TP 2- ORGANIZACIÓN DE DATOS - 7506

Introducción:

En el presente informe se describe el trabajo que realicé para poder entrenar un algoritmo de Machine Learning que pueda precedir si un postulante va a postularse o no, a un determinado aviso.

Temas:

Link del repositorio:

https://github.com/NicoGallegos/fiuba-7506-2018

Archivos

Tiene el desarrollo con las primeras pruebas, y elección de modelo de Machine Learning.

Seleccion de Modelo ML.ipynb

Código que tiene mucha de las pruebas realizadas.

Todas las Pruebas TP 2.ipynb

Contiene las generación de la mejor prueba, sin código basura/intermedio de pruebas.

TP 2- Limpiado.ipynb

Tiene pruebas de generadores de no postulados

AvisosNoPostuladoGenerator.ipynb

Selección de Algoritmo de Machine Learning

Para elegir el algoritmo con el cual trabajar, me quedo con un set básico de datos, para luego poder hacer el entrenamiento.

Utilicé la transformación de postulantes género, y de los detalles de los avisos, para luego poder hacer pruebas con diferentes algoritmos de Machine Learning.

	edad	idaviso	idpostulante	nivel_Gerencia / Alta Gerencia / Dirección	nivel_Jefe / Supervisor / Responsable	nivel_Junior	nivel_Otro	nivel_Senior / Semi-Senior	sepostulo	sexo_FEM	...	tipo_Pasantia	tipo_Por Contrato	tipo_P Hora
0	26.0	1112410160	qekrW81	0	1	0	0	0	0	1	...	0	0	
1	29.0	1112410160	pzjDkN3	0	1	0	0	0	0	1	...	0	0	
2	29.0	1112410160	bOzqo29	0	1	0	0	0	0	1	...	0	0	
3	47.0	1112410160	1avP3N	0	1	0	0	0	0	0	...	0	0	
4	30.0	1112410160	IDLV6Oz	0	1	0	0	0	0	1	...	0	0	

En una primera prueba, entreno con pocos datos, y luego genero cross validation para cada uno de los métodos probados:

```
models = []
models.append(('RFR10', RandomForestRegressor(n_estimators=10, max_features='sqrt')))
models.append(('RFR20', RandomForestRegressor(n_estimators=20, max_features='sqrt')))
models.append(('RFR50',  RandomForestRegressor(n_estimators=50, max_features='sqrt')))
models.append(('RFR60',  RandomForestRegressor(n_estimators=60, max_features='sqrt')))
models.append(('RFR100',  RandomForestRegressor(n_estimators=100, max_features='sqrt')))

models.append(('RFC10', RandomForestRegressor(n_estimators=10, max_features='sqrt')))
models.append(('RFC20', RandomForestRegressor(n_estimators=20, max_features='sqrt')))
models.append(('RFC50',  RandomForestRegressor(n_estimators=50, max_features='sqrt')))
models.append(('RFC60',  RandomForestRegressor(n_estimators=60, max_features='sqrt')))
models.append(('RFC100',  RandomForestClassifier(n_estimators=100, max_features='sqrt')))

models.append(('KNN 2', KNeighborsClassifier(n_neighbors=2)))
models.append(('KNN 3', KNeighborsClassifier(n_neighbors=3)))
models.append(('LR', linear_model.LinearRegression()))
models.append(('RIDGE.05', linear_model.Ridge (alpha = .5)))
models.append(('LASO.01', linear_model.Ridge (alpha = .1)))
models.append(('LASOLARS.05',linear_model.LassoLars (alpha = 0.5)))
models.append(('BAYESIAN',linear_model.BayesianRidge()))
models.append(('SVR',svm.LinearSVR()))
models.append(('DES-TREE',tree.DecisionTreeRegressor()))
models.append(('NB', GaussianNB()))
models.append(('LDA', LinearDiscriminantAnalysis()))
```

