FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2
Predicción de postulaciones laborales

[75.06] Organizacion de Datos
Primer cuatrimestre de 2018
Grupo: "Scooby Data Doo"

Alumno:	**Número de padrón**
CONDORI,Guillermo	98688
JUSTO NARCIZO, Edson	97775
PICCO, Martín	99289
SPASIUK, Kevin	99849

Índice

1. Introducción

El presente informe reune la documentación de la solución del segundo trabajo práctico de la materia Organización de Datos que consiste en generar modelos de predicción en base a un registro histórico de avisos de busquedas laborales para poder determinar que tan probable es que determinado usuario se postule a un aviso de búsqueda laboral de ciertas características.

2. Archivos trabajados

3. Transformaciones y limpieza de los datos existentes

3.1. Reemplazo de valores nulos

Las primeras medidas a tomar luego de cargar cada archivo de datos era realizar una limpieza de los features que contengan valores no validos, rellenándolos con algún valor considerado "no válido", como lo podría ser un -1 o un string que diga "Ninguno".

3.2. Codificación de variables categóricas

Los primeros datos que requirieron tratamiento fueron todas las features que representen variables categóricas. Dado que los algoritmos de ML trabajan con números, no se pueden usar datos de tipo string, etc. Por lo que antes de entrenar el modelo hay que realizar un preprocesamiento de esta clase de features.

Para la conversión se usó la clase LabelEncoder provista por el modulo preprocessing de sklearn. Dicha clase procesa un arreglo de valores, generando un número entero para cada categoría.

Features codificadas vinculadas al postulante:

- sexo
- nombre_edu (nombre del tipo de educacion)
- estado_edu (finalizado, en curso, etc.)

Features codificadas vinculadas a los avisos:

- nombre_zona
- ciudad
- tipo_de_trabajo
- nivel_laboral
- nombre_area
- denominacion_empresa

3.3. Limpieza de textos

Observaciones iniciales mostraron que los textos (título y descripción) de los avisos contenían código HTML y caractéres que podrían dificultar el procesamiento. Las medidas tomadas fueron limpiar los tags HTML a través de una expresión regular, remover los caracteres especiales (\n, \t, _s) y reemplazar aquellas vocales con tilde por su contraparte sin el mismo.

3.4. Limpieza de duplicados de Educación

Los archivos que contienen los datos sobre la educación de los postulantes presentan *todos* los niveles de educación del postulante. Como solo nos interesa el mayor nivel de educación logrado, al unir con los datos de los postulantes hacemos un filtrado y solo nos quedamos con el mayor de las educaciones. Aprovechando que la feature fue codificada de en el rango 0-6, con un ordenamiento del par (id_postulante, nivel_educacion) y remover duplicados es suficiente.

4. Feature Engineering

4.1. Features integradas

En esta sección se abarcan aquellas features generadas que ayudaron a mejorar la precisión de los resultados

4.1.1. Aviso Online

Se generó una columna de tipo booleano a partir de la cruza de los datos de avisos con aquellos indicados en el archivo *avisos\online*. Aquellos casos que resultan nulos de la cruza se les asigna un 0 en lugar de 1.

4.1.2. Aviso Visto

Un concepto que resulta de los más intuitivos a partir de mirar los datos. ¿Vió el postulante el aviso?

La columna se genera partiendo de agregar un feature *visto* en 1 a todos los registros cargados del archivo *vistas*. Al momento de hacer merge con los datos de postulantes y avisos, lo hacemos de tipo *left*, de manera que todas aquellas combinaciones de id de postulante y aviso que no están en la tabla de vistas van a tener un valor nulo en el campo *visto*, mientras que todos aquellas presentes tendrán el valor 1. A continuación se asigna un 0 a todos aquellos registros que tengan el feature en valor nulo.

A pesar de la simplicidad de la feature, fue uno de los cambios que mejoró la precisión del modelo en gran medida. Previo a la adición de la columna se obtenían resultados rondando los 80 %, mientras que luego de agregarla, los submits a la competencia pasaron a tener un 92 % de precisión.

