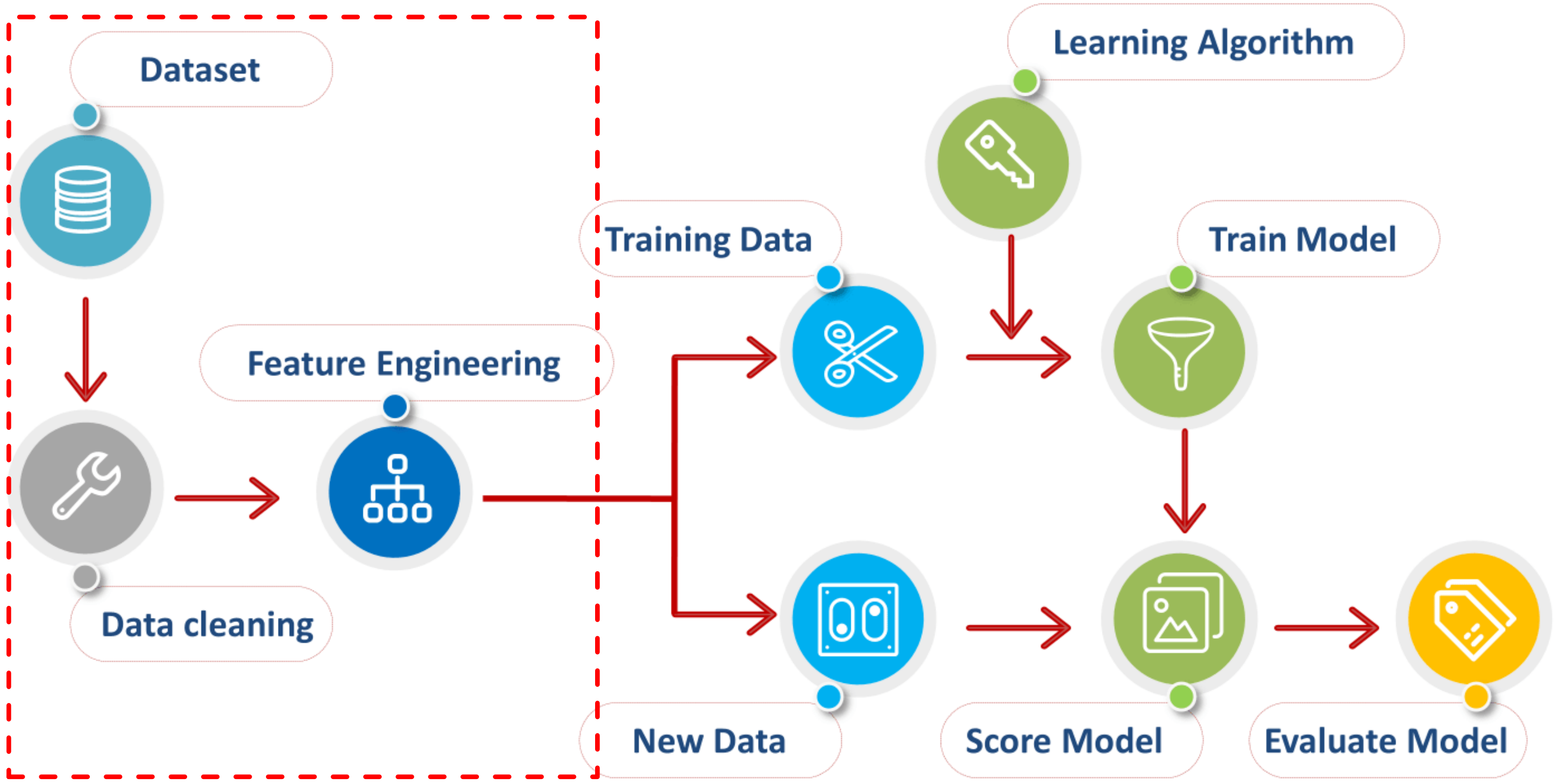
THE STATE OF INFODEMIC ON TWITTER

BRANDON ATTAI, KELTEN FALEZ ,TAHSIN CHOWDHURY

PROBLEM OVERVIEW – WHAT IS BEING SOLVED?

- Identifying misinformation about covid using text(NLP) and tweet metadata
  - Classify tweet as factual/false

ML DEVELOPMENT PROCESS OVERVIEW



Source: https://medium.datadriveninvestor.com/

DATA COLLECTION

Use Twitter API

- Tweepy
- Set up account
- Set access token
- Authenticate

Query

- Use verified article titles as search query.
- Limited to tweets in English only.
- Work around the APIs rate limit.

Inspect Response

- Pulled tweets were relevant to the titles.
- All tweets were very recent.

Parse Response

- Response is a massive JSON object with lots of metadata.
- Use Tweepy methods to access and save the ones relevant to the project.

Prepare Dataframe

- Use the saved data from the response to build a dataframe.
- Save it as csv.

LABELLING METHODOLOGY

- Does the tweet agree/disagree with well-known guidelines set by WHO, CDC, etc?
- Verify claim using fact-checking websites such as Politifact, Snopes, Healthfeedback.org
- Check account info such as account creation date, account handle, number of followers, replies etc.

AGREEMENT ANALYSIS – DISAGREEMENTS RESOLVED

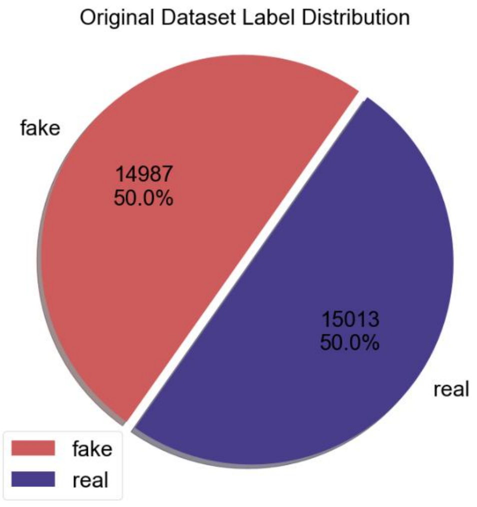
- Inter-Rater Reliability Method used to judge the quality of labelling: **Percentage Agreement**.
  - Add the number of agreements between each rater pair for each sample and then calculate the mean of that for the whole dataset.

