NO	1	0	1	Part-time	Mid-Senior level	Bachelor's Degree

Vemos que la columna de 'benefits' aún tiene más de un 30% de missing después de hacer todo el preprocessing, sabemos que no hay ningún motivo importante por el cual deba tener tantos datos perdidos, por lo tanto decidimos que eliminarla es la mejor opción.
También, los datos categóricos que sean missing, los sustituiremos por una nueva categoría que llamaremos 'non defined'. Esto lo hacemos con el objetivo de no eliminar filas ya que no queremos sesgar el conjunto de datos.

data [24]

Visualize

title object	location object	department object	salary_range obj...	company_profile o...	description object	requirements obj...
Software En... 1.7% Account Ma... 1.4% 2197 others ... 96.9%	GB, LND, Lo... 6.1% US, NY, New... 3.7% 1255 others ... 90.2%	Non defined ... 26.4% Sales ... 7.3% 578 others ... 66.4%	0-0 ... 3.5% 40000-500... 1.5% 984 others ... 95%	Non defined ... 23.3% We Provide ... 2.2% 1064 others ... 74.5%	General Sum... 0.5% We're lookin... 0.2% 4684 others ... 99.3%	Non defined ... 4% - Sales expe... 0.5% 4017 others ... 95.5%
Marketing Intern	US, NY, New York	Marketing	0-0	We're Food52, and we've created a...	Food52, a fast-growing, James...	Experience with content...
Accounting Clerk	US, MD,	Non defined	24007-29008	Non defined	Job OverviewApex is an environment...	Accounts Payable & Accounts...
Head of Content (m/f)	DE, BE, Berlin	ANDROIDPIT	20000-28000	Founded in 2009, the Fonpit AG ros...	Your Responsibilities: ...	Your Know-How: ...
ASP.net Developer Job opportunity a...	US, NJ, Jersey City	Non defined	100000-120000	Non defined	Position : #URL_86fd830a...	Position : #URL_86fd830a...
VP of Sales - Vault Dragon	SG, 01, Singapore	Sales	120000-150000	Jungle Ventures is the leading...	About Vault Dragon Vault...	Key Superpowers3-5...
Visual Designer	US, NY, New York	Product	35000-50000	Kettle is an independent digit...	Kettle is hiring a Visual...	Own and develop our visual identity...
Marketing Assistant	US, TX, Austin	Marketing	21000-26500	IntelliBright was created to levera...	IntelliBright is growing fast and ...	Job RequirementsAss...
Vice President, Sales and...	US, CA, Carlsbad	Businessfriend.com	100000-120000	WDM Group is an innovative, forwar...	#URL_eda2500ddcedb60957fcd7f5...	Job Requirements:A...
Software Applications...	US, KS,	Non defined	50000-65000	Non defined	Day to Day-Install, upgrade and...	Must Have's3+ years of...
Jr. Developer	US	Non defined	40000-50000	Non defined	Entry level Software...	Non defined

Finalmente, solo nos queda transformar el dataset a valores numéricos, para poder aplicar algoritmos de data mining. Para ello recurrimos al mean encoding, quedando el dataset con el siguiente formato.

data [26]

Visualize

title float64 0.0 - 0.99988355...	location float64 0.0 - 0.99999420...	department float... 0.0 - 0.99988355...	salary_range floa... 0.0 - 0.99988355...	company_profile f... 0.0 - 0.99999921...	description float... 7.816336044707...	requirements flo... 5.775510773520...
6.333615537613 8344e-09	0.041025641025 641026	0.0	0.022222222222 222223	5.127506722231 591e-05	0.056281213983 86477	0.056281213983 86477
0.300307911154 4562	0.380665639000 2519	0.064139941690 9621	0.056281213983 86477	0.144389438943 8944	0.056281213983 86477	0.015136349685 249436
0.056281213983 86477	0.0	0.056281213983 86477	0.015136349685 249436	0.015136349685 249436	0.056281213983 86477	0.056281213983 86477
0.015136349685 249436	0.002669185716 6782603	0.064139941690 9621	1.160042320025 2607e-10	0.144389438943 8944	0.056281213983 86477	0.056281213983 86477
0.056281213983 86477	2.555047162850 63e-06	0.034300791556 72823	0.000139162232 9800143	0.015136349685 249436	0.056281213983 86477	0.056281213983 86477
3.458016055410 5546e-07	0.041025641025 641026	0.0	0.024390243902 439025	0.000139162232 9800143	0.015136349685 249436	0.006708885161 828739
5.127506722231 591e-05	0.109375	0.0	0.000376681775 03177614	0.015136349685 249436	0.056281213983 86477	0.056281213983 86477
0.056281213983 86477	1.721654869556 5304e-08	0.056281213983 86477	1.160042320025 2607e-10	3.458016055410 5546e-07	0.015136349685 249436	0.056281213983 86477
0.056281213983 86477	0.015136349685 249436	0.064139941690 9621	0.037735849056 60377	0.144389438943 8944	0.056281213983 86477	0.056281213983 86477
0.015136349685 249436	0.093749999999 99872	0.064139941690 9621	0.0125	0.144389438943 8944	0.000376681775 03177614	0.048076923076 92308

