Computer Science Tripos Part II

2020-2021
Coursework submission
Cover sheet

Student name……Kacper Walentynowicz……

College……Trinity College……

CRSID……kpw29……

Submit to Student Admin
Date of submission

Unit name Data Science - Principles and Practice

Assessor mark

SA Records……..

CRSID……kpw29……

Unit Name……Data Science - Principles……

Assignment No……Final Assignment……

SA stamp date and initials

Prediction of premium products purchases

Data Science Principles & Practice Final Assignment

1. Introduction

The report describes the process of constructing a Machine Learning pipeline for prediction of premium product purchases based on the Clickstream Data in Online Shopping dataset.

1.1. Dataset description

The dataset on the web page is accompanied by a short authors' description. Let's inspect what we can learn from that, and also what a simple look at the data shows us.

```
In [15]: df = pd.read_csv('e-shop clothing 2008.csv', sep=';')

In [16]: df.head()
```

Out[16]:

	year	month	day	order	country	session ID	page 1 (main category)	page 2 (clothing model)	colour	location	model photography	price	price 2	page
0	2008	4	1	1	29	1	1	A13	1	5	1	28	2	1
1	2008	4	1	2	29	1	1	A16	1	6	1	33	2	1
2	2008	4	1	3	29	1	2	B4	10	2	1	52	1	1
3	2008	4	1	4	29	1	2	B17	6	6	2	38	2	1
4	2008	4	1	5	29	1	2	B8	4	3	2	52	1	1

There are 14 features, as below. Descriptions in italics come from researchers, the rest are my thoughts.

1. Year -> 2008
2. *Month -> from April (4) to August (8).*
3. *Day -> day number of the month*
4. *ORDER -> sequence of clicks during one session.*
   This variable is probably quite important, as it indicates the order in which purchases happened.
5. *COUNTRY -> variable indicating the country of origin of the IP address.*
   This is a categorical variable with almost 50 different categories, it definitely needs some preprocessing.
6. *SESSION ID -> variable indicating session id (short record).*
   This is the only variable which allows us to link several orders to the same customer.
7. *PAGE 1 (MAIN CATEGORY) -> concerns the main product category: trousers, skirts, blouses, sale.*

Why is “sale” a category? It surely isn’t a type of clothing.

8. *PAGE 2 (CLOTHING MODEL) -> contains information about the code for each of the 217 products.*
9. *COLOUR -> colour of product*
   14 different colours, we need to do something with it.
10. *LOCATION -> photo location on the page, the screen has been divided into six parts.*
    This is another categorical variable with a high importance. If we investigate importance of various classes, then we may be able to instruct the shop owners where to position their products on the webpage.
11. *MODEL PHOTOGRAPHY -> variable with two categories: en-face, profile.*
12. *PRICE -> price in US dollars*
13. *PRICE 2 -> variable informing whether the price of a particular product is higher than the average price for the entire product category. 1-YES, 2-NO.*
    This is our target variable, tells whether the product is considered to be premium. It is very important not to confuse the two classes, so convert the {1,2} into {1, 0}, as this is more natural.
14. *PAGE -> page number within the e-store website (from 1 to 5)*

After performing the described changes, and renaming a few columns into a bit more clearer names, the head of the dataset looks as follows:

```
In [47]: df.head()
```

Out[47]:

	year	month	day	order	country	session_id	main_category	clothing_model	colour	location	photo	price	premium	page
0	2008	4	1	1	29	1	1	A13	1	5	1	28	0	1
1	2008	4	1	2	29	1	1	A16	1	6	1	33	0	1
2	2008	4	1	3	29	1	2	B4	10	2	1	52	1	1
3	2008	4	1	4	29	1	2	B17	6	6	2	38	0	1
4	2008	4	1	5	29	1	2	B8	4	3	2	52	1	1

2. Data investigation and preparation

2.1. First steps

Our goal is a classification task - the target is a binary categorical variable. The dataset contains 165474 entries, and a quick check reveals that there are no NULL entries. The number differs from the original paper’s description by ~50.000, but there is no information why.

```
In [69]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 165474 entries, 0 to 165473
Data columns (total 13 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   month           165474 non-null  int64
 1   day             165474 non-null  int64
 2   order           165474 non-null  int64
 3   country         165474 non-null  int64
 4   session_id      165474 non-null  int64
 5   main_category   165474 non-null  int64
 6   clothing_model  165474 non-null  object
 7   colour          165474 non-null  int64
 8   location        165474 non-null  int64
 9   photo           165474 non-null  int64
 10  price           165474 non-null  int64
 11  premium         165474 non-null  int64
 12  page            165474 non-null  int64
dtypes: int64(12), object(1)
memory usage: 16.4+ MB
```

```
In [67]: summary = pd.DataFrame(df.dtypes, columns=['Feature type'])
         summary['Is any null?'] = pd.DataFrame(df.isnull().any())
         summary
```

Out[67]:

	Feature type	Is any null?
month	int64	False
day	int64	False
order	int64	False
country	int64	False
session_id	int64	False
main_category	int64	False
clothing_model	object	False
colour	int64	False
location	int64	False
photo	int64	False
price	int64	False
premium	int64	False
page	int64	False

Clothing_model looks no good, as it's not a numerical value.

```
In [24]: df['clothing_model']
```

```
Out[24]: 0         A13
         1         A16
         2          B4
         3         B17
         4          B8
                  ...
         165469    B10
         165470    A11
         165471     A2
         165472     C2
         165473     B2
         Name: clothing_model, Length: 165474, dtype: object
```

```
In [22]: dic = {"A":1, "B":2, "C":3, "P":4}
         iter = 0
         for row in df.values:
             desc = str(row[6])
             if dic[desc[0]] != row[5]:
                 print(row, iter)

             iter += 1
```

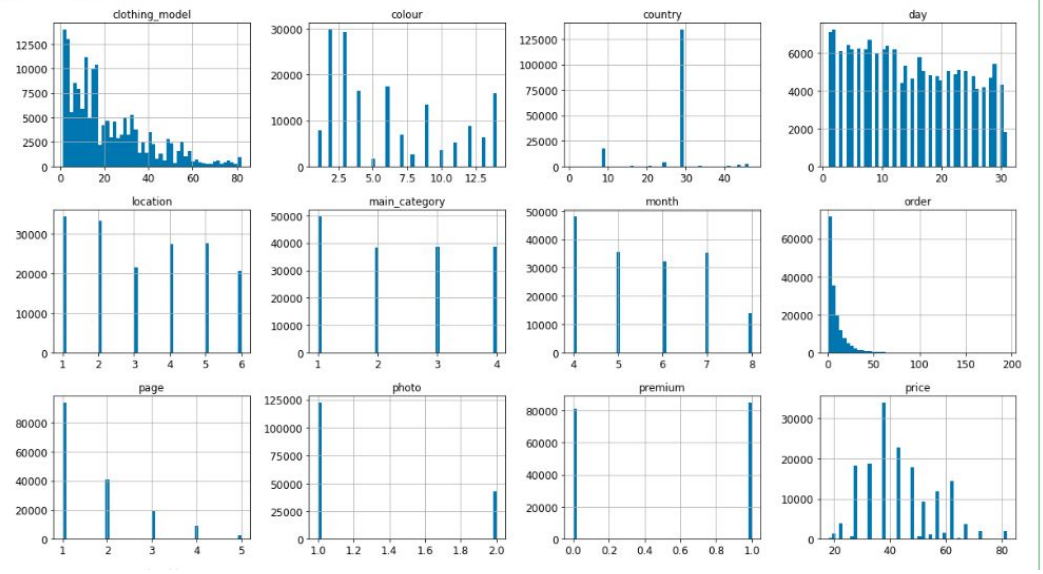
```
[4 10 27 29 2766 2 'A18' 4 6 1 38 0 1] 19001
```

A quick inspection suggests that main_category is mapped to the letter in clothing_model, so we can simply drop the letter and get a numeric value. The only problem is one row where the product models do not match, this is probably an error in dataset preparation. There are 938 entries for product A18, and I believe it's enough to classify this as a mistake and fix it manually. Additionally, I found it useful to combine these two informations into one, so that product class A12 becomes 112, B3 becomes 203, etc.