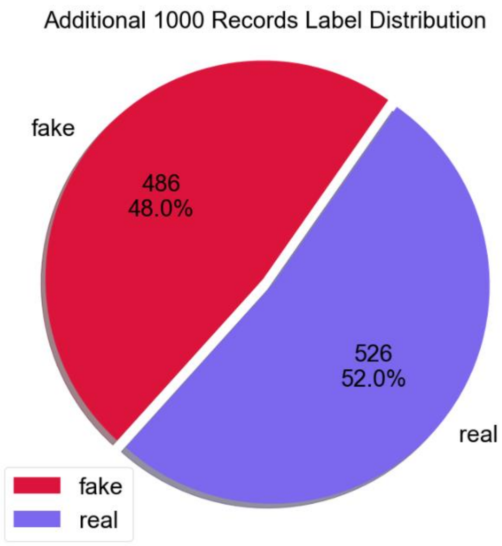
$$\boxed{60\%}$$

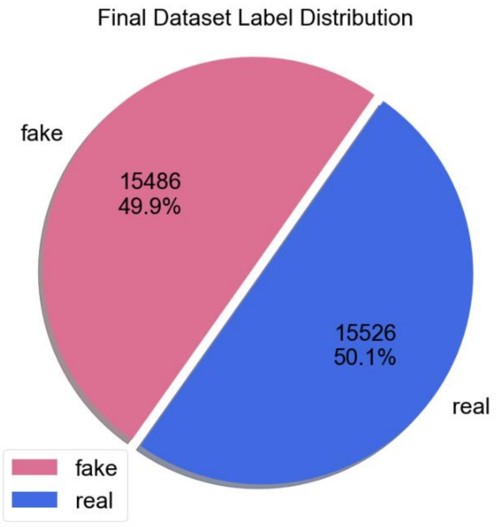
- Disagreements were resolved by accepting the label that was the majority.

DATASET COMPARISON

LABEL DISTRIBUTION : ~APPROX. 50-50 SPLIT







COMPARING THE DATASETS

- In order to check how the additional 1000 data samples compare to the original dataset, the following features were used:
  - Most used platforms to send out the tweet
  - Average retweets
  - Average likes
  - Average followers
  - Average following
  - Percentage of tweets with URLs attached
  - Percentage of accounts with no bio
  - Average account age
  - Average tweet length

ORIGINAL VS NEW DATA – FEATURE COMPARISON

	original	aditional
top_three_sources	[Twitter Web App, Twitter for iPhone, Twitter ...	[Twitter Web App, Twitter for iPhone, Twitter ...
avg_rts	5	14
avg_likes	6	20
avg_followers	2327	2079
avg_following	52294	76924
perc_with_links	83.11	82.5099
perc_acc_with_bio	87.0533	86.6601
avg_acc_age_years	7.75603	7.5751
avg_tweet_length	13	13

Note: avg_tweet_length was calculated after both sets were preprocessed to check if data preprocessing result in different outputs.

PRE-PROCESSING PIPELINE

Replacing Emojis

- Demoji Toolkit
- Replaces emojis with word representation
- ☺ - becomes 'smiley face'

Remove URLs

- URLs add noise to the tweet data
- Twitter API has feature for obtaining URLs in tweet metadata
- Regex created to remove all URLs

Digits and Punctuation Removal

- Regex to remove digits from the document
- Regex to remove punctuation such as capitalization, apostrophes etc.

Spell Correction

- Replace incorrectly spelled words with correctly spelled words
- SpellChecker Toolkit
- Eg. 'yesssssssss' -> 'yes'

PRE-PROCESSING PIPELINE (CONTINUED)

Stop Word Removal

- Removal of stop words within the 'english' word set
- Removal of individual letters and words that are less than 3 characters

Stemming

- Snowball stemmer toolkit used
- Stem the words in each tweet

Tokenization

- Twokenize toolkit used designed for social media text deta from Twitter
- Tokenize each tweet

Evaluation

- Dataset evaluated to ensure the steps were correctly done before next processing stages.

DATA PRE-PROCESSING - IMPLEMENTATION

- Custom Transformer class that can be used used in Pyspark ML's Pipeline() in succession with DocumentAssembler(),Tokenizer(), Stemmer(), StopWordsCleaner
- This Custom transformer has a udf that removes URLS, remove symbols and digits, convert to lowercase, replace emojis, and fix spellings.

CUSTOM TRANSFORMER

```
class CustomTransformer(Transformer, HasInputCol, HasOutputCol, DefaultParamsReadable, DefaultParamsWritable):
  input_col = Param(Params._dummy(), "input_col", "input column name.", typeConverter=TypeConverters.toString)
  output_col = Param(Params._dummy(), "output_col", "output column name.", typeConverter=TypeConverters.toString)

  @keyword_only
  def __init__(self, input_col: str = "input", output_col: str = "output"):
    super(CustomTransformer, self).__init__()
    self._setDefault(input_col=None, output_col=None)
    kwargs = self._input_kwargs
    self.set_params(**kwargs)

  @keyword_only
  def set_params(self, input_col: str = "input", output_col: str = "output"):
    kwargs = self._input_kwargs
    self._set(**kwargs)

  def get_input_col(self):
    return self.getOrDefault(self.input_col)

  def get_output_col(self):
    return self.getOrDefault(self.output_col)


  def _transform(self, df: DataFrame):

    def process_URLs(text):
      return re.sub(r'''(?i)\b((?:https?://|www\d{0,3}[.]|[a-z0-9.\-]+[.][a-z]{2,4}/)(?:[^\s()<>]+|\(([^\s()<>]+|(\([^\s()<>]+\)))*\))+(?:\(([^\s()<>]+|(\([^\s()<>]+\)))*\)|[^\s`!()\[\]{};:'".,<>?«»“”‘’]))''', "", text)

    def keep_alphabets_lower(text):
      cleaned_text = re.sub(r"[^a-zA-Z0-9]", " ", text)
      #cleaned_text = " ".join(re.findall("[A-Z][^A-Z]*", text))
      cleaned_text = re.sub(r'[0-9]+', '', cleaned_text)
      cleaned_text = re.sub(' +', ' ', cleaned_text)
      return cleaned_text.lower()
```