Criterio de evaluación de los modelos

Para elegir la mejor técnica con la que evaluar nuestros resultados (cross-validation o k-fold cross-validation), hay diversos aspectos a tener en cuenta, el tamaño del dataset, conjunto de datos balanceado y tiempo de ejecución para el caso de k-fold, respecto al tamaño actual de nuestro dataset.
Usaremos k-fold ya que se utiliza una proporción más alta del conjunto de datos para entrenar el modelo, lo que puede resultar en un modelo más preciso. También creemos que no habrá mucha diferencia en el tiempo de ejecución entre uno u otro.
Usaremos k=10 ya que proporciona un buen compromiso entre la precisión y la velocidad de ejecución. Tendremos suficientes datos para el entrenamiento y será de ejecución relativamente rápida.
Para la métrica de evaluación, tenemos varias posibilidades. En nuestro caso tenemos que predecir el valor de un caso binario. Es decir, si la oferta de trabajo será fraudulenta o no.
Fijándonos en los resultados de está columna, nos damos cuenta que el dataset está desbalanceado.

```
Representación de la clase "fraudulento": 5.63%
Representación de la clase "no fraudulento": 94.37%
```

Esto provoca que el accuracy no sea una buena medida de evaluación.
Si escogieramos el accuracy, lo que pasaría es que probablemente nos saliera un accuracy muy alto, pero realmente lo que nos importa son los casos en los que la oferta de trabajo es fraudulenta, que sería en los que fallaría más.
A partir de aquí podemos escoger entre precisión, recall o f1-score.
Como el f1-score se utiliza para combinar las medidas de precisión y recall en un sólo valor, nos resultará realmente práctico, ya que hace más fácil el poder comparar el rendimiento combinado de la precisión y la exhaustividad entre varias soluciones. Por lo tanto, usaremos el f1-score.

Ejecución métodos machine learning

Naive Bayes

Para utilizar Naive Bayes, consideramos cada variable condicionalmente independiente de la clase, ya que esto es 'naive'. Esto quiere decir que las variables no estarán condicionadas de si los otros sucesos ocurren o no.
Como ya dijimos, nuestro dataset es desbalanceado. Sabiendo esto y que el dataset tiene un buen tamaño, podemos asumir que tendremos suficientes elementos para obtener probabilidades fiables.

Dicho esto, podemos ejecutar el modelo, y utilizando un k-fold con k=10 obtenemos los siguientes resultados:

	precision	recall	f1-score	support
0.0	0.99	0.97	0.98	4913
1.0	0.62	0.90	0.74	293
accuracy			0.96	5206
macro avg	0.81	0.93	0.86	5206
weighted avg	0.97	0.96	0.97	5206

Obtenemos un f1 de 98% para no fraudulentos y un 74% para fraudulentos, haciendo así una media de precisión del 86%.

Mejora del rendimiento con Threshold

Ya que tenemos un dataset binario y desbalanceado podemos ajustar el valor de threshold para mejorar el valor de f1. Esto se hace para que el modelo sea más sensible a la clase menos frecuente, de manera que el modelo sea más propenso a clasificar una predicción como positiva para la clase menos frecuente.
Después de ajustar el threshold obtenemos los siguientes resultados:

Selected threshold in 10-fold cross validation: [0.99998176 0.99998176]

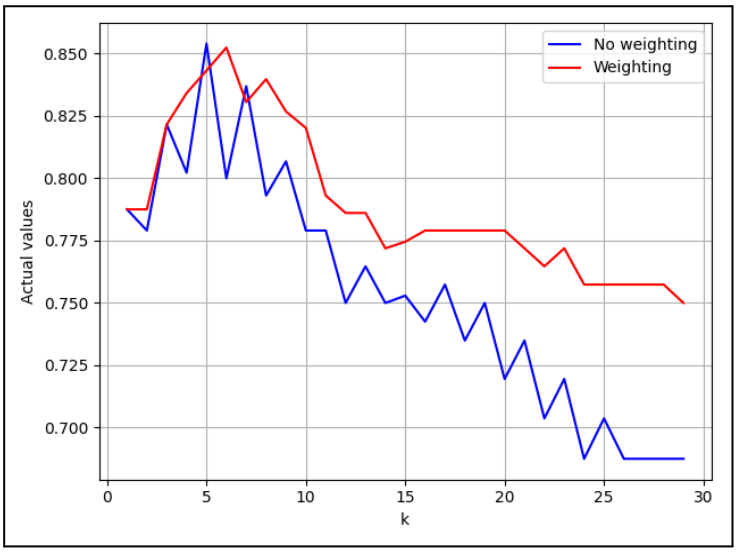
	precision	recall	f1-score	support
0.0	0.99	0.98	0.98	1467
1.0	0.70	0.85	0.77	95
accuracy			0.97	1562
macro avg	0.84	0.91	0.88	1562
weighted avg	0.97	0.97	0.97	1562

Podemos observar una mejora de 86% a 88%, gracias a que ahora la clase de fraudulento pasa de tener un f1 de 0.74 a 0.77.

KNN

Encontrar mejor-K para KNN y Weighted-KNN

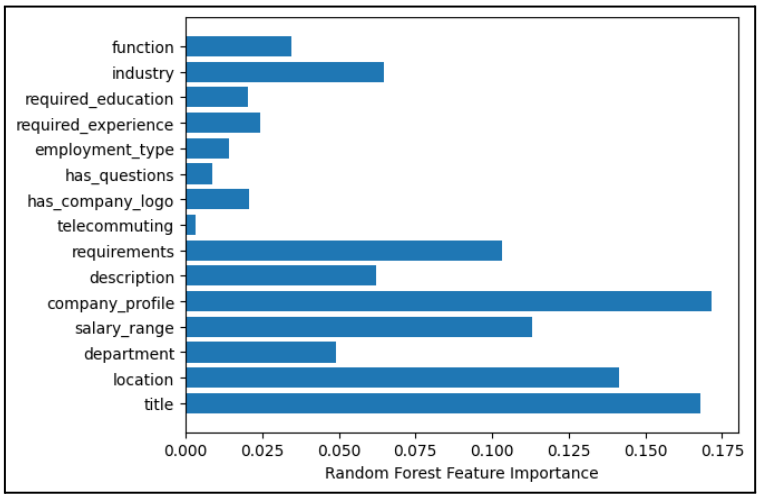
Para encontrar la mejor K normalmente se usa basándose en la k con la que se obtiene más accuracy. En nuestro caso, como lo que más importa es el f1-score, usaremos este valor.
Para ello hemos optado por hacer una gráfica donde se representa el valor de k en función del valor de f1. El valor de k irá cambiando a través de un bucle en el que irá de 1 a 30.