Se obtuvo los siguientes resultados:

```
RFR10: 0.081268 (0.005587)
RFR20: 0.083233 (0.004481)
RFR50: 0.089590 (0.007050)
RFR60: 0.091051 (0.004208)
RFR100: 0.090027 (0.005576)
RFC10: 0.082198 (0.007145)
RFC20: 0.090133 (0.005974)
RFC50: 0.091368 (0.004178)
RFC60: 0.089136 (0.004179)
RFC100: 0.609500 (0.002733)
KNN 2: 0.596742 (0.002619)
KNN 3: 0.589947 (0.016717)
LR: 0.144936 (0.016587)
RIDGE.05: 0.145063 (0.016453)
LASO.01: 0.144984 (0.016571)
LASOLARS.05: -0.002537 (0.001803)
BAYESIAN: 0.144456 (0.015281)
SVR: -0.610361 (0.377650)
DES-TREE: 0.061187 (0.006413)
NB: 0.632290 (0.004506)
LDA: 0.630385 (0.011350)
```

Se puede apreciar, que Random Forest Classifier Gaussian, y Linear Discriminant Analysis fueron los mejores con esta pequeña prueba.

Para estos, vuelvo a ejecutar más cross validation test.

```
---RFC----
[ 0.26030369  0.55639913  0.5320304   0.52334419  0.22692725]
---GAUSS----
[ 0.1670282   0.66160521  0.61237785  0.62540717  0.61780673]
---LDA----
[ 0.17353579  0.59327549  0.56026059  0.56677524  0.2019544 ]
```

Una vez que ya tengo unos candidatos. Lo que hice fue aumentar la cantidad de datos para entrenamiento, y volver a procesar. Volvemos a procesar todos los algoritmos, aunque ya tenemos algunos cantidatos, obteniendo los siguientes restulados:

```
RFR10: 0.353495 (0.003226)
RFR20: 0.353552 (0.003170)
RFR50: 0.353517 (0.003224)
RFR60: 0.353573 (0.003202)
RFR100: 0.353556 (0.003218)
RFC10: 0.353473 (0.003255)
RFC20: 0.353491 (0.003256)
RFC50: 0.353580 (0.003195)
RFC60: 0.353543 (0.003205)
RFC100: 0.757973 (0.001559)
KNN 2: 0.666637 (0.010344)
KNN 3: 0.698880 (0.004891)
LR: 0.274874 (0.002613)
RIDGE.05: 0.274874 (0.002613)
LASO.01: 0.274874 (0.002613)
LASOLARS.05: -0.000004 (0.000003)
BAYESIAN: 0.274873 (0.002613)
SVR: -0.257657 (0.633758)
DES-TREE: 0.353423 (0.003229)
NB: 0.577928 (0.000302)
```

Una vez más, vemos que tenemos a Random Forest Classifier como el mejor de los algoritmos para la prueba. Volvemos a generar más validaciones entre los algoritmos con mejor resultado:

```
---RFC----
[ 0.73955803  0.77456053  0.79105053  0.78274581  0.69454762]
```

```
---LDA----
[ 0.71511166  0.74360801  0.76693064  0.76833228  0.67665335]
```

En base a las pruebas, veo que el mejor algoritmo para trabajar es Random Forest Classifier.

Transformaciones Aplicadas

Postulantes Género

En este caso, lo que hizo fue generar una columna por cada valor en el campo sexo, quedandonos: sexo_masc, sexo_fem, sexo_no_declara

Con la fecha de nacimiento, lo que hice fue calcular la edad en base a la fecha actual.

	idpostulante	fechanacimiento	sexo_FEM	sexo_MASC	sexo_NO_DECLARA	edad
0	6MM	1985-01-01	0	1	0	33.0
4	ebE	1952-07-07	0	1	0	65.0
7	NAjM	1962-06-09	1	0	0	56.0
10	ZjlZ	1970-01-25	0	1	0	48.0
11	8wPl	1973-03-13	0	1	0	45.0

Otra segunda prueba que hice con las edades, fue pasarlas a rangos de edad, creando estas categorias:

‘Menor a 24’ ; “Menor a 31” ; “Menor a 36” ; “Menor 40” ; “Menor 50” “Mayor 60”; Resto

```
trainingConCantNombre['menor24'] = np.where(trainingConCantNombre['edad'] < 25, 1, 0);
trainingConCantNombre['menor31'] = np.where(trainingConCantNombre['edad'] < 31, 1, 0);
trainingConCantNombre['menor36'] = np.where(trainingConCantNombre['edad'] < 36, 1, 0);
trainingConCantNombre['menor40'] = np.where(trainingConCantNombre['edad'] < 40, 1, 0);
trainingConCantNombre['menor50'] = np.where(trainingConCantNombre['edad'] < 51, 1, 0);
trainingConCantNombre['mayor50'] = np.where(trainingConCantNombre['edad'] > 51, 1, 0);
trainingConCantNombre['resto']   = np.where(trainingConCantNombre['edad'].isnull(), 1, 0);
```

Este caso fue descartado, puesto que en las pruebas, me dio mejor resultado la edad.

Postulantes Educación

Aquí tenemos dos columnas para transformar, una que es nombre y la otra estado. La primera aproximación fue generar ambas columnas con One Hot Enconding, obteniendo así columnas por cada uno de los estados y por el nombre de lo que se cursaba. Finalmente, lo que hice fue agruparla por postulante, y sumar los valores de cada columna.

El problema de la transformación anterior, es que pierdo la relación entre estado y la educación, puesto que puedo tener 4 abandonados, y 1 finalizado, pero no sé a qué se corresponde cada uno de esos abandonados.

Lo que hice entonces, fue unir las dos columnas en una única, y generar las columnas. Esto me permite mantener la relación entre su estado actual, y el estudio.

Generación de no postulados

Para poder generar los avisos para los cuales los postulantes no se postularon, utilicé diferentes métodos. En mis primeras aproximaciones, lo que intenté hacer para generarlos es basarme en los datos de postulaciones y vistas, entonces, si no teníamos una postulación en el dataset, significa que no se postuló.

Para ello, utilicé varios métodos que partían con la misma premisa, los métodos variaban simplemente en la forma de mergear los datos, puesto que al tener tantos valores, no podía realizar un merge del dataset completo.

Entonces, lo que hice aquí fue generar samples pequeños, en muchas iteraciones, que me permitían hacer un right join, para luego quedarme con los nulos en postulación.

Este método tenía como desventaja, la generación de muchas tuplas repetidas, puesto que los sets eran reducidos.

```
import datetime as dt

def generoNOAvisos(cantParaArmar):
    sumaNoPostulados = pd.DataFrame();
    for i in range(int(cantParaArmar)):
        print("---------------------- Iteracion de sample " + str(i) +" de "+ str(cantParaArmar) +"---------");
        vistasSample = vistasForMerge.sample(frac=0.1);
        postulacionesSample = postuChico.sample(frac=0.1);

        avisosNoVistos = pd.merge(postulacionesSample,vistasSample,left_on='idaviso',right_on='idAviso',how='right', indic
        avisosNoVistosPosta = avisosNoVistos[avisosNoVistos['_merge'] == 'right_only']
#avisosNoVistosFinal = avisosNoVistosPosta.drop(['idpostulante_x','datePostulacion','cantPostulaciones','_merge'],axis=1)
        avisosNoVistosFinal = avisosNoVistosPosta.drop(['idpostulante_x','_merge'],axis=1)
        avisosNoVistosFinal['idaviso'] = avisosNoVistosFinal['idAviso'];
        avisosNoVistosFinal['idpostulante'] = avisosNoVistosFinal['idpostulante_y'];
        avisosNoVistosFinal.drop(['idpostulante_y','idAviso'],axis=1,inplace=True);
        sumaNoPostulados = pd.concat([sumaNoPostulados,avisosNoVistosFinal])
    return sumaNoPostulados;


def generoArchivosNOPostulado(cantArchivos,cantInicial,cantIteracionesPorArhivo):
    print("--- HORA DE INICIO PROCESO TOTAL "+ str(dt.datetime.now()) + "-----");
    for i in range(int(cantInicial),int(cantArchivos)):
        print("---------- Generando archivos " + str(i) +" de "+ str(cantArchivos) +"-------");
        print("---------- HORA DE INICIO "+ str(dt.datetime.now()) + "-------");
        sumaTotalNoAvisos = generoNOAvisos(int(cantIteracionesPorArhivo));
        sumaTotalNoAvisosSinDuplicados = sumaTotalNoAvisos.drop_duplicates().reset_index()
        sumaTotalNoAvisosSinDuplicados['sepostulo'] = 0
        sumaTotalNoAvisosSinDuplicados = sumaTotalNoAvisosSinDuplicados[['idpostulante','idaviso','sepostulo']]
        sumaTotalNoAvisosSinDuplicados.to_csv('nopostulados-hoy/nopostulados-'+str(i)+'.csv');
        print("---------- HORA DE FIN "+ str(dt.datetime.now()) + "--------------------------");
    print("--- HORA DE FIN PROCESO TOTAL "+ str(dt.datetime.now()) + "-----");