4.1.3. Análisis *naive* de los títulos y descripciones

Como forma de sacar provecho a los campos de título y descripción se decidió hacer una búsqueda *naive* de términos asociados a algún tema o concepto y tratar de generar columnas a partir de ello.

Se definió un diccionario con el tópico (por ejemplo: se ofrece obra social, requiere secundario completo, etc.) como clave y una lista de palabras o términos como valor. El procesamiento consiste simplemente en verificar si cualquier término se encuentra presente en el texto. En caso de estarlo se asigna 1 a la columna, mientras que en el caso contrario un 0.

La mejora que presenta la feature no es muy notable, muy probablemente por el hecho de que los "escaneos" terminan cubriendo una parte chica del datasets

4.1.4. Matriz de frecuencia/relevancia de términos en descripciones

Uno de los mayores desafíos de la modelación es el procesamiento de los títulos y descripciones de los avisos. Dado que se tratan de textos, es necesario llevarlos a una representación distinta para poder aprovecharlos al trabajar con los algoritmos de ML.

El enfoque tomado fue generar una matriz que indique la frecuencia/relevancia de cada término para cada documento o texto presente. La intuición detrás de la decisión es lograr representar de

alguna manera los avisos y mantener las similitudes entre ellos, dado que aquellos avisos "similares" tendrían descripciones que compartiesen términos y/o estructura. De haberse postulado un usuario para un aviso, se pensaría de una mayor probabilidad de que el usuario se postulase a avisos que son parecidos. Otros enfoques posibles hubiesen podido ser la distancia de todos los avisos contra todos, utilizando alguna métrica como la de Jaccard.

El proceso para generar esta matriz "tf-idf" implica primero un pre-procesamiento de los textos. Dado que los datos crudos son HTML, hay que remover todos los tags y caracteres especiales, incluyendo el reemplazo de vocales acentuadas por su contrapartida sin tilde[1]. Posterior a esto, se realiza una "tokenización" de las palabras, extrayendo cada término de cada texto[2]. Finalmente se construye la matriz de relevancias sobre los tokens extraídos previamente[3].

Dada la alta cardinalidad de datos que maneja la matriz, el uso directo no iba a ser posible por una simple cuestión de escacez de recursos. Para ello se decide recurrir a algún método de reducción de dimensiones. Debido a la simplicidad y familiaridad se decidió utilizar SVD. A la vez, para aprovechar el formato disperso con el que fue generada la matriz, se usó la variante `SVDS` provista por `numpy`.

En las primeras instancias de prueba se decidió usar las primeras 6 columnas/autovalores de la descomposición. La adición al set de entrenamiento resultó en una mejora de las predicciones, particularmente en el caso de Random Forest. Luego de una rápida grid search para encontrar hiper-párametros, los submits a la competencia llegaron a tener un 95

Para entender un poco mejor la situación, se hizo un gráfico de los primeros 50 autovalores obtenidos en la reducción para poder visualizar la "energía" presente en la matriz. La figura 1 muestra de forma gráfica los autovalores mencionados.