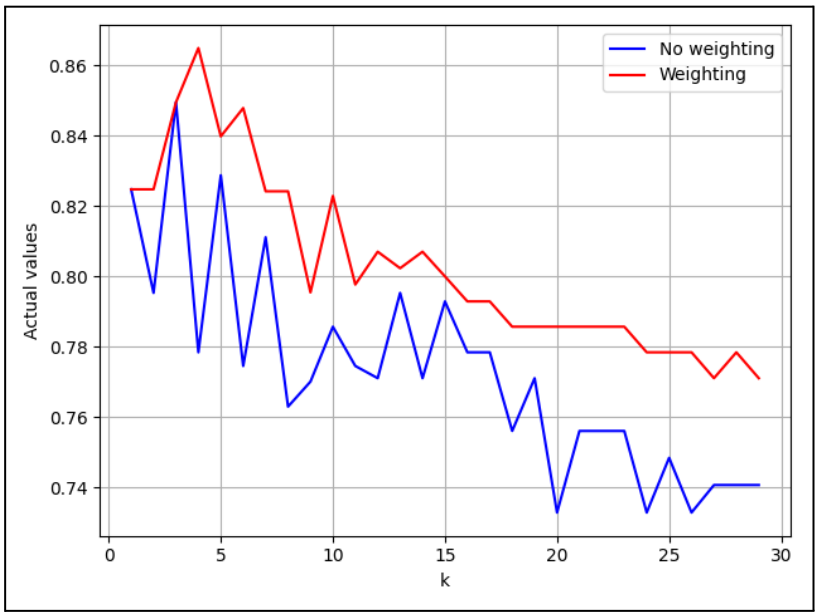
Podemos observar que el mejor valor para K en K-NN es el 5, y para Weighted K-NN es el 6, y obtenemos un valor de 0.8524 para f1.

Eliminar features irrelevantes

Como sabemos K-NN es sensible a los features más irrelevantes, por lo que vamos a probar a eliminar aquellas columnas que lo sean.
Para encontrar cuales de ellas lo son, utilizaremos la función feature_importances de Random Forest, obteniendo los siguientes resultados.



Podemos observar que las features más irrelevantes son telecommuting, has_questions y has_company_logo. Vamos a eliminarlas y volver a comprobar el mejor valor de K.



Viendo el gráfico observamos que hemos mejorado el valor de f1 a 0.8650, y que también han cambiado los valores de K. Ahora, el mejor valor para K en K-NN es el 3, y para Weighted K-NN es el 4.

Decision Trees

Parámetros usados

Para encontrar los mejores parámetros para el Decision Tree utilizaremos RandomizedSearchCV de sklearn. Esto nos permitirá obtener los mejores parámetros a través de prueba y error. Es decir, RandomizedSearchCV probará todas las combinaciones de parámetros posibles a partir de una plantilla con todos los parámetros a probar que nosotros le daremos, y nos devolverá la información del DT obtenido con mejor f1.
Dicho esto introducimos los parámetros a probar, y creamos el objeto de RandomizedSearch con k-fold (k=10) y haciendo que sus resultados se basen en el f1.

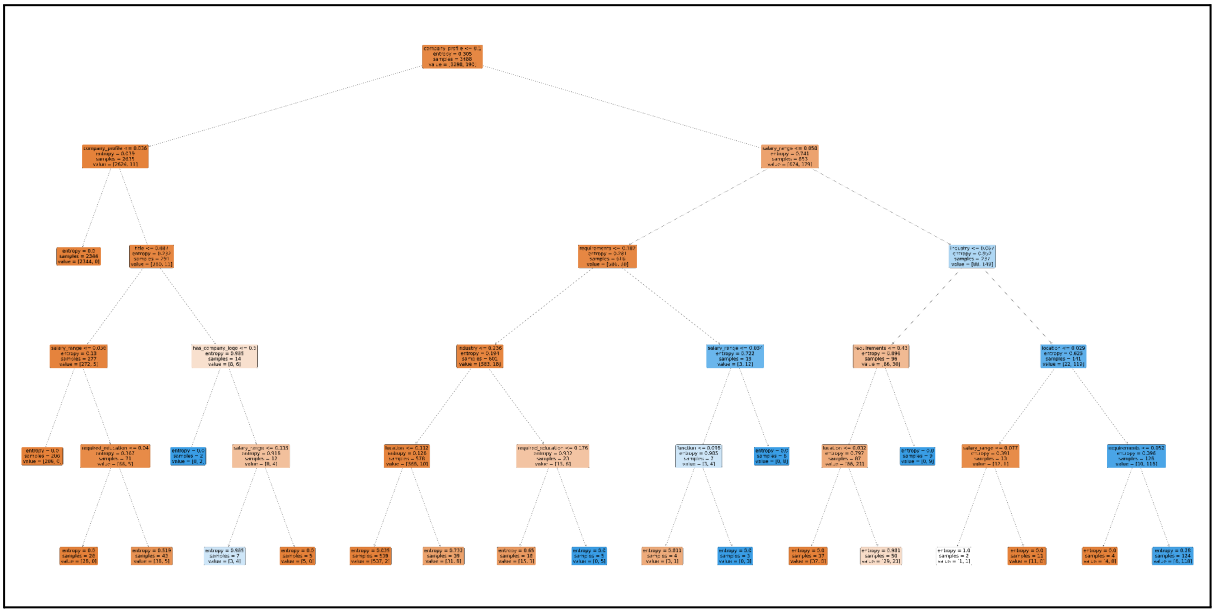
```
# Define the parameters that you want to optimize
parameters = {'criterion': ['gini', 'entropy', 'log_loss'],
              'splitter': ['best', 'random'],
              'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'min_samples_split': [1,2,3,4,5],
              'max_features': [4,5,6,7]}

# Create a randomized search object using the decision tree classifier and the parameters dictionary
rs = RandomizedSearchCV(dt, parameters, cv=10, scoring='f1')
```