2.2. Insights from a histogram

Let's plot a very simple histogram to get more understanding of the data.

```
In [66]: df.hist(bins=50, figsize=(20,15))
         plt.show()
```



Some plots make little sense, but a few points can be inferred:

- There is roughly the same number of instances of the two target classes in **premium**
- An overwhelming majority of customers come from one country (Poland), so it makes sense to divide **country** into two groups: Poland and Other.
- **Order** and **page** seem to follow an exponential distribution, which is perfectly reasonable given what they represent
- **Price** most likely doesn't follow a normal distribution, but it may be close. Let's apply a test for this.

```
In [72]: for feature in ['price']:
             alpha = 0.05
             p_value = scipy.stats.normaltest(df[feature])[1]
             if(p_value < alpha):
                 print('For feature \'' + feature +'\' null hypothesis can be rejected. NOT a normal distribution.')
             else:
                 print('For feature \'' + feature +'\' null hypothesis can not be rejected.')

         For feature 'price' null hypothesis can be rejected. NOT a normal distribution.
```

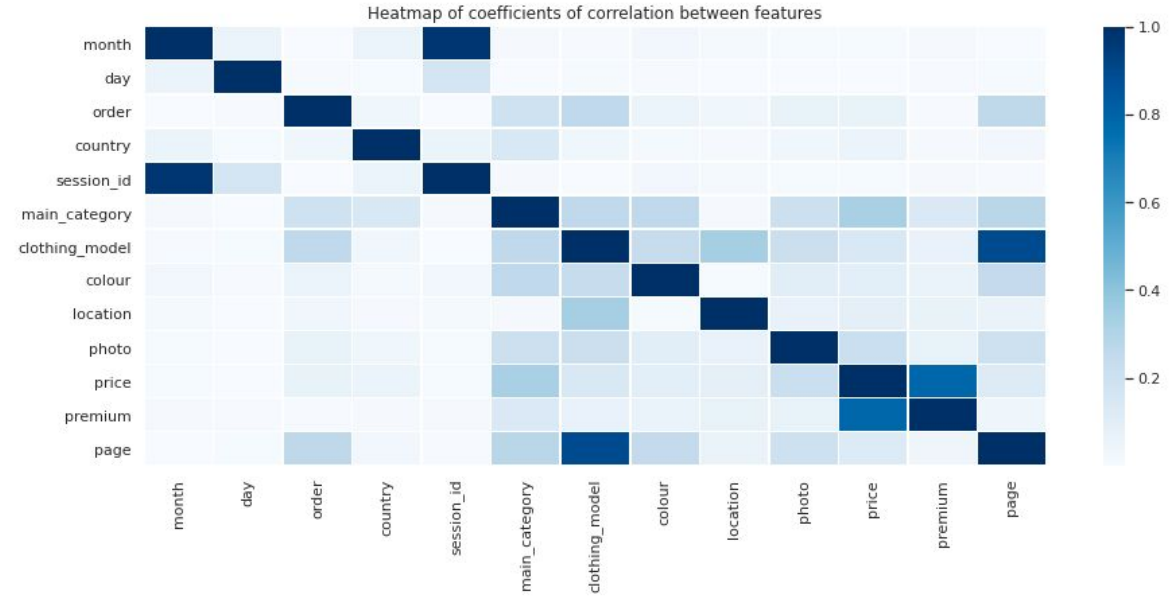
Therefore, it's most likely inappropriate to use data preprocessing methods which assume a normal distribution, such as StandardScaler().

2.3. Correlation of variables.

Let's investigate how the variables are correlated with each other, especially how they're correlated with the target variable. I've decided to use Spearman's correlation rather than the default one (Pearson's), as it's more appropriate in case of most variables being categorical, as explained in this blog.

```
In [137]: corr_matrix = df.corr(method='spearman')

In [133]: corr_matrix["premium"].sort_values(ascending=False)

Out[133]: premium           1.000000
          price             0.783931
          main_category     0.143976
          colour            0.058974
          month             0.013237
          session_id        0.012684
          order             0.004731
          day              -0.004560
          country          -0.012512
          page             -0.041586
          clothing_model   -0.067573
          photo            -0.072944
          location         -0.074084
          Name: premium, dtype: float64
```

Unsurprisingly, the target variable is strongly correlated with **price.**
We may also get more understanding from the entire correlation matrix.

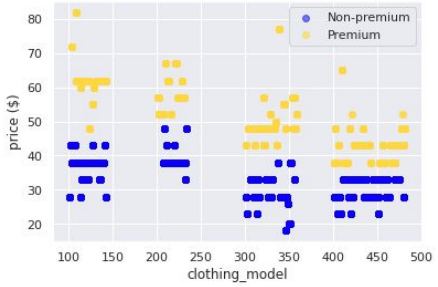


At first glance I found surprising the strong correlation of **session_id**, with **month**, but not with **day**.
Additionally, **clothing_model** has high correlation with **page**. Perhaps the models were ordered on the site by their ID.

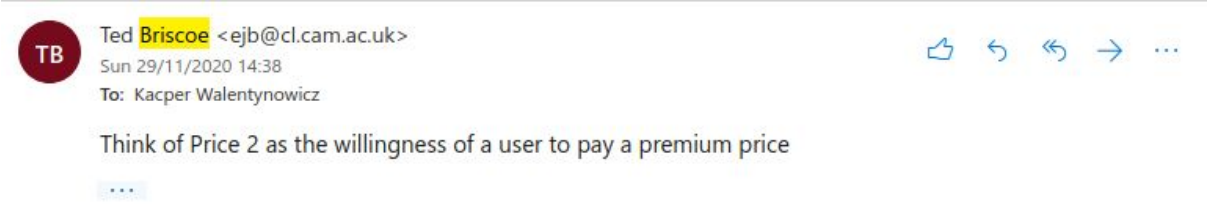
2.4. Analysing product categories

The product categories seem to be key to understanding what the dataset really contains, and the key to building a sensible and interpretable model. Let’s see how clothing models influence prices.

```
In [29]: fig, ax = plt.subplots()
clr = ["blue", "gold"]
names = ["Non-premium", "Premium"]

for premium in [0, 1]:
    x = df['clothing_model'].loc[df['premium'] == premium]
    y = df['price'].loc[df['premium'] == premium]
    ax.scatter(x, y, c=clr[premium], label=names[premium],
               alpha=0.5)

ax.legend()
plt.xlabel('clothing_model')
plt.ylabel('price ($)')
ax.grid(True)

plt.show()
```



WAIT. Something is really suspicious here. Why is there a clear separation between the two? I stopped for a while to think about this and have reached a conclusion that target variable is an alias for **“is this product’s price more than average of this category’s price?”.** Following a clarification from Professor Briscoe:



I’ve started even more cautious investigation of the data. It turns out that there’s much more dependency going on behind the scenes than one would expect at first glance.