CUSTOM TRANSFORMER (CONTINUED)

```
def fix_spellings(text):
  spellchecker = SpellChecker()
  words = word_tokenize(text)
  corrected = ""
  for word in range(len(words)):
    corrected += (spellchecker.correction(words[word]) + " ")
  return corrected

def replace_emojis(text):
  emojis = demoji.findall(text)
  for k, v in emojis.items():
    text = text.replace(k,v)
  return text

def do_all_preprocessing(text):
    no_emojis = replace_emojis(text)
    no_urls = process_URLs(no_emojis)
    just_lower_alphabets = keep_alphabets_lower(no_urls)
    # fixed_spellings = fix_spellings(just_lower_alphabets)

    return just_lower_alphabets #fixed_spellings

input_col = self.get_input_col()
output_col = self.get_output_col()
# The custom action: concatenate the integer form of the doubles from the Vector
transform_udf = F.udf(lambda x: do_all_preprocessing(x), StringType())
return df.withColumn(output_col, transform_udf(input_col))
```

FURTHER DATA PRE-PROCESSING - IMPLEMENTATION

- Create new features such as length of the token
- Scale numeric data
- Generate TF-IDF feature vectors using the tokens
- Encode the labels

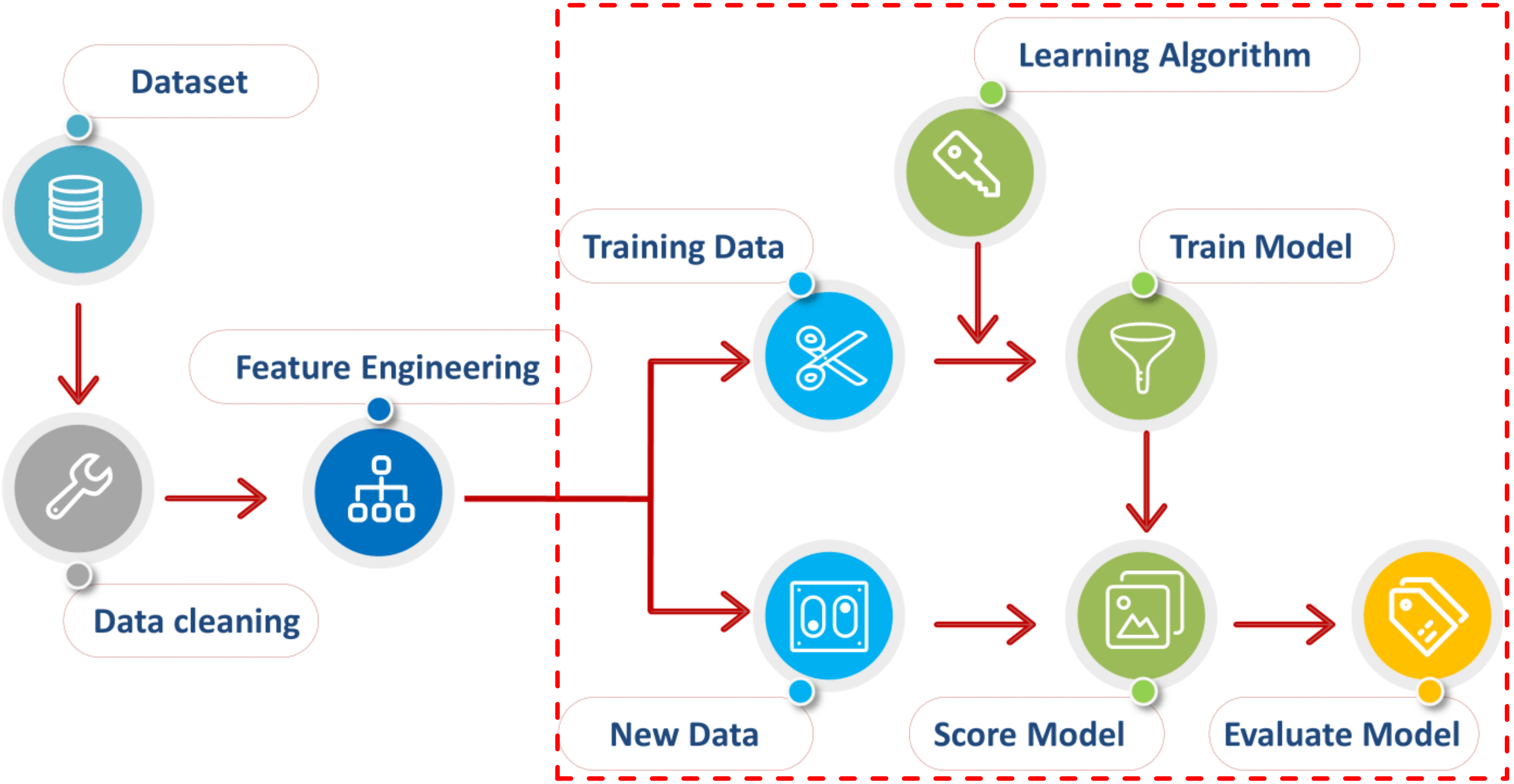
```python
def preprocess(df):
    df = df.withColumn("text", col("content"))

    custom_transformer = CustomTransformer(input_col = 'text', output_col ="clean_text")
    assembler = DocumentAssembler().setInputCol('clean_text').setOutputCol('doc')
    tokenizer = Tokenizer().setInputCols(['doc']).setOutputCol('tokens_annotations')
    stemmer = Stemmer().setInputCols(["tokens_annotations"]).setOutputCol("stem")
    stop_words = StopWordsCleaner.pretrained("stopwords_en", "en").setInputCols(["stem"]).setOutputCol("cleanTokens")
    finisher = Finisher().setInputCols(['cleanTokens']).setOutputCols(['tokens']).setOutputAsArray(True)
    pipeline = Pipeline().setStages([custom_transformer, assembler, tokenizer, stemmer ,stop_words, finisher]).fit(df.select('text')) #Pass the custom transformer before the Assembler
    processed = pipeline.transform(df.select('text'))

    modified_df = df.withColumn("mid",monotonically_increasing_id()).join(processed.withColumn("mid",monotonically_increasing_id()),["mid"]).drop("mid").drop('text')

    len_udf = udf(lambda s: len(s), IntegerType())
    modified_df = modified_df.withColumn("token_count",len_udf(col('tokens')))

    columns_to_scale = ["num_retweets", "num_likes", "followers", "following", "cum_tweets"]
    columns_to_drop = ["num_retweets_vec", "num_likes_vec", "followers_vec", "following_vec", "cum_tweets_vec"]

    #Convert the columns to vectors
    assemblers = [VectorAssembler(inputCols=[col], outputCol=col + "_vec") for col in columns_to_scale]
    scalers = [MinMaxScaler(inputCol=col + "_vec", outputCol=col + "_scaled") for col in columns_to_scale]

    #Build a pipeline
    pipeline = Pipeline(stages=assemblers + scalers)

    #Fit and Transform the data
    scalerModel = pipeline.fit(modified_df)
    scaledData = scalerModel.transform(modified_df).drop(*columns_to_drop)
    scaledData = scaledData.cache()

    #Hashing Vector TF
    hashing_vec=HashingTF(numFeatures = 2000, inputCol='tokens',outputCol='tf_features')
    hashing_df=hashing_vec.transform(scaledData)
    # hashing_df.select(['text','tokens','tf_features']).show(4,False)

    #TF-IDF
    tf_idf_vec=IDF(inputCol='tf_features',outputCol='tf_idf_features')
    df_with_features = tf_idf_vec.fit(hashing_df).transform(hashing_df)
    df_with_features = df_with_features.cache()
    #df_with_features.show(5)

    temp_va = VectorAssembler(inputCols=['token_count','num_retweets_scaled',
                                         'num_likes_scaled',
                                         'followers_scaled',
    final_df = temp_va.transform(df_with_features)
    final_df = final_df.cache()

    indexer = StringIndexer(inputCol="label", outputCol="categoryIndex")
    df_fully_encoded = indexer.fit(final_df).transform(final_df)
    df_fully_encoded = df_fully_encoded.cache()

    return df_fully_encoded
```