Después de esto ya podemos entrenar el modelo y comprobar los resultados que obtenemos, que son los siguientes:

```
print(rs.best_score_)
print(rs.best_params_)
BestPar = rs.best_params_
```

```
0.8621638524077548
{'splitter': 'best', 'min_samples_split': 3, 'max_features': 7, 'max_depth': 10, 'criterion': 'gini'}
```

Finalmente podemos ver el DT que se obtiene añadiendo estos parámetros



Interpretación del Decision Tree con ejemplo

Vamos a realizar una comprobación del funcionamiento del DT utilizando una muestra del set de validación en la que de como resultado 1 (fraudulent):

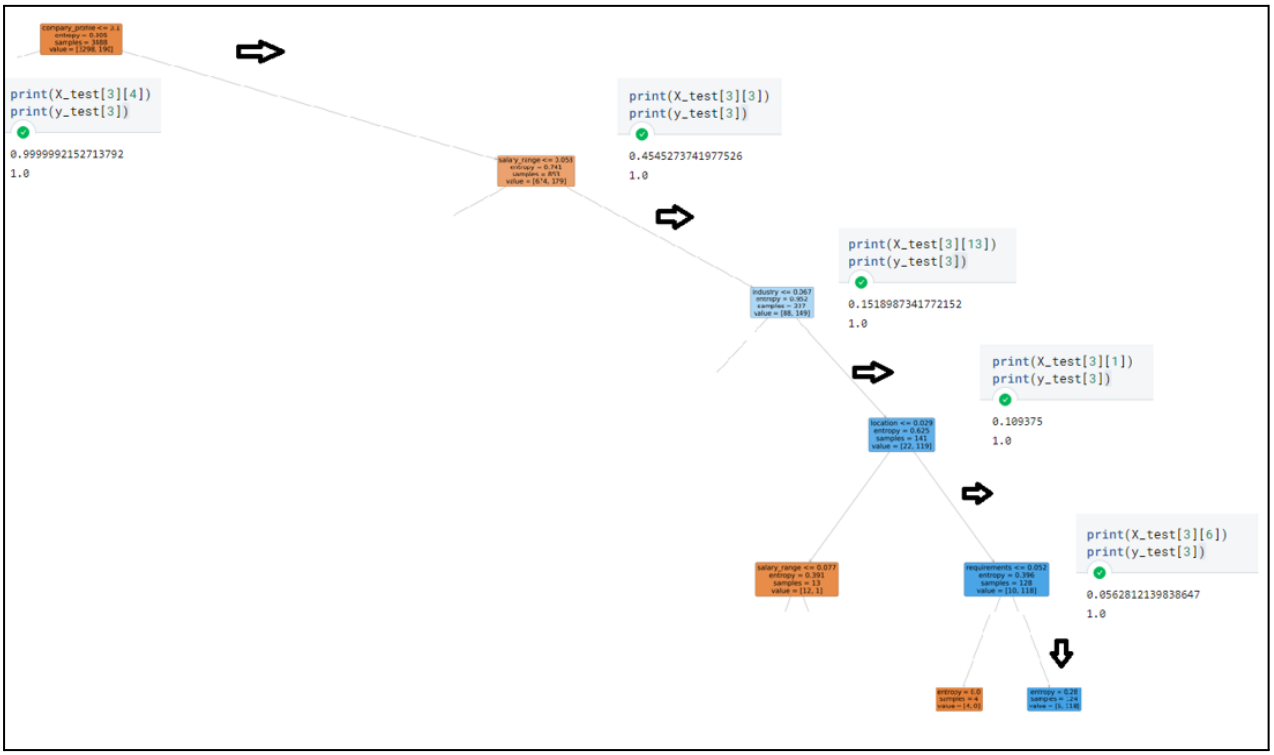
```
(X_train, X_test,  y_train, y_test) = cv.train_test_split(X, y, test_size=.33, random_state=1)
```

utilizando X_test escogeremos una fila [i] que dé como resultado en y_test[i] un 1(fraudulent), encontramos que la fila 3 cumple esa condición y para esta comprobaremos los valores de los atributos que se muestran en el árbol de decisión. Con tal de movernos en el orden necesario por cada nodo, accederemos indexando las columnas de la siguiente manera:

```
for colName in data:
    print(colName, data.columns.get_loc(colName))
```

```
title 0
location 1
department 2
salary_range 3
company_profile 4
description 5
requirements 6
telecommuting 7
has_company_logo 8
has_questions 9
employment_type 10
required_experience 11
required_education 12
industry 13
function 14
fraudulent 15
```

Con y_test[3] seguimos la ruta indicada por el decision tree obtenido teniendo en cuenta el valor de cada feature para cada nodo, y podemos ver que acaba en una hoja correcta(azul=”fraudulent”, naranja=”no fraudulent”):