Let’s select columns which relate to the model of the item.

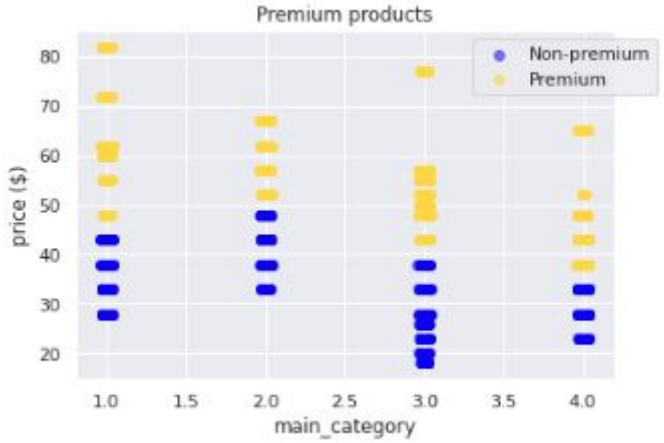
```
In [55]: product_info = df[['clothing_model', 'colour', 'location', 'photo', 'price', 'premium']]
gmin = product_info.groupby('clothing_model').min()
gmax = product_info.groupby('clothing_model').max()
equality = gmin == gmax

print("Different models: ", len(gmin))
equality.sum()
```

```
Different models:  217
```

```
Out[55]: colour      217
location    217
photo       217
price       217
premium     217
dtype: int64
```

It turns out that **all the information about one model is always the same.** For the attributes of colour or photo, it’s not really surprising. But there’s so much more to it: each product model always has the same location on the website, and always has the same price.

There comes a conclusion: we can immediately predict the target if we know which clothing model it is. Therefore, if we want a generalizable classifier, we need to exclude **clothing_model** from the data. Let’s repeat the plot, but with the broader clothing category.



Price needs to be removed, too.

Additionally, the above observations confirm the website is entirely **static**. We have no way to link clients among different sessions, and therefore year, month and day of the session are completely useless as well.

```
In [152]: clickstream = df.drop(['clothing_model', 'price', 'year', 'month', 'day'], axis=1)
          clickstream.info()

          <class 'pandas.core.frame.DataFrame'>
          RangeIndex: 165474 entries, 0 to 165473
          Data columns (total 9 columns):
           #   Column         Non-Null Count   Dtype
          ---  ------         --------------   -----
           0   order          165474 non-null  int64
           1   country        165474 non-null  int64
           2   session_id     165474 non-null  int64
           3   main_category  165474 non-null  int64
           4   colour         165474 non-null  int64
           5   location       165474 non-null  int64
           6   photo          165474 non-null  int64
           7   premium        165474 non-null  int64
           8   page           165474 non-null  int64
          dtypes: int64(9)
          memory usage: 11.4 MB
```

2.5. Clickstream vs Purchase

There is at least one more subtlety - the data is actually a clickstream. We have **no** information if clients bought these products. Perhaps the original paper did have some records about this, but in this dataset we only have the clicks. Therefore, we need to rephrase understanding of the datapoints to: “the product with characteristics X, location Y, …, attracted customer’s attention”. What about the purchases? Let’s assume a **linear relationship** between number of views and number of purchases (because we know nothing about the purchases), This is how many salespeople think anyway, for example when they estimate cost and benefits of advertising space.

2.6. Data preparation

Let’s bring various columns to a more appropriate format.

2.6. 1. Colour.

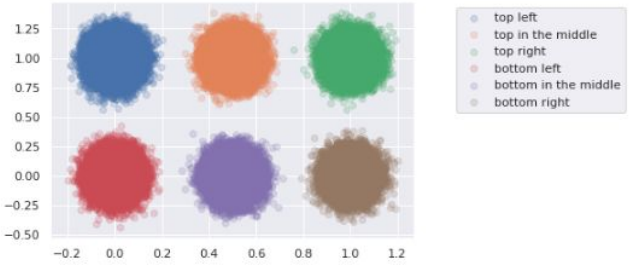
Colours are divided into 14 categories. The values are completely meaningless, because they don’t convey information about colour similarities. To resolve this, I’m using a data embedding known to many - colour scales. Precisely, I’m converting the values into HSV (Hue, Saturation, Value) colour scale, as this is the scale which better models human perception of colour than machine-readable RGB.

```
In [111]: from matplotlib import colors
          import colorsys

          dict_colors = dict(colors.cnames)
          manually_found_hexes = {"burgundy": "#800020",
                                  "navy blue":"#000080",
                                  "nothing": "#000000", #not important, never used
                                  "multicolor": "#000000"
                                 }
          def hsv_from_name(cname):
              if cname in dict_colors:
                  hexc = dict_colors[cname]
              elif cname in manually_found_hexes:
                  hexc = manually_found_hexes[cname]
              else:
                  print("FAIL")

              r,g,b = colors.hex2color(hexc)
              return colorsys.rgb_to_hsv(r, g, b)

          #tests
          orange_hsv = hsv_from_name('orange')

          my_colors = ['nothing', 'beige', 'black', 'blue', 'brown', 'burgundy', 'gray', 'green', 'navy blue', 'multicolor', '

In [112]: my_hsvs = [hsv_from_name(x) for x in my_colors]

In [119]: clickstream['color_h'] = clickstream['colour'].apply(lambda clr : my_hsvs[clr][0])
          clickstream['color_s'] = clickstream['colour'].apply(lambda clr : my_hsvs[clr][1])
          clickstream['color_v'] = clickstream['colour'].apply(lambda clr : my_hsvs[clr][2])


In [122]: clickstream = clickstream.drop(['colour'], axis=1)
```