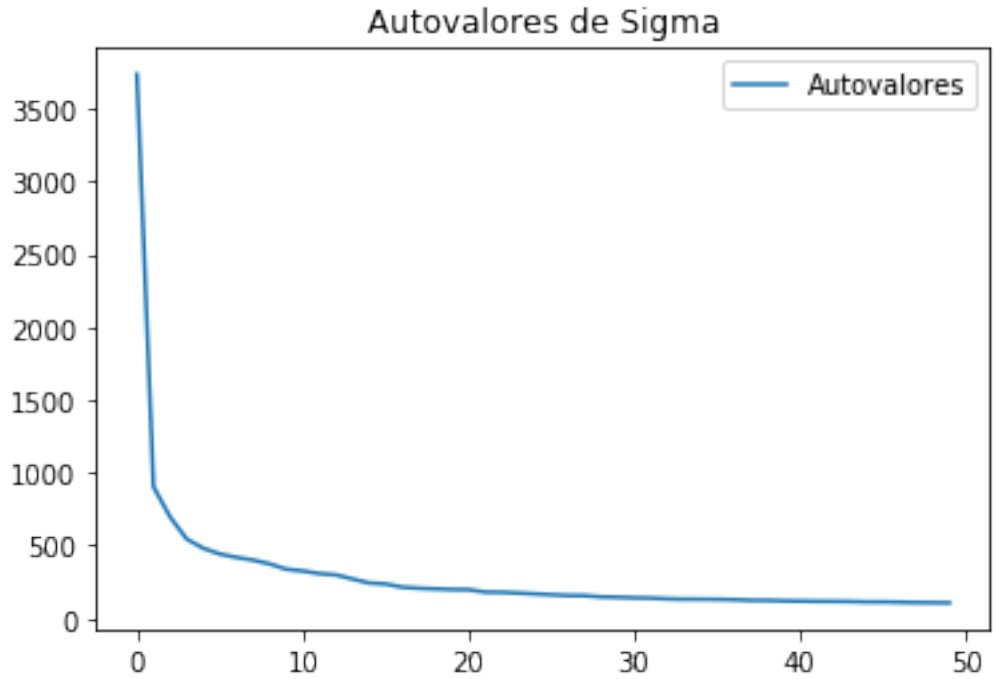


Figura 1: Primeros 50 autovalores de la matriz "tf-idf" a partir de los datos de los títulos y avisos

Como muestra la figura, el "codo" se da en los primeros autovalores, dando una fuerte caida para el 4to/5to y tomando un ritmo de disminución medianamente estable.

[1] La limpieza se realizó usando la clase `HtmlParser` provista por el modulo `html.parser` de python
[2] Se usó la libreria `nltk` (https://www.nltk.org/)
[3] La construcción de la matriz se realisó con la librería `gensim` https://radimrehurek.com/gensim/

Por cuestiones de recursos, para los submits se tomaron las primeras 10 a 15 columnas/autovalores.

4.1.5. Clustering sobre features generados a partir de términos en descripciones

Una idea surgida fue aplicar un algoritmo de clustering sobre los datos obtenidos de los títulos y descripciones de los avisos. Se aplicó K-Means con distintos valores arbitrarios para K, pero se terminó optando por la $\sqrt{n}$, con n la cantidad de avisos.

A partir del cambio se percibieron leves mejoras de los resultados para el algoritmo XGBoost pero una degradación para Random Forest, motivo por el que la feature se usa solo para el primero.

4.2. Features descartadas

Varios intentos de feature engineering resultaron en una degradación de los resultados

4.2.1. Minhash sobre título/descripción del aviso

Entre los primeros intentos de aprovechar los títulos y descripciones se trató de aplicar minhashes. Habiendo limpiado los textos, se los dividía en n-gramas (se utilizó n=5) y se aplicaba Jenkins hashing sobre cada n-grama, tomando el mínimo de todos estos hashes como la representación. Los dos features generados no resultaron ser de utilidad, dado que se apreció una degradación en los submits a la competencia.

En retrospectiva, se debería haber usado un enfoque con más dimensiones y no la reducción a una sola. La aplicación de alguna herramienta cómo Hashing Trick, hasheando a k nuevas features, podría haber dado mejores resultados.

4.2.2. Features vinculadas al tiempo

Se trató de generar columnas independientes para mes/día/hora de las postulaciones a partir del dato existente *fecha_postulacion*. Sin embargo, nos encontramos con el problema de que no existe una fecha de postulación para las "no postulaciones". Se intentó reemplazar los valores faltantes con -1 para indicar un valor no existente. Aún así, posteriores corridas presentaron una degradación considerable.

4.2.3. Distancia temporal entre postulación y vista

Una interrogante que surgió fue *¿Cuánto tiempo pasa entre que un usuario ve un aviso y se postula?*. La feature intentaría tener en cuenta la diferencia de tiempo a la hora del entrenamiento,

El enfoque tomado fue crear una feature a partir de la resta de la fecha de postulación y la fecha de la vista, siendo ambos valores previamente convertidos a segundos desde una fecha definida.

Sin embargo, los problemas que se presentaron fueron los mismos al tratar de usar la fecha de la postulación. El completado de valores nulos para los casos de "no postulaciones" degeneraban la precisión de las predicciones enormemente.

4.2.4. Promedio de postulaciones/vistas por postulante

Se buscó generar una feature con el promedio de postulaciones o vistas de un postulante. Al igual que en otros casos, la adición empeoraba los resultados predecidos.

4.2.5. Cantidad de postulaciones por aviso

Siguiendo la hipótesis de que un aviso con muchas postulaciones es "popular", se trató de traducirlo a una feature. Sin embargo, no trajo una mejoría aparente en los resultados.

5. Algoritmos

5.1. Primeros intentos

En las primeras instancias del trabajo se hicieron pruebas tanto locales como submits a la competencia usando los siguientes algoritmos:

- Decision Tree
- Random Forest
- XGBoost (Gradient Boosting)
- AdaBoost
- SVM (variante RBF)
- MLP (Multi-Layer Perceptron)

Todos los algoritmos salvo XGBoost y Random Forest daban una precisión de aproximadamente 50 % (Decision Tree y AdaBoost), tardaban demasiado tiempo en entrenarse (SVM, MLP) o al reajustar los parámetros para entrenar en un tiempo deseable daban muy malos resultados (MLP), mientras que los primeros daban resultados de 70 % y 60 %, respectivamente. Debido a esto, se decidió profundizar sobre estos dos algoritmos.

Dado que con cada de feature agregada los resultados cambiaban, a lo largo del trabajo se fue haciendo un tuning de los hiper-parámetros a prueba y error, y en algunos casos un grid-search para determinar cuál sería una combinación apropiada. La dificultad de hacer un grid-search para cada cambio era el costo de tiempo y que a lo mejor la precisión obtenida en el entorno de prueba no se asemejaba a la obtenida por los submits a la competencia.

5.2. Algoritmo con mejor resultado

Teniendo en cuenta cada algoritmo de forma singular, Random Forest fue el que mejor resultados dió comparado a XGBoost, siendo aproximadamente 95.5 % contra 93 %.

Sin embargo, se llegaron a resutlados de aproximadamente 96.4 % a través de una ponderación de las predicciones de ambos algoritmos. A través de pesos asignados a cada algoritmo, por lo general de los ordenes de 90 % para XGBoost y 10 % para Random Forest, se recalcula cada una de las predicciones y se genera un nuevo set que es el envíado a la competencia.

6. Conclusiones

6.1. Progresión de los resultados

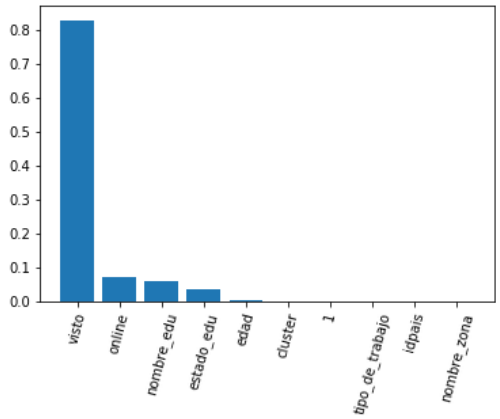
Siguiendo un orden cronológico, la progresión de los resultados se puede sumarizar de la siguiente manera:

1. Datos originales de los archivos (descartando Título y Descripción), aplicando codificación de las variables categóricas. Ninguna feature adicional.
2. Feature *online*
3. Feature *visto*
4. Features surgidas de análisis naive de los títulos y descripciones
5. Features obtenidas a partir de la reducción de dimensiones por descomposición SVD de la matriz de tf-idf generada a partir de los títulos y descripciones
6. Clusterización de las features obtenidas a partir de títulos y descripciones
7. Ponderación de resultados de los algoritmos XGBoost y RandomForest