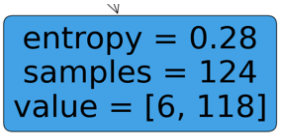


Fiabilidad del Decision Tree con las hojas

Para comprobar la fiabilidad del DT utilizaremos el valor de cada hoja del árbol. Este determina la predicción del árbol. De esta manera podemos comparar la predicción del modelo con los casos reales. Para ello utilizaremos tres hojas aleatorias del DT.

```
count = 0
for i in range(0, len(X_test)):
    if(X_test[i][4] > 0.1 and X_test[i][3] > 0.058 and X_test[i][13] > 0.067 and X_test[i][1] > 0.029 and
       X_test[i][6] > 0.052):
        count += 1
print(count)
```

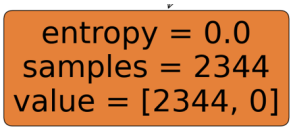
```
62
```



Como podemos observar hay una gran diferencia a la hora de calcular los casos fraudulentos. Esto es algo completamente normal ya que al ser un dataset desbalanceado es mucho más fácil que prediga los casos más abundantes, en este caso los no fraudulentos.

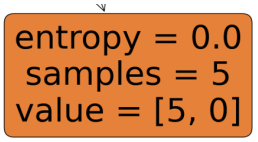
```
count2 = 0
for i in range(0, len(X_test)):
    if(X_test[i][4] <= 0.1 and X_test[i][4] <= 0.036):
        count2 += 1
print(count2)
```

```
1109
```




```
count3 = 0
for i in range(0, len(X_test)):
    if(X_test[i][4] <= 0.1 and X_test[i][4] > 0.036 and X_test[i][0] > 0.087
       and X_test[i][8] > 0.5 and X_test[i][3] > 0.135):
        count3 += 1
print(count3)
```

```
5
```

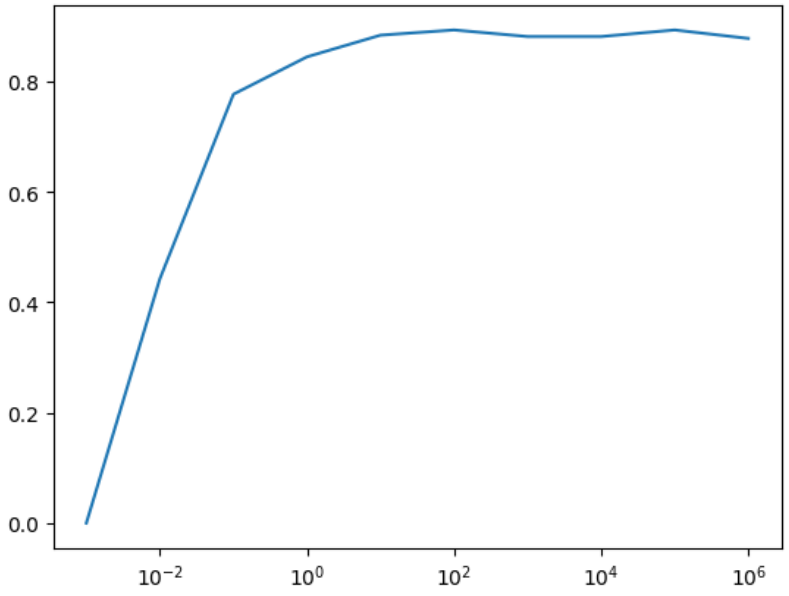


En cambio si hacemos que se centre en los casos en que da como resultado los no fraudulentos, se observa que predice con mucho más acierto.

Support Vector Machines

Parámetros usados

Primero de todo hemos buscado las mejores C para el modelo que usa kernel lineal y el modelo que usa el kernel polinomial, obteniendo los siguientes resultados.

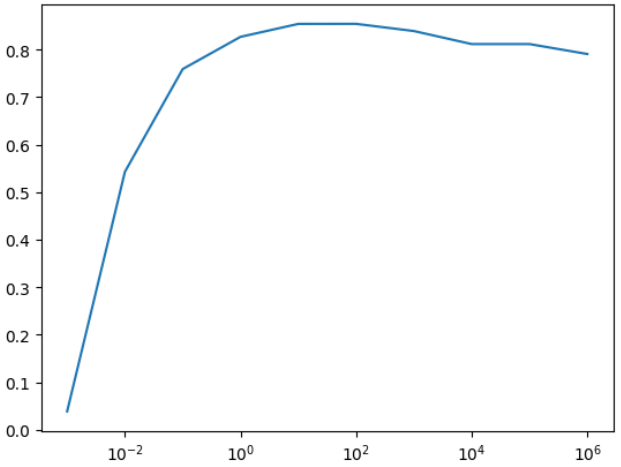


Kernel Lineal

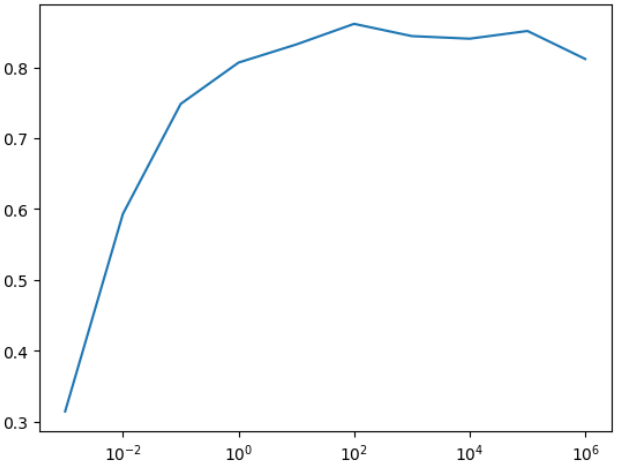
Podemos observar que las mejores C son 100 y 10000. Como tienen el mismo valor de f1, nos quedaremos con la más simple que es 100.

