75.06
Organización de Datos

TP 2

1er cuatrimestre 2018

Profesor titular: Argerich, Luis

Alumnos: Bollero, Carlos - 93542
Botalla, Tomás - 96356
Mangiafave, Bruno - 96420

Índice

Github: https://github.com/tbotalla/Datos1Q2018

Kaggle: https://www.kaggle.com/tbotalla/datos1q2018

Enunciado

El objetivo del presente trabajo práctico, es utilizar las herramientas aprendidas durante el curso de Análisis de datos , junto con las de Machine Learning para realizar predicciones de un set de datos real provisto por la empresa Navent.
Se intenta predecir para un determinado aviso, y un determinado usuario, cuál es la probabilidad de que el usuario postule a ese aviso.

La competencia se desarolla en la plataforma Kaggle, y se utilizaron una serie de datos provistos por la empresa Navent, en los siguientes links:
https://drive.google.com/file/d/1K4uRag5nmGtfuvzyJV9RL_73lzsh_iTO/view?usp=sharing
https://drive.google.com/file/d/1Pudf2TrUn_hfd8Dks4UTTJLf9ZdnGUd_/view?usp=sharing
https://drive.google.com/file/d/1ic7saV_7q-vpaUBkrta83nHRDn2d5qMb/view?usp=sharing
https://drive.google.com/file/d/1K7E7gxh6O24BHCGShXKM9cy4cRas7WeF/view?usp=sharing
Los datos vienen dados en forma de CSV, como se describen a continuacion:

- postulantes_educacion.csv
- postulantes_genero_y_edad.csv
- vistas.csv
- postulaciones.csv
- avisos_online.csv
- avisos_detalles.csv

La idea fue probar varios algoritmos de Machine Learning para predecir las probabilidades de un usuario a postularse a cierto aviso. A medida que se fueron realizando las distintas pruebas, se realizó el correspondiente submit en Kaggle para evaluar el resultado de las mismas.

El link a la competencia es https://www.kaggle.com/t/3917603da7044a8ba47cfc606a94e235

Preprocesamiento

Se adoptaron distintas estrategias para el procesamiento de cada set de datos según la información contenida en cada uno.

En todos los casos se mergearon los datos de los sets previos al 15 de marzo con los de los días posteriores, se eliminaron datos repetidos aplicando distintos criterios, por ejemplo el último estudio alcanzado en el set de Educacion, en caso que se repitiera un idPostulante y columnas donde los datos no aportan información relevante.

- ## Set de Postulaciones

	idaviso	idpostulante	fechapostulacion
1882925	1112293594	8MrZwpO	2018-02-15 21:37:20

Se realizó un análisis de cuantas postulaciones recibió cada aviso y fue agregado como una variable categórica en nuestro set final de entrenamiento, a su vez fue usado a la hora de mergear el set final de entrenamiento como otra variable categórica booleana llamada "*sepostulo*".

Se desestimó la columna *fechapostulacion* ya que según nuestro criterio no aportaba datos relevantes.

- ## Set de Vistas

	idaviso	fechavista	idpostulante
6232567	1112428133	2018-03-28T17:36:23.566-0400	RzMd8kR

Se realizó un análisis de cuántas visitas recibio cada aviso y tambien fue agregado como una variable categórica en nuestro set final de entrenamiento.

De la misma manera que en el set de postulaciones, se desestimó la columna *fechavista* ya que no consideramos como relevantes los datos aportados.

- ## Set de Edad y género

	idpostulante	fechanacimiento	sexo
7934	Zrp9x5	1976-12-22	MASC

Analizando las columnas del DataFrame notamos como interesante poder determinar la edad de cada usuario procesando la fecha de nacimiento mediante la función to_datetime() y calculando la diferencia con el año actual. A la columna *sexo* se la transformó en una columna booleana mapeando el valor 0 a *FEM* y 1 a *MASC*. Ambas de estas columnas consideramos importantes para formar parte de nuestro set de entrenamiento final.

- ## Set de Educación

	idpostulante	nivel	estado
167076	KBdNZOq	Secundario	Graduado

En primera instancia se mergearon Set de Educacion y Set de Edad y Género, en función de crear un perfil del idPostulante, en los casos que se requiriera información faltante de algún Postulante se procedió a utilizar la función fillna(), adoptando el criterio de llenar los NaN con el valor más frecuente para esa variable en el Dataset. El nivel y el estado de la educación formaron parte de nuestras columnas categóricas.

