Pachamama Labs

Grupo 36

Informe final - Navent - Predicción de Postulaciones a Avisos Laborales

Alumnos

- Federico Diaz
- Mariano Stampella

Preparación de Datos

En relación a la manera que elegimos organizar los datos buscamos ser lo más simples posibles y organizar los datos de una forma que fuera útil para la obtención de features y simple para trabajar.

Las entidades que utilizamos fueron avisos, vistas, postulantes y postulaciones y postulaciones. Lo que hicimos fue tratar de concatenar los diferentes archivos tratando de evitar duplicados.

Postulaciones

De todas las entidades fue sobre la que menos trabajamos

```
def get_postulaciones(size=None):
    postulaciones = pd.read_csv('data/FiubaHasta15Abril/fiuba_4_postulaciones.csv', nrows =size)
        .drop_duplicates(subset=['idpostulante', 'idaviso'], keep='last')
    columns_rename = {'idaviso': 'id_aviso', 'idpostulante': 'id_postulante', 'fechapostulacion': 'fecha_postulacion'}
    postulaciones = postulaciones.rename(columns=columns_rename)
    postulaciones['fecha_postulacion']=pd.to_datetime(postulaciones['fecha_postulacion'])
    return postulaciones
```

El resultado es muy simple, los ids y las fechas de cada una de las postulaciones.

	id_aviso	id_postulante	fecha_postulacion
0	1112248724	NjlD	2018-01-19 07:39:16
1	1112286523	ZaO5	2018-01-24 15:07:39
2	1112272060	ZaO5	2018-01-24 15:20:10
3	1112288401	ZaO5	2018-01-26 08:37:04
4	1112300563	ZaO5	2018-01-30 13:35:48

Avisos

En el caso de los avisos, el proceso fue más complejo, y lo vamos a explicar en partes.

Para empezar lo más simple fue simplemente traernos los tres archivos, concatenar y evitar duplicados.

```
#Función que devuelve los avisos preparados para su procesamiento
def get_avisos_detalle():
    avisos1 = pd.read_csv('data/fiuba_6_avisos_detalle.csv')
    avisos2 = pd.read_csv('data/FiubaDesde15Abril/fiuba_6_avisos_detalle.csv')
    avisos3 = pd.read_csv('data/FiubaHasta15Abril/fiuba_6_avisos_detalle.csv')
    avisos4 = pd.read_csv('data/fiuba_6_avisos_detalle_missing_nivel_laboral.csv')
    avisos_detalle = pd.concat([avisos1,avisos2,avisos3,avisos4])
    .drop_duplicates(subset=['idaviso'], keep='last').reset_index(drop=True)
    columns_rename = {'idpostulante': 'id_postulante', 'idaviso': 'id_aviso'}
    avisos_detalle = avisos_detalle.rename(columns=columns_rename)
```

Luego empezamos a trabajar sobre varios features a los que convertimos a valores jerárquicos.

```
#Los niveles de las búsquedas se ordenaron jerárquicamente
to_nivel_laboral_nro = {'Senior / Semi-Senior' : 2, 'Junior':1, 'Otro':0,
    'Jefe / Supervisor / Responsable':3,
    'Gerencia / Alta Gerencia / Dirección':4}

#Se definió una jerarquía también en relación al tipo de trabajo.
to_tipo_trabajo_nro={'Full-time':0, 'Part-time':1, 'Teletrabajo':2, 'Por Horas':3, 'Pasantia':4,
    'Temporario':5, 'Por Contrato':6, 'Fines de Semana':7, 'Primer empleo':8,
    'Voluntario':9}

#Aplicamos las jerarquías que definimos antes
avisos_detalle['nivel_laboral_nro']= avisos_detalle['nivel_laboral'].map(to_nivel_laboral_nro)
avisos_detalle['tipo_de_trabajo_nro'] = avisos_detalle['tipo_de_trabajo'].map(to_tipo_trabajo_nro)
avisos_detalle['nombre_area_nro'] = avisos_detalle['nombre_area'].map(to_nombre_area_numero)
```

El nivel laboral y el tipo de trabajo fueron dos features bastante simples ya que venían con los datos ya definidos. Lo único que hicimos en este caso fue definir el orden en base a criterios que consideramos útiles.

