

Big Data II

Memoria de trabajo sobre Apache Spark

Antonio Manuel Milán Jiménez

 antoniomj@correo.ugr.es

Introducción	3
Técnicas de preprocesamiento	3
Clasificadores	6
Parámetros utilizados para preprocesamiento y clasificadores	6
Métricas de acierto utilizadas	7
Resultados obtenidos	7
Decision Tree	7
Support Vector Machine with Stochastic Gradient Descent	10
Random Forest	11
PCARD	11
k-Nearest Neighbors	12
Conclusión	13
Paquetes de Apache Spark utilizados	14

Introducción

En este trabajo se propone trabajar sobre la base de datos ‘Susy’ aplicando diferentes técnicas de preprocessamiento y clasificación sobre Apache Spark. Se trata de un problema de clasificación binario (clases 0.0 y 1.0) con 18 características y un millón de instancias, tanto el conjunto de datos de entrenamiento como el de test, sin presentar valores perdidos ninguno de ellos.

Quizás lo más característico de esta base de datos es que presenta un fuerte desbalance a razón de 90%-10%, es decir, se tienen 900000 instancias para la clase negativa (1.0) y únicamente 100000 instancias de la clase positiva (0.0). Este hecho hace que seguramente sea necesario aplicar alguna técnica de balanceado de clases para que el modelo pueda aprender correctamente ambas clases.

Técnicas de preprocessamiento

Tal y como se acaba de comentar, será prácticamente “obligatorio” aplicar alguna técnica de balanceado de clases. Así, se propone emplear **Random Undersampling**, haciendo que para ambas clases se tengan 100000 instancias, u **Random Oversampling**, aumentando el número de instancias de la clase minoritaria y hacerla más equitativa. En la siguiente tabla se muestran diferentes ratios de balanceo de Oversampling que podrían ser interesantes junto con el número de instancias finales para cada clase:

Ratio de Oversampling	Nº instancias positivas	Nº instancias negativas
0%	100000	900000
100%	899724	900000
90%	809830	900000
80%	719644	900000
65%	584114	900000
50%	448506	900000
105%	945026	900000
115%	1034987	900000
130%	1169998	900000

A continuación se propone utilizar un **filtrado de ruido**, instancias anómalas respecto a los valores de sus características o clasificación que propician que el modelo aprenda sobre datos posiblemente erróneos y le hagan equivocarse. Por lo tanto, una vez que se han balanceado las clases y seguramente haya un mayor ruido en los datos si ya en los originales se presentaban instancias anómalas, es interesante eliminar todas estas instancias.

Así, se utilizará el método **HME_BD** para realizar este filtrado de ruido. En la siguiente tabla se muestran el número de instancias detectadas como ruido para diferentes balances previos:

Balance	Nº Instancias eliminadas	Prop. Instancias eliminadas	Nº Instancias finales
None	96908	9.69%	903092
Undersampling	42348	21.17%	157652
Oversampling 100%	369134	20.51%	1430590
Oversampling 90%	355162	20.77%	1354668
Oversampling 80%	338304	20.88%	1281340
Oversampling 65%	309381	20.84%	1174733
Oversampling 50%	274875	20.38%	1073631
Oversampling 105%	374996	20.32%	1470030
Oversampling 115%	386939	19.99%	1548048
Oversampling 130%	402721	19.45%	1667277

Dentro de lo esperado, en los datos que no se ha realizado ningún tipo de balanceo es donde menos instancias se detectan como ruido, no llegando a alcanzar el 10%. Respecto al Undersampling se encuentra el mayor porcentaje de ruido detectado, 21.17%, explicándose con que muchas de las instancias que entonces eran de la clase mayoritaria se han eliminado, propiciando que otras instancias más “extremas” ahora estén más aisladas y sí se detecten como anomalías. Por último, de los datos con Oversampling se detecta en torno a un 20% como ruido, lo cual se debe a que también se habrá realizado Oversampling sobre instancias ruidosas.

La siguiente técnica propuesta es realizar una **selección de instancias**. Dado el alto número de instancias que se presentan, es probable que un número considerable de ellas estén aportando información redundante, sobre todo si se ha realizado Oversampling.