- ## Set de Avisos

	idaviso	titulo	descripcion	nombre_zona	tipo_de_trabajo	nivel_laboral	nombre_area	denominacion_empresa
3050	1112100565	Administrativo E-Commerce	<p>Adecco Office está especia...	Gran Buenos Aires	Full-time	Senior / Semi-Senior	E-commerce	Adecco - Región NORTE & OESTE GBA

En este caso, este set de datos presentaba en principio una cantidad de información mayor a extraer. Por un lado se presentan diversas columnas categóricas como **nombre_zona**, **tipo_de_trabajo**, y **nivel_laboral**. Analizando más detenidamente pudimos notar que la columna **nombre_zona** no proveía datos relevantes puesto que una gran mayoría de los datos tenían un mismo valor, por este motivo decidimos descartar la columna. Por otro lado pensamos que dos columnas en particular podrían resultar aún más interesantes para analizar.

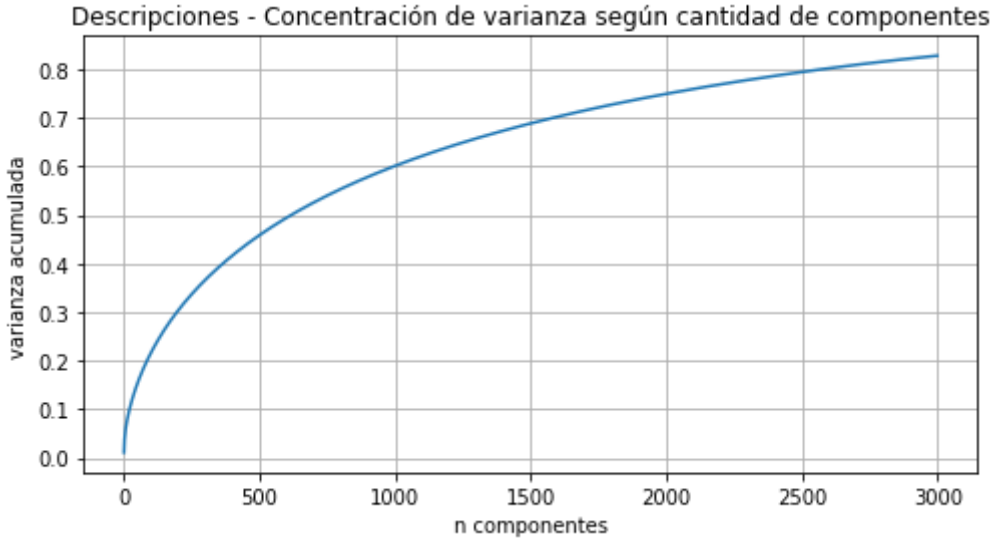
Tomando como referencia un análisis realizado previamente para la primera parte del trabajo práctico, donde determinamos que la parte que resulta más atractiva de un aviso está en su *descripción* y *título* estudiamos cuáles eran las palabra con mayor cantidad de repeticiones dentro de estas columnas de los avisos. Para esta segunda parte del trabajo práctico decidimos que no íbamos simplemente a contar cuál era la palabra más repetida sino que resultaba más interesante poder agrupar tanto *títulos* como *descripciones* según algún criterio y así poder determinar una relación entre usuarios y los títulos y descripciones agrupadas. Tomamos la decisión

de aplicar un algoritmo de clustering para conseguir que ese agrupamiento esté asociado a la distancia entre textos.

Para las descripciones de los avisos aplicamos como primer paso una vectorización de cada descripción contando repeticiones de cada término sin tener en cuenta los palabras que nada aportan a la temática de un texto (stop words) y aplicando además IDF a fin de minimizar el impacto de los términos que se repiten a lo largo de todos los textos.