Kernel Polinomial

Degree 2



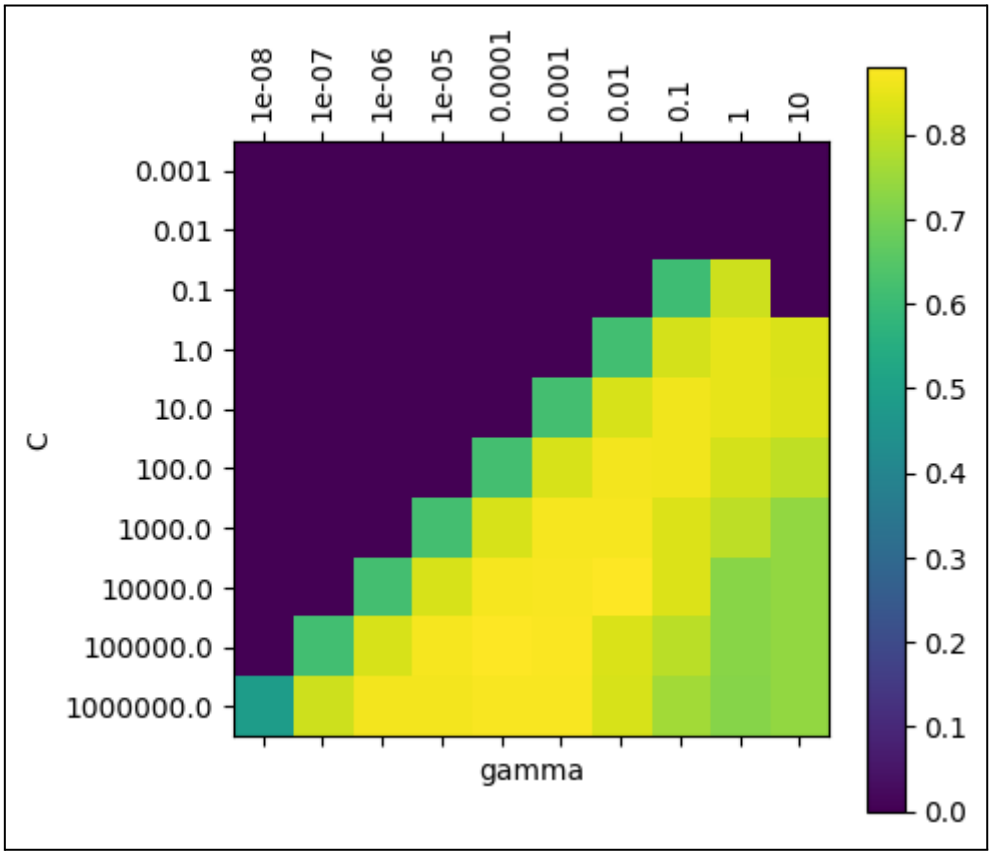
Degree 3



Podemos observar que las mejores C son 100 para ambos casos, degree 2 y degree 3.

RBF

Para RBF hemos utilizado GridSearchCV para obtener tanto la mejor C como la mejor gamma.



Prácticamente no se observa pero los resultados nos dicen que:

```
Mejor combinación parmetros encontrados: {'C': 10000.0, 'gamma': 0.01}
```

Una vez hemos obtenido los mejores parámetros para cada kernel y utilizando RandomSearchCV obtendremos cual es el kernel que nos da mayor valor de f1.

Parámetros utilizados:

```
# Best params for each kernel
parameters.append({'kernel': ['poly'],
                   'C': [100],
                   'degree': [2, 3]})
parameters.append({'kernel': ['linear'],
                   'C': [100]})
parameters.append({'kernel': ['rbf'],
                   'C': [10000],
                   'gamma': [0.01]})
```

Resultado

```
f1: 0.8936170212765957 con parámetros {'kernel': 'linear', 'C': 100}
```

Numero de supports y interpretación

Supports para linear kernel

```
Number of supports:  115 ( 99 of them have slacks)
Prop. of supports:  0.03297018348623853
```

Supports para poly kernel

```
Number of supports:  138 ( 52 of them have slacks)
Prop. of supports:  0.03956422018348624
```

Supports para rbf kernel

```
Number of supports:  163 ( 44 of them have slacks)
Prop. of supports:  0.046731651376146786
```

En todos los casos de SVM obtenemos un número muy bajo de supports, y en el que menos obtenemos es aquel en el que obtenemos mejor performance. Esto es algo común en SVMs. Además, sabiendo que nuestro modelo realiza una clasificación binaria, el linear kernel es una buena opción de por sí siempre y cuando los datos sean linealmente separables, es decir, en dos clases.