DATA PRE-PROCESSING - RESULTS

- The preprocessed feature columns are used to generate a feature vector using PySpark's VectorAssembler().
- The feature vector and the encoded label columns are then used to create train and test splits that are used in model training.

```
temp_va = VectorAssembler(inputCols=['token_count','num_retweets_scaled',
                                     'num_likes_scaled',
                                     'followers_scaled',
                                     'following_scaled',
                                     'cum_tweets_scaled',
                                     'tf_idf_features'],outputCol='features_vec')
final_df = temp_va.transform(df_with_features)
final_df = final_df.cache()

indexer = StringIndexer(inputCol="label", outputCol="categoryIndex")
df_fully_encoded = indexer.fit(final_df).transform(final_df)
df_fully_encoded = df_fully_encoded.cache()
```

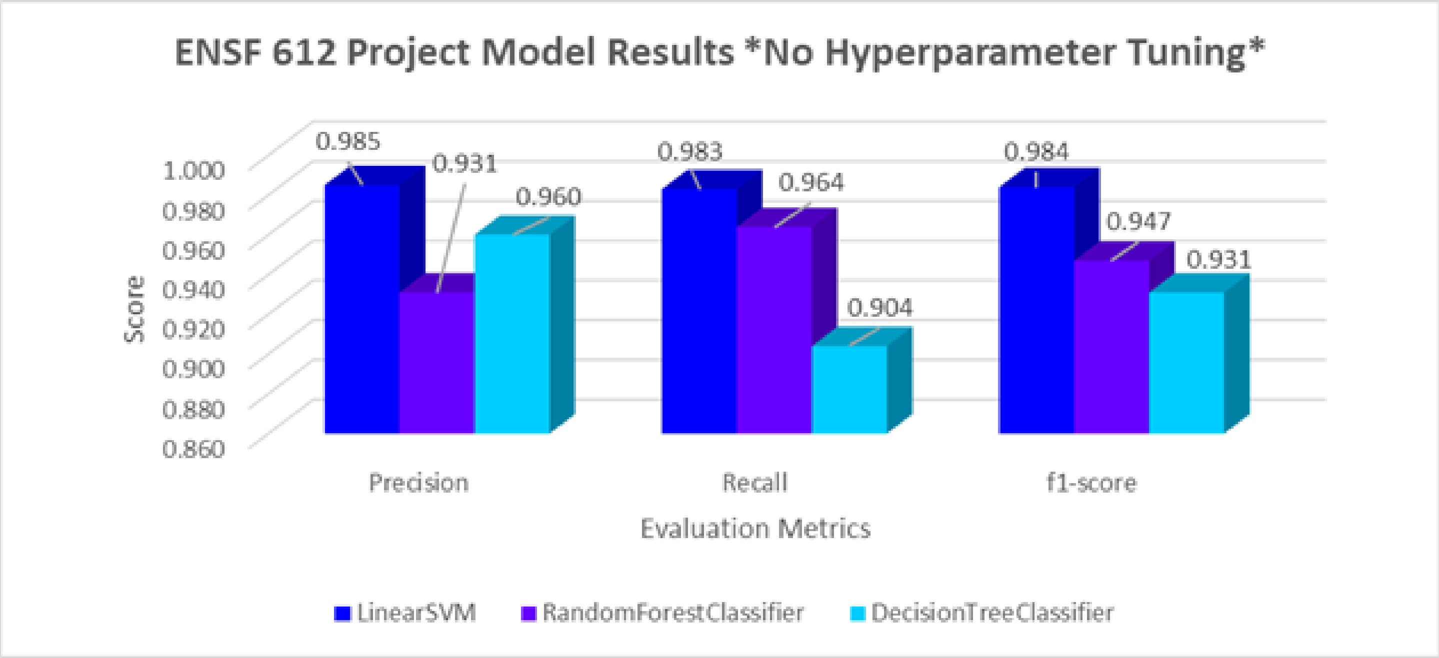
ML DEVELOPMENT PROCESS OVERVIEW



Source: https://medium.datadriveninvestor.com/

MODELS – TRAINING: BASELINE MODELS

- Models used:
  - Decision Tree Classifier
  - LinearSVM
  - Random Forest Classifier
- Train-Test split: 75%-25%
- Results for the extended dataset ->