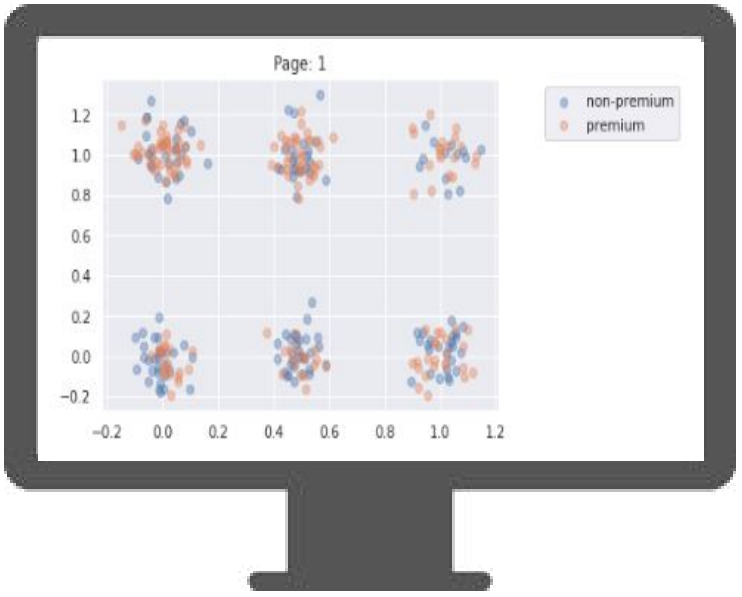
2.6.2. Location

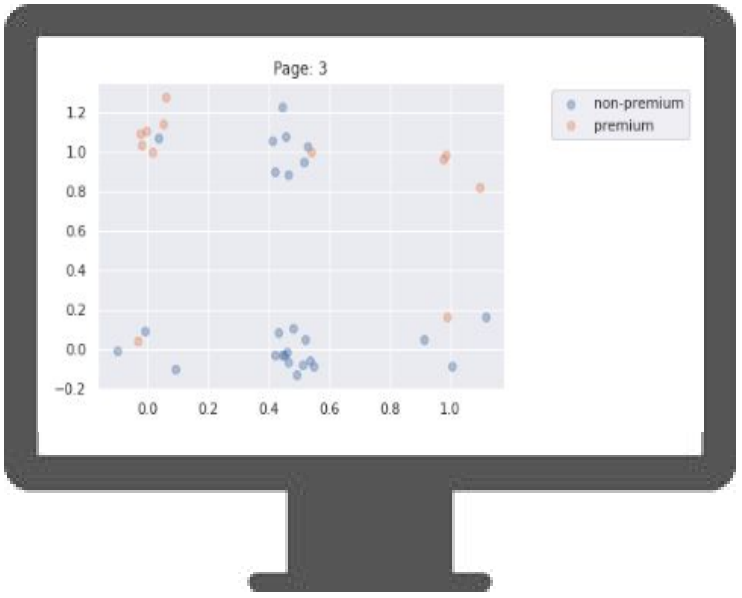
Locations are more naturally expressed based on their relative position on the screen, with x/y coordinates. Let's also add a bit of noise, to reflect that in real life there can be several pictures around each other in "top-right-corner" or so, and we don't know exact positions which may be different.

```
In [364]: #[0-nothing, 1-top left, 2-top in the middle, top right, bottom left, bottom in the middle, bottom right]
          positions = [(0.0, 0), (1.0, 0), (1.0, 0.5), (1.0, 1), (0.0, 0), (0.0, 0.5), (0.0, 1)]
          text = ['nothing', 'top left', 'top in the middle', 'top right', 'bottom left', 'bottom in the middle', 'bottom righ
          clickstream['loc_y'] = clickstream['location'].apply(lambda value: positions[value][0] + np.random.normal(0, 0.1))
          clickstream['loc_x'] = clickstream['location'].apply(lambda value: positions[value][1] + np.random.normal(0, 0.05))

          #let's plot positions on the screen in order to verify that they are correct
          data_len = 34532
          for lvl in np.arange(1, 7):
              i = (clickstream['location'] == lvl)
              plt.scatter(clickstream['loc_x'][i],
                          clickstream['loc_y'][i],
                          label=text[lvl], alpha=0.2)

          plt.legend(loc='upper left', bbox_to_anchor=(1.1, 1))

Out[364]: <matplotlib.legend.Legend at 0x7f1c5b8a1250>
```



Additionally, I looked at the scatterplot of target values depending on page and location on the screen. There are notable differences between frequencies of clicks on different pages.





2.6.3. Main category of the item

The four classes of items are better represented with one-hot-encoding than with the initially given uninformative values 1,2,3,4.

```
In [281]: onehot_features = ['trousers', 'skirts', 'blouses', 'sale']
          index = 0
          for feature in onehot_features:
              index += 1
              clickstream['is_' + feature] = (clickstream['main_category'] == index).astype('float')
```

2.6.4. Scaling for other features

Order and Page are already in appropriate form, but our algorithms may benefit from scaling. As we've previously found, they look like they come from exponential distributions, so it's best to use a MinMax scaler to just bring them to range [0, 1].

Additionally, Session_id has large values and must be scaled, too.

```
: from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
for feature in ['order', 'page', 'session_id']:
    clickstream[feature] = scaler.fit_transform(clickstream[feature].values.reshape(-1, 1))
```

2.6.5. Preparing train and test datasets.

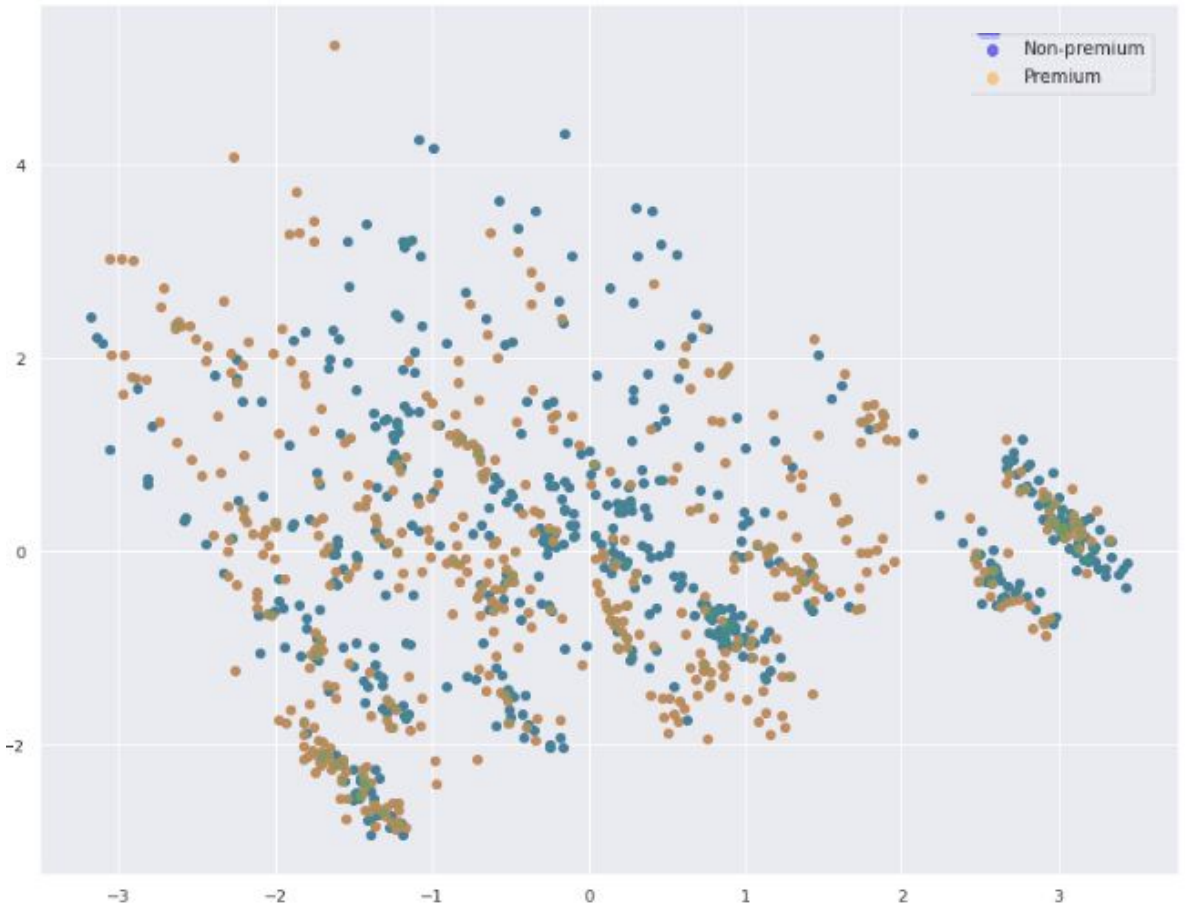
I'm using StratifiedShuffle to ensure that target value is equally represented in the two datasets.

```
In [382]: from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(clickstream, clickstream['premium']):
    X_train = clickstream.loc[train_index]
    X_test = clickstream.loc[test_index]
```