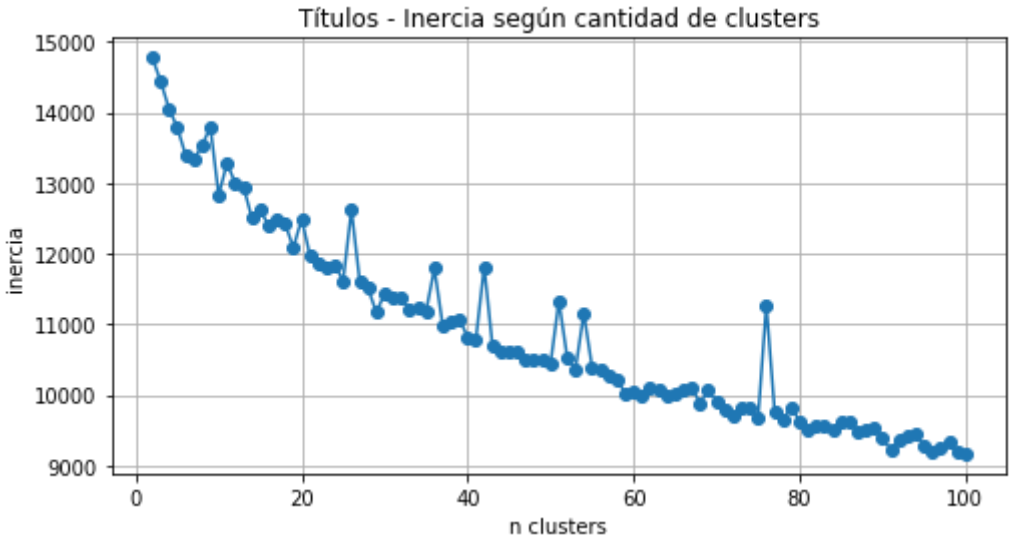
En el caso de los títulos, como la cantidad de palabras es más acotada y el impacto de las palabras repetidas a los largo de todos los textos es mucho menor decidimos aplicar el mismo algoritmo pero descartando la aplicación de IDF.

Como resultado de aplicar la vectorización obtuvimos matrices de tamaño demasiado grande que no permitían aplicarles directamente algún algoritmo de clustering sin tener inconvenientes de memoria o del tipo de dato esperado por el algoritmo. Por esto como siguiente paso decidimos como mejor opción aplicar una reducción de dimensiones a las matrices obtenidas quedándonos con las columnas cuyos componentes principales acumulaban la mayoría de la varianza de los datos.



Luego de esto, en ambos casos optamos por aplicar *KMeans* como algoritmo de clustering, específicamente la versión *MiniBatchKMeans* (*http://www.eecs.tufts.edu/~dsculley/papers/fastkmeans.pdf*) aprovechando que reduce el tiempo computacional requerido y que los resultados obtenidos son prácticamente idénticos a los de su versión regular.

Para determinar la cantidad de cluster ideal aplicamos el “método del codo” probando distintos números de clusters y observando a partir de que valores el algoritmo deja de aportar información.



Ya con la cantidad de clusters determinada, aplicamos el KMeans donde a cada detalle le asigna un cluster. A la columna resultado la transformamos en columnas booleanas (una por cada cluster).

Para las columnas categóricas como nivel_laboral y tipo_de_trabajo aplicamos one hot encoding. Para cada categoría posible de esos features se agregó una nueva columna booleana indicando si está presente o no.

El feature nombre_area tenía 186 categorías con una distribución poco uniforme donde, pocas áreas concentraban la mayor parte de los datos. El criterio adoptado fue aplicar one hot encoding pero solo de las 20 áreas principales, es decir agregar 20 features booleanos indicando pertenencia a cada área.

Además, juntando con los datos de vistas y postulaciones agregamos dos columnas nuevas que contabilizan la cantidad de vistas y de postulaciones que tiene cada aviso. Para los casos que no se tenían datos se dejaron en 0 estas dos columnas.

- ## Set de Avisos online

Consideramos que este set de datos no aportaba ningún tipo de información valiosa que pueda tener algún peso en nuestra predicción, por lo que decidimo no utilizarlo.

Preparación de datos

Para entrenar y predecir lo que hicimos fue unir los datos de los postulantes con los de los avisos. Por un lado para el set de entrenamiento la forma de unir estos dos fue mediante el set de postulaciones, mientras que por el otro lado para el set de test de Kaggle unimos los avisos con los postulantes mediante el set de la entrega test_final_100k.csv ya que tiene los pares (idAviso, idPostulante).

Al juntar los datos de avisos y postulantes observamos que algunos datos estaban nulos, por lo tanto los completamos con el promedio o la moda de dicho atributo para las columnas numéricas y categóricas respectivamente.

Generación de casos negativos

Una vez que juntamos los avisos con los postulantes en el set de entrenamiento lo que tenemos son todos casos “positivos”, es decir casos en que postulantes aplicaron a avisos. Para entrenar a los algoritmos y no caer en overfitting necesitamos registros de postulantes que no apliquen a avisos, a éstos los llamamos “negativos”.

Para generar los casos negativos lo que hicimos fue tomar una muestra aleatoria de avisos y otra de postulantes para luego unirlas y tener los casos deseados. La cantidad de casos negativos debía estar balanceada con la cantidad de positivos para no caer en overfitting ni underfitting.