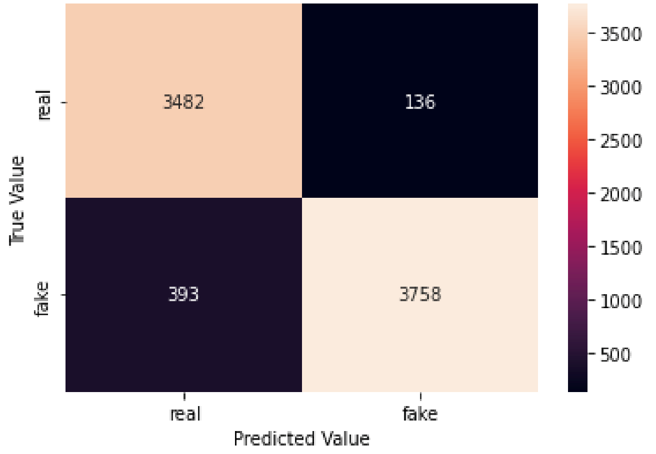


LinearSVM was the best performing baseline model

ORIGINAL + ADDITIONAL 1000 DATA POINTS (BASELINE MODEL)

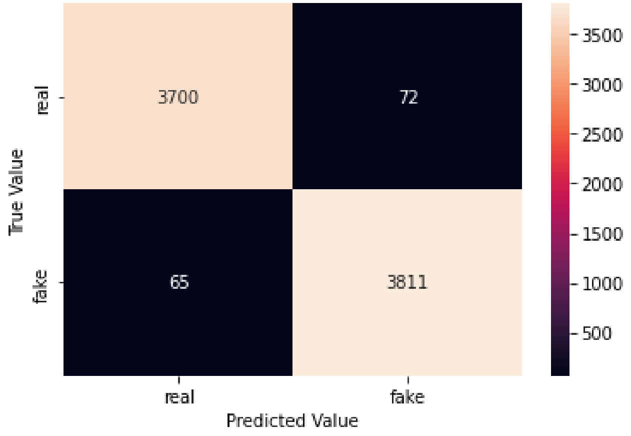
Decision Tree Results

```
precision: 0.9650744735490498
recall: 0.9053240183088412
f1 score: 0.9342448725916719
```



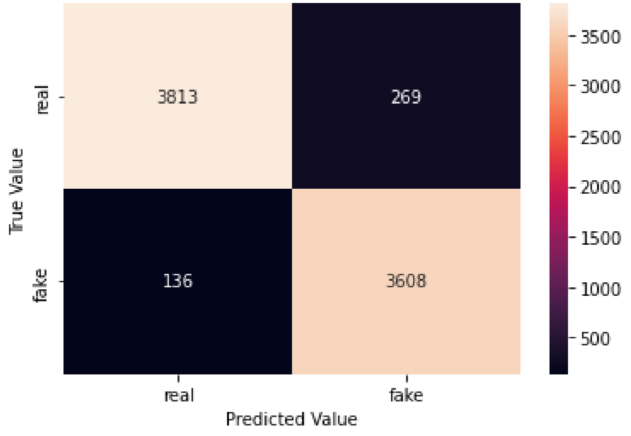
Linear SVM Results

```
precision: 0.9814576358485707
recall: 0.9832301341589267
f1 score: 0.9823430854491558
```



Random Forest Results

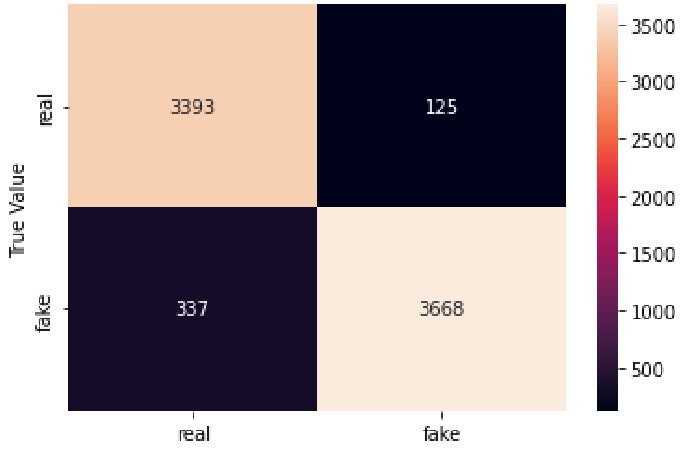
```
precision: 0.9306164560226979
recall: 0.9636752136752137
f1 score: 0.9468573677995012
```



ORIGINAL DATASET (BASELINE MODEL)

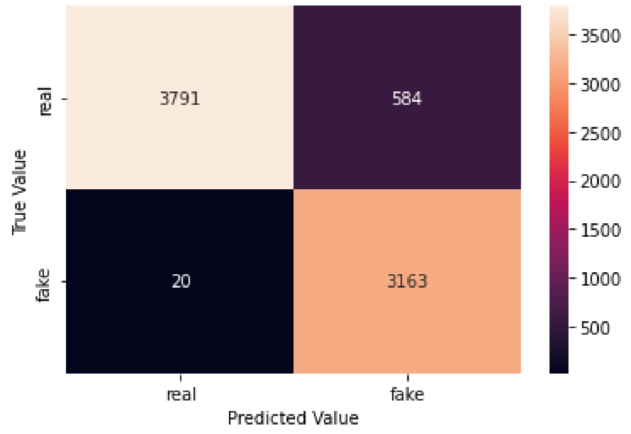
Decision Tree Results

```
precision: 0.9670445557606117
recall: 0.9158551810237203
f1 score: 0.9407540394973071
```



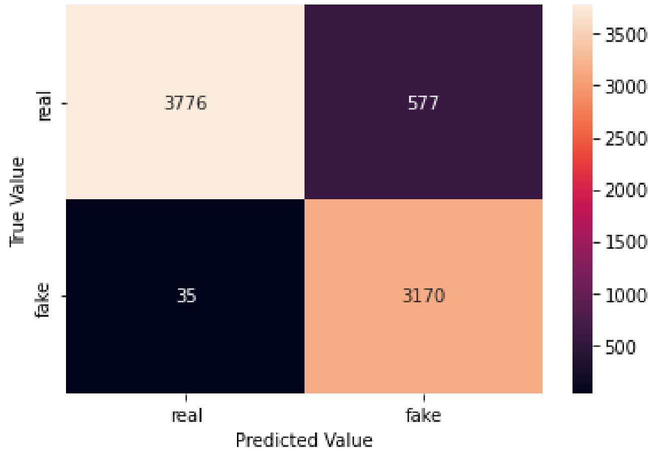
Linear SVM Results

```
precision: 0.8441419802508674
recall: 0.9937166195413132
f1 score: 0.9128427128427129
```