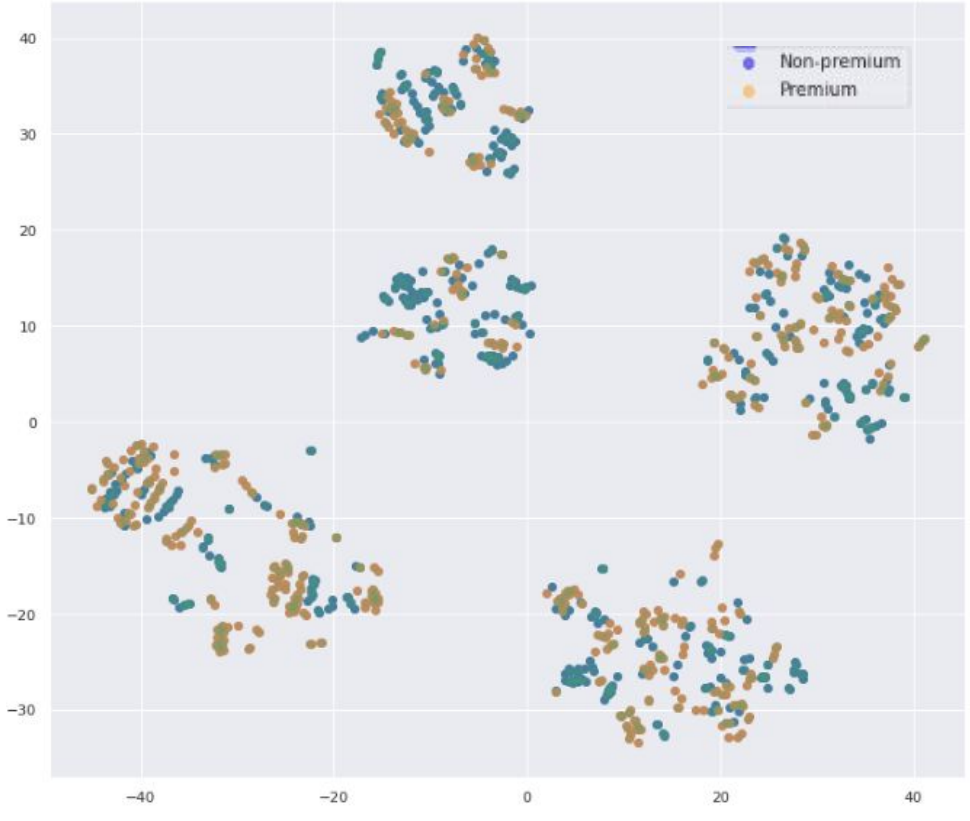
2.7. Dimensionality reduction

There aren't many features left, so the goal of applying dimensionality reduction is to learn whether there are some interesting clusters in the data.

The scatter below was obtained by applying PCA on first 1000 records to discover any patterns. It didn't lead me to interesting conclusions, other than a suspicion that simple models may not be performing well.



tSNE gives a bit more interesting results by confidently identifying a few clusters for different values of perplexity. Sadly, each of these appear to have a lot of instances of both classes.



3. Machine learning algorithms application

3.1. Evaluation

By default, I am using a simple cross-validation function (3-5 CV runs depending on learning time) to evaluate the models. I've decided that doing too detailed investigation for models which perform poorly won't be useful, and I'd rather focus on fine-tuning the better ones.
In this business prediction, I don't think that either precision or recall should be prioritized and therefore I've decided to use the balanced metrics - accuracy and F1 for evaluation.

```
from sklearn.model_selection import cross_val_score
def run_cross_validation(model, X, y, cv_param):
    scores_f1 = cross_val_score(model, X, y,
                                scoring = "f1", cv=cv_param, n_jobs=-1)

    scores_acc = cross_val_score(model, X, y,
                                scoring = "accuracy", cv=cv_param, n_jobs=-1)

    print("Running cross-validation for: ", type(model))
    print("Av Accuracy: ", np.mean(scores_acc))
    print(scores_acc)
    print("Av F1: ", np.mean(scores_f1))
    print(scores_f1)
```

I am also plotting Precision vs Threshold and ROC curves for the models, using mostly the code provided in the previous practicals.
Although confusion matrices are among my favourite representations, they are not going to be super useful, as we have only 2 classes. I will only report them for final evaluation.

```
from sklearn.metrics import roc_curve

def plot_roc_curve(model, fpr, tpr, label=None):
    tt = 'ROC curve for: ' + str(type(model))
    plt.title(tt)
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], "k--")
    plt.axis([0, 1, 0, 1.01])
    plt.xlabel("False positive rate (fpr)")
    plt.ylabel("True positive rate (tpr)")
```

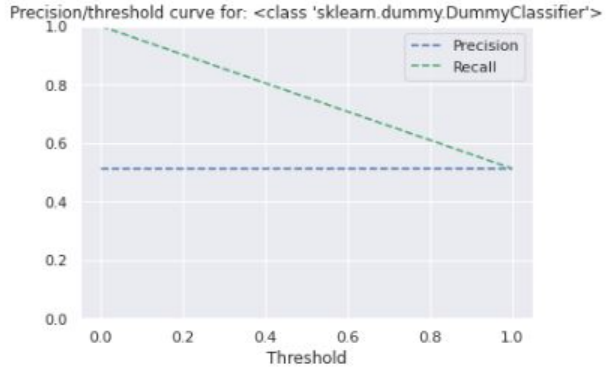
```
from sklearn.metrics import precision_recall_curve
def plot_pr_vs_threshold(model, precisions, recalls, thresholds):
    tt = 'Precision/threshold curve for: ' + str(type(model))
    plt.title(tt)
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
    plt.plot(thresholds, recalls[:-1], "g--", label="Recall")
    plt.xlabel("Threshold")
    plt.legend(loc="upper right")
    plt.ylim([0, 1])

def fit_model_analyse(model, X, y):
    model.fit(X, y)
    y_scores = model.predict(X)
    precisions, recalls, thresholds = precision_recall_curve(y, y_scores)
    plt.title('')
    plot_pr_vs_threshold(model, precisions, recalls, thresholds)
    plt.show()

    fpr, tpr, thresholds = roc_curve(y, y_scores)
    plot_roc_curve(model, fpr, tpr)
    plt.show()
```

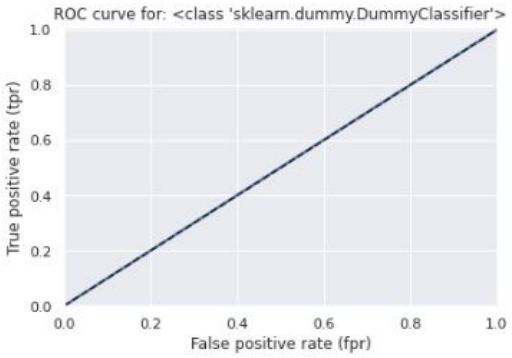
3.2. Baseline

Let's check a dummy classifier first and to have a baseline model to compare to. This also allows me to verify correctness of my plotting and measurement functions, as I know what output to expect.

```
from sklearn.dummy import DummyClassifier
dummy_clf = DummyClassifier()
run_cross_validation(dummy_clf, X_train, y_train, 5)
fit_model_analyse(dummy_clf, X_train, y_train)
```

```
Running cross-validation for:  <class 'sklearn.dummy.DummyClassifier'>
Av Accuracy:  0.4988026362999697
[0.49712948 0.49920683 0.50207735 0.50260613 0.49299339]
Av F1:  0.5150121544845472
[0.51490872 0.51430467 0.51149743 0.5136608  0.52068915]
```



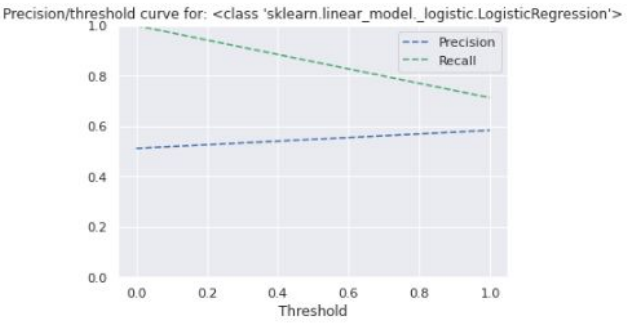


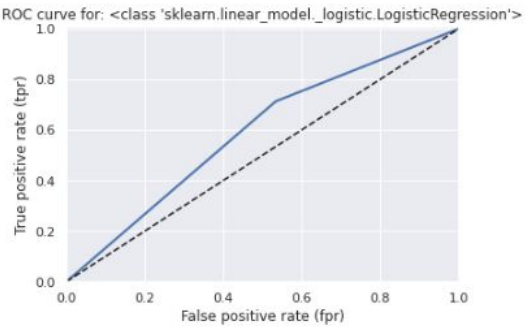
3.3. Logistic Regression models

Default Logistic Regression model performs a bit better than baseline.

```
from sklearn.linear_model import LogisticRegression
simple_log_reg = LogisticRegression()

run_cross_validation(simple_log_reg, X_train, y_train, 5)
fit_model_analyse(simple_log_reg, X_train, y_train)
```

```
Running cross-validation for:  <class 'sklearn.linear_model._logistic.LogisticRegression'>
Av Accuracy:  0.6079212074548318
[0.60775797 0.6069648  0.60609609 0.60934431 0.60944287]
Av F1:  0.637338665500496
[0.63781258 0.63696623 0.63421136 0.63738036 0.64032281]
```





I've tried improving this with the usage of pipelining it together with preprocessing of PolynomialFeatures(), and it did the job. For the 3rd degree polynomial, I obtained 87% accuracy!

```
run_cross_validation(lin_reg_poly, X_train, y_train, 5)
fit_model_analyse(lin_reg_poly, X_train, y_train)
```

```
Running cross-validation for:  <class 'sklearn.pipeline.Pipeline'>
Av Accuracy:  0.8727366218034014
[0.87381024 0.87290376 0.87177066 0.87120411 0.87399433]
Av F1:  0.876808761216855
[0.87754279 0.87705068 0.87581843 0.87600902 0.87762289]
```

3.4. Naive Bayes model

There were two Naive Bayes models we have encountered during the practicals. I've started with Gaussian NB, as it works for negative values in the dataset as well.

```
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()
run_cross_validation(gnb, X_train, y_train, 5)
fit_model_analyse(gnb, X_train, y_train)
```

```
Running cross-validation for:  <class 'sklearn.naive_bayes.GaussianNB'>
Av Accuracy:  0.6028750700654153
[0.59786221 0.60156368 0.60594501 0.60722919 0.60177526]
Av F1:  0.6456818572713002
[0.64449564 0.64539985 0.64550984 0.64668909 0.64631487]
```

These are similar values to the simple LinearRegression model. Perhaps both of these are too simple to capture the richness of this data.

3.5. Support Vector Machines

Let's try a classification based on SVMs, implemented in sklearn under the name SGDClassifier.

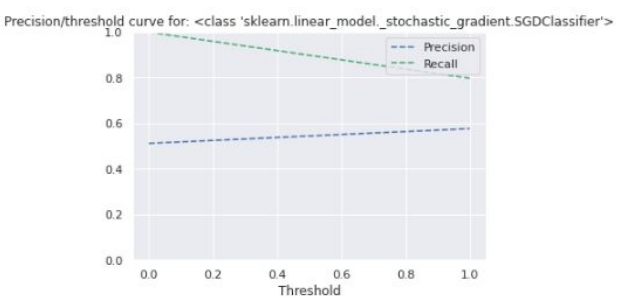
```
from sklearn.linear_model import SGDClassifier

simple_sgd = SGDClassifier(max_iter=700, random_state=42, loss="perceptron", eta0=1, n_jobs=-1, penalty=None)
simple_sgd.fit(X_train, y_train)
y_scores = simple_sgd.predict(X_train)
y_scores, len(y_scores), sum(y_scores), sum(y_train)
```

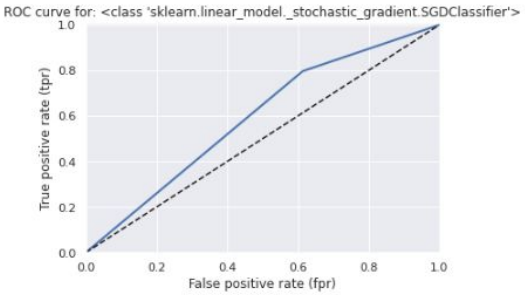
```
(array([0, 1, 1, ..., 0, 1, 1]), 132379, 93697, 67756)
```

```
run_cross_validation(simple_sgd, X_train, y_train, 3)
fit_model_analyse(simple_sgd, X_train, y_train)
```

```
Running cross-validation for:  <class 'sklearn.linear_model._stochastic_gradient.SGDClassifier'>
Av Accuracy:  0.5559109697301787
[0.61764906 0.52930245 0.5207814 ]
Av F1:  0.6516815665755976
[0.6010593  0.67698289 0.67700251]
```

Accuracy worse than previously, not good. At least the ROC curve looks somewhat different than previously.





Before we move on to the model which actually captures the data well, let's try one more variation - kernel approximation for the SVM. I've tried various values of the gamma hyperparameter, these improve a bit on previous results.

```
from sklearn.kernel_approximation import RBFSampler
sgd_kernel = Pipeline([('rbf', RBFSampler(gamma=1, random_state=42)),
                       ('sgd', SGDClassifier(max_iter=700, random_state=42, loss="perceptron", eta0=1, n_jobs=-1, penalty
sgd_kernel.fit(X_train, y_train)
y_scores = sgd_kernel.predict(X_train)
y_scores, len(y_scores), sum(y_scores), sum(y_train)
```

```
(array([0, 0, 0, ..., 0, 0, 1]), 132379, 24262, 67756)
```

```
run_cross_validation(sgd_kernel, X_train, y_train, 3)
fit_model_analyse(sgd_kernel, X_train, y_train)
```

```
Running cross-validation for:  <class 'sklearn.pipeline.Pipeline'>
Av Accuracy:  0.6779399251165227
[0.66859292 0.65204188 0.71318497]
Av F1:  0.6869646483952118
[0.61164223 0.71182432 0.73742739]
```

3.6. Decision Trees

Decision Trees make powerful, interpretable models, and they are simple and understandable to humans. Additionally, recent EU laws demand the retailers to be able to explain their ML algorithms' decisions, so investigating these in a greater detail sounds like a good idea.
To prevent overfitting, I am constraining the model's depth and stop the process when the impurity decreases are negligible.

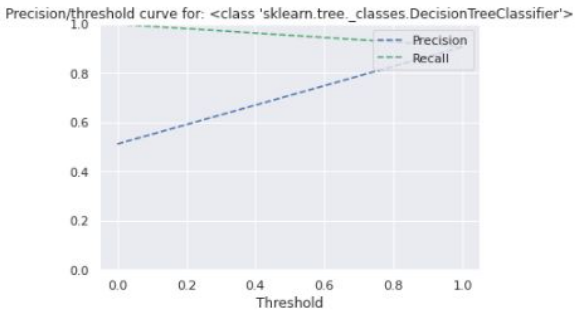
```
from sklearn.tree import DecisionTreeClassifier

dec_tree = DecisionTreeClassifier(random_state=42, max_depth=15, min_impurity_decrease=0.0001)

dec_tree.fit(X_train, y_train)
y_scores = dec_tree.predict(X_train)
y_scores, len(y_scores), sum(y_scores), sum(y_train)
```

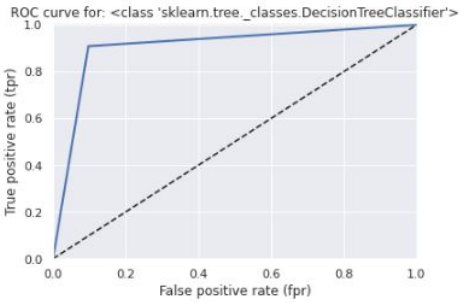
```
(array([0, 1, 1, ..., 1, 1, 1]), 132379, 65756, 67756)
```

```
run_cross_validation(dec_tree, X_train, y_train, 3)
```

```
Running cross-validation for:  <class 'sklearn.tree._classes.DecisionTreeClassifier'>
Av Accuracy:  0.9219060399224169
[0.92224715 0.92102162 0.92244935]
Av F1:  0.922724702414221
[0.92291793 0.92171178 0.92354439]
```

This looks heartwarming. Let's also plot the curves to see if they look equally good.





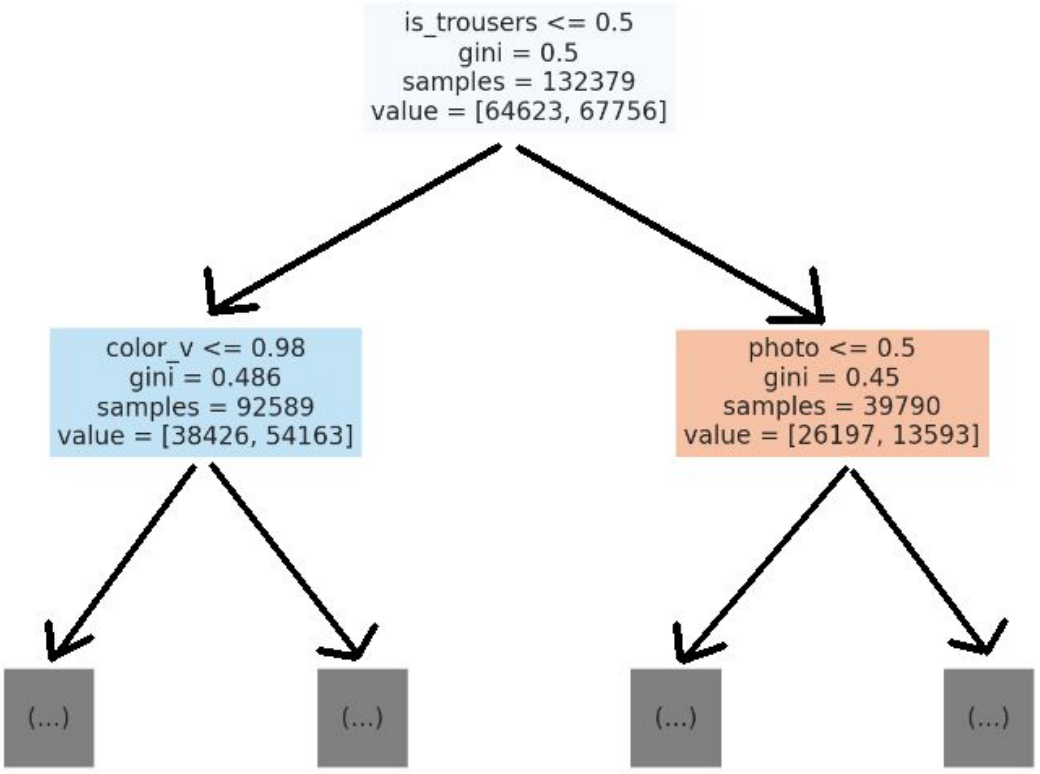
These results make me confident that the Decision Trees are a solid basis for further experiments with the dataset. Let's investigate which features are considered important.

```
feature_importances = dec_tree.feature_importances_
feature_importances
attributes = list(X_train_pd.columns)
sorted(zip(feature_importances, attributes), reverse=True)
```

```
[(0.2276715834814278, 'page'),
 (0.19378796025921535, 'loc_x'),
 (0.14983054111368604, 'loc_y'),
 (0.07385009904432138, 'photo'),
 (0.07164635180667951, 'color_s'),
 (0.06531675363995405, 'color_v'),
 (0.05333875453335018, 'session_id'),
 (0.04983256823617723, 'is_trousers'),
 (0.02764508626457631, 'is_skirts'),
 (0.02359280186618278, 'is_blouses'),
 (0.023196583023703397, 'order'),
 (0.018901232665693577, 'color_h'),
 (0.017871477165771355, 'is_sale'),
 (0.0035182068992609308, 'country')]
```

Unsurprisingly, features related to position of the product on the webpage (page and location) are the most meaningful. According to the classifier, good photo and the model of the product are not of primary importance. A sales lesson here - expensive products need exposure to clients.

This decision tree is huge, but let's look at the top of it to see what the start of the decision process looks like. Interestingly, the very first question is: "Are we classifying trousers?".



Let's improve the Decision Tree model now by tuning the appropriate hyperparameters. I've started with RandomSearchCV technique, but the results were worse than my beginning random gues. Therefore, I needed to use more computationally expensive GridSearchCV.

```
parameters = {'criterion':('entropy', 'gini'),
              'splitter':('best','random'),
              'max_depth':np.arange(10, 20),
              'min_impurity_decrease': [0.1, 0.01, 0.001, 0.0001],
              'min_samples_leaf':np.arange(1,5)}
```

```
from sklearn.model_selection import GridSearchCV
rnd_search = GridSearchCV(DecisionTreeClassifier(), parameters, cv=5, n_jobs=-1)
```

```
rnd_search.fit(X_train, y_train)
best_parameters_tree = rnd_search.best_params_
```

```
best_parameters_tree
```

```
{'criterion': 'entropy',
 'max_depth': 16,
 'min_impurity_decrease': 0.0001,
 'min_samples_leaf': 1,
 'splitter': 'best'}
```

```
improved_tree = DecisionTreeClassifier(**best_parameters_tree)
run_cross_validation(improved_tree, X_train, y_train, 5)
```

```
Running cross-validation for:  <class 'sklearn.tree._classes.DecisionTreeClassifier'>
Av Accuracy:  0.9231826908001274
[0.92415773 0.92298686 0.92200483 0.92215591 0.92460812]
Av F1:  0.9241484452951916
[0.92493458 0.92400015 0.9226099  0.92343698 0.92576062]
```

The fine-tuned Decision Tree consistently achieves more than 92% accuracy and F1-score on all 5 cross-validation runs.