Por lo tanto, sería interesante eliminar todos estos datos que no aportan nueva información útil al modelo, haciendo que sea menos eficiente. De esta forma, se propone utilizar el método **FCNN_MR**, mostrando en la siguiente tabla la proporción de instancias eliminadas:

Nº instancias (balanceo + filtrado de ruido)	Nº Instancias eliminadas	Prop. Instancias eliminadas	Nº Instancias finales
903092 (None + HME_BD)	872072	96.56%	31020
157652 (Under + HME_BD)	83493	52.96%	74159
1430590 (100% + HME_BD)	907103	63.40%	523487
1354668 (90% + HME_BD)	844580	62.34%	510088
1281340 (80% + HME_BD)	791297	61.75%	490043
1174733 (65% + HME_BD)	716912	61.02%	457821
1073631 (50% + HME_BD)	671912	62.58%	401719
1470030 (105% + HME_BD)	939224	63.89%	530806
1548048 (115% + HME_BD)	1009461	65.20%	538587
1667277 (130% + HME_BD)	1111450	66.66%	555827

Empezando por los datos sobre los que se había realizado Oversampling y posteriormente un filtrado de ruido, para todos ellos se han detectado más del 60% como información irrelevante, lo cual muestra que se había creado una gran cantidad de información redundante y con este filtrado se conseguirá mejorar notablemente la eficiencia de los clasificadores que trabajen sobre ellos. Respecto a los datos sobre los que se había realizado Undersampling, se ha detectado como instancias irrelevantes más del 50%; obteniendo finalmente un conjunto de datos de tan solo 83493 observaciones de más calidad que su original de un millón de instancias. Por último, sorprende que del ‘dataset’ que no había sufrido ningún balanceo se haya detectado más del 96%, lo cual puede indicar que al tratarse de unos datos tan desbalanceados se estén detectando muchas instancias de la clase mayoritaria sin importancia.

Una vez que se han realizado todas estas técnicas de preprocesamiento que afectan directamente a la elección de las instancias, es interesante utilizar ahora alguna técnica de **selección de características**, determinando aquellas características que más información útil aportan al modelo. Una vez realizado esto se lograría un conjunto de datos bastante más reducido que el original y de mayor calidad (Smart Data), ayudando directamente a la eficacia del clasificador y a su comportamiento.

Por lo tanto se propone utilizar el método **PCARD**, el cuál se trata de un ‘ensemble’ del PCA (Análisis de componentes principales) y de un discretizador aleatorio. Además, este método incluye directamente un Árbol de Decisión para realizar ya la clasificación de los datos.

Adicionalmente mencionar que, en el caso de utilizar como clasificador kNN (vecino más cercano), será muy recomendable emplear alguna **normalización** sobre los datos (concretamente **MinMaxScaler**) para evitar que el rango de los valores de las características influyan en las distancias que se contemplan en el clasificador y sean más relevantes unas que otras simplemente por este hecho.

Clasificadores

Respecto a los clasificadores con los que se construirán los modelos se encuentra **Decision Tree** (tanto por sí solo como en PCARD), **Random Forest**, **kNN** (vecino más cercano) y **SVM SGD**(Support Vector Machine con Descenso de Gradiente Estocástico).

Por lo tanto se han utilizado diferentes clasificadores con la idea de intentar abarcar diferentes naturalezas de los mismos y observar cómo se comportan cada uno de ellos trabajando con el conjunto de datos.

Parámetros utilizados para preprocesamiento y clasificadores

A continuación se muestran para las diferentes técnicas de preprocesamiento y los diferentes clasificadores los parámetros utilizados:

- **Random Undersampling**: nº partitions → 250
- **Random Oversampling**: nº partitions → 100, nº repartitions → 250, oversampling rate → [100,90,80,65,50,105,115,130]
- **HME_BD**: nTrees → 100, maxDepthRF → 10, partitions → 4
- **FCNN_MR**: k → 3
- **MinMaxScaler**: min → 0, max → 1
- **PCARD**: cuts → 5, trees → 10
- **Decision Tree**: impurity → “gini”, maxDepth → 20, maxBins → 32
- **Random Forest**: numTrees → 100, featureSubsetStrategy → “auto”, impurity → “gini”, maxDepth → 4, maxBins → 32
- **SVM SGD**: numIterations → 100
- **kNN**: dist → “euclidean”, numPartitionMap → 128, numReduces → 32, numIterations → 1, maxWeight → 5, k → [5,7,9,15,21]

Métricas de acierto utilizadas

Al trabajar con unos datos tan desbalanceados resulta más importante todavía qué métricas se van a utilizar para medir la bondad de los modelos que se estén construyendo.