Random Forest Results

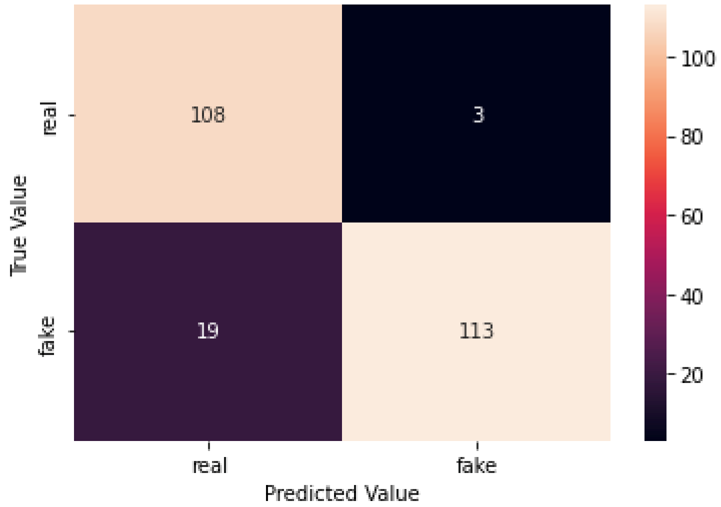
```
precision: 0.8460101414464906
recall: 0.9890795631825273
f1 score: 0.9119677790563867
```



ADDITIONAL 1000 DATA POINTS (BASELINE MODEL)

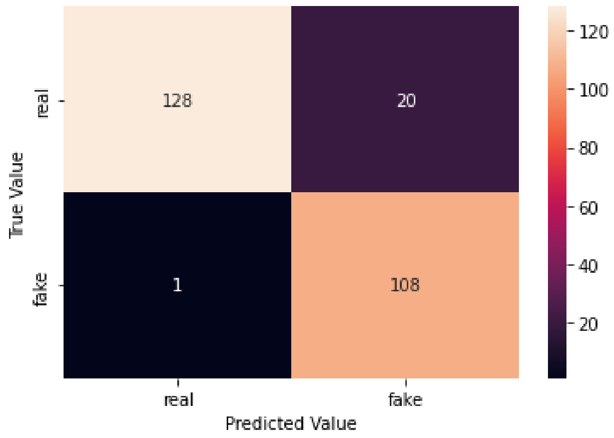
Decision Tree Results

```
precision: 0.9741379310344828
recall: 0.8560606060606061
f1 score: 0.9112903225806451
```



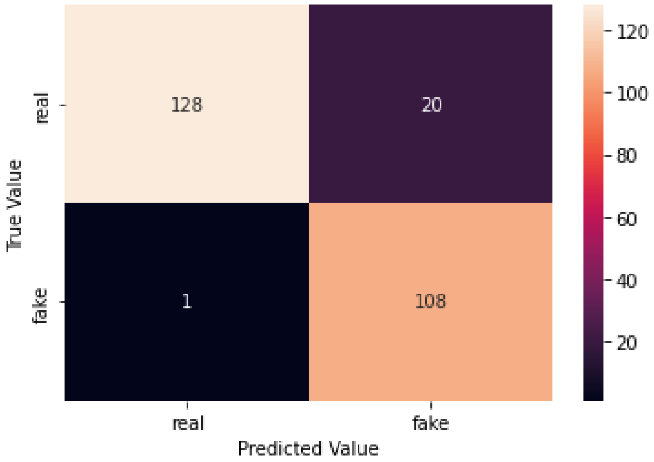
Linear SVM Results

```
precision: 0.84375
recall: 0.9908256880733946
f1 score: 0.9113924050632912
```



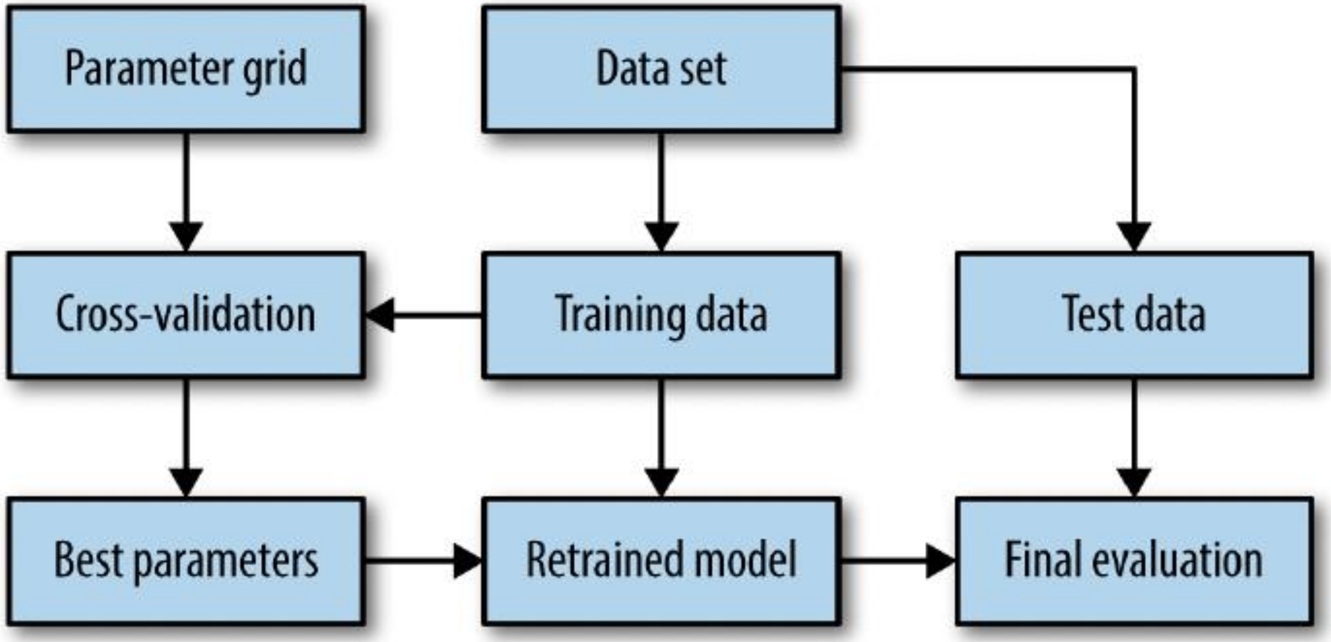
Random Forest Results

```
precision: 0.84375
recall: 0.9908256880733946
f1 score: 0.9113924050632912
```



MODEL TUNING
GRID SEARCH

- Grid Search the various hyperparameters of the models to determine the most optimal.
- For a better estimate of the generalization performance, cross-validation to evaluate the performance of each parameter combination.