```

Una mejora a este método, fue trabajar con rangos de postulantes, entonces reducía mi set de para el merge, y además me garantizaba generar mucho mayor volumen de datos.

```
noPostuladosTotal = pd.DataFrame();
for i in range(0,238482,5000):
    print("----" + str(i) + "-----")
    pfRows = pf.iloc[i:i+5000]
    avisosConPostulacion = pd.merge(postulaciones,pfRows,on='idpostulante',how='inner')
    todos = pd.merge(vistasChiquito,avisosConPostulacion,on='idaviso',how='left')
    noPostuTodos = todos[todos['idpostulante_y'].isnull()];
    noPostuFiltrados = noPostuTodos.sample(n=10);
    noPostuladosTotal = pd.concat([noPostuFiltrados,noPostuladosTotal])


noPostuladosTotal.to_csv('nopostulados/noPostuladosNG.csv')
```

Como mis pruebas siguientes siguieron empeorando con este método (y , a medida que aumentaba mis no postulados, mis scores daban peor), cambié este método por otra forma:

En este método, lo que hice fue hacer un outer join entre postulantes y vistas, y nuevamente quedarme con aquellos que no tenían parte de postulantes. Éste método es mucho más simple y rápido que los anteriores, además de poder obtener todos los datos del dataset. Sin embargo, a pesar de tener todos los datos, mis resultados siguieron empeorando.

Luego de varias pruebas fallidas, con scores muy bajos, llegué a la conclusión de que mi forma de generación de no postulaciones estaba planteada con una premisa errónea. Hasta ese momento, lo que consideraba era sólo los avisos que esa persona vio y no se postuló, lo cual está mal, porque si esa persona vio ese aviso, hay posibilidades concretas de que se postule, puesto que al verlo, demuestra que el postulante tiene un cierto interés, hay algo que le llamó la atención. Por ende, los no postulados que debo generar, son todos aquellos en dónde la persona ni siquiera lo vio. Esto lo hice generando muestras aletorias entre avisos detalles y postulantes. Luego puedo chequear si se postuló o no.

Avisos Detalle

Para el caso de la denominación de empresa, lo que hice fue agruparlas por cantidad, para así poder darle un mayor valor de importancia, a las empresas que más apariciones tienen.

:

	denominacion_empresa	cantidadEmpresa
2025	Manpower	963909
2490	RANDSTAD	893729
161	Adecco -Región Office	697775
1444	Grupo Gestión	689770
1236	Farmacity	588353
259	Assistem	575936
303	BBVA Francés	549225
301	BAYTON	439058
2462	Pullmen Servicios Empresarios S.A.	396778
157	Adecco - Región NORTE & OESTE GBA	328406

Para el caso del tipo de trabajo, nombre_zona, y nivel_laboral, fue convertir cada uno de los valores a columnas, con 0 y 1, en el caso que corresponda.

	idaviso	zona_Capital Federal	zona_GBA Oeste	zona_Gran Buenos Aires	tipo_Fines de Semana	tipo_Full-time	tipo_Part-time	tipo_Pasantia	tipo_Por Contrato	tipo_Por Horas
0	1111556097	0	0	1	0	1	0	0	0	0
1	1111949392	0	0	1	0	1	0	0	0	0
2	1112145935	0	0	1	0	1	0	0	0	0
3	1112146010	0	0	1	0	1	0	0	0	0
4	1112211475	0	0	1	0	1	0	0	0	0

