Contenido

Introducción

El siguiente informe fue realizado en el contexto de la materia Organización de Datos de la Universidad de Buenos Aires. El objetivo general fue, a través de la prueba de distintos algoritmos del área de Machine Learning, obtener una predicción de postulaciones para el dataset de Navent del sitio zonajobs.

Para esto, el informe estará dividido en dos grandes secciones: en primer lugar, una sección donde se explicará el preprocesamiento realizado sobre los datos y los distintos features que se han podido obtener a través de los mismos. En segunda instancia, habrá una sección donde se detallarán los distintos algoritmos utilizados y los modelos entrenados a partir de los mismos, explicando su desarrollo y mostrando los resultados obtenidos.

Preprocesamiento de los datos

A continuación, se detalla el procesamiento realizado sobre los datos provistos por la cátedra para poder utilizarlos para entrenar nuestros modelos.

En primera instancia se procedió a combinar los múltiples datasets provistos en tres archivos con formato csv bien definidos, concatenando los datos previos al 15 de abril de 2018 junto con los posteriores. Esto se hizo para las vistas, las postulaciones y los postulantes (para estos últimos además se combinaron los distintos archivos con sus distintos features).

Una vez obtenidos los datasets completos, se procedió a eliminar las columnas con una elevada cantidad de valores nulos o incompletos, evitando de esta forma utilizar datos poco representativos. También se eliminaron datos que resultan irrelevantes, como por ejemplo el identificador de país de los avisos, dado que es el mismo para todos los anuncios.

A partir de éstos datasets, procedimos a extraer nuevos features que nos parecieron relevantes para el desarrollo de las predicciones. El general, el criterio para elegir features fue quedarnos con aquellos que nos parecieran de importancia para un posible postulante a la hora de elegir a qué aviso postularse y, también, aquellos que pudieran ayudar a nuestros algoritmos a relacionar a los avisos y postulantes entre sí y así obtener una mejor puntuación.

Una gran cantidad de features fue extraída de las descripciones y los títulos de los anuncios, dado estos proveen un detalle de los requerimientos de los postulantes. Este tipo de información es la que un postulante analiza a la hora de postularse y por lo tanto debería ser un factor determinante a la hora de reconocer los comportamientos de cada candidato.

Features

De los postulantes

Edad: Para las edades tomamos la fecha de nacimiento y lo pasamos a años de edad, en caso de tener la fecha de nacimiento faltante o bien lo rellenamos con 200 (un valor inválido) o con el promedio de las otras edades según el algoritmo utilizado.

Género: Usamos el género de cada postulante o bien de forma categórica o con algún encoding según el algoritmo utilizado.

Nivel educativo: Si tomamos por ejemplo **Secundario**, esta resulta una variable categórica cuya clasificación puede ser “Graduado”, “En curso”, “Abandonado” o “-” si no hay datos. A veces cada una de estas categorías es encodeada con one hot encoding. Con la misma idea hicimos los features **Master, Posgrado, Otro nivel educativo, Doctorado, Universitario y Terciario/Técnico.**

De los avisos

Nombre de la zona: La zona como variable categorica o one hot encoding.

Tipo de trabajo: Como variable categórica o one hot encoding.

Nombre del área del empleo: Dado que se cuentan con un alto número de áreas laborales, optamos por manejarlo como variable categórica, count encoding o mean encoding para evitar incrementar el tamaño del dataset excesivamente.

Nivel laboral: Como variable categorica o one hot encoding.

Denominación de la empresa: Como variable categórica. A primera vista parecería ser irrelevante, sin embargo, probamos empíricamente que puede llegar a mejorar los puntajes.

A continuación, se listan aquellos features extraídos de la descripción de los avisos:

Pide un hombre: El anuncio solicita específicamente un hombre. Más de mil anuncios lo hacen.

Pide una mujer: ¿Pide el anuncio específicamente por una mujer? Casi tres mil anuncios lo hacen.

Pide inglés: Si el anuncio pide manejo del inglés. Consideramos que al ser uno de los idiomas más populares en Argentina, debía tener una importancia significativa. Ocurre en 4597 anuncios.

Pide experiencia: El anuncio pide experiencia previa en algún área. Esto debería influir en aquellos postulantes que no cumplen el requisito. Ocurre en 13580 avisos.

Paquete office: Pide manejo de office. Siendo el paquete de software más solicitado, nos pareció un dato importante. 4037 anuncios lo hacen.

Pide excel: Pide excel específicamente. 10065 anuncios lo hacen.

Liderazgo: Pide habilidades en liderazgo. Notamos que múltiples empresas buscan candidatos con cualidades de líder. 1292 anuncios lo piden.

Secundario: El aviso solicita candidatos graduados del secundario. Siendo el nivel educacional más frecuentemente solicitado, nos pareció importante analizar si ésto era pedido por los avisos. 4892 anuncios lo solicitan.

Técnico: Menciona “técnico” en la descripción. Esto puede indicar que se buscan candidatos con perfiles técnicos o egresados de secundarios técnicos. 4499 anuncios lo hacen.

Universitario: Menciona “Universitario” en la descripción. Esto puede indicar que buscan un estudiante/graduado universitario.

Viajar: Menciona “viajar” en la descripción. Nos pareció interesante dado que puede llegar a ser un factor determinante para quién considera postularse a un trabajo. Una persona muy establecida puede directamente descartar el aviso, mientras que

otra puede estar orientando su búsqueda a trabajos de perfil más internacional.1064 anuncios lo hacen.

Empresa importante o empresa líder: Se trata, a criterio del anunciante, de una empresa importante o empresa líder en algún área. Esto comprende a 6726 anuncios.

Capacitación: Menciona "capacitación" en el anuncio. Si una empresa ofrece capacitación para el puesto, resulta lógico que esto influencie al candidato a la hora de postularse, sobre todo para aquellos que recién comienzan su carrera laboral. 3898 anuncios lo hacen.

Remuneración pretendida: Solicita al postulante que indique la remuneración pretendida. Ocurre en 1683 anuncios.

Graduados: Menciona "Graduados" en su descripción, implicando que excluye a estudiantes del secundario por lo menos. Ocurre en 3968 anuncios.

Estudiantes: Menciona "estudiantes" en su descripción. Al revisar las descripciones observamos que en general se refiere a estudiantes universitarios. Ocurre en 5020 anuncios.

Conocimientos específicos: Pide algún tipo de conocimiento específico. Esto puede ser decisivo para aquellos postulantes que no posean títulos en áreas específicas. 3263 anuncios lo hacen.

Buen ambiente: Menciona que tiene un buen ambiente de trabajo. Un factor determinante a la hora de postularse a un trabajo suele ser, además de sueldo y competencias, el ambiente laboral. Es lógico que, al ofrecer un buen ambiente, aumente la cantidad de postulaciones. Ocurre en 3717 avisos.

Sueldo: Incluye "Sueldo" en la descripción. Podría indicar que el aviso explicita el sueldo del trabajo. Ocurre en 2780 avisos.

Discriminación: Menciona algo acerca de que no discriminan postulantes. Ocurre en 1507 anuncios, siendo la gran mayoría de la empresa Accenture.

Sueldo: Contiene la palabra "Sueldo" en su descripción. Ocurre en 2780 avisos.

Part Time: Ofrece algún indicio de que el empleo es de medio tiempo. Ocurre en 2780 avisos.

Full Time: Ofrece algún indicio de que el empleo es de tiempo completo. Ocurre en 4412 avisos.

Ofrece Crecimiento: Ofrece algún tipo de crecimiento laboral. Otro factor determinante a la hora de elegir postularse, sobre todo para quienes desean hacer carrera en alguna empresa. Ocurre en 3410 avisos.

Ofrece Beneficios: Menciona la palabra "beneficios" en la descripción del aviso, indicando que el mismo explicita los beneficios ofrecidos por la empresa en la contratación. Ocurre en 2964.

Obra Social: Ofrece alguna de las principales obras sociales en la descripción. Ocurre en 1521 avisos.

Multinacional: Incluye frases o palabras que indican que la empresa o el trabajo ofrecido es de carácter internacional. Ocurre en 3659 avisos.

Nacional: Incluye frases o palabras que indican que la empresa o el trabajo ofrecido es de carácter nacional. Ocurre en 5553 avisos

Largo Descripción: Consideramos que el largo de la descripción del aviso es relevante como feature. Un aviso más largo puede ofrecer más información al candidato a la hora de postularse. El largo promedio resulta 1330.7 caracteres.

Supera Largo Promedio: Consideramos relevante si la longitud de la descripción del aviso supera al largo promedio de las descripciones. Ocurre en 9507 avisos.

Requisitos Excluyentes: El aviso solicita algún tipo de requisito excluyente. Esto puede ser decisivo ya que, si un candidato no posee los requisitos excluyentes, en la mayoría de los casos no se postula. Ocurre en 8223 avisos.

Perfil Específico: El aviso incluye "Perfil" en su descripción. Esto puede indicar que se busca un perfil en particular para el trabajo. Aquellos postulantes con el perfil solicitado pueden sentirse más atraídos a postularse. Ocurre en 6274 avisos.

Contratación Inmediata: Incluye “contratación inmediata” en la descripción. Aquellos candidatos con disponibilidad inmediata podrían sentirse inclinados a postularse.

Proactivo: El aviso incluye palabras o frases que indicarían que se busca una persona extrovertida o proactiva para el trabajo. Ocurre en 1991 avisos.

Trabajo en equipo: El aviso incluye frases que indican que el trabajo requeriría trabajo en equipo. Ocurre en 2795

Consideramos que los títulos podían proveer información valiosa para analizar las postulaciones, dado que es lo primero que los postulantes ven. Si bien muchos de ellos ya están presentes en el nombre del área, ciertos algoritmos vuelven muy complicado utilizar el mismo y estos booleanos compensan esa falta de información.

Pide Vendedor: El título incluye la palabra “Vendedor”. Ocurre en 1870 avisos.
Pide Administrativo: El título incluye la palabra “Administrativo”. Ocurre en 1823 avisos.
Pide Técnico: El título incluye la palabra “Técnico”. Ocurre en 1313 avisos.
Pide Analista: El título incluye la palabra “Analista”. Ocurre en 3339 avisos.
Pide Comercial: El título incluye la palabra “Comercial”. Ocurre en 1317 avisos.
Pide Asistente: El título incluye la palabra “Asistente”. Ocurre en 831 avisos.
Pide Ejecutivo: El título incluye la palabra “Ejecutivo”. Ocurre en 726 avisos.
Pide Ingeniero: El título incluye la palabra “Ingeniero”. Ocurre en 620 avisos.
Pide Operario: El título incluye la palabra “Operario”. Ocurre en 925 avisos.
Pide Desarrollador: El título incluye la palabra “Desarrollador”. Ocurre en 549 avisos.
Título en Mayúscula: El título contiene al menos una palabra completamente en mayúscula.

De las postulaciones

Decidimos no extraer features de las fechas de las postulaciones ya que no tenemos ninguna fecha para el set que queremos predecir.

Utilizando datos del set de avisos, podemos obtener los siguientes features del set de postulaciones:

Postulaciones por área: Cuenta la cantidad de postulaciones de un postulante determinado para un área particular. Consideramos que si un candidato se postuló a muchos avisos de una misma área, es más probable que se postule a otro de esa área.
Con la misma idea extrajimos los features **postulaciones por tipo de trabajo** y **postulaciones por nivel laboral**.

De las vistas

Las vistas están altamente correlacionadas con la probabilidad de postularse, cuestión que trataremos más adelante en el informe. Por esto, consideramos que extraer distintos "puntajes" en base a ellas es crucial.
No tuvimos en consideración el tiempo en que ocurrió la visita porque no tenemos los tiempos del set a predecir, pero además quizá ver el aviso antes de postularse es tan importante como verlo después (quizá para verificar el estado del anuncio al que se postuló, para mostrárselo a alguien...).

Lo vió: ¿El postulante vio el anuncio? Es posible que un candidato se postule sin ver con anterioridad el anuncio, como puede ser el caso de recibir una recomendación de un conocido sobre cierto anuncio.

Vistas del postulante al anuncio: Siguiendo el criterio expuesto arriba, consideramos relevante la cantidad de veces que el candidato vió el anuncio antes de postularse.

Puntaje: Es un puntaje que se basa en la cantidad de veces que vio el anuncio un determinado postulante con la cantidad de vistas de ese postulante en todos los anuncios. Este feature escala las vistas en un número de 0 a 1, lo que es bueno en algoritmos donde la escala importa. Además, consideramos que no debería ser igual si un candidato entra a un anuncio o grupo de anuncios en particular y un candidato

que entra indiscriminadamente a buscar anuncios. Como el tipo de búsqueda es distinto, consideramos que este feature es de suma relevancia.

Vistas por área: Cantidad de vistas de un postulante para una determinada área. Al igual que con las postulaciones, consideramos que la cantidad de vistas del candidato a avisos de cierta área indica que su búsqueda estaría localizada a esa área y por lo tanto sería un fuerte indicador de si el mismo se postularía a un aviso de la misma.
Con la misma idea se hacen **vistas por tipo de trabajo** y **vistas por nivel laboral**.

No postulaciones

Para entrenar a nuestros algoritmos de clasificación necesitamos datos de candidatos que no se postularon a avisos y la forma en que generamos estos puede no ser trivial. En todos los medios para generar no postulaciones o bien elegimos postulantes, o bien avisos, o bien ambos al azar. Esto se debe a que los avisos son aproximadamente 30000 y los postulantes 500000 por lo que la probabilidad de que de las $1{,}5.10^{10}$ combinaciones posibles alguna sea una postulación del set de entrenamiento o bien esté en el set de predicción nos pareció escasa. De todos modos filtramos para ambas cosas antes de entrenar los modelos.
Intentamos deducir y hacer algunas suposiciones del set de predicción que nos fue dado, e imitarlo lo más posible en el set de entrenamiento.

Intentamos 5 tipos de no postulaciones:

Cambiando postulantes: En este caso tomamos el set de postulaciones y cambiamos los postulantes por otros al azar. Esto al principio mostraba un buen desempeño pero encontramos alternativas mejores. La intención era que de alguna manera cada aviso tuviera al menos un "vecino" de sí mismo que no se haya postulado.
Cambiando avisos: Usamos el mismo principio de antes pero cambiando los avisos y dejando los postulantes. Esta alternativa no presentó mejoras respecto de lo anterior.
Considerando las vistas: Notamos que hay ciertos datos en el set de predicción con visitas y como conocemos que la mitad del set de predicción son no postulaciones no había manera en que todos los anuncios visitados en este set sean postulaciones, por lo que generamos un set entero de no postulaciones que si habían sido visitadas para mezclar con las mismas no postulaciones de arriba en una proporción muy

chica. Esto empeoro fuertemente los scores o bien no tenía ningún efecto dependiendo la proporción.
Totalmente al azar: Tomamos muestras al azar tanto de las vistas como las postulaciones y las concatenamos. Este es el tipo de no postulaciones que mejor se desempeñó durante todos los entrenamientos.
Basándose en categorías: Intentamos generar no postulaciones al azar pero con áreas de avisos que tengan la misma proporción o parecida a la de el set de postulaciones. Este último no tuvo los mejores resultados, aunque resultaron buenos.

También intentamos excluir totalmente a los postulantes y/o avisos que aparecen en el set de predicción, pero esto empeoró los resultados.

Un punto clave sobre las no-postulaciones es la cantidad utilizada en el set final de datos. Probamos con múltiples proporciones con respecto a las postulaciones. En la gran mayoría de casos las mejores proporciones fueron 0.75 - 1, 1.5 - 1 y 1-1 (no postulaciones - postulaciones).

Encodings Utilizados

El dataset contenía varias variables categóricas, las cuales no son aceptadas por múltiples algoritmos de clasificación como por ejemplo XGBoost o Naive Bayes. Para poder utilizar las mismas, se procedió a codificarlas con los siguientes tipos de codificación:

Count Encoding: Consiste en reemplazar las categorías por la cantidad de apariciones de las mismas en el set de datos. Resultó la codificación más simple de implementar y fue la primera que utilizamos.

One Hot Encoding: Por cada valor de cada categoría, se genera un feature en el dataset con un valor booleano que indica si la entrada poseía o no el valor de la categoría. Esta codificación incrementa mucho el tamaño del dataset, al añadir muchas columnas al mismo. Debido a esto, no pudimos codificar el área laboral, dado que la misma cuenta con 188 posibles valores. De haberlo hecho, el sample utilizado para entrenar los algoritmos hubieran sido muy pequeños, por lo que desestimamos esta opción.

Mean Encoding: Esta codificación consiste en reemplazar las categorías con el promedio de los valores a predecir en la categoría. Esto implica que por cada

categoría, se calcula el promedio entre las postulaciones y no postulaciones y se reemplaza la categoría por ese promedio.
Ésta codificación nos permitió codificar el nombre del área e incluirlo en el set de entrenamiento para algoritmos como XGBoost, lo que permitió romper la barrera del 96.5%.

Algoritmos utilizados

Se realizaron más de 200 corridas de algoritmos, muchas de las cuales fueron descartadas y no sólo no fueron subidas a kaggle sino que muchos de sus notebooks fueron eliminados, por ejemplos en caso de no mejorar puntajes anteriores o de considerar que no realizaban un aporte nuevo. Contamos con más de 100 notebooks en github. Esto vuelve imposible documentar todo, algunos algoritmos como LightGBM, XgBoost y KNN se probaron con muchisimas combinaciones de features, a veces en todos los encodings posibles y con todas las no postulaciones.

Algunos intentos sin Machine Learning

Mandar todos 1s o todos 0s

Esto resultó en un score de 0.5 en kaggle, lo que nos hizo dar cuenta de que la mitad de los datos a predecir eran no postulaciones y la otra mitad postulaciones. Esto es muy útil como métrica extra de la predicción ya que de allí se desprende que el promedio del campo “sepostulo” debe ser cercano a 0.5 al realizar la predicción. Para lograr esto muchas veces cambiamos el balance de la cantidad de no-postulaciones respecto a las postulaciones, ya sea para que los modelos sean más o menos propensos a predecir unos o ceros. Esta herramiento nos ayudó a mejorar mucho los algoritmos entrenados.

Una predicción muy barata

En un punto de la competencia nos preguntamos qué tan importante podía ser la cantidad de visitas. Apenas lo agregamos nuestros puntajes rompieron la barrera del 90% en kaggle pero nos dimos cuenta de hay usuarios que visitan cientos o más de mil anuncios y otros que visitan tan solo uno y ya se postulan. Aquí una gran diferencia: hay usuarios mucho más “visitadores” y hay otros que visitan un anuncio y se postulan al mismo (quiza por alguna recomendacion por mail o un link de alguien que le haya recomendado postularse).
Lo que pretendíamos era que las visitas de usuarios que suelen visitar menos sean mucho más valiosas, entonces agregamos el feature puntaje, pero antes de entrenar

con el mismo nos percatamos de que era un número entre 0 y 1, ¿por qué no podría ser también una probabilidad a postularse? Esto nos daría una buena medida de la importancia del feature que agregamos.
El submit de una predicción que tenga ese feature como probabilidad nos dio un puntaje de **83.9%** lo que hizo considerar muy malos todos los puntajes que fueran resultado de entrenar algún modelo y puntuaron menos que eso. También nos dio una buena medida de que tan predecibles son los datos debido a sus visitas.

Decission Tree

Es el primer algoritmo que probamos con one hot encoding para todas las categóricas, sin usar el nombre del área de empleo y sin considerar nada en las visitas. Los puntajes en esta etapa temprana del trabajo con este algoritmo rondaron para el set de prediccion el 70%. Abandonamos este algoritmo por su simpleza esperando lograr mejores puntajes de otras formas.

Random Forest

Random Forest es un ensamble de decission trees que usa para cada árbol un subsample de datos y features y genera árboles aleatorizados para luego promediarlos todos a la hora de predecir.
El mejor puntaje logrado por random forest fue cercano al 80% sin tener en cuenta la visitas y el área de trabajo, lo que es muy bueno y nos hace pensar que de seguir su entrenamiento con más y mejores features habríamos podido mejorar mucho, pero no mejorando nuestro mejor puntaje.

KNN

Uno de los algoritmos utilizados fue K-Nearest-Neighbors. El algoritmo es bastante simple y consiste básicamente en, para cada uno de los puntos que queremos clasificar, buscar los k vecinos más cercanos, siendo k un hiper-parámetro que se pasa cuando se instancia el clasificador.
A partir de encontrar estos k vecinos más cercanos, es posible predecir la probabilidad para clasificar una clase, evaluando cuántos vecinos pertenecen a la clase que queremos clasificar y dividiendo por el total de los k vecinos. Este es uno de los algoritmos más simples de Machine Learning, pero se decidió de probarlo ya que suele arrojar buenos resultados.

Para aplicar este algoritmo hubo que resolver varios inconvenientes. En primer lugar, hay que tener en cuenta que la mayoría de las variables son categóricas, y el algoritmo de sklearn solamente acepta variables numéricas o booleanas, traduciendo estas últimas a unos y ceros según sean True o False respectivamente. Para resolverlo, se utilizó ONE-HOT encoding.
No todos los features generados mejoraron los resultados. Fueron realizadas varias pruebas y se llegó a la conclusión empírica de que los mejores features para entrenar un modelo son los correspondientes a educación, rango de edad y sexo de los postulantes; algunos features correspondientes a palabras que contienen los anuncios; features con palabras que contienen los títulos de los anuncios; y features correspondientes al área de los anuncios.
Otro problema que se tuvo que afrontar fue el escalamiento de los datos, por ejemplo para la categoría de edad, donde las diferencias de edades pueden incrementar muy fuertemente la distancia, agregando demasiado valor a este feature respecto de otros features booleanos que puede ser más importantes. Debido a esto, se crearon distintos features booleanos correspondientes a distintos rangos de edad.
Sin embargo, el problema más resonante tiene que ver con el tamaño de los datos. Tomando todas las postulaciones y una cantidad de no postulaciones que duplica a la de postulaciones, se llega a aproximadamente dieciocho millones de entrada. Esta cantidad quizá no es tan grande para otros algoritmos, pero KNN tiene orden cuadrático, por lo que procesar todos los datos llevaría una cantidad de tiempo inconmensurable. Aquí se tuvo que tomar la decisión de trabajar con samples, generalmente de un millón (aproximadamente 2 hs de entrenamiento) o de 2 millones (aproximadamente 8 hs de entrenamiento). Los resultados para estos tamaños de samples prácticamente no varían, solamente encontramos una leve mejora. Lógicamente, es posible que si se hubieran aprovechado todas las postulaciones, la precisión y el mejor resultado hubiera mejorado, pero esto fue imposible de llevar a cabo por el inconveniente descrito. Realizamos un cálculo aproximado de cuánto tardaría KNN usando los dieciocho millones de datos, y el resultado indicaba que el entrenamiento estaría en el orden de los ¡años! en cuanto a tiempo.

Buscando K

Para buscar el mejor k fue realizado un grid search con distintos valores, comparando el resultado obtenido. Aquí la dimensión de los datos fue un factor muy limitante, ya que encontrar el mejor k podría llevar mucho tiempo con samples muy grandes, por lo que los samples tomados fueron bastante pequeños, de entre 100000 y 200000 entradas.

Inicialmente se probó con una cantidad k bastante pequeña, yendo de 8 a 15 vecinos, pero posteriormente se llegó a la conclusión de que habría que usar una cantidad más grande de vecinos, por lo que probó con otro grid search con saltos de 10 para k, en un rango entre 10 y 100.

Buscando la mejor distancia y métrica

Por otro lado, también mediante grid search y para distintas cantidades de k (tomando pequeños samples), fueron probadas distintas métricas. Estás fueron: manhattan, euclidean, minkowski, hamming, canberra, braycurtis, jaccard, matching, dic, kulsinski, rogerstanimoto, russellrao, sokalmichener, sokalsneath. Todas estas métricas se encuentran en la librería de sklearn. La mejor de todas ellas resultó ser la euclidea, aunque la distancia de BrayCurtis se aproximó bastante.
A su vez fueron probados distintos pesos: uniform y distance. El primero consiste en darle un peso uniforme a los k vecinos más cercanos a la hora de predecir, mientras que el segundo asigna un mayor peso a los k vecinos que más cerca se encuentran. Empíricamente, el primero funcionó mejor que el segundo.
A continuación, dos tablas elaboradas en base a distintas iteraciones para métricas distintas (se confeccionaron utilizando samples de 200k sumando postulaciones y no postulaciones):

	'euclidean'	'chebyshev'
10	0,769920936	0,631629644
20	0,784281061	0,680506061
30	0,786821638	0,683004095
40	0,789823379	0,670754713
50	0,792437160	0,675905045
60	0,792436489	0,672509689
70	0,791424570	0,677412623
80	0,791661237	0,674719794
90	0,791358169	0,666539933
100	0,789678409	0,668285520
110	0,790142257	0,661098319
120	0,789006447	0,662518416
130	0,789453623	0,658723601
140	0,787292864	0,654879536

	'hamming'	'canberra'	'braycurtis'	'matching	'dice'	'kulsinki'
10	0,78009567	0,780542367	0,791320995	0,777597118	0,7869	0,78753798
20	0,79226356	0,792468336	0,800424371	0,789381185	0,7984	0,80066259
30	0,79694626	0,796807790	0,802971942	0,796915286	0,8025	0,80389327
40	0,79926933	0,798939843	0,805563585	0,795782033	0,8033	0,80514038

	'rogerstanimoto'	'russellrao'	'sokalmichener'	'sokalsneath'	'minkowski' (p=3)
10	0,777571120	0,76412964	0,777571120	0,786863008	0,770781721
20	0,789347954	0,78253212	0,798496952	0,798496952	0,785737681
30	0,797095805	0,79090538	0,797095805	0,802481564	0,789102857
40	0,796090587	0,79448576	0,796090587	0,803338809	0,789025125

La razón por la que para las métricas de la segunda tabla se hayan usado menor cantidad de vecinos es que la búsqueda terminaba por insumir un tiempo del que no se disponía, por lo que se decidió utilizar una cantidad de k vecinos más acotada que para las primeras métricas de la tabla uno.

Si bien con una gran cantidad de métricas se obtiene resultados similares, hay que resaltar que experimentalmente la métrica que corresponde a la distancia euclidiana tardaba una cantidad de tiempo menos considerable respecto de las demás. Además, como se puede ver en las tablas, se obtienen buenos resultados para esta métrica, por lo que fue elegida para entrenar con samples más grandes.

A pesar de lo que se puede apreciar en las tablas, en base a varios entrenamientos realizados con samples más grandes (probablemente por esto aparece esta diferencia), el algoritmo funciona mejor cuando se utiliza una cantidad de vecinos un tanto más grande de lo que muestra la tabla, por lo que los mejores modelos para KNN fueron entrenados con tomando 150 vecinos.

Resultados

En sus primeras versiones, con el algoritmo se lograron predicciones con aproximadamente 75% de precisión, por lo cual se consideró dejarlo de lado rápidamente. Sin embargo, cuando se descubrió la importancia del feature de si el postulante había visto el anuncio al cual se postuló, se experimentó una mejora llegando a 88%. Luego, incorporando distintos features y sacando otros que bajaban el score, se llegó a una precisión máxima de 93,6%. Creemos que este resultado podría haber sido mejorado tomando un sample más grande de datos, pero como se describió previamente esto resultó imposible.

LightGBM

LightGBM es un algoritmo de gradient boosting que utiliza árboles. A diferencia de otros algoritmos de este tipo, LightGBM construye el árbol hoja por hoja y no por niveles, lo que puede ayudar a reducir el error aunque también puede resultar en un árbol desbalanceado, por lo que es importante fijar bien el parámetro de profundidad máxima y su relación las hojas. Una de las principales ventajas de LightGBM por lo que se suele usar es su velocidad y poco uso de memoria. Otra de las razones para elegir LightGBM y la principal en nuestro caso es que acepta variables categóricas y hemos demostrado en distintas corridas que el score respecto de usar LightGBM con One Hot Encoding (siempre que se podía), Mean Encoding y/o Count Encoding, funciona casi igual que pasarle las variables categóricas sin ningún encoding, por lo que sospechamos que podemos confiar en lo que hace internamente con estas variables.

Hicimos mas de 50 corridas de LightGBM explorando distintos parámetros y sets de datos y la amplia mayoría de nuestros mejores puntajes se deben al mismo.
Hemos entrenado LightGBM con absolutamente todos los features que listamos previamente, y hemos realizado pruebas con menos features, de forma random y no random y pareciera ser que LightGBM se maneja muy bien en grandes dimensiones y que rara vez los features sobran, y si lo hacen la diferencia de puntajes es del orden de 10^{-4}.

La cantidad de no postulaciones utilizada varía mucho. Desde la misma cantidad de postulaciones hasta dos veces esa cantidad los mejores resultados se obtienen con una proporción de no postulaciones respecto de las postulaciones de 5/4 y 3/2.

LightGBM tambien es responsable de las predicciones mas baratas que hemos logrado, llegando a puntuar 95.1% de precisión en kaggle con un entrenamiento de 10 iteraciones que se realiza en 4 minutos. Sobre estas corridas rapidas realizamos un grid search de parámetros para buscar los mejores usando la métrica score.

Leaves	Depth	L_rate	colsample	subsample	subsample_freq	Resultado
6000	13	0,005	0,8	0,8	0,5	78,6694744
6000	14	0,005	0,8	0,8	0,5	79,3063470
6000	15	0,005	0,8	0,8	0,5	79,6770387
6000	16	0,005	0,8	0,8	0,5	79,9513121
6000	17	0,005	0,8	0,8	0,5	80,2253368
6000	18	0,005	0,8	0,8	0,5	80,4380666
6000	18	0,01	0,8	0,8	0,5	80,4958803
4098	18	0,01	0,8	0,8	0,5	80,3383534
7000	18	0,01	0,8	0,8	0,5	80,5497776
7000	18	0,01	0,8	0,8	0,6	80,5497776
7000	18	0,01	0,8	0,8	1	80,5497776
7000	18	0,01	0,8	0,9	1	80,5722193
7000	18	0,01	0,8	1	1	80,5635162
7000	18	0,01	0,9	1	1	80,7999309
7000	18	0,01	1	1	1	80,2064386

Luego, descubrimos que obtenemos los mejores resultados con 12000 hojas para entrenamientos más largos, y que ciertas mejoras ligeras de puntaje vistas en esta búsqueda de hiperparametros en entrenamientos muy largos o bien no se aprecian o son contraproducentes.
Los entrenamientos más largos de LightGBM no superaron las 3 horas.

Redes neuronales

Descubrir que un puntaje de las vistas nos hace obtener una muy buena predicción nos hizo sospechar que quizá hay una funciona no lineal que aproxima muy bien la clase en nuestros set de entrenamiento, por lo que decidimos intentar con redes neuronales.
Los mejores resultados los obtuvimos con una red de neuronas de entrada igual a la cantidad de features y 4-6 capas ocultas de la misma cantidad de neuronas. Estos puntajes superan ligeramente el 94% en kaggle, teniendo en cuenta que usamos la variable del área del anuncio como count encoding y no hicimos ninguna prueba con mean encoding.
Entrenar estas redes neuronales nos resultó muy barato y el modelo en disco ocupa 190KB por lo que es muy ligero. Hicimos aproximadamente 10 corridas de este algoritmo.

Gaussian Naive Bayes

Naive Bayes fue una grata sorpresa ya que, a pesar de no mostrar los mejores resultados, entrenando con todos los features (que son demasiados y todos tienen el mismo peso) el ROC AUC score sobre el set de test llegó a 87 en un entrenamiento muy veloz, por lo que es una excelente alternativa si se cuenta con poco tiempo. Hicimos una sola corrida de este algoritmo por sí solo y lo abandonamos porque teníamos en claro que no era un algoritmo que pudiera llegar al podio de la competencia, pero creemos que se puede mejorar muchísimo y tener un buen potencial para entrenamientos muy cortos.

Random Gaussian Naive Bayes

Con Naive Bayes solo notamos que las predicciones eran de mucha confianza. Casi todas eran 0.9999 o 0.00001, por lo que perdíamos la oportunidad de reducir el error para las predicciones en las que estuviera menos seguro.
Por esto es que programamos una clase que entrena un primer naive bayes con todos los features y después entrena otros n naive bayes con k features al azar (n y k hiperparametros). La predicción de probabilidad era siempre el promedio de lo que dijeran todos los modelos. Si un nuevo entrenamiento reducía el ROC AUC score sobre el set de test era descartado y hacía una cantidad fija de reintentos que se mandaba como hiperparámetro. Si al reintentar se fallaba en mejorar el score o bien terminaban las iteraciones, el algoritmo se detenia. Esto con 10 iteraciones y 3 reintentos en un entrenamiento de 5 minutos nos llevó a puntuar naive bayes con 90% tanto en el set de test como en kaggle, lo que es una mejora de 3% sobre naive bayes solo. Los datos observados en la predicción tenían muchos más números cercanos a 0.5, que es lo que queríamos.
Creemos que la clase podría mejorarse mucho más y hacer, por ejemplo, que la probabilidad de elegir dos features juntos sea inversamente proporcional a qué tan correlacionados están estos, ya que naive bayes trabaja mejor con features que no están correlacionados, pero por cuestión de tiempo y por saber que este muy probablemente no es el mejor algoritmo abandonamos la idea.
Pudimos usar esta clase para hacer que otros algoritmos se entrenen con la misma idea, el problema principal es que no pudimos persistir esta clase en disco para cargarla.

XGBoost

XGBoost es uno de los algoritmos que no podía faltar. Este es un algoritmo de Boosting que genera un árbol para cada iteración, y es considerado como uno de los mejores algoritmos de clasificación.
En un principio, XGBoost fue dejado un tanto de lado ya que con LightGBM se estaban obteniendo muy buenos resultados y además este último acepta variables categóricas, es decir que no era necesario realizar ningún encoding para aquellas variables con muchísima cantidad de clasificaciones (por ejemplo el nombre del área). Sin embargo, una vez que nos atoramos en un puntaje de 95% fue necesario comenzar a explorar distintas variantes. XGBoost fue una en la que más se hizo énfasis, incluso con este algoritmo se logró el mayor de todos los resultados.
Es fundamental destacar que sí o sí se necesitan utilizar uno o más encodings. El criterio nuestro fue utilizar One Hot encoding siempre que pudiéramos, y en su defecto Mean encoding. Por otro lado, fue el algoritmo para el que más features se probaron, ya que se tomaron absolutamente todos aquellos desarrollados en la sección de features.

Dado que XGBoost es un algoritmo que utiliza muchos recursos, nunca se pudo entrenar con más del 70% del set de datos. Se probaron distintos tamaños de muestra y se hallaron los mejores resultados con un 50% del set completo.

CatBoost

CatBoost es otro algoritmo de gradient boosting que, al igual que LightGBM, acepta variables categóricas, aunque según se investigó no suele tener tan buenos resultados. Buscamos con un set reducido de datos los mejores hiperparámetros para CatBoost. Los scores de la siguiente tabla corresponden a la precisión sobre la categoría de forma discreta, sin probabilidad, sobre el set de test.

Depth	L2	L_rate	Resultado
10	2	0,005	75,60062538
9	2	0,005	75,44670415
11	2	0,005	75,70748750
12	2	0,005	76,22948934
13	2	0,005	76,34144901
13	1	0,005	76,35742549
13	1,5	0,005	76,32186695
13	1	0,01	76,38098613

Entrenamos este algoritmo 3 veces con todo el set de datos y conseguimos una precisión del 94% sobre el set de predicción, que en su momento era menor a lo conseguido con LightGBM, y al ser tener CatBoost un entrenamiento más costoso, fue abandonado su uso.

Gradient Boosting Machine (H2O)

H2O es una plataforma de Machine Learning que corre sobre Java para ser usada principalmente sobre sistemas distribuidos. Llegamos leer que el algoritmo de Gradient Boosting que incluía era muy bueno y trataba muy bien a las variables categóricas.
Instalamos el módulo para python y corrimos H2O Flow, que es una interfaz web muy similar a Jupyter con sus propias notebooks y un "lenguaje" de instrucciones muy sencillo para cargar los archivos y entrenar algoritmos.
Ofrece muchas comodidades, como mostrar gráficos de las variables más importantes, gráficos que nos sirvieron más tarde para pensar algunas ideas, también muestra el progreso del algoritmo con una interfaz gráfica muy agradable.
El problema es que las predicciones resultaron pésimas. No logramos entender qué era lo que estábamos haciendo mal y los scores resultaron inaceptables (<50%), por lo que lo dejamos de lado rápidamente.

Perceptrón

El algoritmo de clasificación lineal también fue probado. Si bien no se esperaba lograr mucha precisión, se llegó a un score del 85%. Rápidamente, se encontraron algoritmos que superaban ampliamente este puntaje, por lo que no se indagó demasiado en intentar hacer funcionar mejor el algoritmo, dejándolo de lado rápidamente.

SVM

El algoritmo de Support Vector Machines también fue testeado. Sin embargo, este fue utilizado una vez que la competencia ya había avanzado bastante y, al obtener malos resultados y al observar que entrenarlo llevaba un tiempo considerable, se lo abandonó instantáneamente.

La mejor solución

Si bien con distintos algoritmos se llegaron a buenos resultados, estos no alcanzaron para ganar la competencia, e incluso por sí solos no hubieran logrado estar en los puestos top.
Nuestro mejor score se logró de una forma un tanto inesperada. Combinando dos de los mejores algoritmos que entrenamos, KNN y XGBoost, se realizó una ponderación entre los mismos obteniendo así la mejor puntuación. La ponderación toma con un 60% de importancia a KNN y un 40% a XGBoost, es decir que la probabilidad de postulación se calculó como:

P(postulación) = P(postulación en KNN) * 0.6 + P(postulación en XGBoost) * 0.4

Con dicha fórmula se logró un score de 0.97360.

Aquí, hay que frenar y hacer una aclaración. Además del entrenamiento de los algoritmos, se aprovechó que en el set de predicción había postulaciones que se encontraban en el set de entrenamiento (unas 34500). Realizando una combinación entre dichos dataframes se pudo elevar la precisión de la predicción. Por sí solos y aprovechando esta falla encontrada en los datos, con XGBoost se había logrando un score de 0.96861 y con KNN uno de 0.96151.

También, fueron probadas ponderaciones con LightGBM, pero ninguna de estas pudo superar el máximo antes mencionado.