MODEL TUNING GRID SEARCH

- Hyperparameter tuning was based on the various parameters available for each model.
- Research was conducted to determine the parameters to tune that would have a meaningful impact on the model performance.

Model	Parameter	Definition
Decision Tree	maxBins	Allows the algorithm to consider more split candidates and make fine-grained split decisions.
	maxDepth	A stopping rule that sets the maximum node depth of the recursive tree construction
Linear SVM	Regularization Parameter	Limits the importance of each point
	maxIter	Hard limit on iterations within solver
RandomForest	numTrees	The number of trees in the forest.
	maxBins	Allows the algorithm to consider more split candidates and make fine-grained split decisions.
	bootstrap	Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree.
	maxDepth	A stopping rule that sets the maximum node depth of the recursive tree construction

MODEL TUNING
GRID SEARCH

- Overview of the hyperparameter values that were tuned for the grid search.

Parameter Grid for Models

Model	Parameter	Value			
Decision Tree	maxBins	16	32	64	-
	maxDepth	5	7	10	20
Linear SVM	Regularization Parameter	0.0001	0.001	0.01	0.1
	maxIter	10	100	1000	-
RandomForest	numTrees	20	50	64	-
	maxBins	16	32	64	-
	bootstrap	TRUE	FALSE		-
	maxDepth	5	7	10	20

CODE OVERVIEW OF GRID SEARCH AND CROSS VALIDATION

Parameter Grid and Cross Val for Decision Trees

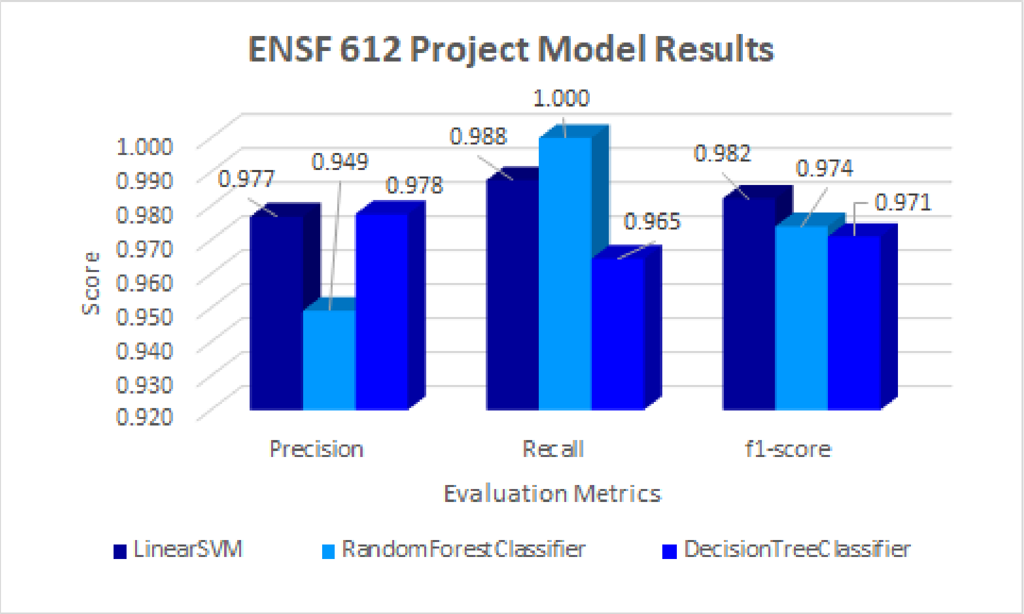
```
paramGrid = ParamGridBuilder()\
            .addGrid(dt.maxBins, [16, 32, 64])\
            .addGrid(dt.maxDepth, [5, 7, 10, 20])\
            .build()


crossval = CrossValidator(estimator=dt,
                          estimatorParamMaps=paramGrid,
                          evaluator=BinaryClassificationEvaluator(),
                          numFolds=3,
                          collectSubModels=True)
```

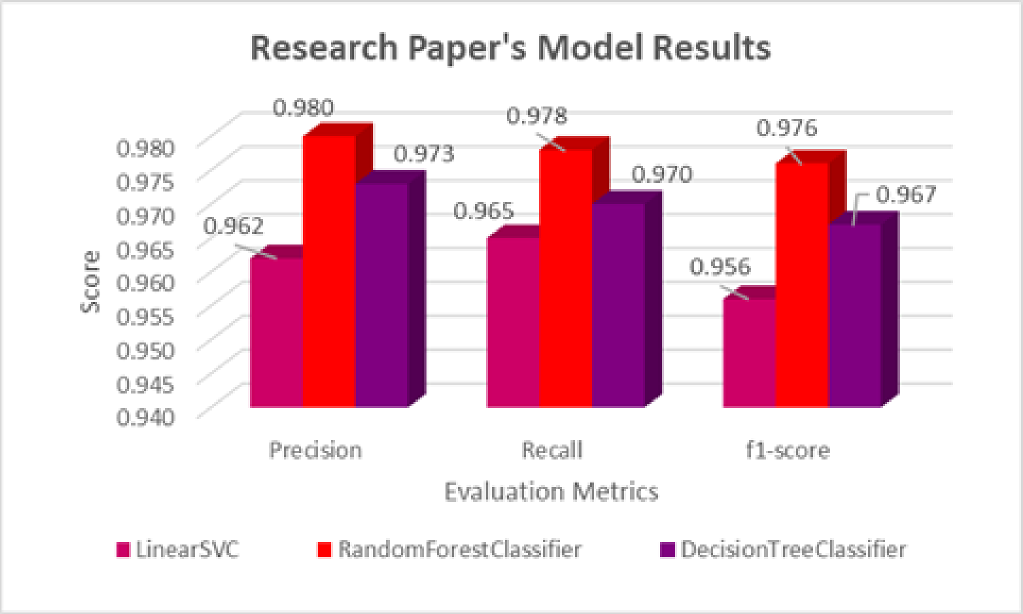
Parameter Grid and Cross Val for LinearSVM

```
paramGrid = ParamGridBuilder()\
            .addGrid(svm.regParam, [0.0001, 0.001, 0.01, 0.1])\
            .addGrid(svm.maxIter, [10, 100, 1000])\
            .build()

crossval = CrossValidator(estimator=svm,
                          estimatorParamMaps=paramGrid,
                          evaluator=BinaryClassificationEvaluator(),
                          numFolds=3)
```

