# Universidad de Buenos Aires

# Facultad de Ingeniería


(75.06) Organización de Datos

Cátedra: Luis Argerich

**Repositorio: https://github.com/freedocx/ZonaJobsPrediction**

**En el marco de una competencia en la plataforma Kaggle**

*1° Cuatrimestre 2018*

Alumnos:

GARATE, Julián Matías (93043)

IGLESIAS, Ignacio (95050)

SHOKIDA, German (96172)

TULIPANI, Gaston (96570)

| Materia: 75.06 Organización de Datos | | Repo: https://github.com/freedocx/ZonaJobsPrediction |
|---|---|---|
| Grupo: Bitcoin | Cuatrimestre: 1C 2018 | Informe:Navent |

# Contenido

| Materia: 75.06 Organización de Datos | | Repo: **https://github.com/freedocx/ZonaJobsPrediction** |
|---|---|---|
| Grupo: Bitcoin | Cuatrimestre: 1C 2018 | Informe:Navent |

# Introducción

Link: http://www.navent.com/
El objetivo del trabajo es en base a datos de postulantes y avisos de trabajo, es dar una probabilidad de que un usuario se postule.
Para resolver este problema se utilizan algoritmos de *Machine Learning*.

# Análisis del set de Datos

El dataset contiene cuatro partes que hacen referencia a distintas franjas temporales que van desde el verano hasta la fines de abril.

Para realizar el entrenamiento hay una franja temporal del 17 de abril en adelante los cuales no tienen datos de postulaciones y en reemplazo se nos proveyó de un test set con supuestas postulaciones las cuales deberíamos clasificar como reales o falsas asignando una probabilidad.

# Lenguaje de Programación

Para este trabajo, se utilizó Python 3, sobre la plataforma de Anaconda y las siguientes librerías

- Sklearn: brinda todos los algoritmos para el procesamiento y el formato de datos
- Pandas: lectura y trabajo de análisis sobre las estructuras de datos
- Regex: procesamiento de texto
- Xgboost: algoritmo de machine learning

# Selección del Algoritmo

El primer algoritmo que decidimos empezar a implementar fue Naive Bayes. Luego
Comenzamos a realizar pruebas con el clasificador Naive Bayes y obtuvimos resultados muy bajos, por ende comenzamos a preguntarnos qué tipo de algoritmo deberíamos utilizar. Decimos afrontar el problema con Random Forest y no logramos mejorar mucho. El mejor resultado que obtuvimos fue 65%. Luego realizamos pruebas con un tercer algoritmo, en este caso XGBoost, con el cual obtuvimos mucho mejores resultados que RF, pasando de un 65% a un 85%. Cambiando el tipo de algoritmo del clasificador 1 o 0 a un cálculo directo de probabilidades de un “1” hicimos el último salto a nivel predicción por arriba del 90%.

**RandomForest**

Random Forest es una aplicación de árboles de decisión donde cada uno usa un bootstrap del set de entrenamiento y un cierto conjunto de atributos tomados al azar. El resultado se combina, promediando o votando. El ajuste de los hiper parámetros pasa por variar la cantidad de árboles y la profundidad de los mismos.

**XGBoost**

También utiliza árboles pero a diferencia de RF, xgboost utiliza el método de Boosting. Sencillamente el algoritmo se compone de un una función error, una función de regularización para mantener conservador el modelo (y controlar la complejidad de los árboles) y un conjunto de árboles anidados los cuales trabajan sobre el error del anterior. El objetivo del algoritmo es minimizar la suma del error con la regularización, de esa manera el modelo resultante evita el sobreajuste, derivado de una excesiva complejidad, mientras decrece el error.
En este caso el ajuste de los hiper parámetros, pasa por ajustar la función de regularización, la tasa de aprendizaje (ganancia), el tamaño y profundidad de los árboles.