Finalmente podemos observar cuales son los main supports de la versión con kernel lineal, haciendo un print de ellos:

```
array([   0,   11,   31,   41,  163,  186,  253,  277,  278,  303,  387,
        403,  489,  552,  656,  754,  793,  824,  828,  942, 1044, 1055,
       1119, 1194, 1199, 1256, 1268, 1277, 1337, 1344, 1438, 1456, 1566,
       1582, 1656, 1682, 1823, 1826, 1871, 2048, 2132, 2261, 2268, 2349,
       2414, 2427, 2532, 2579, 2625, 2739, 2742, 2814, 2846, 2908, 3038,
       3072, 3143, 3266, 3369, 3464,    3,  168,  256,  335,  454,  470,
        548,  604,  610,  678,  703,  819,  907,  988, 1005, 1021, 1050,
       1169, 1393, 1433, 1499, 1583, 1662, 1664, 1751, 1792, 1820, 1881,
       1891, 1921, 2000, 2097, 2212, 2509, 2563, 2577, 2616, 2646, 2688,
       2867, 2875, 2900, 2920, 2941, 3116, 3144, 3168, 3176, 3197, 3212,
       3374, 3426, 3445, 3484, 3487], dtype=int32)
```

Métodos meta-learning

Performance Majority Voting

Para el majority voting usaremos los mejores modelos que hemos obtenido en cada algoritmo de cada método.

```
modelos = [('nb', GaussianNB()),
           ('knn', nb.KNeighborsClassifier(n_neighbors=5)),
           ('dt', tree.DecisionTreeClassifier(criterion=BestPar['criterion'], max_depth=BestPar['max_depth'],
                                              max_features=BestPar['max_features'],
                                              min_samples_split=BestPar['min_samples_split'], splitter=BestPar['splitter'])),
            ('svc', SVC(kernel='linear', C=100))]
```

Una vez definidos, podemos ejecutar el majority voting, donde obtenemos el siguiente f1:

```
f1: 0.866 [Majority Voting]
```

También vamos a ejecutar la variante de majority voting weighted voting, donde ahora cada modelo tendrá un pesaje a la hora de votar diferente. En nuestro caso, todos los modelos votarán por igual excepto Decision Tree ya que es el menos fiable. Obtenemos el siguiente resultado, en el que observamos una mejora.

```
f1: 0.875 [Weighted Voting]
```

Bagging

En este caso utilizaremos BaggingClassifier() con un estimador base DecisionTreeClassifier() y iremos iterando el numero de estimadores(n_estimators) entre:

```
[1,2,5,10,20,50,100,200]
```

para cada caso obtenemos los siguientes resultados tomando como medida de scoring el valor f1:

```
f1: 0.824 [1]
f1: 0.829 [2]
f1: 0.881 [5]
f1: 0.879 [10]
f1: 0.877 [20]
f1: 0.882 [50]
f1: 0.888 [100]
f1: 0.884 [200]
```

Después de obtener estos resultados, optamos por hacer uso del parámetro max_features, para el cual vamos a tener que buscar con prueba y error el mejor valor, es decir, el que muestre unos mejores resultados respecto al Bagging anterior, para ello utilizamos:

```
max_features=0.75
```

En estos valores podemos ver una clara mejoría a partir de 50 estimadores:

```
f1: 0.792 [1]
f1: 0.772 [2]
f1: 0.848 [5]
f1: 0.867 [10]
f1: 0.874 [20]
f1: 0.892 [50]
f1: 0.883 [100]
f1: 0.891 [200]
```

RandomForest

En nuestro proyecto ya usamos RandomForest, para obtener las features más importantes para el algoritmo KNN. Ahora volveremos a usarlo para ver los diferentes f1-score que obtenemos en función del número de estimadores. Obtenemos los siguientes resultados:

```
f1-score: 0.765 [1]
f1-score: 0.769 [2]
f1-score: 0.851 [5]
f1-score: 0.874 [10]
f1-score: 0.879 [20]
f1-score: 0.888 [50]
f1-score: 0.878 [100]
f1-score: 0.883 [200]
```

Adaboost

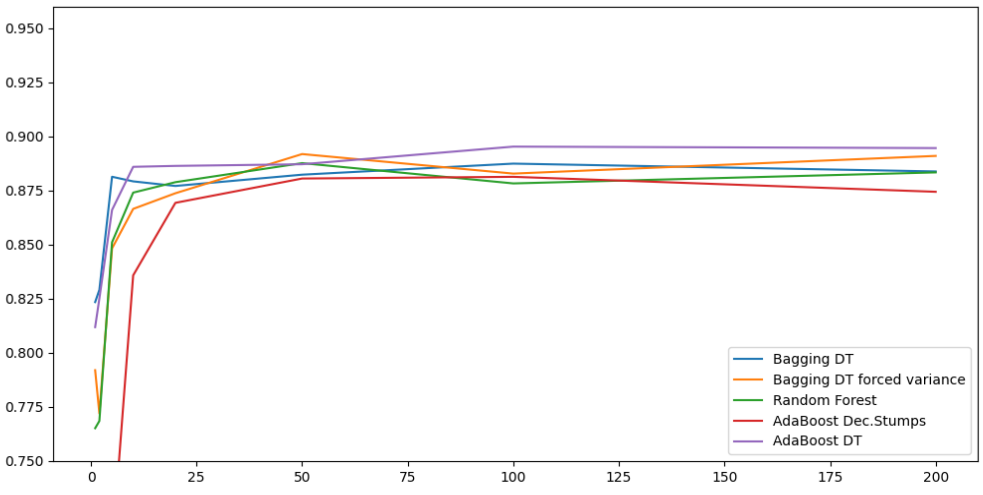
Aquí utilizamos AdaBoostClassifier() con los mismos parámetros que con Bagging. Vemos que con pocos estimadores obtenemos un valor de f1 score menor respecto a los métodos anteriores, pero a medida que aumenta la cantidad de estimadores, se obtienen unos valores bastante similares a los anteriores.

```
f1: 0.412 [1]
f1: 0.423 [2]
f1: 0.708 [5]
f1: 0.836 [10]
f1: 0.869 [20]
f1: 0.881 [50]
f1: 0.881 [100]
f1: 0.874 [200]
```

Con max_depth=5

```
f1: 0.812 [1]
f1: 0.825 [2]
f1: 0.866 [5]
f1: 0.886 [10]
f1: 0.886 [20]
f1: 0.887 [50]
f1: 0.895 [100]
f1: 0.895 [200]
```

Podemos comparar el resultado de todos los métodos en una misma gráfica,donde comprobamos que la mejor performance la realiza AdaBoost con Decision Tree.