Luego trabajamos sobre el nombre_area para armar clusters de postulantes, en base al tipo de aviso que miraron o al que se postularon.

```
#Preparamos las áreas para un fúturo K-Mean de avisos
to_nombre_area_numero = pd.Series(avisos_detalle['nombre_area'].unique()).to_dict()
to_nombre_area_numero  = {v: k for k, v in to_nombre_area_numero.items()}
```

Feature Hasher

Luego trabajamos sobre el título de los avisos, la idea fue generar tokens, medir sus frecuencias y en base a eso armar features que nos puedan dar información relevante sobre los avisos.

La idea fue basarnos en una función de tokenizado:

```
def tokens(doc):
    return (tok.lower() for tok in re.findall(r"\w+", doc))


def token_freqs(doc):
    freq = defaultdict(int)
    for tok in tokens(doc):
        freq[tok] += 1
    return freq
```

Nada muy complejo pero con esta información de frecuencia x token llamamos a Feature Hasher de scikit y mediante el método transform obtenemos las columnas que luego agregamos al dataframe de los avisos.

```
#Procesamos los títulos para generar features que describan a cada título
#En particular utilzamos FeatureHasher de scikit, probamos con diferentes cantidades y finalmente nos quedamos con 40
#Esto lo hicimos antes de definir los clusters por lo tanto son muchos avisos.
h = FeatureHasher(n_features = 40, input_type='string', dtype='float32')

#Aplicamos una función que considere las frecuencias de los tokens.
avisos_detalle['titulo_as_token_freq'] = avisos_detalle.titulo.apply(lambda x: token_freqs(x))
#Aplicamos el FeatureHasher a la columna
x = h.transform(avisos_detalle['titulo_as_token_freq'])
avisos_detalle['titulo'] = list(x.toarray())
```

Finalmente lo que hacemos es concatenar y sacar la columna de título que ya no la necesitamos más al haber sido reemplazada por los tokens.

```
titulos_como_lista = avisos_detalle.titulo.apply(pd.Series)
avisos_detalle = pd.merge(avisos_detalle, titulos_como_lista, left_index = True, right_index = True)
avisos_detalle = avisos_detalle.drop(['titulo'], axis=1)
```

Vistas

El procesamiento de las vistas nos deja sobre un feature que tuvo mucho éxito en nuestro modelo que es cuantas veces visualizó un postulante un determinado aviso.
Esto lo resolvemos con un groupby muy simple:

```
#Obtenemos todas las vistas por parte de los postulantes
def get_vistas(size=None):
    vistas1 = pd.read_csv('data/FiubaHasta15Abril/fiuba_3_vistas.csv', nrows =size)
    vistas2 = pd.read_csv('data/FiubaDesde15Abril/fiuba_3_vistas.csv', nrows =size)
    vistas3 = pd.read_csv('data/fiuba_3_vistas.csv', nrows =size)
    vistas = pd.concat([vistas1, vistas2, vistas3])

    #Sumamos las visualizaciones que hizo un postulante sobre un aviso
    vistas_sumarizadas = vistas.groupby(['idpostulante', 'idAviso'], as_index=False)['timestamp'].count()
    columns_rename = {'idAviso': 'id_aviso', 'idpostulante': 'id_postulante', 'timestamp': 'visualizaciones'}
    vistas_sumarizadas = vistas_sumarizadas.rename(columns=columns_rename)

    return vistas_sumarizadas
```

Postulantes

Los postulantes tienen varios archivos que aportan información sobre ellos, para obtener la lista de postulantes la función es muy simple:

```
def get_postulantes_limpios():
    postulantes = pd.merge(get_postulantes_genero_edad(), get_postulantes_nivel_educativo(),
                           on='id_postulante', how='outer')
    order_for_columns = ['id_postulante','edad_postulante', 'genero_postulante', 'genero_postulante_nro',
                         'maximo_nivel_educativo_postulante']
    return postulantes[order_for_columns]
```

Los datos sin embargo son obtenidos de otras dos funciones

En la primera obtenemos los datos relacionados a los estudios del postulante:

```
#Obtenemos los niveles educativos
def get_postulantes_nivel_educativo_para(path):
    postulantes_nivel_educativo = pd.read_csv(path)
    columns_rename = {'idpostulante': 'id_postulante', 'nombre': 'formacion_postulante',
                      'estado': 'estado_formacion_postulante'}
    postulantes_nivel_educativo=postulantes_nivel_educativo.rename(columns=columns_rename)

    #Definimos una variable categórica con jerarquía.
    formacion_to_number={'Secundario' : 10, 'Otro': 20, 'Terciario/Técnico' : 30, 'Universitario' : 40,
                         'Posgrado' : 50,'Master' : 50, 'Doctorado' : 50}
    postulantes_nivel_educativo['formacion_postulante_numero']= postulantes_nivel_educativo['formacion_postulante']
    .map(formacion_to_number);
    estado_to_number = {'En Curso': 4, 'Abandonado': 0, 'Graduado': 8}
    postulantes_nivel_educativo['estado_formacion_postulante_numero']=
        postulantes_nivel_educativo['estado_formacion_postulante'].map(estado_to_number)
    postulantes_nivel_educativo['nivel_educativo_postulante_numero'] =
        postulantes_nivel_educativo['formacion_postulante_numero'] +
        postulantes_nivel_educativo['estado_formacion_postulante_numero']
    postulantes_nivel_educativo['nivel_educativo_postulante_texto'] =
        postulantes_nivel_educativo['formacion_postulante'] + ' - ' +
        postulantes_nivel_educativo['estado_formacion_postulante']
    relevant_columns = ['id_postulante','nivel_educativo_postulante_texto', 'nivel_educativo_postulante_numero']
    postulantes_nivel_educativo = postulantes_nivel_educativo[relevant_columns]
    grouped=postulantes_nivel_educativo.groupby(['id_postulante']).agg({'nivel_educativo_postulante_numero':['max']})
    df=grouped.reset_index()
    df.columns = ['id_postulante', 'maximo_nivel_educativo_postulante']
    return df
```

En la siguiente tomamos los datos de edad y género del postulante.

```
def get_postulantes_genero_edad():
    postulantes1 = pd.read_csv('data/fiuba_2_postulantes_genero_y_edad.csv')
    postulantes2 = pd.read_csv('data/FiubaDesde15Abril/fiuba_2_postulantes_genero_y_edad.csv')
    postulantes3 = pd.read_csv('data/FiubaHasta15Abril/fiuba_2_postulantes_genero_y_edad.csv')
    postulantes_genero_edad = pd.concat([postulantes1,postulantes2,postulantes3])
        .drop_duplicates(subset=['idpostulante'], keep='last').reset_index(drop=True)
    postulantes_genero_edad['año_nacimiento_postulante']=get_year_of_birth(postulantes_genero_edad)
    postulantes_genero_edad['edad_postulante']=postulantes_genero_edad['año_nacimiento_postulante']
        .map(get_age, na_action=None)
    postulantes_genero_edad['rango_edad_postulante']=postulantes_genero_edad['año_nacimiento_postulante']
        .map(get_age_range, na_action=None)
    columns_rename = {'idpostulante': 'id_postulante', 'fechanacimiento': 'fecha_nacimiento_postulante',
                      'sexo': 'genero_postulante'}
    postulantes_genero_edad = postulantes_genero_edad.rename(columns=columns_rename)
    postulantes_genero_edad = postulantes_genero_edad[['id_postulante', 'genero_postulante',
                                                       'fecha_nacimiento_postulante',
                                                       'edad_postulante',
                                                       'rango_edad_postulante']]
    postulantes_genero_edad['genero_postulante_nro'] = postulantes_genero_edad['genero_postulante']
         .map({'FEM': 0, 'MASC': 1, 'NO_DECLARA': 2})
    return postulantes_genero_edad
```

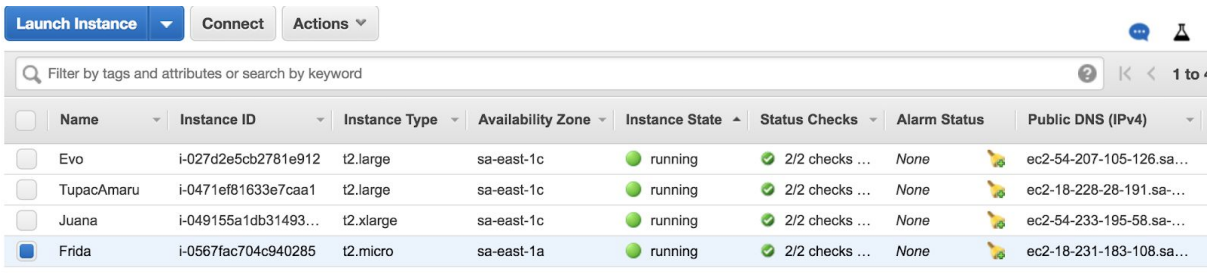
Set de entrenamiento

Luego de obtener y procesar todos los datos, armamos el set de entrenamiento.

```python
#Obtener los datos para el set de entrenamiento
def get_datos_entrenamiento(postulantes_masinfo, size=None):

    #Obtengo las postulaciones
    postulaciones_aplicadas = get_detalle_postulaciones(size)

    #Defino las columnas que van a ser necesarias para el set de entrenamiento.
    columnas_relevantes = x_entrenamiento() + y_entrenamiento()

    #Definimos los casos positivos.
    postulaciones_aplicadas['sepostulo'] = True

    #Consideramos los casos negativos a aquellos que visualizaron pero no se postularon
    postulaciones_no_aplicadas = get_detalle_vistas(size)
    vistas = postulaciones_no_aplicadas[['id_postulante', 'id_aviso', 'visualizaciones']]
    postulaciones_aplicadas = pd.merge(postulaciones_aplicadas, vistas, on=['id_postulante', 'id_aviso'], how='left' )
    postulaciones_aplicadas['visualizaciones'].fillna(value=0, inplace=True)
    postulaciones_no_aplicadas['sepostulo'] = False

    #Balanceamos los positivos y negativos
    postulaciones_aplicadas = postulaciones_aplicadas[:postulaciones_no_aplicadas.shape[0]]
    postulaciones_no_aplicadas = postulaciones_no_aplicadas[:postulaciones_aplicadas.shape[0]]

    #Primero hacemos un append entre los positivos y negativos, luego hacemos un merge con los datos de clustering
    postulaciones_no_aplicadas = postulaciones_aplicadas.append(postulaciones_no_aplicadas)
         .drop_duplicates(subset=['id_aviso', 'id_postulante'], keep='first')
    postulaciones_no_aplicadas = pd.merge(postulaciones_no_aplicadas
                                          , postulantes_masinfo
                                          , on='id_postulante'
                                          , how='inner')
    return postulaciones_no_aplicadas[columnas_relevantes].dropna()
```

Infraestructura

Para poder llevar a cabo varias pruebas en simultáneo, y poder probar diferentes modelos armamos un conjunto de servidores, estos nos sirvieron para entender un poco la performance de los modelos y de nuestros datos, aunque no las supimos utilizar con inteligencia.



Si bien no tiene que ver con estrictamente con el trabajo fue muy interesante el aprendizaje sobre los entornos, anaconda, jupyter y demás en varias instancias EC2.

Modelos de predicción

Lineales

El primer modelo de predicción que evaluamos fue el de regresión lineal. Este modelo lo utilizamos solamente con set de entrenamientos menor a 100K

```python
# Para un set de entrenamiento superior a 100K el tiempo de ejecucion era intratable. Para esos set cercanos
# a 100K el escore no llegaba a 0.6
# Sin embargo obtuvimos un sorprendente score de 0.3 para un set de 5k... nuestro "no predictor
def get_predictor(set_entrenamiento):
    X=pd.get_dummies(set_entrenamiento.loc[:, x_entrenamiento()])
    y=set_entrenamiento.loc[:, 'sepostulo']
    clf = svm.SVC()
    return clf.fit(X, y)
```

Random Forest

El segundo modelo que utilizamos fue el clasificador Random Forest. Con este modelo obtuvimos el más alto score. Para los datos de la competencia 0.83906 y 0.99409 para nuestro set de test.
Los mejores hiperparametros que encontramos utilizando greed search fueron

- n_estimators=50,min_samples_split=5,min_samples_leaf=5, max_depth=5

```
In [84]: # Mejor score
         clf = RandomForestClassifier(n_estimators=50,min_samples_split=5,min_samples_leaf=5, max_depth=5)
         predictor =  clf.fit(X_train, y_train)
         predictor.score(X_test, y_test)

Out[84]: 0.99409213989509759
```

Ensamble o Pipeline

Luego de no poder encontrar los mejores hiperparametros para subir el score alcanzado con nuestro Random Forest, decidimos mudarnos a macondo y ensayar la alquimia. Es decir, buscar un ensamble.
El ensamble que probamos es utilizando un metodo de Boosting, basado en transformar nuestras features en una matriz dispersa de grandes dimensiones. Luego entrenar un modelo lineal sobre esas features.
En este pipeline participan los siguientes modelos basados en arboles

- Random forest (RF). Los hiperparametros utilizados fueron los que nos dieron el mas alto score: (n_estimators=50,min_samples_split=5,min_samples_leaf=5, max_depth=5)
- Random Tree (RT).
- Gradient Boosting of Regression Trees Machine (GBT) [ref-GBT], fue el metodo que orquesto el ensamble.

Ensamble

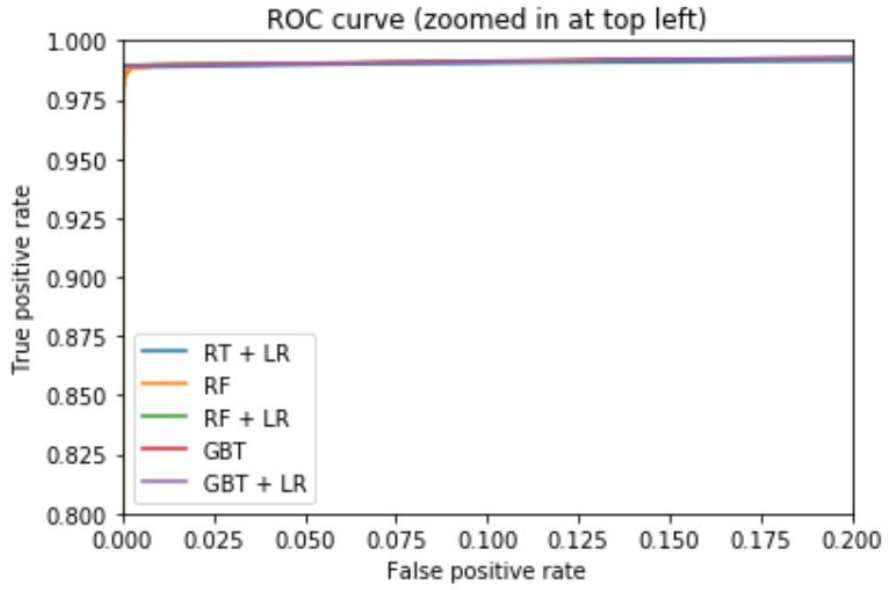
```
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import (RandomTreesEmbedding, RandomForestClassifier,
                              GradientBoostingClassifier)
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from sklearn.pipeline import make_pipeline

n_estimator = 50
X_train, X_test, y_train, y_test = train_test_split(set_entrenamiento.loc[:, x_entrenamiento()],
                                                    set_entrenamiento.loc[:, 'sepostulo'],
                                                    test_size=0.5)
X_train, X_train_lr, y_train, y_train_lr = train_test_split(X_train,
                                                            y_train,
                                                            test_size=0.5)
```

```
# Transformacion de features (no supervisada) basada totalmente en randoms trees
rt = RandomTreesEmbedding(max_depth=3, n_estimators=n_estimator,
    random_state=0)

rt_lm = LogisticRegression()
pipeline = make_pipeline(rt, rt_lm)
pipeline.fit(X_train, y_train)
y_pred_rt = pipeline.predict_proba(X_test)[:, 1]
fpr_rt_lm, tpr_rt_lm, _ = roc_curve(y_test, y_pred_rt)
```

```
# Transformacion supervisada basada en nuestro Random Forest
rf = RandomForestClassifier(n_estimators=50,min_samples_split=5,min_samples_leaf=5, max_depth=5)
rf_enc = OneHotEncoder()
rf_lm = LogisticRegression()
rf.fit(X_train, y_train)
rf_enc.fit(rf.apply(X_train))
rf_lm.fit(rf_enc.transform(rf.apply(X_train_lr)), y_train_lr)

y_pred_rf_lm = rf_lm.predict_proba(rf_enc.transform(rf.apply(X_test)))[:, 1]
fpr_rf_lm, tpr_rf_lm, _ = roc_curve(y_test, y_pred_rf_lm)
```

```
grd = GradientBoostingClassifier(n_estimators=n_estimator)
grd_enc = OneHotEncoder()
grd_lm = LogisticRegression()
grd.fit(X_train, y_train)
grd_enc.fit(grd.apply(X_train)[:, :, 0])
grd_lm.fit(grd_enc.transform(grd.apply(X_train_lr)[:, :, 0]), y_train_lr)

y_pred_grd_lm = grd_lm.predict_proba(
    grd_enc.transform(grd.apply(X_test)[:, :, 0]))[:, 1]
fpr_grd_lm, tpr_grd_lm, _ = roc_curve(y_test, y_pred_grd_lm)


# El modelo gradient boosted, puro
y_pred_grd = grd.predict_proba(X_test)[:, 1]
fpr_grd, tpr_grd, _ = roc_curve(y_test, y_pred_grd)


# El modelo random forest puro
y_pred_rf = rf.predict_proba(X_test)[:, 1]
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_pred_rf)

plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_rt_lm, tpr_rt_lm, label='RT + LR')
plt.plot(fpr_rf, tpr_rf, label='RF')
plt.plot(fpr_rf_lm, tpr_rf_lm, label='RF + LR')
plt.plot(fpr_grd, tpr_grd, label='GBT')
plt.plot(fpr_grd_lm, tpr_grd_lm, label='GBT + LR')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
plt.figure(2)
plt.xlim(0, 0.2)
plt.ylim(0.8, 1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_rt_lm, tpr_rt_lm, label='RT + LR')
plt.plot(fpr_rf, tpr_rf, label='RF')
plt.plot(fpr_rf_lm, tpr_rf_lm, label='RF + LR')
plt.plot(fpr_grd, tpr_grd, label='GBT')
plt.plot(fpr_grd_lm, tpr_grd_lm, label='GBT + LR')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve (zoomed in at top left)')
plt.legend(loc='best')
plt.show()
```

Sin embargo no pudimos mejorar nuestro score, solamente mantenerlo. En este grafico vemos la comparacion entre los distintos predictores por separado y los ensambles, respecto de los falsos positivos obtenidos (promedio). Cómo se puede observar no obtuvimos un resultado esperado que nos entusiasme.



Armado de una grilla para mejorar los hiper parámetros

Si bien probamos muchos modelos diferentes enseguida encontramos en random forrest lo que nos pareció el mejor por performance y resultados. Tal vez fue un error, no sabemos pero probamos durante días otras alternativas y ninguno fue mejor.

Para poder optimizar el uso del algoritmo utilizamos GridSearchCV de Scikit de la siguiente manera:

```
# Definir los parámetros que buscamos mejorar
tuned_parameters = [{'n_estimators': [5, 10, 15, 20], 'max_depth': [2, 5, 7, 9],
                     'min_samples_split': [5,10,15], 'min_samples_leaf': [5,10,15]}]

scores = ['precision', 'recall']

for score in scores:
    print("# Optimizando los parámetros para %s" % score)
    print()

    clf = GridSearchCV(RandomForestClassifier(), tuned_parameters, cv=5,
                       scoring='%s_macro' % score)
    clf.fit(X_train, y_train)

    print("Los mejores parámetros para este set de entrenamiento son:")
    print()
    print(clf.best_params_)
    print()
    print("Resultados: ")
    print()
    means = clf.cv_results_['mean_test_score']
    stds = clf.cv_results_['std_test_score']
    for mean, std, params in zip(means, stds, clf.cv_results_['params']):
        print("%0.3f (+/-%0.03f) for %r"
              % (mean, std * 2, params))
    print()

    print("Reporte Final:")
    print()
    print()
    y_true, y_pred = y_test, clf.predict(X_test)
    print(classification_report(y_true, y_pred))
    print()
```

Esto nos permitió entender cuales eran los mejores parámetros para poder utilizar el Random Forrest

Como resultado finalmente nos dice que:

```
# Optimizando los parámetros para precision

Los mejores parámetros para este set de entrenamiento son:

{'max_depth': 5, 'min_samples_leaf': 5, 'min_samples_split': 5, 'n_estimators': 40}

Resultados:

0.986 (+/-0.029) for {'max_depth': 2, 'min_samples_leaf': 5, 'min_samples_split': 5, 'n_estimators': 15}
0.995 (+/-0.015) for {'max_depth': 2, 'min_samples_leaf': 5, 'min_samples_split': 5, 'n_estimators': 20}
0.986 (+/-0.039) for {'max_depth': 2, 'min_samples_leaf': 5, 'min_samples_split': 5, 'n_estimators': 30}
0.997 (+/-0.006) for {'max_depth': 2, 'min_samples_leaf': 5, 'min_samples_split': 5, 'n_estimators': 40}
0.996 (+/-0.008) for {'max_depth': 2, 'min_samples_leaf': 5, 'min_samples_split': 5, 'n_estimators': 50}
0.993 (+/-0.012) for {'max_depth': 2, 'min_samples_leaf': 5, 'min_samples_split': 5, 'n_estimators': 60}
0.986 (+/-0.042) for {'max_depth': 2, 'min_samples_leaf': 5, 'min_samples_split': 5, 'n_estimators': 70}
```

Resultados obtenidos

El modelo con el cual obtuvimos un score de 0.83906 es un RandomForest con los siguientes hiperparametros

- Cantidad de Arboles: 40
- En cuanto a la cantidad de atributos por arbol quedo acotada por los siguientes hiperparametros: 5 cómo minimo por split y 5 cómo minimo por hoja. Con una profundiad maxima de 5

El set de entrenamiento tiene 2 features engineers destacables que fueron claves para alcanzar el score

1. Clasificacion de avisos: Mediante K means buscamos clusters de avisos basados en el texto del titulo de los avisos (transformado mediante feature hasher) y el resto de atributos: zona, nombre de area, nivel laboral y total de visualizaciones . Este procedimiento fue explcado en la seccion.
2. Clasificacion de postulantes: Mediante K means buscamos clusters de postulantes basados en la cantidad de visualizaciones que realizan sobre los avisos agrupados por area

Acceso al respositorio y obtencion de resultados

El repositorio esta en: `https://github.com/pachamama-labs/navent_visualization.git`
Los pasos para reproducir la obtencion de lo resultados se encuentra en el archivo README del repositorio:
`https://github.com/pachamama-labs/navent_visualization/blob/master/README.md`