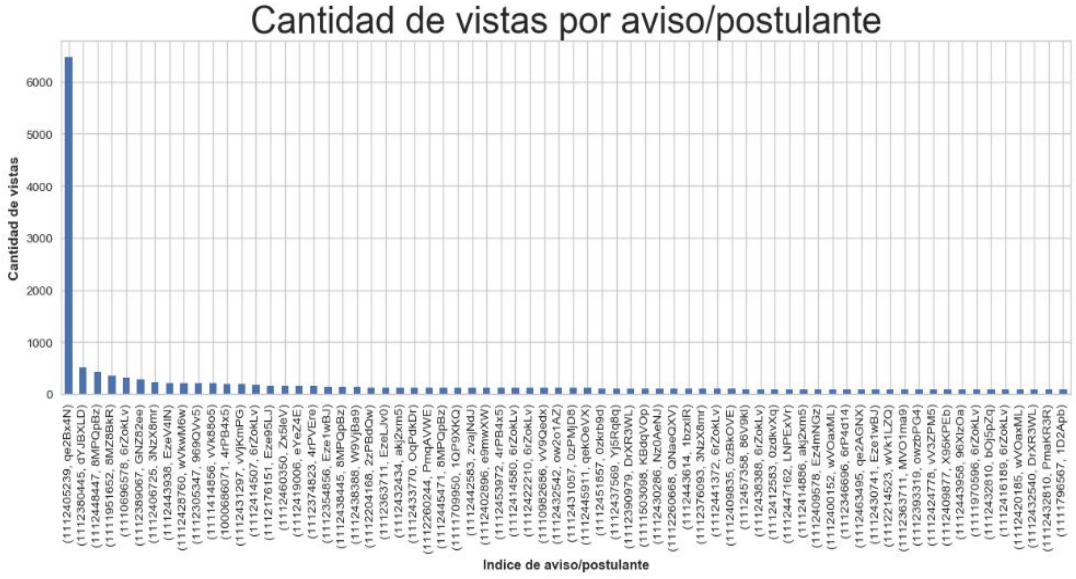
# Desarrollo

## Preparación de los features

### Postulaciones y vistas

Las postulaciones y las vistas son las principales variables para lograr un buen aprendizaje del algoritmo, como se verá a continuación. El tratamiento fue el siguiente:

1. Cantidad de vistas totales de un **postulante a un aviso**.



2. Cantidad de vistas en los últimos días.
3. Cantidad de vistas totales de un **postulante en general**.


Se puede ver que hay usuarios que son muy activos y están observando avisos de manera muy frecuente.

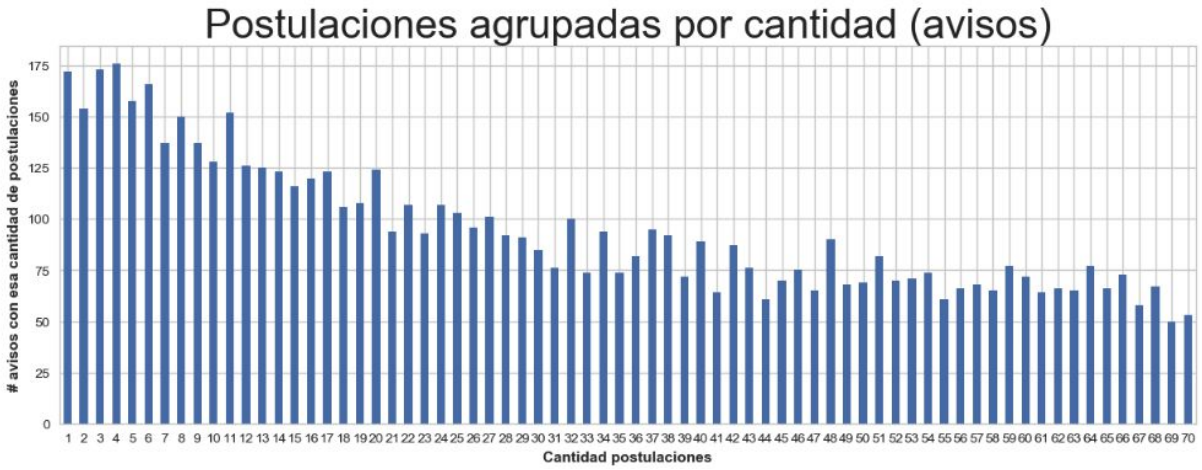
4. Cantidad de vistas de un **postulante en los últimos días**.
5. Cantidad de **vistas totales que tiene un aviso**.


Hay avisos que son más frecuentados que otros, más atractivos.