Por ejemplo, la medida clásica de '**Accuracy**' otorgaría un acierto del 90% a un modelo que siempre optase por la clase negativa, demostrándose que no es una medida que muestre realmente lo bien o mal que lo haga el clasificador para este problema desbalanceado.

Una medida más acertada sería **AUC**, es decir, el área bajo la curva ROC que sí contempla la presencia de falsos positivos. Sin embargo, sería de interés que se contemplasen por separado el acierto para la clase negativa así como para la positiva (True Positive Rate y True Negative Rate), de forma que el producto de ambos ratios sea un indicador directo de cómo de bien se han aprendido y se está clasificando para ambas clases. Al contrario que con '**Accuracy**', TNR en ese caso mostraría un acierto del 100% mientras que TPR sería un acierto del 0%, de forma que al hacer el producto se obtendría un acierto total del 0%, mostrando que realmente no se ha aprendido nada sobre ambas clases.

Por lo tanto, aunque también se calculen las otras métricas, se tomará como medida de acierto real **TPR*TNR**; ayudando además a saber en cada momento si se está aprendiendo más o menos lo mismo o no para ambas clases, algo que no muestran las otras medidas.

Resultados obtenidos

A continuación se muestran y comentan los resultados que se han ido obteniendo por parte de los diferentes clasificadores, observando cómo se comportan las técnicas de preprocesamiento y qué decisiones se han ido tomando.

Decision Tree

Para empezar se ha utilizado como clasificador Decision Tree y se han realizado diferentes combinaciones de las técnicas de preprocesamiento mencionadas para observar cómo evoluciona el acierto del modelo:

Sampling	Noise Filter	Instances Selection	Classifier	Accuracy	Area Under Curve ROC	TPR	TNR	TPR*TNR
None	None	None	Decision Tree	88.91%	67.15%	28.38%	95.64%	27.14%
Oversampling 100%	None	None	Decision Tree	79.7%	62.01%	67.28%	81.08%	54.55%
None	HME_BD	None	Decision Tree	90.32%	76.67%	7.92%	99.47%	7.88%
None	None	FCNN_MR	Decision Tree	83.08%	58.79%	34.94%	88.43%	30.90%
None	HME_BD	FCNN_MR	Decision Tree	90.04%	71.01%	11.12%	98.81%	10.99%
Oversampling 100%	HME_BD	None	Decision Tree	76.72%	62.41%	81..17%	76.22%	61.87%
Oversampling 100%	None	FCNN_MR	Decision Tree	82.61%	61.79%	52.55%	85.95%	45.17%
Oversampling 100%	HME_BD	FCNN_MR	Decision Tree	76.59%	62.3%	80.85%	76.11%	61.54%

Tal y como se comentó en la sección anterior, observando por ejemplo el primer resultado, se obtiene un ‘Accuracy’ de 88.91% que a priori podría parecer un resultado bastante bueno. Sin embargo, observando las métricas de TPR y TNR se descubre que TPR apenas alcanza el 28%, indicando que no se está aprendiendo sobre la clase positiva, la clase minoritaria en este caso. Por lo tanto la métrica de TPR*TNR, que alcanza en este caso 27.14%, es una buena medida que refleja en general cómo de bien o mal se está comportando el modelo.

Siguiendo con esta idea y observando el resto de resultados, parece claro que balancear las clases es prácticamente “obligatorio” para poder aprender tambien sobre la clase positiva y conseguir así un TPR y TNR más o menos equivalentes, que se traducirá en un acierto final mayor.

Respecto al filtrado de ruido realizado, parece interesante aplicarlo pues cuando se ha utilizado sobre los datos balanceados se incrementa notablemente el acierto final hasta el 61.87%.

Respecto a la selección de instancias realizada, considerando únicamente el acierto obtenido, llega a ser contraproducente utilizarla sobre los datos ya balanceados y se obtiene prácticamente el mismo acierto al utilizarla sobre los datos balanceados y tras el filtrado de ruido (61.54%). No obstante, sigue siendo interesante aplicarla pues se consigue que la construcción del clasificador sea más eficiente al no tener que trabajar con información irrelevante e incluso puede llegar a conseguir que un modelo se comporte mejor por este hecho.