Una vez generados los casos negativos los juntamos con los casos positivos resultando en aproximadamente 13 millones de registros. Ahora bien, dado que es posible que al generar aleatoriamente los casos negativos algunos sean válidos, lo que hicimos fue que, al concatenar con los positivos en caso de que coincidiera el par (idAviso, idPostulante) prevaleciera el caso positivo ya que es el que tenemos certeza que es real. Conseguidos los casos negativos pudimos comenzar a probar algoritmos.

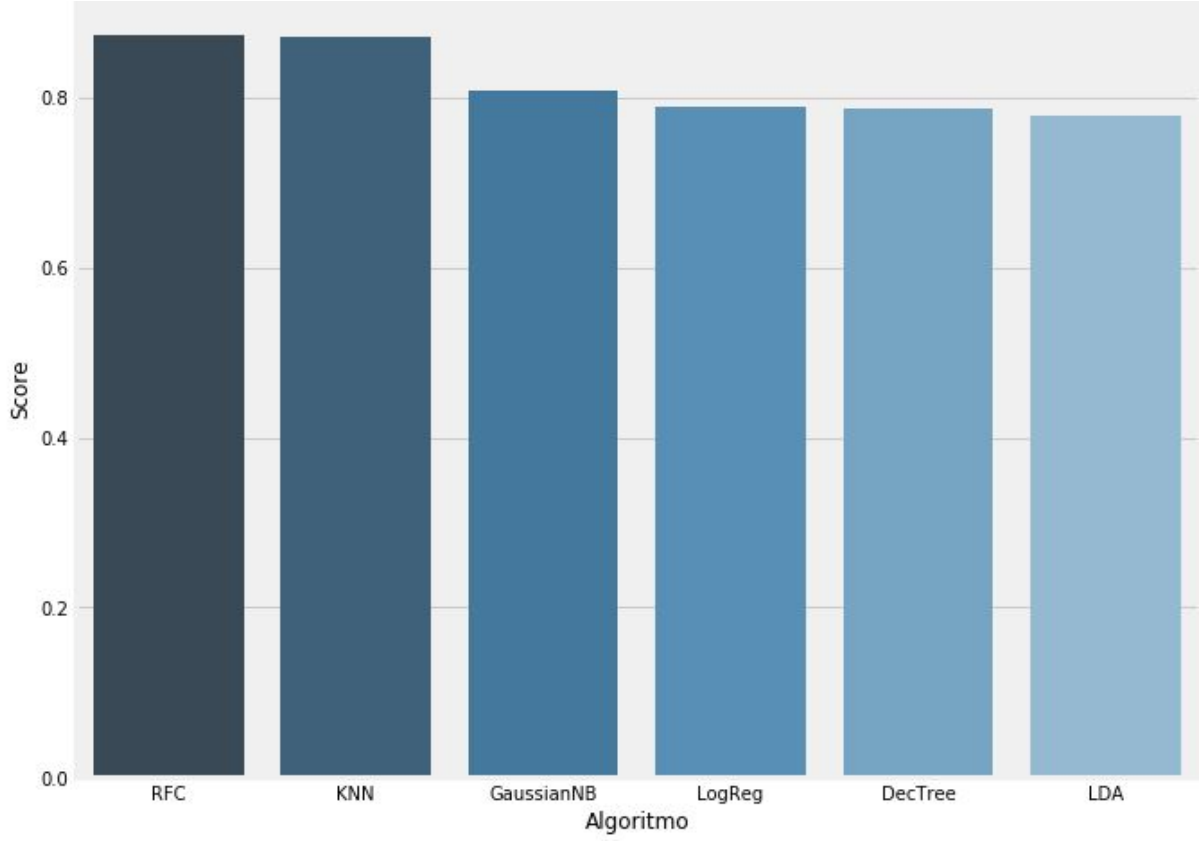
Análisis de algoritmos

Cross-validation

Para la selección del algoritmo a utilizar lo primero que hicimos fue rankear mediante cross-validation ciertos algoritmos de clasificación con los hiper parámetros por defecto. El puntaje usado fue AUROC (Area Under the Receiver Operating Characteristic Curve), el mismo usado para rankear en la competencia de Kaggle.

Probamos 6 algoritmos diferentes: Random Forest, KNN, Gaussian Naive Bayes, Logistic Regression, Decision Tree, y Linear Discriminant Analysis. Se intentó probar otros algoritmos como MultiLayer Perceptron y Support Vector Clustering, pero dada la cantidad de registros, el tiempo para entrenar y predecir era demasiado grande.