Parameter Grid and Cross Val for Random Forest

```
paramGrid = ParamGridBuilder()\
            .addGrid(rf.numTrees, [20, 50, 100])\
            .addGrid(rf.maxBins, [16, 32, 64])\
            .addGrid(rf.bootstrap, [True, False])\
            .addGrid(rf.maxDepth, [5, 7, 10, 20])\
            .build()

crossval = CrossValidator(estimator=rf,
                          estimatorParamMaps=paramGrid,
                          evaluator=BinaryClassificationEvaluator(),
                          numFolds=3,
                          collectSubModels = True)
```



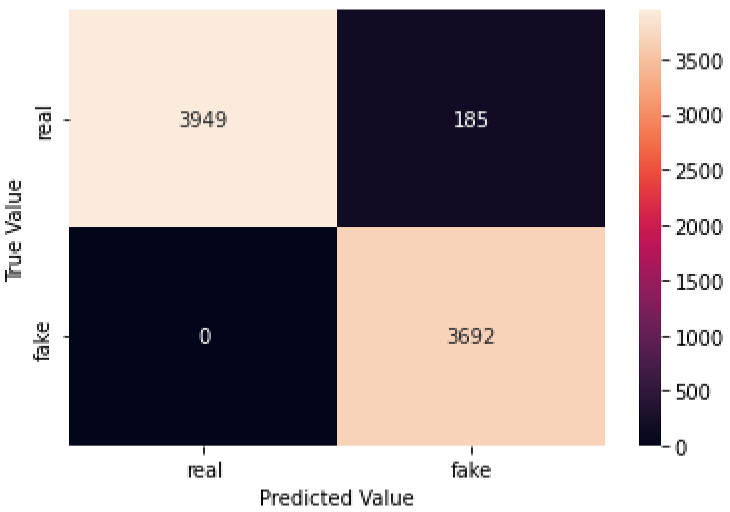


MODELS WITH BEST PARAMETERS - RESULTS

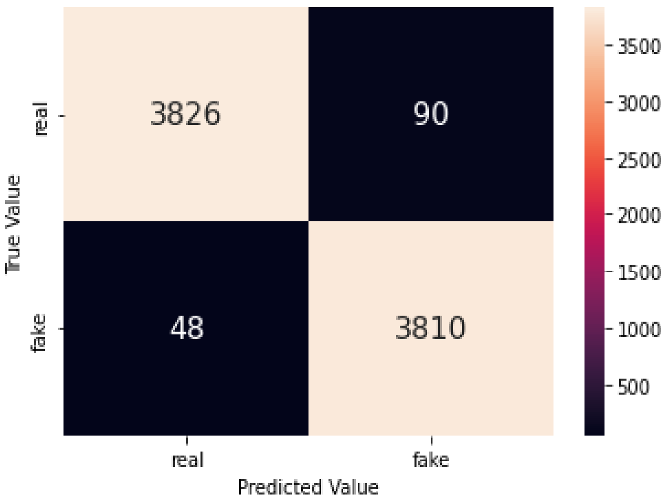
- Our classifiers had higher performance compared to Research article.
- Our best performing model : LinearSVM
- Article's best performing model: RandomForestClassifier

GridSearch Results for Models				
Classifier	**Best Parameters**	**Precision**	**Recall**	**f1-score**
LinearSVM	regParam: 0.001 maxIter: 10	0.9769	0.9876	0.9822
RandomForestClassifier	numTrees: 100 maxBins: 32 bootstrap: false maxDepth: 20	0.9492	1.0000	0.9740
DecisionTreeClassifier	numTrees: 100 maxBins: 16 bootstrap: false maxDepth: 20	0.9776	0.9645	0.9710

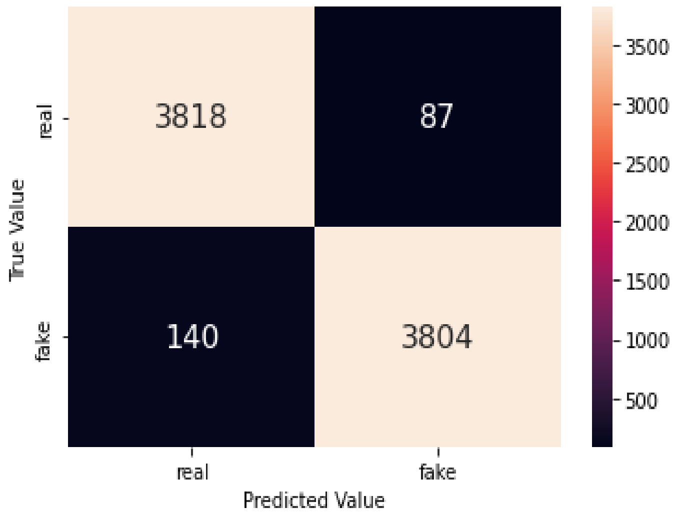
RESULTS – CONTINUED



CONFUSION MATRIX FOR THE BEST **RANDOM FOREST CLASSIFIER**



CONFUSION MATRIX FOR THE BEST **LINEAR SVM CLASSIFIER**



CONFUSION MATRIX FOR THE BEST **DECISION TREE CLASSIFIER**

CONCLUSION

- Dataset had an almost even split of label distribution
- Text data was preprocessed using several NLP techniques Stemming and TF-IDF
- Three classifiers were trained on the data and improved using GridSearch
- Going forward: Deploy LinearSVC model
  - Best performance after performing GridSearch
- Outperformed classifiers in original research paper

THANK YOU