Visto el gran incremento en el acierto al balancear las clases, parece interesante seguir utilizando otras técnicas de balanceo, tales como Undersampling.

Además, observando por ejemplo el TPR y TNR del mejor acierto obtenido, 81.17% y 76.22% respectivamente, sí que parece una buena idea como se mencionó el emplear otros ratios en Oversampling. Se trataría de conseguir que se asemejen más todavía el acierto para cada clase y lograr un acierto final mayor. Siguiendo con esta idea, se van a emplear otros ratios de Oversampling para observar su comportamiento:

Sampling	Noise Filter	Instances Selection	Classifier	Accuracy	Area Under Curve ROC	TPR	TNR	TPR*TNR
Undersampling	HME_BD	None	Decision Tree	74.20%	61.70%	84.03%	73.11%	61.44%
Undersampling	HME_BD	FCNN_MR	Decision Tree	74.43%	61.70%	83.34%	73.44%	61.21%
Oversampling 90%	HME_BD	None	Decision Tree	78.21%	62.86%	78.94%	78.13%	61.68%
Oversampling 90%	HME_BD	FCNN_MR	Decision Tree	77.94%	62.71%	78.94%	77.83%	61.44%
Oversampling 80%	HME_BD	None	Decision Tree	79.55%	63.26%	76.42%	79.90%	61.06%
Oversampling 80%	HME_BD	FCNN_MR	Decision Tree	79.25%	63.10%	76.53%	79.56%	60.89%
Oversampling 65%	HME_BD	None	Decision Tree	81.44%	63.92%	72.46%	82.43%	59.74%
Oversampling 65%	HME_BD	FCNN_MR	Decision Tree	81.01%	63.68%	72.84%	81.92%	59.67%
Oversampling 50%	HME_BD	None	Decision Tree	83.98%	64.99%	65.43%	86.04%	56.30%
Oversampling 50%	HME_BD	FCNN_MR	Decision Tree	83.61%	64.70%	65.75%	85.59%	56.28%
Oversampling 105%	HME_BD	None	Decision Tree	75.54%	62.06%	82.57%	74.76%	61.73%
Oversampling 105%	HME_BD	FCNN_MR	Decision Tree	75.35%	61.99%	82.63%	74.54%	61.59%
Oversampling 115%	HME_BD	None	Decision Tree	74.15%	61.73%	84.38%	73.01%	61.61%
Oversampling 115%	HME_BD	FCNN_MR	Decision Tree	74.04%	61.65%	84.17%	72.91%	61.37%
Oversampling 130%	HME_BD	None	Decision Tree	73.62%	61.60%	84.94%	72.37%	61.47%
Oversampling 130%	HME_BD	FCNN_MR	Decision Tree	73.78%	61.62%	84.69%	72.57%	61.46%

Observando los diferentes resultados obtenidos, se descubren resultados bastante próximos al mejor resultado hasta el momento que se consiguió con un Oversampling de 100% (por ejemplo con un 105% se obtiene un acierto de 61.73%). Al utilizar un

Oversampling del 90% y el filtrado de ruido se consigue que TPR y TNR sea muy similares tal y como se buscaba (78.94 y 78.13 respectivamente).

Sin embargo, no son lo suficientemente altos para conseguir un resultado final mejor, alcanzando un 61.68%, nuevamente muy próximo al 61.87% logrado anteriormente.

Así, a la vista de estos experimentos con Oversampling y los resultados conseguidos, no parece que sea relevante el utilizar un ratio mayor o inferior, logrando el mejor resultado cuando ambas clases están totalmente balanceadas.

Respecto al Undersampling realizado, también se logra un resultado similar (hasta un 61.44%) por lo que esta técnica sí sería interesante seguir utilizandola pues alcanza un resultado muy próximo al mejor con un número de instancias mucho menor, siendo más eficiente para los clasificadores como se comentó anteriormente.

