Feature Hashing for Scalable Machine Learning

Nick Pentreath, IBM

#EUds15

About

- About me
  - @MLnick
  - Principal Engineer at IBM working on machine learning & Apache Spark
  - Apache Spark committer & PMC
  - Author of *Machine Learning with Spark*

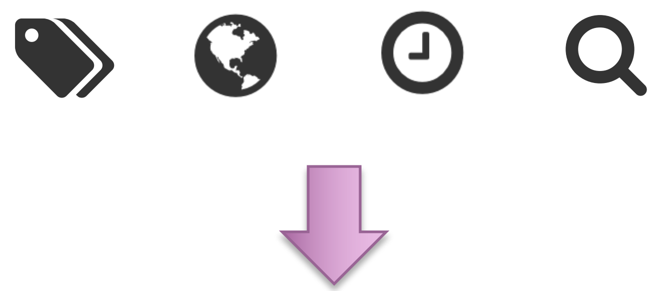
Agenda

- Intro to feature hashing
- `HashingTF` in Spark ML
- `FeatureHasher` in Spark ML
- Experiments
- Future Work

Feature Hashing

Encoding Features

- Most ML algorithms operate on numeric feature vectors
- Features are often categorical – even numerical features (e.g. binning continuous features)



$$[2.1 \quad 3.2 \quad -0.2 \quad 0.7]$$

Encoding Features

- “one-hot” encoding is popular for categorical features
- “bag of words” is popular for text (or *token counts* more generally)

$$[0 \quad 0 \quad 1 \quad 0 \quad 0]$$

$$[0 \quad 2 \quad 0 \quad 1 \quad 1]$$

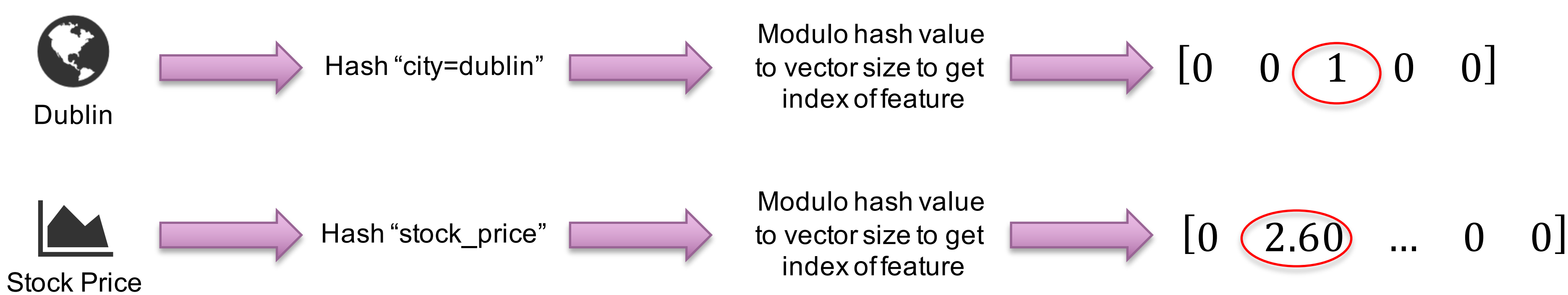
High Dimensional Features

- Many domains have very high *dense* feature dimension (e.g. images, video)
- Here we're concerned with *sparse* feature domains, e.g. online ads, ecommerce, social networks, video sharing, text & NLP
- Model sizes can be very large even for simple models

`[1,...,1,...,1,...,1,...,1,...,3.14,...1,...]`

The “Hashing Trick”

- Use a *hash function* to map feature values to indices in the feature vector



Feature Hashing: Pros

- Fast & Simple
- Preserves sparsity
- Memory efficient
  - Limits feature vector size
  - No need to store mapping feature name -> index
- Online learning
- Easy handling of missing data
- Feature engineering

Feature Hashing: Cons

- No inverse mapping => cannot go from feature indices back to feature names
  - Interpretability & feature importances
  - But similar issues with other dim reduction techniques (e.g. random projections, PCA, SVD)
- Hash collisions …
  - Impact on accuracy of feature collisions
  - Can use signed hash functions to alleviate part of it

HashingTF in Spark ML

HashingTF Transformer

- Transforms text (sentences) -> term frequency vectors (aka “bag of words”)
- Uses the “hashing trick” to compute the feature indices
- Feature value is term frequency (*token count*)
- Optional parameter to only return binary *token occurrence* vector

HashingTF Transformer

```
val tokenizer = new Tokenizer().setInputCol("text").setOutputCol("terms")
val hashingTf = new HashingTF().setInputCol("terms").setOutputCol("features")
val pipeline = new Pipeline().setStages(Array(tokenizer, hashingTf))
val model = pipeline.fit(df)
```

```
+-------------------+------------------------+----------------------------------------------------+
|text               |terms                   |features                                            |
+-------------------+------------------------+----------------------------------------------------+
|The quick brown fox|[the, quick, brown, fox]|(262144,[22323,38208,103838,129637],[1.0,1.0,1.0,1.0])|
|jumps over         |[jumps, over]           |(262144,[179832,252565],[1.0,1.0])                  |
|the lazy dog       |[the, lazy, dog]        |(262144,[51504,75919,103838],[1.0,1.0,1.0])         |
+-------------------+------------------------+----------------------------------------------------+
```

Hacking HashingTF

```scala
val stringsDF = df.select(inputCols: _*).rdd.map { case row =>
  val seq = inputCols.map { colName =>
    val value = row.getString(row.fieldIndex(colName))
    s"$colName=$value"
  }
  (row.getInt(0), seq)
}.toDF("label", "raw")
val hashingTf = new HashingTF().setInputCol("raw").setOutputCol("features")
```

label	i1	i2	i3	i4	i5	i6	i7	i8	i9
0	1	1	5	0	1382	4	15	2	181
0	2	0	44	1	102	8	2	2	4
0	2	0	1	14	767	89	4	2	245
0	NULL	893	NULL	NULL	4392	NULL	0	0	0
0	3	-1	NULL	0	2	0	3	0	0

label	raw	features
0	[i1=1, i2=1, i3=5...	(262144,[2411,726...
0	[i1=2, i2=0, i3=4...	(262144,[5352,934...
0	[i1=2, i2=0, i3=1...	(262144,[14069,15...
0	[i1=NULL, i2=893,...	(262144,[4201,693...
0	[i1=3, i2=-1, i3=...	(262144,[6935,140...

- `HashingTF` can be used for categorical features…
- … but doesn't fit neatly into Pipelines

FeatureHasher in Spark ML

FeatureHasher

- Flexible, scalable feature encoding using *hashing trick*
- Support multiple input columns (numeric or string / boolean, i.e. categorical)
- *One-shot* feature encoder
- Core logic similar to `HashingTF`

FeatureHasher

- Operates on entire Row

```
val hashFeatures = udf { row: Row =>
  val map = new OpenHashMap[Int, Double]()
  localInputCols.foreach { colName =>
```

- … UDF with struct input

```
dataset.select(..., hashFeatures(struct($(inputCols))))
```

FeatureHasher

```
val value = getDouble(row.get(fieldIndex))
val hash = hashFunc(colName)
(hash, value)

val value = row.get(fieldIndex).toString
val fieldName = s"$colName=$value"
val hash = hashFunc(fieldName)
(hash, 1.0)
```

- Determining feature index
  - Numeric: feature name
  - String / boolean: “feature=value”
- String encoding => effectively “one hot” with indicator value of `1.0`

FeatureHasher

- Modulo raw index to feature vector size
- Hash collisions are *additive* (currently)

```
val idx = Utils.nonNegativeMod(rawIdx, n)
map.changeValue(idx, value, v => v + value)

Vectors.sparse(n, map.toSeq)
```

FeatureHasher

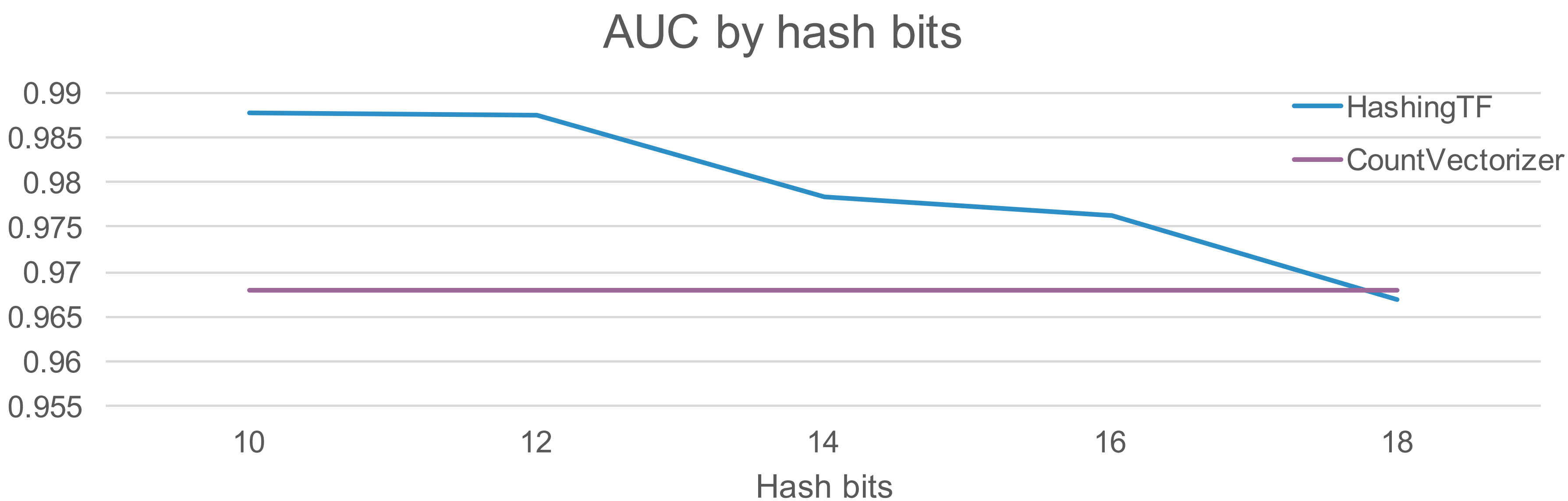
city	country	doubles	ints
Dublin	IE	2.5	5
San Francisco	US	3.5	2
Cape Town	ZA	1.2	3

city	country	doubles	ints	hashedFeatures
Dublin	IE	2.5	5	(262144,[101935,110123,221251,252209],[1.0,2.5,5.0,1.0])
San Francisco	US	3.5	2	(262144,[104522,110123,155525,221251],[1.0,3.5,1.0,2.0])
Cape Town	ZA	1.2	3	(262144,[24936,110123,166603,221251],[1.0,1.2,1.0,3.0])

Experiments

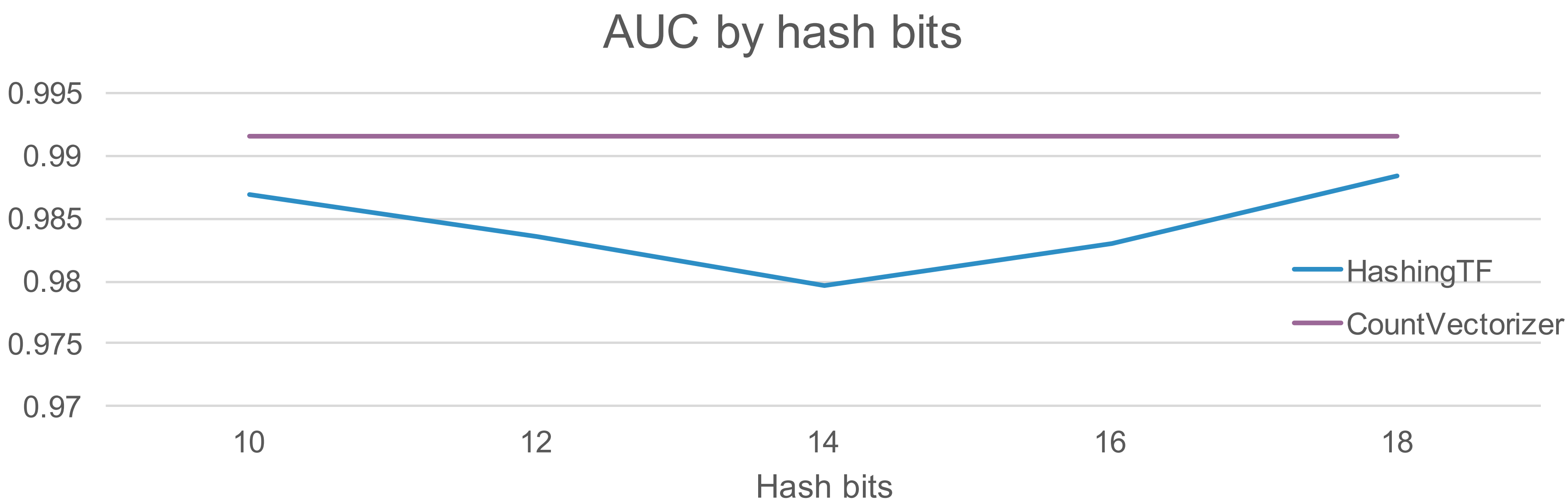
Text Classification

- Kaggle Email Spam Dataset



Text Classification

- Adding regularization (regParam=0.01)

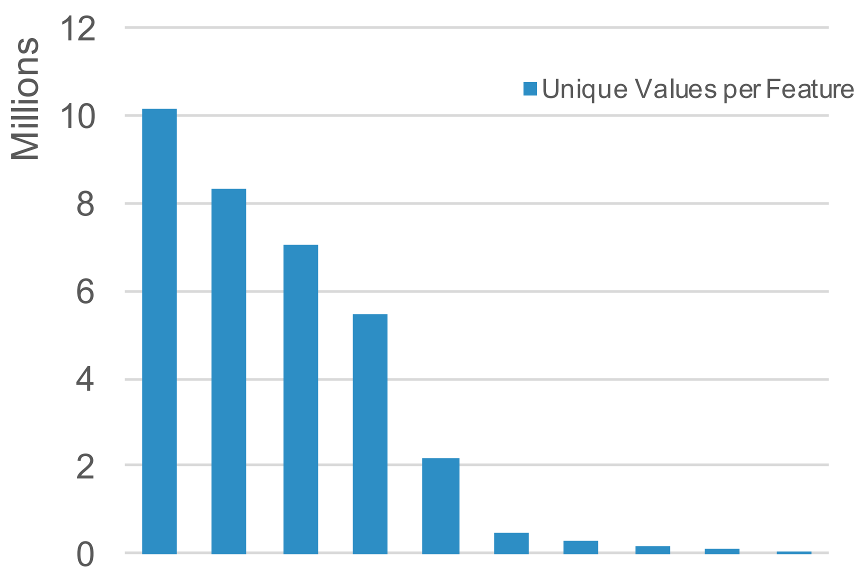


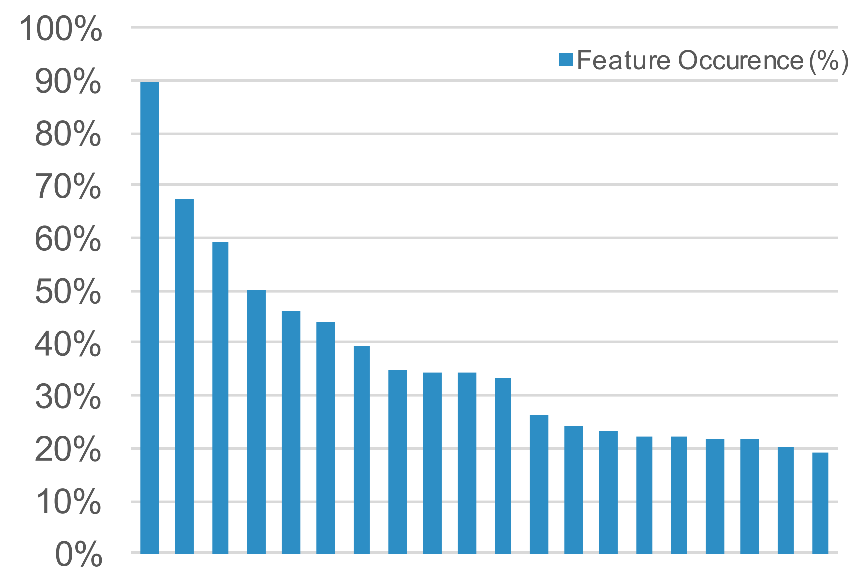
Ad Click Prediction

- Criteo Display Advertising Challenge
  - 45m examples, 34m features, 0.000003% sparsity
- Outbrain Click Prediction
  - 80m examples, 15m features, 0.000007% sparsity
- Criteo Terabyte Log Data
  - 7 day subset
  - 1.5b examples, 300m feature, 0.0000003% sparsity

Data

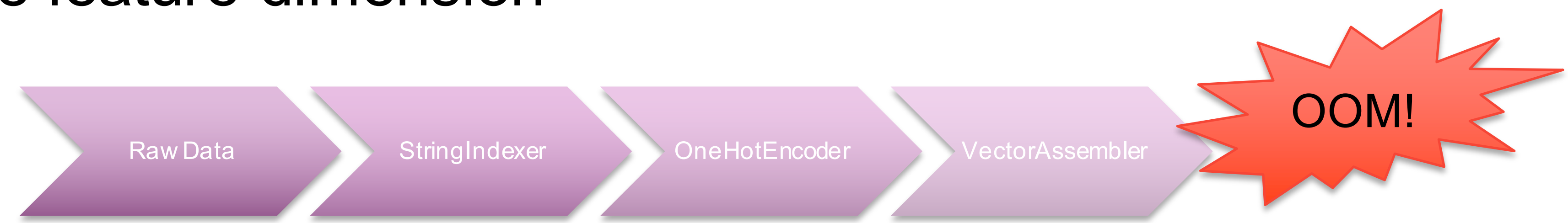
- Illustrative characteristics - Criteo DAC





Challenges

- Typical one-hot encoding pipeline fails consistently with large feature dimension

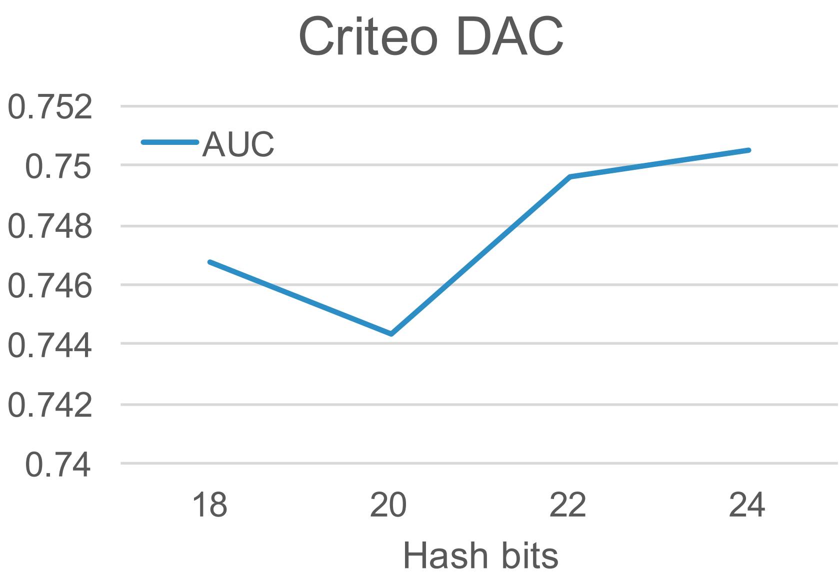


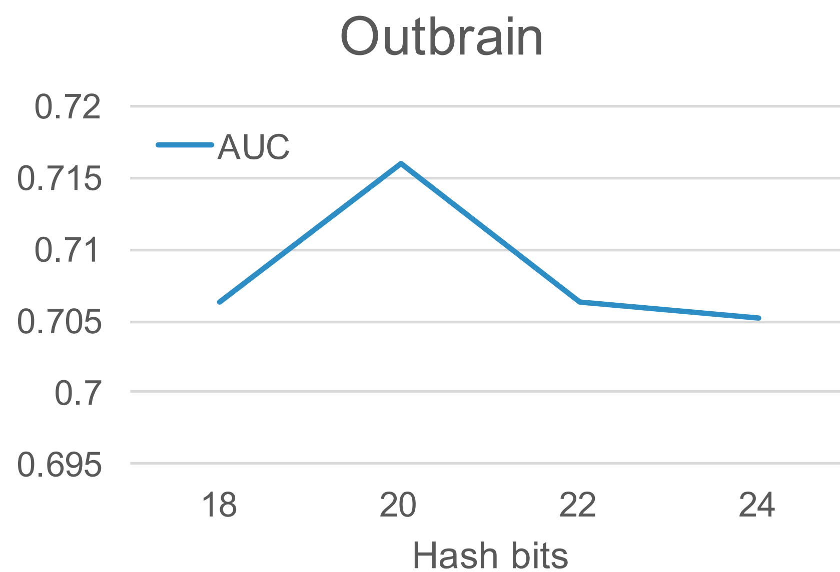
label	i1	i2	i3	i4	i5	i6	i7	i8	i9
0	1	1	5	0	1382	4	15	2	181
0	2	0	44	1	102	8	2	2	4
0	2	0	1	14	767	89	4	2	245
0	NULL	893	NULL	NULL	4392	NULL	0	0	0
0	3	-1	NULL	0	2	0	3	0	0

label	i1_idx	i1_ohe	features
0	2.0	(152,[2],[1.0])	(273492,[2,153,28...
0	3.0	(152,[3],[1.0])	(273492,[3,152,28...
0	3.0	(152,[3],[1.0])	(273492,[3,152,28...
0	0.0	(152,[0],[1.0])	(273492,[0,923,28...
0	4.0	(152,[4],[1.0])	(273492,[4,154,28...

Results

- Compare AUC for different # hash bits





Results

- Criteo 1T logs – 7 day subset
- Can train model on 1.5b examples
- 300m original features for this subset
- 2^{24} hashed features (16m)
- Impossible with current Spark ML (OOM, 2Gb broadcast limit)

Summary & Future Work

Summary

- Feature hashing is a fast, efficient, flexible tool for feature encoding
- Can scale to high-dimensional sparse data, without giving up much accuracy
- Supports multi-column “one-shot” encoding
- Avoids common issues with Spark ML Pipelines using `StringIndexer` & `OneHotEncoder` at scale

Future Directions

- Will be part of Spark ML in 2.3.0 release (Q4 2017)
  - Refer to SPARK-13969 for details
- Allow users to specify set of numeric columns to be treated as categorical
- Signed hash functions
- Internal feature crossing & namespaces (ala Vowpal Wabbit)
- `DictVectorizer`-like transformer => one-pass feature encoder for multiple numeric & categorical columns (with inverse mapping) – see SPARK-19962

References

- Hash Kernels
- Feature Hashing for Large Scale Multitask Learning
- Vowpal Wabbit
- Scikit-learn

Thank You.

@MLnick

spark.tc