Para el nombre del area, lo que hice agruparlas por el nombre, y contarlas, de forma que tengan más peso los valores con más apariciones .

	nombre_area	cantidadNombre
185	Ventas	5721
30	Comercial	3258
2	Administración	3116
143	Producción	2830
145	Programación	1735

También agregué cantidad de postulaciones, donde hice algo similar a lo anterior mencionado:

	idaviso	cantPostulaciones
10405	1112334791	38676
1084	1112094756	35185
931	1112033906	34738
10402	1112334788	26531
1837	1112204682	23379
11224	1112345900	22590
1640	1112196813	22109
9732	1112319451	20814
6977	1112280937	20784
8199	1112298966	19621

Cantidad de vistas, tiene el mismo concepto que postulaciones.

	cantidadVistos	idaviso
13310	71885	1806525
13015	60237	1806178
22436	43857	1112196813
21739	37255	1112094756
14163	31622	1807584

Para el caso de los datos nulos, lo que hice fue trabajar con Imputer, y me quedo con los elementos más frecuentes.

```
from sklearn.preprocessing import Imputer
imp = Imputer(missing_values='NaN', strategy='most_frequent', axis=0)
datosForTrainingSinPostuSinNulos = imp.fit_transform(datosSinPostulacion)
datosForTestFinalSinNulos = imp.fit_transform(datosToPredict)
```

Otras pruebas hechas:

Intenté aplicar TF-IDF para el caso de las descripciones de los detalles de avisos. Lo que hice fue limpiar los tags html que tiene por defecto el documento. Una vez limpiados, apliqué una lista de stopwords, para removerlas.

```
def limpiamosTexto(text):
    text
```

```
avisosDetalle['descripcionSinHTML'] = avisosDetalle['descripcion'].str.strip()
```

```
#elimino tags de descripcion
avisosDetalle['descripcionSinHTML'] = avisosDetalle['descripcion'].str.replace(r"<\/?\w+.*?>","");
avisosDetalle['descripcionSinHTML'] = avisosDetalle['descripcionSinHTML'].str.replace(r"\s{2,}"," ");
```

Luego, con esos datos, apliqué TFIDF Vectorizer, y con ese resultado, apliqué un algoritmo de Machine Learning: (Estas operaciones las realicé madiante un pipeline)

```
estimators = [("tf_idf", TfidfVectorizer()),
              ("ridge", linear_model.Ridge())]
model = Pipeline(estimators)
```

```
model.fit(datosMergeadosPrueba['descripcion'],datosMergeadosPrueba['sepostulo'])
```

Este resultado, fue agregado a una columna nueva, para así poder realizar un entrenamiento con el algoritmo elegido.

Lamentablemente, este resultado no fue mejor que mi mejor resultado.

Mejor Entrega

TP 2- Limpiado.ipynb

Para mi mejor entrega, lo que hice fue unir trabajar un sample de avisos postulados y no postulados:

Reducimos el set de postulaciones para trabajar ¶

Me quedo con un subset mucho mas chico, puesto que son demasiados datos.

[49]:
```
avisosPostuladosSample = postulaciones.sample(frac=0.08)
avisosPostuladosSample = avisosPostuladosSample[['idaviso','idpostulante']]
len(avisosPostuladosSample)
```

t[49]: 937031

[50]:
```
avisosNOPostuladosSample = avisosNoPostulado.sample(frac=0.07)
avisosNOPostuladosSample = avisosNOPostuladosSample[['idaviso','idpostulante']]
len(avisosNOPostuladosSample)
```

t[50]: 541588

[51]:
```
#Concateno los no postulados, con los postulados
avisosNOPostuladosSample['sepostulo'] = 0;
avisosPostuladosSample['sepostulo'] = 1;
datosMergeados = pd.concat([avisosNOPostuladosSample,avisosPostuladosSample])
```

[52]:
```
len(datosMergeados)
```

t[52]: 1478619

Los archivos no postulados generados mediante elementos random entre vistas y postulaciones. (Método 5 en el archivo de generación de postulaciones, y explicado en el apartado de Generación).

Con estos datos, lo que hice fue calcular la cantidad de vistas, cantidad de postulaciones por aviso, se convirtió a columnas de 0 y 1 los datos de los avisos detalles (nombre zona, tipo de trabajo y nivel laboral). De la misma manera, se pasaron las educaciones y los géneros, de forma de tener columnas con valores 0 y 1, dependiendo el sexo, y la educación con estados.

Mergeo con Genero y Edad

```
]: len(datosForTraining)
]: 2454734

]: datosForTrainingConGyE = pd.merge(datosForTraining,generoYEducacionOHC,on='idpostulante',how='inner')
   datosForTestingConGyE = pd.merge(datosForTesting,generoYEducacionOHC,on='idpostulante',how='left')

]: len(datosForTrainingConGyE)
]: 2394400
```

Mergeo con nombre area

```
]: #Mergeamos con los nombres

datosForTrainingConGyE['nombre_area'] = datosForTrainingConGyE['nombre_area'].fillna('Otros')
datosForTestingConGyE['nombre_area'] = datosForTestingConGyE['nombre_area'].fillna('Otros')

datosForTrainingConNombre = pd.merge(datosForTrainingConGyE,nombreConCantidad,on='nombre_area',how='left')
datosForTestingConNombre = pd.merge(datosForTestingConGyE,nombreConCantidad,on='nombre_area',how='left')

datosForTrainingConNombre['cantidadNombre'] = datosForTrainingConNombre['cantidadNombre'].fillna(0)
datosForTestingConNombre['cantidadNombre'] = datosForTestingConNombre['cantidadNombre'].fillna(0)
```

Mergeamos con cantidad vistas

```
]: datosConVistas = pd.merge(datosForTrainingConNombre, vistasGP,on=['idaviso'],how='left')
   testingConVistas = pd.merge(datosForTestingConNombre,vistasZIPGP,on=['idaviso'],how='left')

]: len(datosConVistas)
]: 2394400
```

Luego hice un trabajo con las descripciones:

Primero eliminé los tags, y quité todos aquellos espacios dobles que podríamos encontrar.

Trabajo con las descripciones

```
In [146]: # Trabajo con la descripcion
def limpioTextos(df):
    df['descripcionSinHTML'] = df['descripcion'].str.strip()
#elimino tags de descripcion
    df['descripcionSinHTML'] = df['descripcionSinHTML'].str.replace(r"<\/?\w+.*?>"," ");
    df['descripcionSinHTML'] = df['descripcionSinHTML'].str.replace(r"\s{2,}"," ");
    return df

In [147]: descripcionesLimpiadas = limpioTextos(datosConVistas)
testingDescripcionesLimpiadas = limpioTextos(testingConVistas)

In [153]: descripcionesLimpiadas['zonas'] = descripcionesLimpiadas['descripcionSinHTML'].str.contains('(zona de|ubicac
descripcionesLimpiadas['beneficios'] = descripcionesLimpiadas['descripcionSinHTML'].str.contains('(obra soci
descripcionesLimpiadas['experiencia'] = descripcionesLimpiadas['descripcionSinHTML'].str.contains('(expercie
descripcionesLimpiadas['excluyente'] = descripcionesLimpiadas['descripcionSinHTML'].str.contains('(requisito

#Testing Set
testingDescripcionesLimpiadas['zonas'] = pd.to_numeric(testingDescripcionesLimpiadas['descripcionSinHTML'].s
testingDescripcionesLimpiadas['beneficios'] = pd.to_numeric(testingDescripcionesLimpiadas['descripcionSinHTM
testingDescripcionesLimpiadas['experiencia'] = pd.to_numeric(testingDescripcionesLimpiadas['descripcionSinHT
testingDescripcionesLimpiadas['excluyente'] = pd.to_numeric(testingDescripcionesLimpiadas['descripcionSinHTM
```

Creé 4 columnas nuevas: zonas, beneficios, experiencia, excluyente. Definí algunos términos que pueden corresponderse a cada categoría, de forma que si alguno de estos apareciera, marque la columna.

beneficios	zona	experiencia	requisito
obra social	Ubicad.	experciencia comprobable	requisito
osde	zona	referencia	excluyente
bono		conocimientos en	nivel
home office		deseable	obligatori.
vacaciones		valora	
gimnasio		certificado	
franco		título	
		años minimo	
		paquete office	

Finalmente, en caso de tener algún dato nulo, se convierte al valor más frecuente.

```
from sklearn.preprocessing import Imputer
imp = Imputer(missing_values='NaN', strategy='most_frequent', axis=0)
datosForTrainingSinPostuSinNulos = imp.fit_transform(datosSinPostulacion)
datosForTestFinalSinNulos = imp.fit_transform(datosToPredict)
```

prueba5-PostArchivosFaltantesRFC100.csv 0.74356

15 minutes ago by Nico Gallegos

Agrego un analisis de descripcion

Explicación de Submits bajos

Durante el desarrollo del documento, mis resultados fueron bastante malos, y recién al final pude corregir la serie de errores que tuve .

El principal error / problema durante mi trabajo fue la mala generación de datos de no postulados, lo cual no me permitía hacer un buen entrenamiento.

Como ya mencioné en el apartado de generación de archivos no postulados, partí siempre con la premisa de generar avisos no postulados, con un merge entre avisos y postulaciones. Muchos de mis submits estaban basados con esa parte, entonces, a medida que iba a agregando más archivos de no postulados, mis generaciones no mejoraban, o hasta empeoraban. La gran diferencia entre mi mejor resultado, y el resto, es que cambié la premisa por una random.

La segunda explicación es que no había cargado la totalidad de los documentos provistos por la cátedra, si bien afecta al resultado, considero que era mucho peor lo anterior explicado.

Por último, las últimas generaciones desastrosas. Aquí se trataba de un error de código en el merge entre la transformación de generos, edades y los datos. Este “sutil”error, provocaba que perdiera la gran mayoría de features. Además, mientras hacía esta prueba, iba agregando datos generados con los nuevos métodos, lo cual acrecentaba el error, porque había muchos más datos incoherentes.

Elementos que quise agregar

- Intenté aplicar XGBoost como algoritmo de Machine Learning, sin embargo, no pude instalarlo en Windows, por lo que tuve descartarlo.
- Desarrollar las pruebas mediante Pipelines y FeatureUnion. Encontré estos elementos cuando quise aplicar TF-IDF, y me parecieron muy buenos para aplicar, pero por tiempo no lo implementé.
- Mejorar el análisis de las descripciones. Si bien se aplica en mi mejor análisis, es una primera aproximación que podría haberse refinado más , lo mismo que los títulos. Se entiende que los títulos importantes, y no pude aprovecharlos.
- Aplicar de mejor manera TF-IDF. No supe bien cómo aplicar la unión de los resultados con mis valores númericos, por eso opté por hacer un entrenamiento sólo con las descripciones, y luego colocar el resultado, pero siento que no lo aproveché al 100%, y debería haber sido mejor ese submit.
- Buscar más maneras de prorizar la educación. Si bien mi aplicación sirve, seguramente pude haber mejorado un poco más el desarrollo, para poder hacer que le de más importancia a los grados altos, y que la edudcación que tiene la persona, compararla con la descripción del aviso.

Conclusiones

La principal conclusión es que la mayor diferencia en la utilización de herramientas de Machine Learning, se da en los datos que se usan para entrenar, más que el algoritmo en sí. Una prueba con errores / mala verificación de datos , significó obtener menos del 30% de acierto, y un poco más del 70% con los datos de forma correcta, siempre utilizando los mismos features y mismo algoritmo.

A modo de crítica , los análisis planteados en el trabajo práctico 1 no fueron lo suficientemente profundos, lo cual me imposibilitó aplicar alguna relación entre las vistas y los postulantes que uno no sea visible a simple vista.