Support Vector Machine with Stochastic Gradient Descent

A continuación se muestran los resultados obtenidos con Support Vector Machine empleando Stochastic Gradient Descent (SVM SGD). En base a los resultados estudiados del anterior clasificador, se han guiado los experimentos que podrían ser más interesantes:

Sampling	Noise Filter	Instances Selection	Classifier	Accuracy	Area Under Curve ROC	TPR	TNR	TPR*TNR
Oversampling 100%	HME_BD	None	SVM SGD	65.43%	59.53%	88.90%	62.82%	55.85%
Oversampling 100%	HME_BD	FCNN_MR	SVM SGD	71%	60.33%	82.97%	69.67%	57.80%
Undersampling	HME_BD	None	SVM SGD	66.17%	59.62%	88.19%	63.72%	56.20%
Undersampling	HME_BD	FCNN_MR	SVM SGD	70.69%	60.34%	83.77%	69.23%	58%

En general, los resultados obtenidos con SVM son ligeramente inferiores a los que se obtuvieron con Decision Tree. Observando por separado TPR y TNR, cuando únicamente se ha utilizado el filtrado de ruido, se ha aprendido bastante bien sobre la clase positiva (88.9% y 88.19%), aunque en detrimento del aprendizaje de la clase negativa (62.82% 63.72%). Por lo tanto, sería interesante emplear un Oversampling con 80% u 65%, buscando que se aprendiese más sobre la clase negativa y se “desaprendiese” lo mínimo posible sobre la positiva; logrando un mayor equilibrio que quizás se convierta en un mejor resultado:

Sampling	Noise Filter	Instances Selection	Classifier	Accuracy	Area Under Curve ROC	TPR	TNR	TPR*TNR
Oversampling 90%	HME_BD	None	SVM SGD	69.02%	60.05%	85.50%	67.18%	57.45%
Oversampling 80%	HME_BD	None	SVM SGD	72.96%	60.65%	80.27%	72.15%	57.92%
Oversampling 65%	HME_BD	None	SVM SGD	78.51%	61.66%	69.61%	79.50%	55.34%

Se descubre que, aunque se consigue que se equilibren más las tasas de acierto para ambas clases, decrece demasiado la positiva y no se consigue mejorar la tasa de acierto final.

Random Forest

Estos son los resultados que se han obtenido por el clasificador Random Forest:

Sampling	Noise Filter	Instances Selection	Classifier	Accuracy	Area Under Curve ROC	TPR	TNR	TPR*TNR
Oversampling 100%	HME_BD	None	Random Forest	72.76%	60.79%	81.72%	71.77%	58.65%
Oversampling 100%	HME_BD	FCNN_MR	Random Forest	72.14%	60.71%	82.77%	70.95%	58.73%
Undersampling	HME_BD	None	Random Forest	72.53%	60.77%	82.22%	71.46%	58.75%
Undersampling	HME_BD	FCNN_MR	Random Forest	73.63%	61.10%	81.54%	72.75%	59.32%

Los resultados muestran que Random Forest se comporta ligeramente peor que Decision Tree y algo mejor que SVM SGD, con acierto en torno a 58%-59%. Además, vuelve a suceder que se está aprendiendo mejor sobre la clase positiva que sobre la negativa.

PCARD

Visto anteriormente el comportamiento por parte del clasificador Decision Tree, se muestran ahora los resultados obtenidos con PCARD, el cuál realizaba un ensemble de PCA y un

discretizador aleatorio sobre los datos ya preprocesados para finalmente emplear un clasificador Decision Tree.

Además, dado que vuelve a suceder que hay una amplia diferencia entre TPR y TNR, se ha experimentado también con unos datos no balanceados completamente:

Sampling	Noise Filter	Instances Selection	Classifier	Accuracy	Area Under Curve ROC	TPR	TNR	TPR*TNR
Undersampling	HME_BD	None	PCARD	72.33%	77.63%	84.26%	71%	59.83%
Oversampling 100%	HME_BD	None	PCARD	71.26%	77.45%	85.20%	69.71%	59.39%
Oversampling 80%	HME_BD	None	PCARD	75.39%	77.27%	79.62%	74.92%	59.65%

Con PCARD se obtienen unos valores muy cercanos al 60% de acierto final. Se observa además que al utilizar Oversampling con 80% e intentar equilibrar las tasas de aciertos, no se consigue una mejora significativa en TPR*TNR pues, aunque se ha aumentado el acierto sobre la clase negativa, se ha “desaprendido” a la vez sobre la clase positiva.

k-Nearest Neighbors

Por último se muestran los resultados obtenidos por kNN (k-nearest neighbors). Recordar que para un mejor comportamiento por parte del modelo se ha aplicado este clasificador sobre los datos normalizados tras el preprocesamiento:

Sampling	Noise Filter	Instances Selection	Classifier	k	Accuracy	Area Under Curve ROC	TPR	TNR	TPR*TNR
Oversampling 100%	HME_BD	None	kNN	5	74.49%	77.75%	81.83%	73.67%	60.28%
Undersampling	HME_BD	None	kNN	5	70.58%	77.68%	86.56%	68.80%	59.56%
Oversampling 100%	HME_BD	FCNN_MR	kNN	5	74.55%	78.01%	82.33%	73.69%	60.67%
Undersampling	HME_BD	FCNN_MR	kNN	5	72.68%	78.21%	85.13%	71.29%	60.69%

Además, se ha experimentado empleando otras “k” para estudiar si mejoraba o no el comportamiento del clasificador el considerar un número diferente de vecinos:

Sampling	Noise Filter	Instances Selection	Classifier	k	Accuracy	Area Under Curve ROC	TPR	TNR	TPR*TNR
Oversampling 100%	HME_BD	None	kNN	3	76.32%	77.42%	78.80%	76.05%	59.93%
Oversampling 100%	HME_BD	None	kNN	7	73.19%	77.81%	83.60%	72.03%	60.22%
Oversampling 100%	HME_BD	None	kNN	9	72.32%	77.83%	84.71%	70.95%	60.10%
Oversampling 100%	HME_BD	None	kNN	15	71.16%	77.78%	86.05%	69.50%	59.81%
Oversampling 100%	HME_BD	None	kNN	21	71.08%	77.84%	86.29%	69.39%	59.88%
Oversampling 100%	HME_BD	None	kNN	27	70.95%	77.90%	86.59%	69.21%	59.93%
Oversampling 100%	HME_BD	None	kNN	41	70.59%	77.90%	87.03%	68.77%	59.85%
Oversampling 100%	HME_BD	FCNN_MR	kNN	7	73.78%	78.06%	83.41%	72.70%	60.65%
Oversampling 100%	HME_BD	FCNN_MR	kNN	9	73.42%	78.11%	83.97%	72.25%	60.67%
Oversampling 100%	HME_BD	FCNN_MR	kNN	15	73.14%	78.26%	84.67%	71.86%	60.85%
Oversampling 100%	HME_BD	FCNN_MR	kNN	21	73.13%	78.44%	85.07%	71.80%	61.09%

Respecto a los resultados obtenidos utilizando el clasificador kNN, se vuelven a conseguir unos resultados muy similares al resto de clasificadores. Así, el mayor acierto conseguido ha sido de 61.09% utilizando Oversampling con filtrado de ruido y selección de instancias con k=21. Además, utilizando otras “k” no se han conseguido cambios relevantes, no variando los resultados en más de 0.5%..

Conclusión

Como conclusión final mencionar que la mejor tasa TPR*TNR conseguida ha sido de 61.87% utilizando Decision Tree y trabajando sobre unos datos balanceados con Oversampling 100% y filtrando el ruido con HME_BD.

Al incluir además la selección de instancias para eliminar aquellas observaciones redundantes, se consiguen aciertos muy similares con la ventaja de trabajar con un conjunto de datos más pequeños y logrando que sea más eficiente el aprendizaje del modelo.

Respecto al balanceo de clases, se ha probado la necesidad de realizarlo pues de lo contrario no se aprendía sobre la clase minoritaria y realmente no se estaba logrando un buen comportamiento por parte del modelo.

También relacionado con esta técnica, Undersampling ha mostrado en general un acierto muy próximo a los mejores con la ventaja de utilizar un conjunto de datos mucho más pequeño, por lo que se convierte en una opción muy interesante si lo que se quiere es primar a la eficiencia.

Paquetes de Apache Spark utilizados

A continuación se detallan los diferentes paquete utilizados de Apache Spark:

- **Imb-sampling-ROS_and_RUS**: Random Oversampling, Random Undersampling
- **NoiseFramework**: HME_BD
- **SmartReduction**: FCNN_MR
- **MLlib.feature**: MinMaxScaler
- **PCARD**: PCARD
- **MLlib.tree**: Decision Tree, Random Forest
- **MLlib.classification**: SVM with SGD, kNN