Predicción de postulaciones a avisos laborales

Sportelli Castro, Luciano Liberali, Franco Iglesias, Matias

28 de junio de 2018

1. Introducción

El objetivo de este trabajo es generar un modelo predictivo basado en algoritmos de Machine Learning capaz de decidir si un determinado usuario de zonaJobs se postulará o no a un determinado aviso laboral.

Para esto se utilizó la información provista por Navent para generar distintos features para los pares aviso-postulante. Entre la información provista se encuentran datos sobre los avisos y los postulantes, así como también qué avisos vieron y a qué avisos se postularon.

2. Ideas iniciales

En la reunión inicial del equipo para decidir como abordar el trabajo, se presentaron algunas ideas basadas en clustering, principalmente intentar hacer clustering sobre los avisos, de modo de encontrar avisos similares, determinar cuántos avisos vio/se postuló el usuario de cada cluster y en base a esto emitir una probabilidad de que el usuario se postule. Una idea similar surgió sobre hacer clustering sobre los postulantes. El problema que se presentó sobre estas ideas es que en el caso de los avisos, no era simple definir una medida de distancia; la comparación de descripciones no surgió hasta el final de la etapa de feature engineering y hubiera retrasado mucho el desarrollo del proyecto. Por otra parte, determinar la probababilidad por sobre caer en un determinado cluster tal vez no sería lo ideal ya que no tendría en cuenta otros factores como sí podría tenerlo un árbol de decisión. Al realizar las pruebas iniciales optamos por descartar los algoritmos de clustering y arrancar directamente con árboles de decisión.

Otro factor analizado fue cómo representar numéricamente los datos, ya que teníamos por un lado muchas columnas categóricas y por el otro las descripciones en lenguaje natural, donde estaba contenida gran parte de la información del aviso. Para las columnas categóricas la solución fue intentar encontrar métricas sobre cada columna basándonos en el significado; en el caso del sexo que suelen tener los postulantes de un aviso, por ejemplo, se decidió usar una medida "polar", donde un polo es mujer, el otro polo es hombre y los valores intermedios indican tendencias hacia un predominio de ese sexo en la cantidad de postulaciones. Para el caso del área laboral, tipo de trabajo y nivel educativo del postulante, el primer acercamiento fue utilizar una codificación de tipo one-hot. Esto nos sirvió para empezar pero luego fue modificado en la etapa de feature engineering.

3. Desarrollo

El desarrollo del trabajo consistió en cuatro etapas: una etapa temprana, una etapa de feature engineering, una etapa de armado del set de entrenamiento y una etapa final.

En la etapa temprana se hicieron algunas pruebas para observar el entorno en que se iba a desenvolver la competencia. Básicamente se generaron algunas predicciones en base a pocos features obtenidos directamente de los avisos con la finalidad de probar las herramientas y ver unos primeros resultados.

La etapa de feature engineering fue la que más tiempo consumió y consistió en crear distintos features tratando de sintetizar la mayor cantidad de información posible a partir de los datos provistos. Esta etapa inicialmente tuvo un desarrollo un tanto desordenado que culminó en la creación de un programa para estandarizar la forma en que se creaban features. El desarrollo de este programa llevó más tiempo del esperado pero su funciona-

miento tuvo buenos resultados. Durante esta etapa se emitieron muy pocos submits ya que el flujo de trabajo se enfocó en la estandarización de la creación de features.

Una vez finalizada la etapa de feature engineering nos dispusimos a generar un set de entrenamiento. Previo a esta etapa se trabajaba con un set de entrenamiento *primitivo* que fue creado simplemente a partir de marcar las postulaciones con un `1` y las vistas que no se convirtieron en postulaciones con un `0`. El problema de este set es que no contemplaba los pares de avisos que no fueron ni vistos ni postulados. Para resolver este problema, con los features ya creados utilizamos una modalidad similar a aprendizaje semi-supervisado para generar un conjunto de pares de no-postulaciones aleatorias.

Finalmente en la última etapa, con los sets ya creados se probaron distintos algoritmos y ensambles de algoritmos para la competencia. Los algoritmos probados fueron Random Forests, Decision Trees y Gradient Boosting, y se hicieron ensambles de los mismos utilizando AdaBoost y Voting.

3.1. Etapa temprana

Durante esta etapa se realizaron distintas pruebas para interiorizarnos con los datos y con la modalidad de la competencia en Kaggle.

Como un primer acercamiento, se creó un set de entrenamiento contenía todas las postulaciones, marcadas como `1` y todas las vistas que no estaban en el set de postulaciones marcadas como `0`. Con el set ya creado se generaron los primeros features que simplemente contenían la edad, sexo y nivel educativo del postulante, e información sobre la zona, el área y el tipo de trabajo del aviso. Este primer set de features consistió en alrededor de 200 columnas en el set de datos, ya que la información de los avisos se codificó en one-hot. El sexo de los postulantes se codificó en -1/1 según si era mujer u hombre respectivamente.

Una vez generado el set de features, se entrenó un Random Forest y se hizo un primer submit que dió como resultado una precisión del 0.675.

En este punto se observó lo siguiente: el set de entrenamiento no era ideal, ya que únicamente contemplaba los casos en que el usuario vió o se postuló al aviso; y los features no aportaban suficiente información. En base a esto se agregaron "no-postulaciones", pares usuario-aviso tomados de forma aleatoria con correspondiente chequeo de que ese usuario no se hubiera postulado a ese aviso (en realidad nada nos asegura que ese usuario no se hubiera postulado a ese aviso, simplemente que no estaba dentro de la información que nosotros contábamos como postulación, pero la probabilidad es baja) y se crearon algunos features que indicaban la cantidad de veces que el usuario vió y se postulo a avisos de la misma empresa, del mismo área y similares.

Estos cambios empezaban a significar la realización de un proceso de obtención de información sobre los usuarios contenidos en el test final, ya que dejaba de ser simplemente ir al set de información de usuarios y buscar su edad, nivel académico, etc.; sino que requería que estos usuarios sufrieran el mismo proceso que el set de entrenamiento, siendo emparejados con la información de avisos vistos y postulados antes descripta. Con este proceso y la aplicación de algoritmos similares a los anteriores la precisión obtenida saltó de 0.672 a 0.834, mostrando un fuerte cambio debido a modificar el set de entrenamiento y los features elegidos.

A partir de acá se fueron creando distintos features y variaciones de hiperparámetros del algoritmo pero en una forma un tanto desordenada y que no estaba mejorando los resultados obtenidos, lo cual complejizó el avance del trabajo. También se intentó un intento de solución de un problema planteado con anterioridad, que es la consideración

de “no postulados” a aquellas parejas de las cuales no se tenia información. La idea fue básicamente utilizar este método que nos daba un 0.834 de precisión y aplicarlos sobre el set de no postulados para intentar predecir algunas postulaciones allí y luego de este proceso volver a entrenar el algoritmo, lo que resultó en peores resultados pero fue una idea que más adelante fue refinada.

3.2. Feature engineering

El comienzo de la etapa de feature engineering fue marcado por dos eventos: observar que la puntuación se movía muy lentamente y el desorden en la forma de generar los features. Esto nos llevó a construir una pieza de software para estandarizar la forma en que se crean los features, de modo que simplemente se le de como entrada un archivo con los pares aviso-postulante y el programa genere otro archivo con la misma información más los features generados.

La creación de este programa fue muy útil pero tomó más tiempo de lo esperado debido a complicaciones con la cantidad de memoria utilizada o el tiempo que tomaba en generarlos.

Durante el desarrollo del programa se fueron desarrollando distintos features que se dividieron en distintos grupos: información promedio sobre avisos, cantidades por feature, cantidad de vistas y postulaciones, información básica del postulante, vistas por aviso y descripciones. A continuación se detalla cada feature generado.

3.2.1. Información básica del postulante

La información básica del postulante se compone de cuatro features: edad, sexo, nivel educativo y “hay_información_basica”. Este último es sólamente un indicador sobre si se pudo obtener o no información básica del postulante.

El sexo se codificó como -1 si es mujer, 1 si es hombre, 0 si no declara/no hay información. La razón de codificarlo de esta manera es que más adelante al generar otros features para los avisos en base a este, nos permite determinar una tendencia simplemente calculando un promedio del feature. Si el promedio da cercano a -1 entonces la mayor cantidad de gente es mujer; si da cercano a 1 es hombre y si es 0 entonces el género es indiferente.

El nivel educativo se codificó en base a una tabla armada manualmente, indicando con un número entre el 0 y el 10 cual es el nivel educativo de la persona. La idea conceptual que se tomó es que lo más importante es el nivel máximo de educación de la persona; una persona con un doctorado *seguro* que tiene un título secundario y un título universitario. Este tipo de codificación induce también una medida de distancia, la cual es generalmente consistente, hay mucha más diferencia de una persona que tiene un secundario completo a una persona que tiene un doctorado, que de una persona que tiene un secundario completo a una que tiene universitario abandonado. Sin embargo tiene también alguna inconsistencia, particularmente con el caso de la educación terciaria; la distancia de un terciario a un secundario es en principio la misma que del secundario a la universidad -solo se requiere tener el secundario aprobado- pero el peso del título no es el mismo. Debido a que los casos en que se da esta situación son pocos, se decidió mantener la métrica pese al inconveniente.

3.2.2. Información promedio sobre avisos

Aprovechando las métricas armadas para la información de los postulantes, se incorporó esta información también a los avisos. Se agruparon las personas que se postularon a cada aviso, y se tomó el promedio de cada uno de los features. Para evitar que un aviso con pocas postulaciones quede completamente polarizado en sus features, se agregó un poco de ruido sumando la media de cada feature cuatro o cinco veces. De esta forma se necesitan cuatro o cinco personas del mismo grupo para poder modificar el indicador.

3.2.3. Cantidades por feature

Para considerar las preferencias de cada persona, se agregaron features con la cantidad de vistas y postulaciones de cada persona a avisos de la misma zona, del mismo área laboral, del mismo tipo de trabajo y de la misma empresa.

En la etapa inicial solo se tomaban las vistas por zona, área y empresa, y como se observó que dio buenos resultados se extendió al tipo de trabajo y a considerar también las postulaciones.

3.2.4. Cantidad de vistas y postulaciones

Un usuario que tiene más vistas y postulaciones es un usuario que utiliza más el servicio de zonaJobs y por consiguiente es más probable que se postule si el aviso le gusta. Si el usuario tiene pocas vistas y/o postulaciones, es probable que no utilice tanto el servicio. Se generaron dos features con la cantidad de vistas total y la cantidad de postulaciones total del usuario.

Por el contrario, agregarle a los avisos información acerca de cuantas personas de los cuales miraban el aviso se postulaban a él, para tenerlo en cuenta como un aviso que suele ser llamativo para los usuarios no generó aportes a la precisión.

3.2.5. Vistas por aviso

Indica la cantidad de veces que el usuario vio el aviso, si es que lo vió. Al agregar este feature se tuvieron en cuenta dos cosas, la primera es si agrega ruido, la segunda es si se debería considerar también si el usuario se postuló o no al aviso. La segunda consideración nos llevó a pensar que agregar esa columna básicamente haría que el algoritmo quede determinado por la misma, ya que nuestro set de entrenamiento está basado en esa información. En cuanto a la primera, los resultados empíricos nos dieron bien, razón por la cual optamos por dejarla.

3.2.6. Descripciones

El procesamiento de descripciones fue uno de los problemas más complejos que tuvimos y que más tiempo llevó. Debido a que las descripciones están escritas en lenguaje natural, su procesamiento es en sí un problema de Machine Learning. Para atacar el problema y simplificarlo de algún modo, se decidió abordarlo por el lado de búsquedas ranqueadas. De este modo, primero se realizó un preprocesamiento en el cual se realizaron las siguientes operaciones:

- Eliminar todas las etiquetas HTML
- Convertir a minúsculas y eliminar cualquier caracter no alfabético

- Reemplazar letras con tilde por las mismas sin tilde
- Eliminar stop words del inglés, del español y del dominio (avisos laborales)
- Eliminar las palabras que aparecen en menos del 1 % de las descripciones
- Realizar un proceso de stemming para sintetizar familias de palabras en una sola palabra

Aquí nos encontramos con el problema de que algunos avisos estaban en español y otros en inglés; pero ya que los segundos eran minoría y que el inglés y el español son lexicográficamente similares, optamos por asumir que estamos trabajando con anuncios en español (excepto por las stop words).

Realizado este preprocesamiento, el léxico quedó reducido a alrededor de 1000 palabras, y se comprobó que todas las descripciones tenían palabras, excepto un conjunto reducido (11 avisos) que no tenía sentido considerar.

Finalizado el preprocesamiento, se generó un feature basado en la cantidad de palabras que tiene la descripción del anuncio y que el usuario vio. El mismo se calcula sumando la cantidad de veces que el usuario vio la palabra si la palabra está en el anuncio.

Una mejora de este feature que no se llegó a implementar es calcular el score utilizando un peso particular para cada palabra basado en el tf-idf de la misma en la descripción.

3.3. Set de entrenamiento

Con la etapa de feature engineering finalizada y considerando que el tratamiento de los datos es la etapa más importante del desarrollo de cualquier trabajo en el área de Machine Learning, se creó un set de entrenamiento adecuado. En el mismo se tuvo en consideración que hay tres tipos de pares aviso-postulante:

- Par que se postuló
- Par que vió el aviso y no se postuló
- Par que no vió el aviso y/o no tiene sentido y no se postuló

En esta etapa se busca solucionar el problema planteado al inicio, ¿cómo agregar la consideración sobre los pares de los cuales no tenemos información? La solución que utilizamos en este trabajo consiste en una técnica similar a aprendizaje semi-supervisado. Visto que en la etapa inicial, simplemente agregando pares aleatorios de postulantes y avisos la precisión del modelo aumentaba, optamos por aprovechar este fenómeno y hacer un refinamiento iterativo.

El refinamiento consiste en el siguiente proceso: partiendo de un set de entrenamiento que únicamente contiene datos de postulaciones y vistas no postuladas (dos de los tres tipos de pares que planteamos anteriormente), entrenar un Random Forest y hacerlo predecir sobre un conjunto de pares aviso-postulante tomados al azar. Sobre esta predicción, tomar muestras con una probabilidad menor a un umbral dado por una función que depende del número de iteración, adosar esas muestras al set de datos *inicial* y luego reentrenar el algoritmo. El proceso se repite hasta llegar a una cantidad de iteraciones máxima.

La función que define el umbral fue determinada empíricamente bajo el concepto de que la predicción va mejorando con cada iteración, con lo cual, el umbral debería ser

más chico a medida que el algoritmo mejora. Para lograr esto, se realizó una prueba con una función lineal pero no se obtuvieron buenos resultados, ya que la función decaía más rápidamente que lo que el algoritmo aprendía. Finalmente se utilizó la siguiente función:

$$f(i) = e^{-\frac{1,5 \cdot i}{n} + log(b)}$$

donde n es la cantidad total de iteraciones, i es el número de iteración actual, y b indica la probabilidad máxima inicial de aceptar un par. En el trabajo se adoptaron los siguientes valores: $n = 10$, $b = 0{,}5$.

Habiendo creado el set nos encontramos con otro problema, hay tres tipos de pares y dos clases, con lo cual o quedan desbalanceadas las clases, o quedan desbalanceados los tipos. Para resolver este problema se hizo una prueba con cada caso, con mejores resultados para el caso de clases balanceadas. De este modo, el set de entrenamiento terminó distribuido de la siguiente forma: 50 % postulaciones, 25 % vistas no postuladas, 25 % pares no postulados aleatorios.

En base a pruebas realizadas sobre los algoritmos y el hardware, se decidió que el set de entrenamiento tenga un millón de registros. Esto es debido a que comprobamos que entrenando con un millón o dos millones obtuvimos resultados similares, pero con dos millones forzabamos al sistema a mover datos de RAM a swap debido a que el set entero no entraba en memoria. Esto volvía el proceso de entrenamiento mucho más lento. Hicimos también una prueba con 5 millones de datos y, sorpresivamente, obtuvimos una precisión menor que entrenando con dos millones.

3.4. Etapa final

Con el set de entrenamiento ya creado se probaron, principalmente, algoritmos basados en árboles de decisión. También se hicieron algunas pocas pruebas sobre regresores lineales, particularmente stochastic gradient descent sin tener buenos resultados.

Dentro de los algoritmos basados en árboles se utilizó en un primer momento un ensable conocido como Random Forest. El mismo está basado en varios árboles de decisión donde cada uno toma algunos features elegidos al azar y realiza la predicción sobre esos features únicamente, finalmente con los resultados de todos los árboles se realiza una votación para obtener la predicción final. Este algoritmo nos dió buenos resultados en todos los casos pero no fue el mejor de todos; la mejor predicción que nos dio fue de 0,90. Este resultado se obtuvo usando 40 estimadores, cada uno con tantos features como la raiz de la cantidad total de features y un tamaño de hoja de 100 muestras.

Con este resultado, se realizaron pruebas utilizando AdaBoost como método de boosting, lo cual mejoró significativamente los resultados llegando a una precisión de 0,93; sin embargo, nos rindió mejor utilizar AdaBoost con árboles de decisión directamente, llegando a 0,946.

En este punto, se nos ocurrieron dos modificaciones en los features: agregar el requerimiento de un terciario por separado, para solucionar el problema de los estudios terciarios; y agregar un feature indicando la coincidencia en la descripción con los avisos que vió pero no se postuló. En principio pensamos que podría mejorar la puntuación, pero al probarlo con la misma configuración que nos dió el mejor resultado obtuvimos un resultado menor.

3.4.1. Entrenamientos de Random Forest

Random Forest fue el primer tipo de algoritmo que se probó, siendo que es uno de los algoritmos que “funciona bien generalmente”. En nuestro caso nos dió muy buenos

resultados, aunque no fue el mejor de los algoritmos probados. En la primer prueba que hicimos, con un set de entrenamiento que únicamente contenía información sobre el postulante y sobre el aviso, nos dió una precisión de 0,69 con los parámetros por defecto (tamaño de hoja = 2, 10 estimadores).

Basándonos en la premisa de que el resultado de la predicción proviene más del set de entrenamiento que del algoritmo en sí, decidimos no hacer muchos cambios en los hiperparámetros y enfocarnos más en el set de entrenamiento. Simplemente cambiando el set de entrenamiento y agregando features que relacionen al usuario con el aviso logramos obtener una precisión de 0,88 con el mismo algoritmo.

Utilizando AdaBoost como método de boosting logramos llegar a obtener una precisión de 0,92 con 15 estimadores de AdaBoost y 10 estimadores para los Random Forest.

Modificando los hiperparámetros logramos elevar la predicción generada por el algoritmo base a 0,90. Esto se logró cambiando el criterio para el split de "entropía" a "gini" y la cantidad de features por árbol a raíz de la cantidad total. Posteriormente, un entrenamiento de un AdaBoost con árboles de decisión nos dió un resultado de 0,946, considerando que la precisión de los estimadores base era menor que la del random forest modificado (de 0,88 a 0,90), tomamos como hipótesis que el resultado debía mejorar si manteníamos la configuración de AdaBoost y reemplazabamos el estimador base por el Random Forest. Esto nos dió un resultado aceptable, de 0,93, pero no superó al anterior.

3.4.2. Entrenamientos de Decision Trees

Los árboles de decisión no fue un algoritmo contemplado inicialmente, ya que los Random Forest son en el fondo árboles de decisión y nos pareció redundante. Sin embargo, una prueba realizada con un árbol de decisión con hiperparámetros encontrados vía grid-search resultó darnos una precisión muy similar a la encontrada con un Random Forest (0,88) con lo cual decidimos darle una posibilidad.

Si bien por sí solos no superaron mucho más que 0,88, al combinarlos utilizando AdaBoost logramos obtener una precisión de 0,946, siendo la mejor obtenida, inclusive por encima de los RandomForest.

3.4.3. Otros entrenamientos

Por fuera de los árboles de decisión, se probó también Gradient Descent Boosting y Stochastic Gradient Descent. Ninguno de los dos métodos nos dio buenos resultados, con lo cual no se hicieron pruebas exahustivas sobre los mismos. En el caso de Gradient Descent Boosting nos encontramos con un resultado similar a tener un Random Forest sin ningún método de boosting. Para el caso de Stochastic Gradient Descent nos topamos con un tiempo de entrenamiento alto, tal vez debido a una mala configuración. Esto mismo nos detuvo de probar Support Vector Machines, el algoritmo al cual teníamos acceso corría en $O(n^2)$, siendo prácticamente imposible entrenarlo con el mismo set con el que entrenabamos a los otros algoritmos.

4. Conclusiones

La primer conclusión que extraemos, muy mencionada en los apuntes teóricos, es la importancia de definir un set de entrenamiento adecuado. Sólo cambiando el set aumentamos la precisión de 0,70 a 0,89 casi sin modificaciones en los algoritmos utilizados.

En segundo lugar observamos que llegamos a un mejor resultado utilizando un algoritmo sencillo y un método de boosting, en particular un árbol de decisión con AdaBoost que utilizando el mismo método de boosting con un ensamble de algoritmos. Esta observación es, en principio, antiintuitiva, sobre todo considerando que el ensamble del que estamos hablando es un conjunto de árboles de decisión con un subconjunto de features del mismo set de entrenamiento. Como hipótesis, atribuimos este comportamiento a la correlación entre los features. Si los features están muy correlacionados entre sí, dividirlos en subconjuntos podría ocultar algunas correlaciones entre los mismos y por consiguiente, generar una predicción menos firme. Esto, por supuesto, no pasaría en el árbol de decisión que tiene visión sobre todos los features.

Por último, consideramos utilizar un algoritmo sencillo para la puntuación final ya que, además de ser el que mejor resultado nos dió, es muy simple de implementar en general y se lo puede entrenar en un tiempo razonable, en nuestro caso los entrenamientos duraron menos de 2 horas en una PC con especificaciones moderadas[1].

[1]El algoritmo corría en un sólo hilo, RAM: 4 GB; CPU: Intel® Core™ i5-3317U CPU @ 1.70GHz