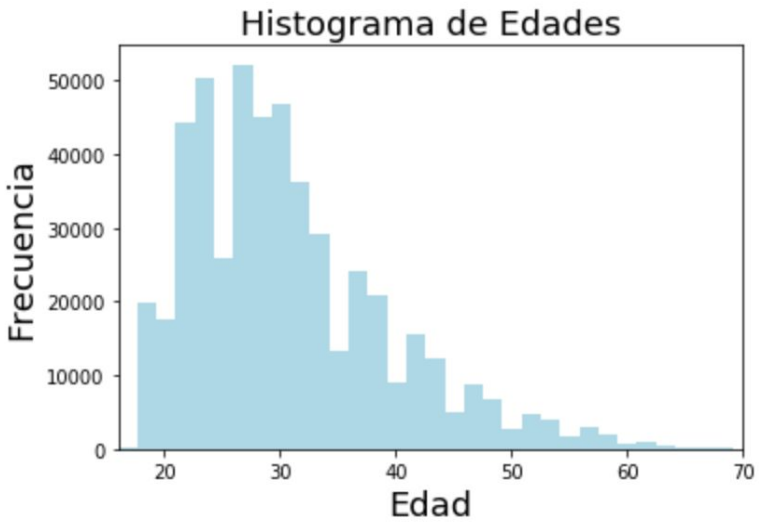
6. Cantidad de vistas totales que tiene un **aviso en los últimos días**.
7. Cantidad de postulaciones que realiza cada postulante


Hay postulantes que tienden a postularse mucho.

También vemos agrupando las cantidades de postulaciones por postulante y por aviso.

**Edad**: la edad de los postulantes se calculó en base a la fecha de nacimiento.

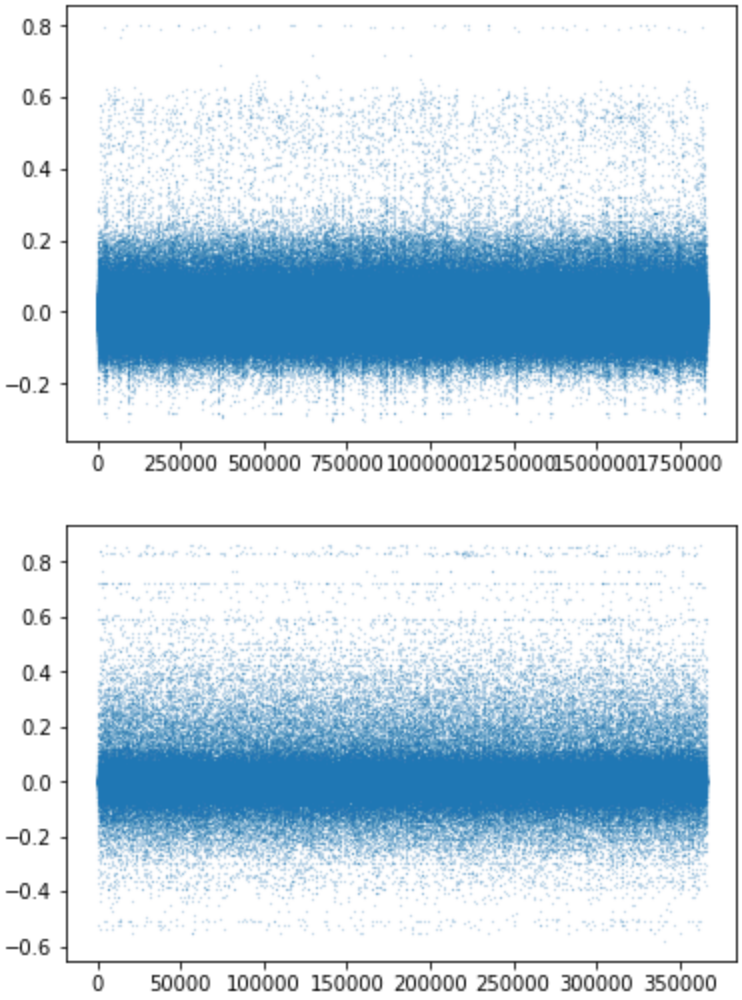



**Nivel académico**: se creó una escala numérica de 0 a 1, donde 0 es “otro” y 1 es posgrado/maestria/doctorado.
**Estado**: 1 terminado, 0 sin completar/abandonado/en curso.
**Zona del trabajo**: se juntaron todas las alternativas del Gran Buenos Aires en una sola categoría y para los trabajos en el interior se rótulo “Interior”.
**Tipo/Área de trabajo**: para estos features categóricos se procedió a binarizar los en columnas.
**Detalles y títulos:** títulos y descripciones de las publicaciones laborales que se ofrecen en el sitio. Tienen una gran cantidad de información oculta en cada palabra, para poder obtenerla se necesito limpiar todos los caracteres no alfabéticos, sacar las “stopwords”, los caracteres provenientes del código HTML, las palabras cortas y otras que a nuestro sentido no aportan especificidad, como ser la palabra “búsqueda”, la cual aparece en casi todos los avisos.
Luego se aplicó **LSI**:

1. Se realizó una vectorización y se aplicó **TF-IDF**.
2. Se calculó La **SVD** y pasamos a quedarnos con un set de datos reducido representado por la matriz U de la descomposición. En base al cálculo de la variabilidad de los valores originales contra los representados en la nueva base V, usando todas las palabras, probamos diversas cantidad de entre 50 y 250 vectores.

A continuación se puede ver una disposición de los valores del conjunto resultante.

Detalles vs Titulos




Como se puede ver la cantidad de palabras en las descripciones es mucho mayor y la matriz resultante es mucho más “dispersa”, esto se deduce de que casi hay muchos más valores girando entorno al 0.

Paralelamente al proceso anterior se aplicó **K-Means** sobre sobre los títulos.
K-Means es un algoritmo simple y bastante rápido que se basa en la generación, en base a iteraciones y cálculos de distancias, de clusters o agrupaciones de datos. De esta forma sobre la matriz de términos resultantes de aplicar TF-IDF se generan las agrupaciones.
El resultado es interesante a medida que se va cambiando la cantidad de grupos; sobre la relación de 5000 palabras sobre 50 grupos se llega a apreciar como las palabras referidas a temáticas parecidas se juntan.

Cluster 38: recepcionista, administrativa, bilingue, secretaria, asistente, zona, telefonista, eventual, concesionario, jurídico, inglés …

Cluster 32: programador, analista, java, tribunales, senior, zona, visual, funcional, plataforma, sistemas Senior, comunicaciones, oracle

Se puede ver como el C38 pareciera agrupar palabras referenciadas a avisos que buscan mujeres puntualmente y por otro lado C32 pareciera agrupar palabras relacionadas con el área informática.

## Tratamiento de los NaNs

En primer término, derivado del análisis exploratorio se detectaron datos faltantes.

- Edad: la edad se completó con el promedio
- Nivel educativo: se completó con el promedio redondeado, o sea nivel secundario.
- Tipo de trabajo: para rellenar el tipo de trabajo, luego de completar la SVD se usó como base para predecir los nans con Random Forest.

## Encode

En pruebas iniciales como se explicó antes utilizamos Random Forest el cual no necesita normalización de las variables continuas.
Posteriormente con el cambio hacia Xgboost, se procedió a probar diferentes alternativas.
Después de varias pruebas con las alternativas que brinda la librería sklearn optamos por MaxMinScaler.

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
X_scaled = X_std * (max - min) + min
```

El resultado es un dataset con una escala unificada para todas las variables.

## CrossValidation-GridSearch

La elección de los hiper parámetros del algoritmo se automatizó con Cross Validation y Grid Search. El cálculo del error se determinó, como en la competencia por el área por debajo de la curva. Las siguientes combinaciones fueron probadas.

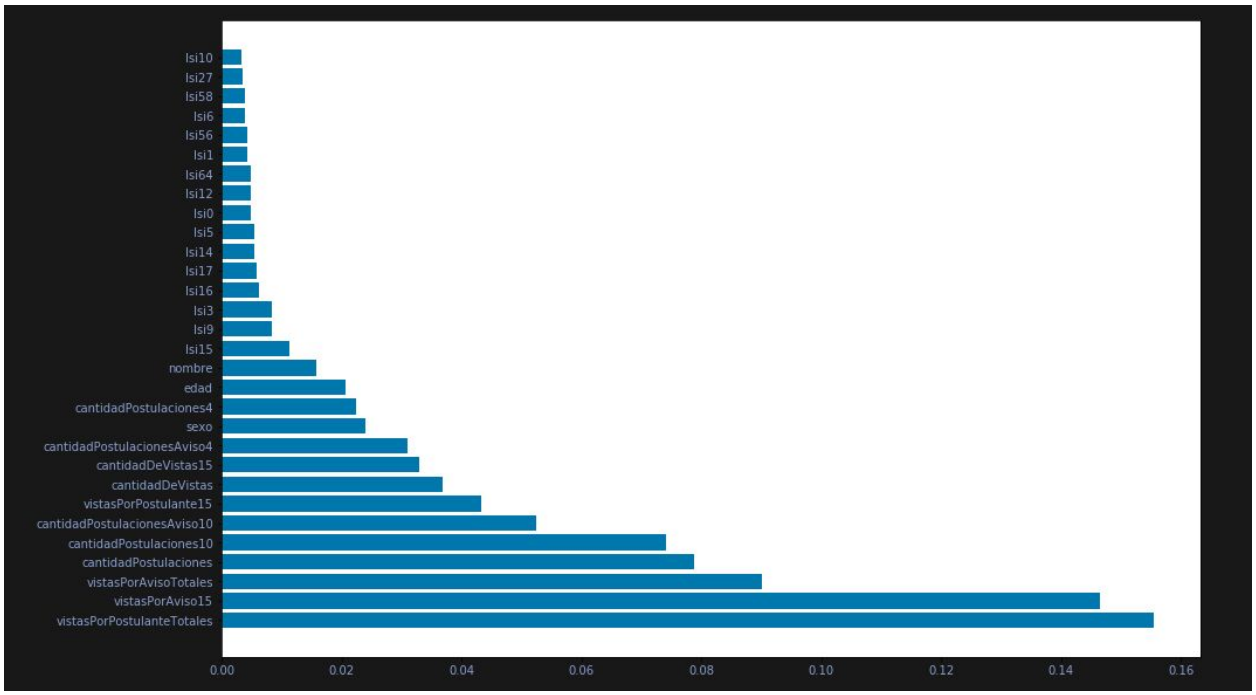
```
parameters = {'learning_rate': [0.1,0.05,0.2,0.5],
              'max_depth': [3,4,5,6,7],
              'gamma': [0,0.1,0.2,0.5,1],
              'n_estimators': [60,70,80]}
```

Ajustando el la tasa de aprendizaje, la regularización para aportar conservadurismo, la profundidad de los árboles, y la cantidad.
El resultado que mejor ajustaba fue el por “default” pero con 5 en el nivel de profundidad.

## Entrenamiento

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, eval_metric='auc', gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=5, min_child_weight=1, missing=None,
       n_estimators=100, n_jobs=4, nthread=None,
       objective='binary:logistic', random_state=0, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=False,
       subsample=1)
```