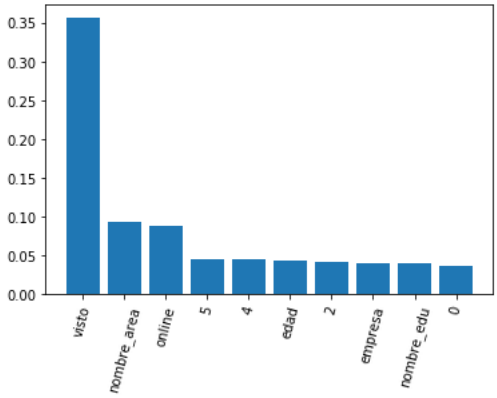
6.2. Reflexiones sobre el proceso

Encontramos que no es suficiente usar solo los datos originales y apoyarse en la búsqueda de los super-parámetros óptimos para cada algoritmo. De esta manera uno puede llegar hasta cierto resultado de precisión y no lograr más. Si queremos mejorar las predicciones requerimos ampliar el modelo a través de Feature Engineering, de manera que los algoritmos de clasificación tengan más información disponible para manejar y por ende mejorar los resultados. Sin embargo, encontramos que no cualquier feature generado mejora los resultados. Como se explicó en la sección de Features, varios de los intentos de ampliar el modelo acabaron resultando en peores resultados.

Los primeros intentos fueron generar features de valores no categóricos, cómo lo son promedios de vistas/postulaciones por postulante o la cantidad de postulaciones por aviso. Ante el fracaso de estos intentos se tomó el enfoque de generar features más estrechamente relacionados a los datos originales y que sean de tipo categórico, cómo lo eran los features *online* y *visto*. Ambas ampliaciones resultaron en una mejora enorme de los resultados, lo que indujo a pensar que este tipo de features eran las que sí iban a generar mejoras notables mientras que sus contrapartes no categóricas no lo harían. Como se muestra en las figuras (2a) y (2b), las features *online* y *visto* acaparan la importancia en el entrenamiento de los algoritmos



(a) Las 10 features de mayor importancia en XGBoost



(b) Las 10 features de mayor importancia en Random Forest

Sin embargo, esta concepción se vio rota con la parte más experimental del trabajo: el tratamiento de los títulos y descripciones. En un principio no se depositaba mucha esperanza en que la conversión a una matriz de "relevancia" de cada termino en cada documento y una posterior reducción de dimensiones fuese a mejorar los resultados en la manera que se vió, por más que hallan sido en unos pocos puntos. Estos resultados positivos fueron una de las mayores sorpresas y trajeron aparejada la duda de si los intentos previos de generar features numéricos fueron encarados de forma correcta, o si con algún tratamiento adicional hubiesen podido mejorar los resultados en lugar de empeorarlos.

Estas features "experimentales" resultaron también ser una buena ocasión para observar el poder de la herramienta de reducción de dimensiones. La idea de reducir una cardinalidad grande a unas pocas columnas que representan los "conceptos" principales de la información y poder trabajar con ellos y aún así poder obtener resultados coherentes. Poder utilizar la herramienta en un caso concreto, por más que sea de índole académica, abre un poco las puertas a entender cómo puede utilizarse en situaciones más reales.

Sumarizando, encontramos que el proceso de feature engineering abarca un poco de análisis de los datos y ver qué patrones, a veces "ocultos", se pueden identificar y sirvan de puntapié para la ideación de nuevas features. Otras veces el proceso implica pasar a algo más experimental y aplicar técnicas y conceptos para poder llevar los datos a otras representaciones y trabajar desde allí. Como se ha mencionado, en el trabajo se han tenido oportunidad de aplicar ambos estilos y

se han observado casos de fracaso y éxito para cada uno.

A manera de cierre, podemos ilustrar las sensaciones del grupo a lo largo del trabajo con la siguiente tira del cómic XKCD:

Figura 3: XKCD 1838 (https://xkcd.com/1838/)