3.7. Random Forests.

Since the Decision Tree performs well on this task, we should consider applying other techniques which are based on it. One is the ensemble model based on Decision Trees - a Random Forest. I've again used basic hyperparameter tuning to find good parameters.

```
tuned_forest = RandomForestClassifier(max_features=4, n_estimators=30, n_jobs=-1)
run_cross_validation(tuned_forest, X_train, y_train, 3)
```

```
Running cross-validation for:  <class 'sklearn.ensemble._forest.RandomForestClassifier'>
Av Accuracy:  0.913747641425222
[0.91472341 0.91154875 0.91497077]
Av F1:  0.9164059727011358
[0.91765643 0.9145473  0.91701419]
```

These are slightly worse results than one Decision Tree. Let's see if more expensive tuning can outperform it.

```
rf_tuned = RandomForestClassifier(**rf_random.best_params_)
run_cross_validation(rf_tuned, X_train, y_train, cv_param=3)
```

```
Running cross-validation for:  <class 'sklearn.ensemble._forest.RandomForestClassifier'>
Av Accuracy:  0.919677576722799
[0.92174859 0.91716902 0.92011512]
Av F1:  0.922348802054963
[0.92388867 0.92068511 0.92247263]
```

After half an hour of waiting, I have got similar, though a bit worse numbers for the tuned forest. Perhaps the classifiers have not been diverse enough to benefit.

3.8. Boosting - XGBoost

Decision Trees outperformed other techniques dramatically, so XGBoost implementation is going to be my choice. It has the reputation of one of the best classifiers according to kaggle community, so the expectations are high.

```
import xgboost as xgb
xgb_model = xgb.XGBClassifier(objective="binary:logistic", random_state=42)
xgb_model.fit(X_train, y_train)

run_cross_validation(xgb_model, X_train, y_train, cv_param=3)
```

```
Running cross-validation for:  <class 'xgboost.sklearn.XGBClassifier'>
Av Accuracy:  0.9193678705956597
[0.92002629 0.91807551 0.92000181]
Av F1:  0.9215069388224286
[0.92205067 0.92029545 0.9221747 ]
```

This looks promising, as I haven't trained any hyperparameters yet.

```
def report_best_scores(results, n_top=3):
    for i in range(1, n_top + 1):
        candidates = np.flatnonzero(results['rank_test_score'] == i)
        for candidate in candidates:
            print("Model with rank: {0}".format(i))
            print("Mean validation score: {0:.3f} (std: {1:.3f})".format(
                  results['mean_test_score'][candidate],
                  results['std_test_score'][candidate]))
            print("Parameters: {0}".format(results['params'][candidate]))
            print("")
```

```
report_best_scores(search.cv_results_, 1)
```

```
Model with rank: 1
Mean validation score: 0.920 (std: 0.001)
Parameters: {'colsample_bytree': 0.9406590942262119, 'gamma': 0.03727532183988541, 'learning_rate': 0.3260660809801
552, 'max_depth': 5, 'n_estimators': 100, 'subsample': 0.679486272613669}
```

```
xgb_best = xgb.XGBClassifier(**search.best_params_)
```

```
run_cross_validation(xgb_best, X_train, y_train, cv_param=5)
```

```
Running cross-validation for:  <class 'xgboost.sklearn.XGBClassifier'>
Av Accuracy:  0.9205463212108217
[0.92264693 0.92083396 0.91973863 0.9175102  0.92200189]
Av F1:  0.9226672417200781
[0.92465048 0.9229978  0.92152591 0.92000586 0.92415617]
```

If more computing power was available, maybe I would have been able to tune the boosting classifier better, but in these circumstances - Decision Tree wins.

3.9. Neural Networks

This task is not an image classification task, nor natural language processing, so I am only testing a traditional feedforward neural network. I've used several hidden Dense layers as follows:

```
model = keras.Sequential()
model.add(Dense(500, input_dim=N_FEATURES))
model.add(Activation('relu'))
model.add(Dense(100))
model.add(Activation('relu'))
model.add(Dense(50))
model.add(Activation('relu'))
model.add(Dense(2))
model.add(Activation('softmax'))
```

The final iteration gives around 92.2% accuracy as well. I haven't used cross-validation or complicated hyperparameter tuning, because of expensive computational cost, just ran a few experiments by hand and selected the best option.

```
Epoch 40/40
4137/4137 [==============================] - 9s 2ms/step - loss: 0.1336 - accuracy: 0.9226
```

4. Evaluation of the final models.

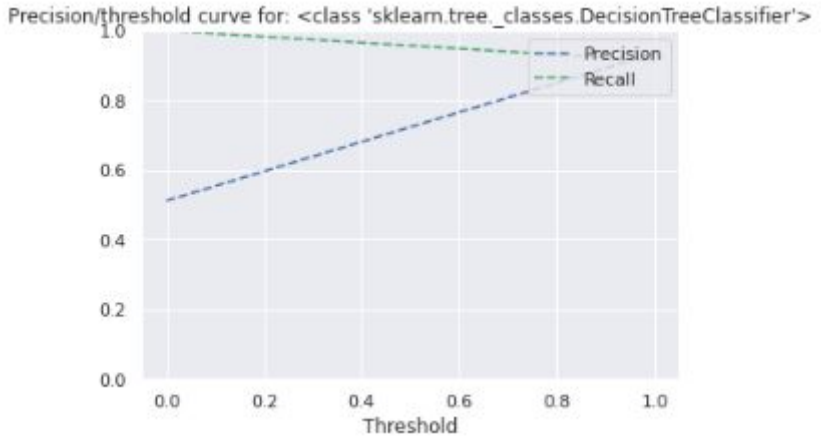
I've selected the final versions of the DecisionTree classifier, the XGBoost classifier and the neural network for evaluation on the test set. All these achieve between 92% and 93% accuracy and F1_score for the test data.

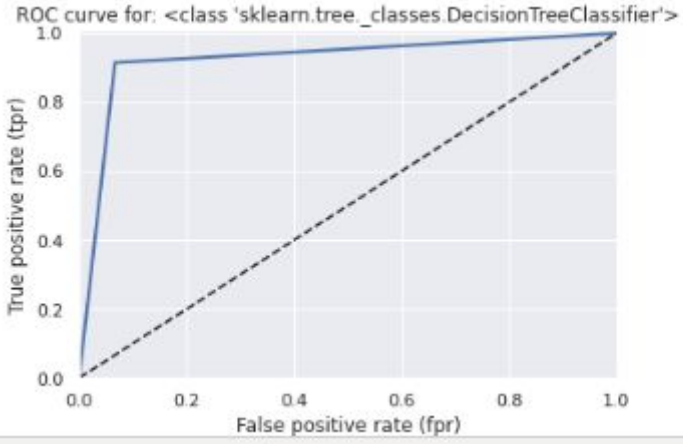
If I were to pick one model to present to the shop owner, it would be the first one. It doesn't require much computing power for predictions, and its interpretability will help in business decisions. Let's look at its performance on the test set.

I am displaying the scores, Pr/Threshold and ROC curves and the normalized confusion matrix. I decided not to plot the confusion matrix, because any color scale for displaying just two different values could be potentially misleading for the reader.

```
improved_tree.fit(X_train, y_train)
final_evaluate(improved_tree, X_test, y_test)
fit_model_analyse(improved_tree, X_test, y_test, fit_model=False)
```

```
Confusion matrix:
[[0.93339936 0.06660064]
 [0.08560128 0.91439872]]
Accuracy: 0.9236742710379211, F1: 0.9246060171919771
```





4.1. Final remarks

If students were expected to spend more time on the assignment, I would have also used the dimensionality-reduced version of the dataset to speedup neural networks training to allow hyperparameter tuning in reasonable time.

I'm also mildly disappointed with XGBoost's results. I expected more from the tuned version of the kaggle's community favourite algorithm.

There are other algorithms, techniques and possibilities, all of which couldn't have been possibly tried in a 24-hour period, and a limited report space.