Los resultados fueron los siguientes:



En el gráfico se ve que los dos que mejores resultados dieron fueron Random Forest y KNN con un puntaje de 0,872 y 0,870 respectivamente. Gaussian Naive Bayes quedó en tercer lugar con un puntaje de 0,808.

Escalado/Normalizado

Una vez identificados los mejores algoritmos pasamos a escalar/normalizar los datos. Para ver qué era más efectivo realizamos cross-validation con los datos estandarizados por un lado y normalizados por otro. El resultado fue un score ligeramente mejor para los datos escalados, mientras que normalizados empeoraron, por lo tanto trabajamos con los primeros.

Búsqueda de hiperparámetros

Para KNN y Random Forest hicimos una búsqueda de hiperparámetros mediante **Grid Search** junto con cross-validation para evaluarlos. Los hiperparámetros que produjeron un mejor score fueron los que usamos para predecir el resultado final.

Predicción (Pruebas)

En base a los algoritmos que probamos, elegimos para la predicción final a **Random Forest**. Algunas de las ventajas de usar este algoritmo son: no necesita escalar los datos ya que el valor de split de los árboles de decisión no dependerá de la magnitud de los features, relativamente rápido de entrenar a diferencia de otros probados (Perceptron por ejemplo), y que implícitamente realiza feature selection lo cual nos sirvió bastante ya que buscar los mejores features tardaba un tiempo muy considerable. Además el algoritmo de random forest es un ensamble, ya que combina los resultados de múltiples árboles de decisión para dar la predicción final.

Para la entrega final entrenamos el algoritmo no con la totalidad de los datos (13 millones aproximadamente), si no con una muestra un poco más chica 8 millones de registros. Ésto tuvimos que hacerlo porque con el set completo teníamos problemas con la memoria para entrenarlo.

El score obtenido mediante cross-validation para el algoritmo final con los mejores hiperparámetros fue de 0,87, pero al hacer el submit a Kaggle obtuvimos un puntaje menor de aproximadamente 0,80. Ésto puede deberse a que el set de Kaggle tenga datos de avisos o postulaciones para las cuales no tenemos datos y por lo tanto no podemos predecir correctamente.

Conclusión

Posibles mejoras:

Una de las posibles mejoras que se puede plantear, es una obviedad, pero sería el hecho de contar con más memoria. Obviamente hacer Hot encoding de cada una de las variables que podemos llegar a considerar categóricas del DataSet, sería inmanejable. Pero al contar con tres columnas que considerábamos bastantes relevantes: Titulo, Descripcion y nombre de Área, había demasiada información como para incluso después de realizar un filtro de StopWords o bien descartar Áreas o keyWords muy poco frecuentes poder labelaziar. Creemos que la solución que optamos de vectorizar la descripción para reducir dimensiones y luego organizarlas en un cluster fue buena, pero con más memoria probablemente hubiera mejorado nuestra predicción.
Otro aspecto que resultó condicionado durante el preprocesamiento por la limitación de la memoria fue la elección de un algoritmo sobre otro, por ejemplo *TruncatedSVD* sobre *PCA* o *MiniBatchKMeans* sobre *KMeans* regular. Como resultado de la vectorización + tfidf se obtenía una matriz dispersa comprimida, tipo de dato que no todo algoritmo soporta. Convertir esa matriz dispersa comprimida en una matriz densa o un array (tipos más aceptados por la mayoría de los algoritmos) resultaban en un consumo excesivo de memoria que derivaban en la interrupción del kernel.

La depuración de los datos resultó clave a la hora de facilitar el armado del modelo, una ventaja de haber realizado anteriormente el TP1 es conocer la características de los datos, por lo que nos fue facil y rapido saber por donde arrancar a limpiar(sobre todo con lo que fue el manejo de la descripciones de los avisos) . Una parte muy importante ya que era un DataSet complejo y muy pesado.

El score que obtuvimos en la competencia no fue el deseado, en el afán de mejorarlo probamos distintos algoritmos y distintas variantes, como ir cambiando los hiperparametros y normalizando los datos. Confiamos en que nuestra performance sea mejor en el Dashboard privado sea por lo menos mejor que la que vimos en nuestro último Submit.

Referencias consultadas

1. Luis Argerich - FIUBA, “*Organización de Datos - Apunte del Curso v2.0”* (24 de Febrero 2018)
2. Scikit-learn, “*Machine Learning in Python”*, *http://scikit-learn.org/*
3. Leo Breiman, Random Forests, *https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf*