HR Analytics: Job Change of Data Scientists

- **Опис на проблем:**

  Една компанија сака да ги вработи оние Data Scientists што успешно ги поминале нивните курсеви. Многу луѓе се пријавиле за нивните курсеви. Компанијата сака да знае кои од овие кандидати сакаат навистина да работат за компанијата или пак ќе бараат нова работа во друга компанија, бидејќи ова ќе им помогне да ги оптимизираат трошоците и времето потрошено на планирање на курсевите, воедно и да го подобрат квалитетот нивни.
- **Опис на податоци:**

  Дадени се податоци за претходни кандидати. За секој од кандидатите имаме демографски податоци, нивното образование, нивното искуство и бинарна одлука - дали се вработил во компанијата, или барал работа на друго место).
- **Карактеристики (features):**
  - **enrollee_id :** уникатен ID за кандидатот
  - **city:** код за градот
  - **city_ development _index :** развоен индекс на градот (скалирано)
  - **gender:** пол на кандидатот
  - **relevent_experience:** релевантно искуство на кандидатот
  - **enrolled_university:** тип на универзитет на кој кандидатот студирал
  - **education_level:** ниво на образование на кандидатот
  - **major_discipline :** насока на кандидатот на која студирал
  - **experience:** вкупно искуство во години на кандидатот
  - **company_size:** Број на вработени во сегашната компанија во која работи кандидатот
  - **company_type :** тип на компанија
  - **last*new*job:** разлика во години помеѓу претходната и сегашната компанија на кандидатот
  - **training_hours:** часови исполнети за тренинг
  - **target: 0 –** Ќе се вработи во оваа компанија**, 1 –** Ќе бара нова работа во друга компанија.

Анализа и визуелизација на множеството и негова подготовка

Користејќи готови модули (SweetViz и pandas-profiling) во Python полесно и побрзо ги анализиравме податоците преку визуелизации.

При анализата, увидовме неколку предизвици кои треба да се решат со цел да креираме модели за предвидување:

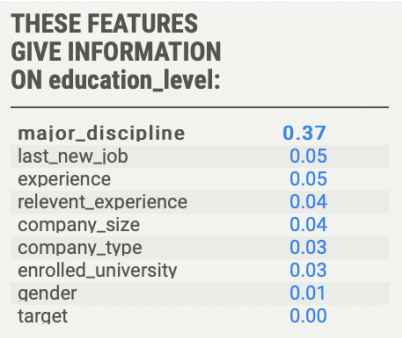
Иницијално, ја тргаме колоната на enrolle_id бидејќи е уникатна колона и нема да ни треба при градење на моделот за класификација.

Понатаму ги енкодираме вредностите во сите категориски колони, но не и NaN вредностите кои ќе треба да ги потполниме.

Она што забележавме, најголемиот предизвик се вредностите кои недостасуваат во дадените колони: **gender, enrolled_university, education_level, major_discipline, company_size, company_tipe, last_new_job**.

Тоа го решаваме со помош на импутација. Но пред да импутираме вредности на местата каде што недостасуваат, го разделуваме множеството на тренирачко и тестирачко, бидејќи мораме да запазиме и на проблемот со Data Leakage, односно да избегнеме допуштање на информации од тренирачкото м-во во тестирачкото м-во.

Користиме метода на импутација Forward Fill, но на поразумен начин.
Пример:
Колоната education_level има редици во кои вредностите се NaN. Ако направиме анализа на оваа колона, тогаш јасно нѝ е дека major_discipline дава некаква додатна информација за education_level.



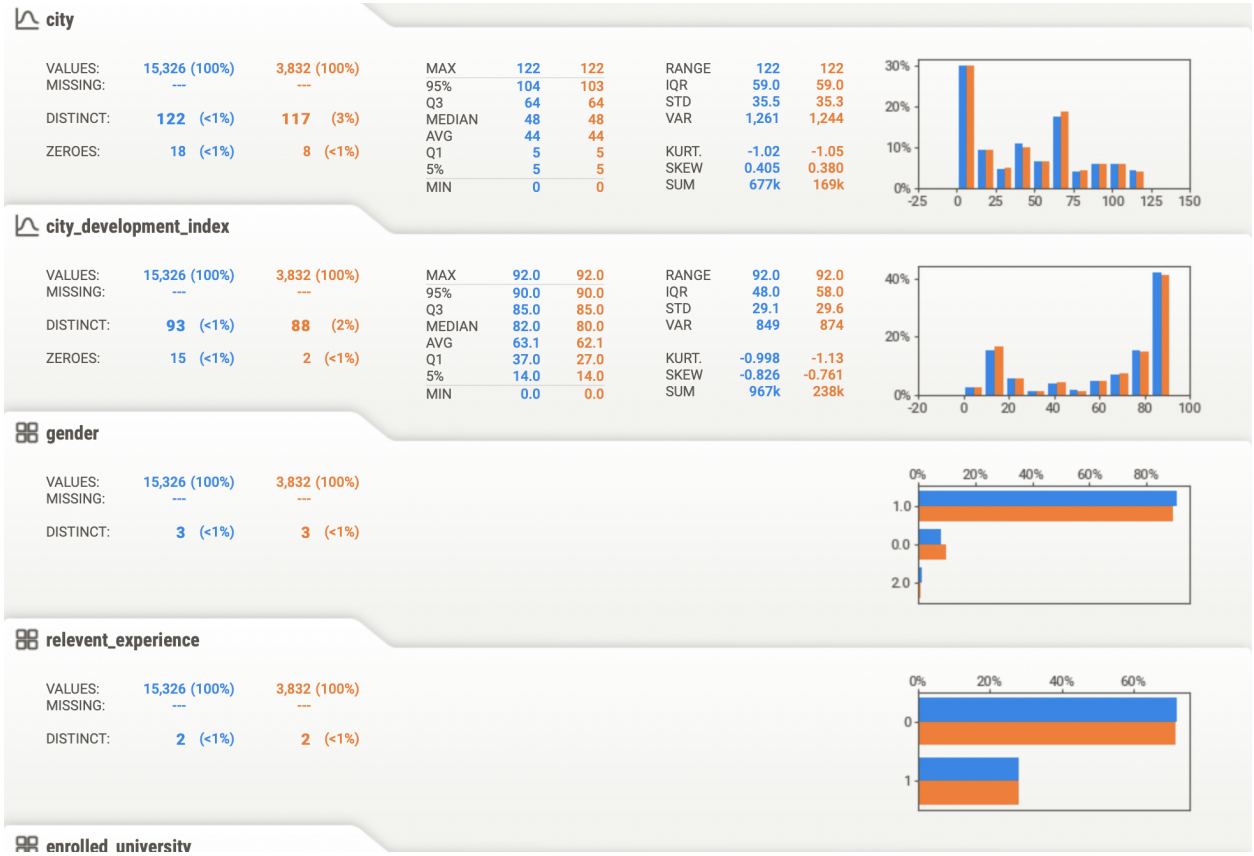
THESE FEATURES GIVE INFORMATION ON education_level:

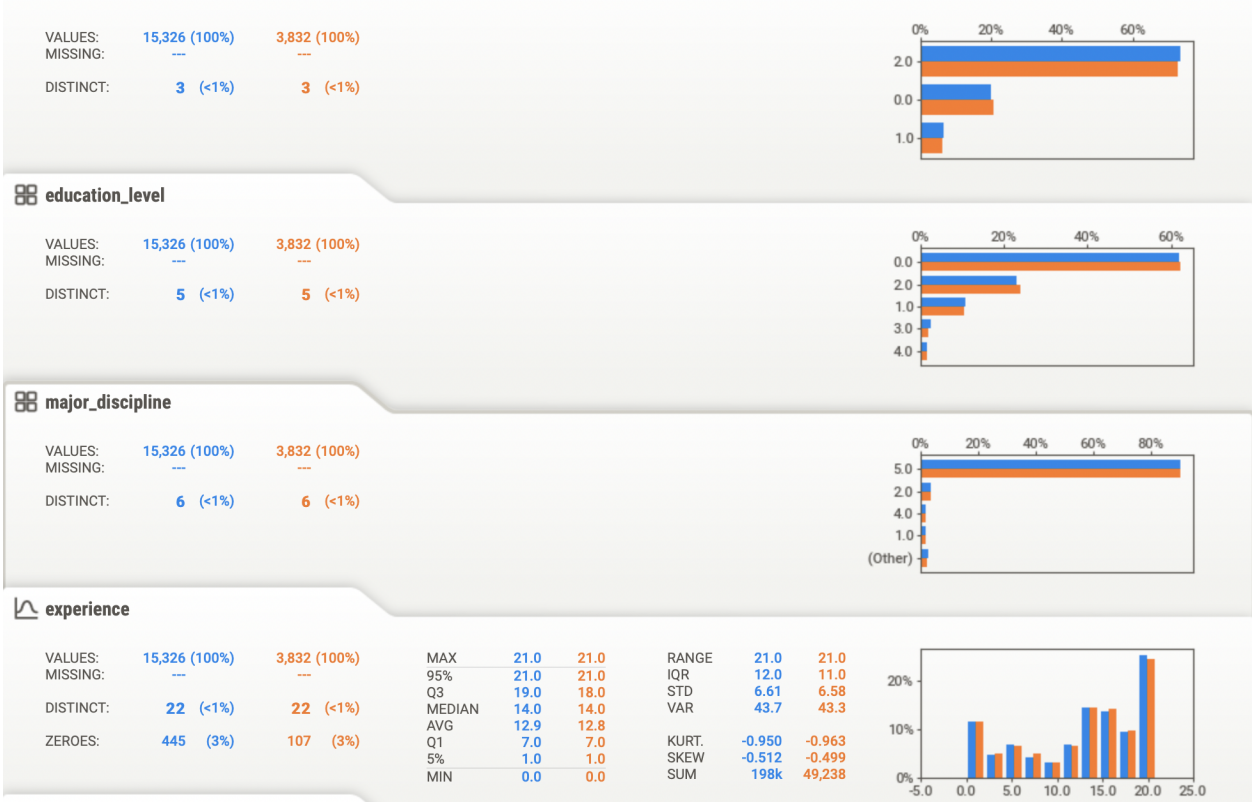
major_discipline	0.37
last_new_job	0.05
experience	0.05
relevent_experience	0.04
company_size	0.04
company_type	0.03
enrolled_university	0.03
gender	0.01
target	0.00

Тогаш тренирачкото множество го сортираме според колоната major_discipline и извршуваме forward fill на колоната education_level. Односно ја запазуваме нивната меѓусебна врска.

Тренирачкото множество е доста небалансирано, па затоа искористивме undersampling концепт на балансирање, поточно методот Edited Nearest Neighbors.

Истата импутација ја извршуваме и врз тестирачкото множество. Правиме компаративна анализа меѓу тренирачкото и тестирачкото множество. Нивните дистрибуции за секоја колона (карактеристики) се скоро еднакви:





Модели

Иницијално, правиме два типа на селекција на карактеристики (колони).
Првиот профил од карактеристики се само нумерички колони, додека пак вториот профил од карактеристики се само категориски колони.

first_setup =
['city_development_index', 'experience','company_size','last_new_job','training_hours']

second_setup =
['city','gender','relevent_experience','enrolled_university','education_level','major_discipline',
'company_type']

Првично градиме модели без нагодување на параметрите (ги градиме по default параметри, односно од самата имплементација од библиотеката)

Ознаки:
P- Precision, R-Recall, F- F1 score, AR - AUC ROC score

Класификатори	Прв профил				Втор профил				Сите карактеристики			
	P	R	F	AR	P	R	F	AR	P	R	F	AR
KNearestNeighbors	0.37	0.61	0.48	0.70	0.47	0.57	0.52	0.71	0.37	0.60	0.48	0.69
Naive Bayes	0.44	0.59	0.50	0.69	0.35	0.46	0.40	0.63	0.41	0.66	0.52	0.73
Logistic Regression	0.42	0.60	0.49	0.69	0.36	0.46	0.40	0.62	0.42	**0.67**	0.52	0.74
RandomForest	0.39	**0.64**	0.49	0.72	0.43	**0.69**	**0.53**	0.75	0.42	**0.67**	0.53	0.76
Support Vector Machine	0.49	0.55	**0.52**	0.69	0.39	0.46	0.43	0.65	0.48	0.55	0.51	0.73
Neural Networks	0.42	0.61	0.49	0.70	0.43	0.59	0.50	0.70	0.50	0.61	**0.55**	0.76
XGBoost	**0.53**	0.49	0.51	**0.75**	**0.52**	0.53	0.52	**0.76**	**0.52**	0.57	0.54	**0.77**

(црвена боја - максимална вредност на секоја метрика соодветно)

Можеме да кажеме дека првите два модела (KNN и Naive Bayes) во ниедна метрика, во ниту еден профил од избрани карактериски(колони), нема максимална вредност (најголема) во однос на другите модели.

RandomForest класификаторот дава најдобар Recall во секоја селекција од колони (карактеристики/полиња/features) во споредба со другите резултати. Значи, RandomForest поточно ги лабелира оние кандидати кои ќе бараат нова работа.

Во секоја селекција од колони, XGBoost има најдобри резултати во однос на Precision и тоа одговара на прашањето „Колку од кандидатите за кои е предвидено дека ќе бараат нова работа, всушност навистина ќе бараат нова работа?“. Исто така и во секоја селекција од колони, XGBoost има највисок скор за AUC ROC, што воедно е најдобар модел со сите карактеристики при негово тестирање.

Хипертунирање на најдобрите модели според AUC ROC

Според дадената табела погоре, можеме да видиме најдобри модели според AUC ROC се XGBoost, Neural Networks и RandomForest заедно сите карактеристики од множеството.

Резултати од хипертунирањето на XGBoost модел:

Параметри	Precision	Recall	F1 score	AUC ROC
{eta: 0.01 , gamma: 0 }	0.52	0.57	0.54	0.761
{eta: 0.01 , gamma: 0.5 }	0.52	0.57	0.54	0.761
{eta: 0.01 , gamma: 0.8 }	0.53	0.57	0.55	0.762
{eta: 0.01 , gamma: 1 }	0.53	0.57	0.55	0.762
{eta: 0.01 , gamma: 2 }	0.53	0.57	0.55	0.762
{eta: 0.01 , gamma: 4 }	0.53	0.57	0.55	0.762
{eta: 0.01 , gamma: 5 }	0.53	0.55	0.54	0.761
{eta: 0.01 , gamma: 6 }	0.54	0.55	0.54	0.762
{eta: 0.01 , gamma: 10 }	0.54	0.54	0.54	0.760
{eta: 0.01 , gamma: 100 }	0.59	0.42	0.49	0.742

{eta: 0.1 , gamma: 0 }	0.52	0.57	0.54	0.761
{eta: 0.1 , gamma: 0.5 }	0.52	0.57	0.54	0.761
{eta: 0.1 , gamma: 0.8 }	0.53	0.57	0.55	0.762
{eta: 0.1 , gamma: 1 }	0.53	0.57	0.55	0.762
{eta: 0.1 , gamma: 2 }	0.59	0.42	0.49	0.762
{eta: 0.1 , gamma: 4 }	0.53	0.57	0.55	0.762
{eta: 0.1 , gamma: 10 }	0.54	0.54	0.54	0.760
{eta: 0.1 , gamma: 100 }	0.59	0.42	0.49	0.741

Во решението се тестирани уште неколку комбинации од параметри, но по она што можеме да видиме, дека резултатите не се разликуваат драстично, без разлика на вредностите на параметрите.

Резултати од хипертунирањето на RandomForest модел:

Параметри	**Precision**	**Recall**	**F1 score**	**AUC ROC**
{n_estimators: 100 , criterion: gini , max_depth: 10 }	0.46	0.69	0.55	0.768
{n_estimators: 100 , criterion: gini , max_depth: 15 }	0.44	0.72	0.55	0.766
{n_estimators: 100 , criterion: gini , max_depth: 30 }	0.42	0.72	0.53	0.759
{n_estimators: 100 , criterion: entropy , max_depth: 10 }	0.46	0.69	0.55	0.770
{n_estimators: 100 , criterion: entropy , max_depth: 15 }	0.45	0.71	0.55	0.767

{n_estimators: 100 , criterion: entropy , max_depth: 30 }	0.42	0.72	0.53	0.759
{n_estimators: 1000 , criterion: gini , max_depth: 10 }	0.46	0.70	0.56	0.770
{n_estimators: 1000 , criterion: gini , max_depth: 15 }	0.44	0.72	0.55	0.768
{n_estimators: 1000 , criterion: gini , max_depth: 30 }	0.42	0.73	0.54	0.762
{n_estimators: 1000 , criterion: entropy , max_depth: 10 }	0.47	0.69	0.56	0.7714
{n_estimators: 1000 , criterion: entropy , max_depth: 30 }	0.45	0.71	0.55	0.769
{n_estimators: 5000 , criterion: gini , max_depth: 10 }	0.46	0.69	0.56	0.770
{n_estimators: 5000 , criterion: gini , max_depth: 15 }	0.44	0.72	0.55	0.767
{n_estimators: 5000 , criterion: gini , max_depth: 30 }	0.42	0.74	0.54	0.762
{n_estimators: 5000 , criterion: entropy , max_depth: 15 }	0.47	0.68	0.56	0.7716

Овде можеме да приметиме дека резултатите најчесто се влошуваат кога се зголемува параметарот max_depth. Исто така постои шема во резултатите од следниве комбинации од параметри

Параметри	AUC ROC score
{n_estimators: 100 , criterion: gini , max_depth: 10 }	0.768
{n_estimators: 100 , criterion: entropy , max_depth: 10 }	0.770
{n_estimators: 1000 , criterion: gini , max_depth: 10 }	0.770
{n_estimators: 5000 , criterion: gini , max_depth: 10 }	0.770
{n_estimators: 1000 , criterion: entropy , max_depth: 10 }	0.7714
{n_estimators: 5000 , criterion: entropy , max_depth: 10 }	0.7716

Видливо дека параметарот max_depth = 10, дава подобри резултати во однос на другите вредности за max_depth ja, додека пак со зголемување на n_estimators согледуваме дека така и се градел подобар моделот. Но, подобрувањето е дури во 4та децимала кога n_estimators = 1000 и n_estimators = 5000. Исто така, кога поделбата се врши по критериумот gini, тогаш не се подобруваат резултатите дури и кога n_estimators = 1000 и n_estimators = 5000.

Резултати од хипертунирањето на невронска мрежа:

Параметри	**Precision**	**Recall**	**F1 score**	**AUC ROC**
{hidden_layer_sizes: 128 activation: logistic , solver: lbfgs , batch_size: 15 }	0.41	0.73	0.53	0.738
{hidden_layer_sizes: 128 , activation: logistic , solver: lbfgs ,	0.41	0.72	0.52	0.742

batch_size: 20 }				
{hidden_layer_sizes: 128 , activation: logistic , solver: adam , batch_size: 15 }	0.41	0.73	0.52	0.745
{hidden_layer_sizes: 128 , activation: logistic , solver: adam , batch_size: 20 }	0.40	0.75	0.52	0.749
{hidden_layer_sizes: 128 , activation: tanh , solver: sgd , batch_size: 15 }	0.46	0.54	0.50	0.710
{hidden_layer_sizes: 128 , activation: tanh , solver: sgd , batch_size: 20 }	0.48	0.55	0.51	0.716
{hidden_layer_sizes: (8, 16) , activation: logistic , solver: lbfgs , batch_size: 25 }	0.45	0.67	0.54	0.761
{hidden_layer_sizes: (8, 16) , activation: tanh , solver: sgd , batch_size: 15 }	0.48	0.51	0.49	0.68
{hidden_layer_sizes: (16, 32, 64) , activation: logistic , solver: sgd , batch_size: 20 }	0.49	0.54	0.51	0.718

Заклучок:
Воглавно и после хипертунирање нема некое значително подобрување на резултатите. Многу често подетална обработка на податоците би довело до подобрување на резултатите. Во нашиот проблем поради небалансираноста на таргет класата, имаме и лоши резултати во останатите метрики. Понатаму може да се проба со менување на алгоритмите за балансирање на податоците, менување на методите за импутација, друга селекција на карактеристики или изведување на други колони.