Importancias de los features para el entrenamiento del algoritmo




1

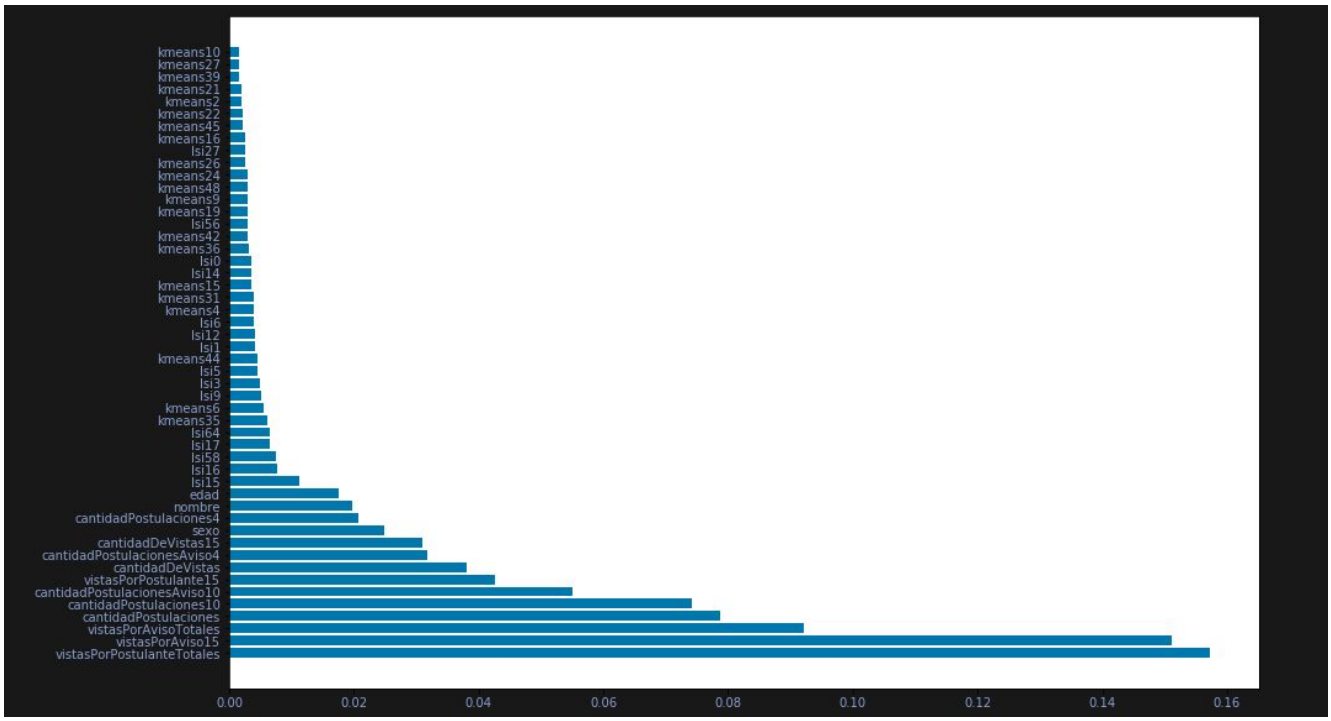
Como se puede apreciar las cantidades de vistas y de publicaciones son las variables a las cuales más peso se les asigna, principalmente a las vistas.
El resultado fue de 98% de aciertos.

[1] Solamente se muestran los 30 primeros features en términos de importancia.

## Re-Entrenamiento

Para reentrenar se seleccionaron las treinta variables con más peso y posteriormente se agregaron los vectores resultantes de aplicar K-Means.
El resultado giró entorno a la predicción anterior. La asignación de pesos fue la siguiente

Importancias de los features para el entrenamiento del algoritmo




El resultado no tuvo mejora solamente sirvió para casi igualar lo anterior pero con menos datos.

# Conclusiones

| # | Tema | Conclusion |
|---|---|---|
| 1 | Cantidad de Vistas | Podemos concluir que la cantidad de veces que un usuario ve un aviso puede ser determinante para la postulación.<br>De igual forma que la cantidad total de vistas de un aviso o de un postulante. Esto se puede deber a dos razones:<br>❏ Aquellas personas que están buscando trabajo es más probable que vean una mayor cantidad de avisos<br>❏ Aquellos avisos sponsoreados o de empresas más conocidas es más probable que tengan mayor cantidad de vistas |
| 2 | Cantidad de Postulaciones | De igual forma que la cantidad de vistas es importante, también lo es la cantidad de postulaciones o un aviso o de un postulante.<br>Esto se puede deber a dos razones:<br>❏ Aquellas personas que están buscando trabajo es más probable que tengan mayor cantidad de postulaciones<br>❏ Aquellos avisos sponsoreados o de empresas más conocidas es más probable que tengan mayor cantidad de postulantes |
| 3 | Edad del Postulante | El algoritmo parece indicar que la edad del postulante tiene un rol importante al momento de decidir por una postulación. Esto se puede deber a múltiples razones:<br>❏ El espectro de edades es muy variado, por lo que puede ser realmente útil al intentar identificar diferencias.<br>❏ Las áreas de trabajo van cambiando a lo largo del tiempo, por lo que personas con distintas edades pueden tener capacidades/preferencias distintas |
| 4 | Títulos y Descripciones | Los textos guardan información útil y las representaciones en pocas dimensiones tanto por el algoritmo LSI como el clustering tienen algún tipo de relevancia para el algoritmo .Incluso la importancia de las otras variables categóricas como el nivel académico, la zona de trabajo y el sexo es mayor cuando se incluyen los datos procesados de los títulos. Además estos algoritmos son flexibles a la incorporación de datos nuevos, en el caso de LSI simplemente hay que escribir el título de el aviso a agregar como combinación lineal de los vectores de la base reducida. |
| 5 | XGBOOST vs RF | El algoritmo XGBOOST es más óptimo, rápido, consume menos recursos y predice mejor.<br>Pero a diferencia de RF necesita de la normalización o la unificación de escalas de todas las variables continuas. |
| | Áreas de trabajo | Las áreas son muchas (185), sería interesante poder generar una relación cuantitativa o distancia entre estas. Pero en nuestra opinión excede este trabajo y necesita conocimientos concretos del Dominio |

| | | |
|---|---|---|
| | | (mercado laboral/estudio del trabajo). |

# Bibliografía


- https://xgboost.readthedocs.io/en/latest/
- http://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html#sphx-glr-auto-examples-preprocessing-plot-all-scaling-py
- https://pandas.pydata.org/
- http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- http://scikit-learn.org/stable/modules/cross_validation.html
- http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

# Anexo

# Pruebas realizadas

## Prueba #1

**Objetivo**: Primer submit a Kaggle
**Algoritmo**: Naive Bayes
**Features**: Edad, nivel académico, su estado, tipo de trabajo, zona de trabajo, rol.
**Resultado**: Consultamos con los profesores que algoritmos nos convendría usar y descartaron Naive. Dato importante, hay que saber cuándo dejar de usar un algoritmo.

## Prueba #2

**Objetivo**: Primer submit a Kaggle
**Algoritmo**: Random Forest
**Features**: Cantidad de vistas totales de un postulante a un aviso, edad, nivel académico, su estado, tipo de trabajo, zona de trabajo, rol.
**Porcentaje**: entre 45% y 50%
**Resultado**: Seguimos intentando con el mismo algoritmo y agregamos más features.

## Prueba #3

**Objetivo**: Pasar los 80%
**Tareas**: Agregar más features para subir el porcentaje
**Algoritmo**: Random Forest
**Features**: Los mismos features más la información de los títulos.
**Porcentaje**: entre 50% y 55%
**Resultado**: Seguimos intentando con el mismo algoritmo y agregamos más features.

## Prueba #4

**Objetivo**: Pasar los 80%
**Tareas**: Dividir el set de entrenamiento en 3 partes y re-entrenar.
**Algoritmo**: Random Forest
**Features**: Los mismos features que la prueba 3.
**Porcentaje**: entre 50% y 55%

**Resultado**: Luego de buscar información nos dimos cuenta que con Random Forest no podemos reentrenar.

## Prueba #5

**Objetivo**: Pasar los 80%
**Tareas**: Cambiamos los valores de la columna de títulos en varias columnas con 0 y 1 .
**Algoritmo**: Random Forest
**Features**: Los diferentes extractos de los títulos en columnas.
**Porcentaje**: entre 60% y 65%
**Resultado**: Vimos que avanzamos poco con Random Forest e intentamos probar con otro algoritmo.

## Prueba #6

**Objetivo**: Pasar los 80%
**Tareas**: Probamos con un nuevo algoritmo.
**Algoritmo**: XGBoost
**Features**: Cantidad de vistas totales de un postulante a un aviso.
Cantidad de vistas en los últimos días.
Cantidad de vistas totales de un postulante en general.
Cantidad de vistas de un postulante en los últimos días.
Cantidad de vistas totales que tiene un aviso.
Cantidad de postulaciones que realiza cada postulante
Edad, nivel académico, su estado, tipo de trabajo, zona de trabajo, tipo/área de trabajo
**Porcentaje**: entre 80% y 85%
**Resultado**: Obtuvimos grandes resultados. Decidimos usar XGBoost ya que nos daba mejores resultados que Random Forest.

## Prueba #7

**Objetivo**: Entrar entre los 3 primeros (Superar los 90%)
**Tareas**: Agregamos más features a la prueba 6.
**Algoritmo**: XGBoost
**Features**: Los mismos que en la prueba 6, más procesamiento de detalles y títulos.
**Porcentaje**: entre 90% y 95%
**Resultado**: Obtuvimos grandes resultados. Decidimos usar XGBoost ya que nos daba mejores resultados que Random Forest.

# Prueba #8

**Objetivo**: Entrar entre los 3 primeros (Superar los %90)
**Tareas**: Probamos con un nuevo algoritmo.
**Algoritmo**: XGBoost
**Features**: Igual que en la prueba 7.
**Porcentaje**: 98%
**Resultado**: Obtuvimos uno de los mejores resultados utilizando predict_proba, o sea, enviando en el submit la probabilidad y no la predicción.