Comparaciones y conclusiones

Tabla comparativa

Método	F1-Score
Naive Bayes	0.88
KNN	0.85
Decision Trees	0.86
SVM	0.89
Majority Voting	0.86
Bagging	0.89
Random Forest	0.88
AdaBoost	0.89

McNemar Test

Para comparar los diferentes métodos utilizados, usaremos el test de McNemar:

```
Naïve Bayes con KNN:
 pvalue      1.802802378388091e-08
statistic   9.0
Naïve Bayes con DecisionTree:
 pvalue      1.802802378388091e-08
statistic   9.0
Naïve Bayes con SVM:
 pvalue      1.802802378388091e-08
statistic   9.0
Naïve Bayes con MajorityVoting:
 pvalue      1.802802378388091e-08
statistic   9.0
Naïve Bayes con Bagging:
 pvalue      1.802802378388091e-08
statistic   9.0
Naïve Bayes con RandomForest:
 pvalue      1.802802378388091e-08
statistic   9.0
Naïve Bayes con AdaBoost:
 pvalue      1.802802378388091e-08
statistic   9.0
```

```
KNN con DecisionTree:
 pvalue      2.4904030624384174e-05
statistic   9.0
KNN con SVM:
 pvalue      2.4904030624384174e-05
statistic   9.0
KNN con MajorityVoting:
 pvalue      2.4904030624384174e-05
statistic   9.0
KNN con Bagging:
 pvalue      2.4904030624384174e-05
statistic   9.0
KNN con RandomForest:
 pvalue      2.4904030624384174e-05
statistic   9.0
KNN con AdaBoost:
 pvalue      2.4904030624384174e-05
statistic   9.0
```

```
DecisionTree con SVM:
 pvalue      0.5966417603730826
statistic   26.0
DecisionTree con MajorityVoting:
 pvalue      0.5966417603730826
statistic   26.0
DecisionTree con Bagging:
 pvalue      0.5966417603730826
statistic   26.0
DecisionTree con RandomForest:
 pvalue      0.5966417603730826
statistic   26.0
DecisionTree con AdaBoost:
 pvalue      0.5966417603730826
statistic   26.0
```

```
SVM con MajorityVoting:
 pvalue      0.13604594767093658
statistic   10.0
SVM con Bagging:
 pvalue      0.13604594767093658
statistic   10.0
SVM con RandomForest:
 pvalue      0.13604594767093658
statistic   10.0
SVM con AdaBoost:
 pvalue      0.13604594767093658
statistic   10.0
```

```
MajorityVoting con Bagging:
 pvalue      1.5236437320709229e-05
statistic   3.0
MajorityVoting con RandomForest:
 pvalue      1.5236437320709229e-05
statistic   3.0
MajorityVoting con AdaBoost:
 pvalue      1.5236437320709229e-05
statistic   3.0
```

```
Bagging con RandomForest:
 pvalue      0.063568115234375
statistic   5.0
Bagging con AdaBoost:
 pvalue      0.063568115234375
statistic   5.0

RandomForest con AdaBoost:
 pvalue      0.063568115234375
statistic   5.0
```

En la mayoría de los casos, el p-valor es menor al valor significativo que hemos establecido, por lo que podemos afirmar que existe una diferencia significativa entre todos los modelos. Excepto entre DT y SVM, SVM y Majority Voting y todos los meta-metodos entre sí. En estos el p-value es mayor, y por lo tanto no podemos afirmar diferencia.

Respecto la statistica de McNemar, como no es igual a cero en ningún caso, podemos concluir que existe una diferencia significativa entre todos los modelos en términos de TP y FN.

Similitudes con cross-validation

	f1 con k-fold (k=10)	**f1 con cross-validation**
Naive Bayes	0.88	0.86
KNN	0.85	0.92
Decision Tree	0.86	0.89
SVM	0.89	0.93

Podemos observar que los resultados obtenidos por cada modelo son bastante similares, es decir, no hay grandes diferencias. Aun así, en general obtenemos mayor f1 con cross-validation y esto es probablemente debido a que el dataset es muy desequilibrado. Es decir, en k-fold algunos pliegues pueden contener una proporción desproporcionada de clases, lo que puede afectar el rendimiento general de los modelos. En cambio, un cross-validation puede ser más adecuado en este caso.

Mejor método

Visto los resultados y por la comodidad al usarlo, nos quedamos con el **SVM de kernel lineal.** Posiblemente sea con el que obtenemos mejor f1 ya que al transformar el espacio de características de manera que sea linealmente separable, aumenta la capacidad de la SVM para clasificar correctamente los datos. También es un gran modelo para hacer predicciones precisas en datos nuevos y no vistos previamente.

Bibliografía

K-fold:
https://www.researchgate.net/post/How_can_I_create_a_single_confusion_matrix_after_K_fold_cross_validation
https://towardsdatascience.com/how-to-plot-a-confusion-matrix-from-a-k-fold-cross-validation-b607317e9874

Mean-encoding:
https://www.youtube.com/watch?v=nd7vc4MZQz4

Python functions:
https://www.softwaretestinghelp.com/python-range-function/

Naive Bayes:
https://towardsdatascience.com/naive-bayes-classifier-how-to-successfully-use-it-in-python-ecf76a995069

KNN:
https://www.yourdatateacher.com/2021/10/11/feature-selection-with-random-forest/
https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html

DecisionTreeClassifier:
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

Plot:
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.scatter.html